PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

Structural Kernels and Neural Network Models for Question Answering Systems

Massimo Nicosia

Advisor:

Prof. Alessandro Moschitti

Università degli Studi di Trento

April 2018

Abstract

Tree kernels and neural networks are powerful machine learning models for extracting patterns from data. Tree kernels compute the similarity between two tree-structured text representations that may incorporate syntactic and semantic information. Neural networks map words into informative embeddings, and learn complex non-linear decision functions by applying a number of transformations to the input. Joining the two approaches is an exciting research direction. In this work, which is set in a Question Answering (QA) context, we apply the individual models to classification and ranking tasks. More importantly, we explore the intersection of tree kernels and neural networks, with the goal of developing more accurate models.

Initially, we focus on a challenging QA task, the resolution of Crossword Puzzles (CPs), and improve an automatic CP solver by tackling two problems: (i) answering crossword clues by reranking snippets from a search engine, and (ii) clue paraphrasing, which is extremely useful for finding clues with the same answers. We apply reranking models based on syntactic structures, and therefore tree kernels, to increase the accuracy and speed of the solver. In addition, we design and evaluate a composite kernel that combines a kernel over structures, and a kernel on neural network induced representations.

Going beyond the neural feature vector approach, we develop a structural kernel that exploits a deep siamese network for evaluating the similarity between words. We assess the resulting model on two classification tasks: question classification and sentiment analysis.

To conclude, we study QA models that establish links between question and candidate answer passages using semantic information. First, we present our tree kernel model for answer sentence selection, which captures relations between important question words and entities in the answer. Then, we build a neural network model that can be trained to extract semantic features from text, and eventually establish links between text pairs. We show that such network is able to better model the notion of question-answer relatedness on several QA datasets, compared to the tree kernel model.

Keywords Question Answering, Tree Kernels, Neural Networks.

Acknowledgements

The past years in academia have been the most important of my life. During this time, I have learned countless valuable lessons that greatly contributed to what I am now. I was lucky to meet a lot of amazing people on my path. Everyone of them taught me something or inspired me to pursue my goals and ambitions. I have been also able to travel to incredible places, attend conferences, and enjoy a freedom that is not easily attainable in other situations.

I would like to thank all the people encountered during this journey. First of all, my advisor Alessandro Moschitti, who helped me navigate through the time spent in doing research; for his advices, the trust put in me, and for his precious mentorship and support. Giovanni Semeraro, who sparked my interest in machine learning and natural language processing with his course at the University of Bari. The people at the Qatar Computing Research Institute, and in particular Preslav Nakov and Lluis Marquez. I would like to thank the extraordinary researchers who hosted me at Google: Katja Filippova, Enrique Alfonseca and Aliaksei Severyn, during my first and third crucial internships at Google Zurich; Bernd Bohnet, Ryan McDonald and Michael Collins, with whom I had the pleasure of working at Google London, during my second intership. A shout-out is for my Google Zurich friends Francesco and Antonio: I cannot wait to join you in July and be your colleague again. An additional thank you goes to Aliaksei, who was not only a superb host, but also an amazing friend and collaborator on several joint papers during our overlapping time in Trento. Here, I was grateful to be part of the iKernels group and to share time and adventures with great friends, students and researchers. Thanks Antonio, Gianni, Irina, Daniele, Olga, Katya, Azad and Liah.

I would like to express my gratitude to my family for its infinite love and support: my special mum, dad, Fabio, Manu, Enzo, and my little nieces Sofia and Gaia, pure twinkles of joy. I hope to always make you proud. I also cheer my friends in Monopoli, who were always happy to see me when I was back. The last thank you is for Clelia. Without your love and caring this journey would have not been so colorful and happy.

Contents

List of Tables

List of Figures

List of Acronyms

API	Application Programming Interface
BOW	Bag Of Words
BiMPM	Bilateral Multi Perspective Matching
CBOW	Continuous Bag Of Words
CNN	Convolutional Neural Network
CP	Crossword Puzzle
CPDB	Crossword Puzzle Database
CWDB	Cross Word Database
DB	Database
DLM	Deep Learning Model
DNN	Deep Neural Network
FC	Focus Classifier
GRU	Gated Recurrent Unit
IR	Information Retrieval
LCT	Lexical Centered Tree
LD	Large Dataset
LDC	Lexical Decomposition
LGR	Logistic Regression
LSTM	Long Short Term Memory
LTR	Learning To Rank
MAP	Mean Average Precision
MRR	Mean Reciprocal Rank
NBOW	Neural Bag Of Words
NE	Named Entity
NER	Named Entity Recognition
NLP	Natural Language Processing
NSPTK	Neural Smoothed Partial Tree Kernel
POS	Part Of Speech
PTK	Partial Tree Kernel
QA	Question Answering
QC	Question Classification
QF	Question Focus
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network

SC Sentiment Classification

SCR Similar Clue Retrieval

SD Small Dataset

SE Search Engine

SPTK Smoothed Partial Tree Kernel

STK Syntactic Tree Kernel

STS Semantic Textual Similarity

SVM Support Vector Machine

TK Tree Kernel

TS Text Snippet

UIMA Unstructured Information Management Applications

cQA Community Question Answering

q/a question/answer

Chapter 1

Introduction

In this section, we present the motivations behind this work, and describe our contributions chapter by chapter, including references to the corresponding publications.

1.1 Motivations

Many Natural Language Processing (NLP) tasks require the understanding of relations between multiple pieces of text. For example in automated Question Answering (QA), the question is put in relation with text fragments that are retrieved from a document collection, usually by a search engine. These fragments may or may not contain the answer to the question, and a machine learning system must determine to what extent this fact is true for a given passage, in order to provide relevant information to a user.

This and other complex tasks often require the manual definition of rules which capture syntactic and semantic patterns useful to characterize the relations between pieces of text. One way to do this is to compute similarities between text fragments, but that does not address the specificity of the given NLP task. Usually, the most accurate methods include the manual design of specialized rules that are triggered when patterns in the pieces of text are found. These rules exploit the output of NLP processors such as parsers. The number and the complexity of rules required for making accurate decisions can be very high, and this leads to systems that are hard to mantain and difficult to develop.

While previous research showed that such systems are effective, they suffer of major drawbacks: (i) being based on heuristics they do not provide a definitive methodology, since natural language is too complex to be characterized by a finite set of rules; and (ii) given the latter claim, new domains and languages require the definition of specific rules, which causes a large effort in engineering NLP systems.

An alternative to manual rule definition is the use of machine learning, which often shifts the problem to the easier task of feature engineering. This is very convenient for simple text categorization problems, such as document topic classification, where simple bag-of-words (BOW) models have been shown to be very effective, e.g., Yang [1999]; Joachims [1998]. Treating words as atomic units has evident limitations. Although this approach is simple, robust and scalable, it does not encode any notion of similarity and it ignores the context in which words appear, unless n-grams are considered. Even so, this only allow the presence of co-occurrent words to be signalled.

When the learning task is more complex, as in QA, features have to encode the combination of syntactic and semantic properties, which basically assume the shape of high-level rules: these are essential to achieve state-of-the-art accuracy. For example, the famous IBM Watson system Ferrucci et al. [2010] also includes a learning to rank algorithm fed with hundreds of features. Some of the latter are extracted with articulated rules, which are very similar to those that can be found in typical manually engineered QA systems.

In order to avoid recurrent engineering efforts, we develop methods focused on automatic feature engineering. In this work, we seek to explore and eventually combine two state-of-the-art approaches for automatic feature engineering: deep neural networks (DNNs) and structural kernels. The former learn to represent text by generating informative embeddings and by combining them. The latter capture similarities between structural representations of text which include syntactic and semantic information. Since the latest results in kernel technology enable the use of semantic similarities between the dimensions of the substructure space, and thus the possibility to integrate the output of DNNs, there are exciting potential research directions in merging the two approaches. In particular, we aim at using DNNs for learning the representations of words and the substructures that they form when combined with syntactic and semantic elements. We want to construct pipelines that take advantage of the automatic feature engineering capabilities of these methods and make them usable in a unified framework.

1.2 Contributions and Structure of the Thesis

In Chapter 2, we introduce the reader to the general concepts which recur in this thesis. Support Vector Machines (SVMs) and kernel methods are at the core of the supervised learning setup of most of our contributions. We also give a formal definition of the structural kernels that we use to tackle the two typologies of NLP tasks in the thesis: reranking and text classification.

In Chapter 3, we give a brief overview of distributional representations for words and neural network architectures. Among those, we describe the siamese network architecture, which give us the opportunity to present the first contribution of this thesis.

Contribution 1 (Chapter 3). *A Hybrid Siamese Network with a multi-loss setup for detecting question similarity and entailment/contradiction relations.*

We describe a siamese network architecture that uses a multi-loss setup, which improves over single-loss setups [Nicosia and Moschitti, 2017a]. This is achieved with less computations than more sophisticated methods, and obtaining strong performances. A loss on the euclidean distance between two encoded sentences acts as a regularizer on their representations, while an additional logistic loss models their interaction.

In Chapter 4, we show that structural representations and kernels are powerful methods for paraphrasing and for answer candidate selection. We apply our models to the automatic resolution of Crossword Puzzles (CPs), which can be seen as an untraditional QA instantiation. We tackle two tasks with the goal of improving the automatic resolution of this challenging linguistic game: the retrieval of similar clues, and the selection of text snippets containing the answer to a target clue. The first task is an instantiation of the question-question similarity problem, while the second is the classical candidate answer passage selection task. Here we detail the contributions from this chapter.

Contribution 2 (Chapter 4). *State-of-the-art retrieval of similar clues and answer passage reranking, resulting in an increase of accuracy and speed for the WebCrow crossword solver, which has been extended with the inclusion of our modules.*

State-of-the-art automatic CPs solvers typically contain modules that given a clue (i) search a database of previously solved puzzles for similar clues which may have the same answer; (ii) retrieve search engine snippets using the clue as query, and determine which of the former may contain the answer. We use tree kernels on structures to rerank clue-clue and clue-snippet pairs, and we feed our reranked list to the CP solver. Our components greatly increase the quality of the solver inputs, which results in more accurate CP completion, and shorter execution time [Barlacchi et al., 2014a,b; Nicosia et al., 2015; Barlacchi et al., 2015].

Contribution 3 (Chapter 4). *UIMA pipeline for the NLP analysis of documents and for the reranking of query-passage pairs.*

The system developed for the reranking of text pairs is based on the UIMA framework, which is also the foundation upon which the famous IBM Watson system is built. Our software artifact [Tymoshenko et al., 2017] has pushed the collaboration in our group and within other groups, resulting in several publications and strong results in semantic evaluation (SemEval) competitions.

In Chapter 5, we conduct our experiments on the interaction between tree kernel and neural network models. We first focus on neural networks as feature extractors, and study the inclusion of such features in our kernel-based classifiers. After this initial exploration, we move into incorporating word representations learned with neural networks directly in the computation of the kernel similarity.

Contribution 4 (Chapter 5). *State-of-the-art retrieval of similar clues for the resolution of CPs, and exploration of the effectiveness of neural network features combined with structural kernels.*

We again tackle the task of retrieving and reranking similar crossword clues, studying tree kernels and deep neural models [Severyn et al., 2015]. This time, we train a neural network on a large dataset of clue pairs. Our network is computationally efficient and can take advantage of the large amount of data, showing a large improvement over the tree kernel approach when the latter uses small training data. In contrast, when data is scarce, tree kernels outperform the network. We also study the interaction of tree kernels and network features, by extracting the activations of the latter, and feeding them into a multiple SVM kernel classifier. Since we use structural representations of text, we effectively combine a kernel on trees and a kernel over the network features.

Contribution 5 (Chapter 5). *Novel semantic kernel that computes the kernel similarity considering neural representations of words, which are optimized for the end task and also include context information.*

We combine tree kernels and neural networks by modeling context word similarity in semantic tree kernels [Nicosia and Moschitti, 2017b]. This way, the latter can operate subtree matching by applying neural-based similarity on tree lexical nodes. We study how to learn representations for the words in context such that tree kernels can exploit more focused information. We found that neural embeddings produced by current methods do not provide a suitable contextual similarity. For this reason, we define a new approach based on a siamese network, which produces word representations while learning a binary text similarity. We define the latter considering examples in the same category as similar. The experiments on question and sentiment classification show that our semantic tree kernel highly improves previous results.

In Chapter 6, we automatically learn complex patterns between pieces of text, such as relational semantic structures that occurr in questions and their answer passages. We focus on convolutional models: convolution tree kernels and convolutional neural networks. We provide the tree kernel with tree representations built from the syntactic trees of questions and passages, after connecting them with relational tags determined by automatic classifiers, such as question and focus classifiers, and Named Entity Recognizers (NERs). This way, effective structural relational patterns are implicitly encoded in the representation, and can be automatically utilized by powerful machine learning models such as kernel methods. In addition to this, we propose a fully convolutional neural network system that encodes the same information of the structural model, but has two important advantages: it is much faster to train and reaches superior performances.

Contribution 6 (Chapter 6). *Semantic links between question and answer passages increase the accuracy of convolutional models. A fully convolutional neural network model shows improved accuracy and shorter training time than its counterpart based on kernels on structural representations.*

We focus on the task of answer sentence selection and propose an SVM model based on a kernel over structural representations. We encode questions and answers using trees containing words, syntactic information, and semantic information in the form of tags. The tags are used to mark the question focus word, and the named entities in the answer. Additionally, tags are typed with the category of the question. We also design a neural network architecture that establishes links between question and answer at the word level. Altough the kernel classifier is very powerful, we show that modern neural network approaches encoding similar information are able to better model the question/answer pair, together with the notion of relevant answer passage.

Chapter 2

Support Vector Machines and Kernel Methods

In this chapter, we describe the concepts regarding supervised learning with Support Vector Machines (SVMs) and kernels. This background will be useful for proceeding to the contribution chapters. We give a general definition of supervised learning and then proceed to describe the main components of discriminative training. After that, we introduce classification and reranking, contextualizing them with respect to the QA tasks tackled in the thesis. We characterize the SVM learning algorithm in its formulations, and finally describe its kernelization, together with the structural kernels widely used in our models.

2.1 Supervised Learning

Supervised learning or predictive learning is the process of learning a function that maps input data into some output. More specifically, given a training set $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{n}$, the goal is to learn a decision function $g \in \mathcal{G}$ from a function space $\mathcal{G}$, usually called the hypothesis space, that maps an input $x \in \mathcal{X}$ to an output $y \in \mathcal{Y}$: $g : \mathcal{X} \rightarrow \mathcal{Y}$. The input x_i may be a D-dimensional vector of numbers, or a structured object. The output y_i could be anything, but most methods treat it as a categorical variable from some finite set, $y_i \in \{1, ..., C\}$, or a real valued scalar, yielding respectively a classification or regression problem. In the classification case the decision function g is specified as follows:

$$g(\mathbf{x}) = \underset{\mathbf{y} \in \mathcal{Y}}{\text{argmax}} f_{\mathbf{w}}(\mathbf{x}, \mathbf{y}), \tag{2.1}$$

where $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ is a discriminant function that, given an input instance $\mathbf{x}$ and a class $\mathbf{y}$, outputs a numerical score. Such function is parametrized by a vector $\mathbf{w}$, which is learned during training. The function f is usually chosen to be linear in the weight vector $\mathbf{w}$:

$$f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle, \tag{2.2}$$

where $\langle \cdot, \cdot \rangle$ is a dot product, and $\Psi : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d$ maps each instance and class into features in $\mathbb{R}^d$.

2.1.1 Empirical Risk Minimization

One of the basic approaches for choosing f is the Empirical Risk Minimization. The function f is determined by the model parameters $\mathbf{w}$, which have to be estimated. A machine learning algorithm finds the function that better fits the training data — finds good values for $\mathbf{w}$ — by minimizing a task-dependent loss function L, computed on the outputs of f and the actual labels associated with the training instances. This process is formalized by the Empirical Risk Minimization:

$$\hat{g} = \arg\min_{g \in \mathcal{G}} R(g) = \arg\min_{g \in \mathcal{G}} \frac{1}{n} \sum_{n=1}^{n} L(g(\mathbf{x_i}), \mathbf{y_i}) \tag{2.3}$$

2.1.2 Loss Function

A loss function $L : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ (or cost function) maps an event into a numerical cost associated with that specific event. The latter is typically a prediction from the model, and the cost is computed by comparing the prediction with the desired event.

A common loss used for "maximum-margin" classification (such as SVMs) is the hinge loss, the closest convex approximation of the standard zero-one loss used in classification. In the most general form the hinge loss can be defined as:

$$l_{hinge}(\mathbf{t}, \mathbf{y}) = \max(0, \max_{y \neq t} (\Delta(\mathbf{y}, \mathbf{t}) + \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{y}) \rangle) - \langle \mathbf{w}, \Psi(\mathbf{x}, \mathbf{t}) \rangle), \tag{2.4}$$

where $\Delta(\cdot, \cdot)$ measures the difference between the predicted output $\mathbf{y}$ and the true output $\mathbf{t}$. According to this definition, the true output $\mathbf{t}$ should have a higher score than any other predicted output $\mathbf{y}$ by at least the quantity specified by $\Delta(\mathbf{y}, \mathbf{t})$.

2.1.3 Discriminative Training

Given the previously defined concepts, we can summarize the main components of discriminative training.

1. The input space, which contains the training and test data represented as feature vectors obtained after applying a joint feature map $\Psi(\cdot, \cdot)$. The design of ψ is of great

importance because it encompasses the feature engineering effort required to extract from the input instances expressive features which directly affect the accuracy of the discriminative model.

2. The output space, which depends on the task — for example -1 or +1 in the case of binary classification, or a confidence value about the class membership of an instance.

3. The hypothesis space, containing the functions which map a point from the input to the output space.

4. The loss function, which is a function mapping a prediction generated by the hypothesis onto a real number representing the cost associated to that prediction, i.e., a penalty for incorrectly classifying examples.

5. An inference process which assigns an output label to an input.

6. An optimization algorithm that learns the parameters of a model by minimizing the empirical risk.

2.2 Supervised Learning Problems in NLP

In this section, we give an overview of the classes of NLP tasks for which we train supervised machine learning models in the thesis. The goal of the next parts is to familiarize the reader with the problems that will be tackled in later chapters.

2.2.1 Classification

Most of the learning problems in NLP are classification problems. In general, a classification problem consists in mapping an input instance $\mathbf{x}$ into an output y belonging to one of K possible classes. Classification can be binary or multiclass, depending on the size of the output space, which is determined by K. If $K = 2$, we have a binary classification task and a binary output space $\mathcal{Y} = \{-1, +1\}$. In the case of $K > 2$, we have a multiclass classification problem, and the output space is $\mathcal{Y} = \{1, ..., K\}$. The joint feature map for binary classification is $\Psi(\mathbf{x}) = y\Phi(\mathbf{x})$, where $\Phi : \mathcal{X} \to \mathbb{R}^d$. For the multiclass case, we have $\Psi(\mathbf{x}) = \Lambda^k \otimes \Phi(\mathbf{x})$, and Λ^k is the one-hot encoding of the k-th label y. $\otimes$ is the tensor product which selects the features corresponding to that y. In this thesis, the reader will encounter many classification problems, and we briefly describe them here.

Paraphrase Identification. Paraphrase Identification is the task of determining if two sentences have the same meaning, although they may be stated using different words. In Chapter 3, we tackle the task of question paraphrase identification: we want to establish if two questions have the same answer, and for this reason they can be marked as duplicates. This is useful in community QA (cQA) settings to quickly retrieve a response to previously answered questions.

Textual Entailment. Textual Entailment (TE) consists in determining if a sentence, called hypothesis, is true given the preceding sentence, called the premise. Typically, TE is set up as a multiclass problem, and the classes are the following: (i) *entailment*, when an hypothesis is entailed by a premise; (ii) neutral, when the hypothesis might be true given a a premise; (iii) contradiction, when the hypothesis is not true given the premise. In Chapter 3, we evaluate a neural network model on a popular TE dataset.

Question Classification. Question Classification (QC) is a question analysis step which consists in classifying a question into one of the classes from a predefined taxonomy. The category of a question can be useful to find related entities and answers, since this information suggests the nature or type of the expected answer. Knowing this enables a series of filtering steps which usually improve the accuracy of a QA system. We perform QC with tree kernel and neural network models in Chapter 5 and 6.

Sentiment Analysis. Text may express affective states and subjective information, especially in the social media domain. Sentiment analysis deals with the automatic extraction of such traits. In Chapter 5, we use Twitter messages to train a model for identifying their polarity, i.e., their emotional content, which may be positive, neutral or negative.

Question Focus Identification. Factoid questions typically contain a simple noun that corefers with the answers. Such noun is the so called question focus, and largely determines the nature of the answer. Candidate answers can be filtered according to their semantic compatibility with the focus. Our QA models use the question focus to establish links between questions and answers, as detailed in Chapter 6.

2.2.2 Reranking

The problem of reranking text pairs recurs in many IR and QA tasks. Search engine or QA systems retrieve a list of documents or paragraphs, whose order is determined with respect to a query or question, according to predefined ranking function. Such functions are designed for scaling to large document collections, and they capture syntactic similarity only to a given extent. Therefore, more complex reranking functions can be applied to the shorter result list produced by the retrieval phase.

More formally, in a supervised learning to rank (LTR) task, we have the following:

- a number of result lists (e.g. search results), where each list is composed by a query $\mathbf{q_i} \in \mathbf{Q}$ with a corresponding list of documents $\mathbf{D_i} = \{\mathbf{d}_i^1, \mathbf{d}_i^2, ..., \mathbf{d}_i^n\}$;
- each document in $\mathbf{D_i}$ has an associated list of relevancy labels $\mathbf{y_i} = \{\mathbf{y}_i^1, \mathbf{y}_i^2, ..., \mathbf{y}_i^n\}$, where a document considered relevant to $\mathbf{q_i}$ has a label equal to 1, and 0 otherwise.

The goal of the reranker is to produce a ranking $\mathbf{R} = (\mathbf{r}_i^1, \mathbf{r}_i^2, ..., \mathbf{r}_i^n)$, for a query $\mathbf{q_i}$ and the associated candidate list $\mathbf{D_i}$, such that each r_i^j is the position of the document d_i^j in the reranked list, and the latter contains all the relevant document at the top.

The ranking function has the following general form:

$$h(\mathbf{w}, \Phi(\mathbf{q_i}, \mathbf{D_i})) \rightarrow \mathbf{R}, \tag{2.5}$$

where $\mathbf{w}$ is a vector containing the model parameters, and the function $\phi(\cdot)$ featurizes a query-document pair. The ranking function h can be modeled using several approaches: the pointwise, pairwise and listwise approaches.

The **pointwise approach** does not model the result list as a whole, but it breaks it into its individual elements, namely the single documents or paragraphs. Therefore, it is a simple approach that treats reranking as a binary classification problem of query-document pairs.

The **pairwise approach** forms pairs of documents from the result list, reducing the reranking problem to pairwise classification. Given a pair of documents, the classifier decides which one is more relevant with respect to the query originating the result list.

The **listwise approach** considers the list to rank in its entirety, without breaking it into single documents or into document pairs. Therefore, it tries to minimize a loss function which compares the predicted permutation of documents with the true ranked list [Xia et al., 2008].

In the thesis the reader will find many tasks that consist in the reranking of the output of our search engines, or of the result lists provided by the authors of benchmark datasets.

Similar Clue Reranking. The task of reranking similar clues, which are definitions from crossword puzzles, is similar to the paraphrase identification task. Also in this case we want to find clues which have the same answer. The difference is that we have lists of clues that need to be reranked for a subsequent consumer component. In addition, we may also extract features from the entire clue list, making the classification decision for a clue dependant on the other clues in the list. Chapter 4 and 5 contain work on this task.

Answer Sentence Selection. Given a query constituted by a crossword clue or a question, the answer sentence selection task consists in reordering a list of sentences retrieved from a document collection by issuing the query to a search engine. The goal of the reordering is to put at the top of the list the sentences or passages which contain the answer to the clue or question. In Chapter 4, we use clues as queries for a Web search engine, and we rerank the search snippets displayed on the result pages. In Chapter 6, we evaluate our models on standard QA datasets, and rerank the provided passage lists associated to a set of questions.

2.3 Support Vector Machines

Support Vector Machines (SVMs) are among the most popular supervised learning algorithms. At the core, an SVM is a linear classifier. However, since the operations between vectorized input examples are expressed in terms of dot products, the SVM can be kernelized. Therefore, if linear separation between points is not possible in the input space, it may be possible in a higher dimensional feature space induced by the "kernel trick".

2.3.1 Primal Formulation

SVMs support binary and multiclass classification, regression, and ranking. In the binary setting the output space is $\mathcal{Y} = \{-1, +1\}$, and $\Psi(\mathbf{x}, \mathbf{y}) = y\Phi(\mathbf{x})$. If we consider linear SVMs, with the input feature map being the identity $\Phi(\mathbf{x}) = \mathbf{x}$, the discriminant function is the following:

$$f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle \tag{2.6}$$

In this case, we can take the sign of the result of f to obtain the output label: $\hat{y} = sign(\langle \mathbf{w}, \mathbf{x} \rangle)$. SVMs find a directed hyperplane dividing the space, and examples are classified according to their side with respect to the hyperplane. The selected hyperplane maximizes the separation margin between the two classes. A classifier confidence margin is the minimal distance between the classifier hyperplane and the nearest examples from both classes. Since the same hyperplane can have infinite equivalent formulation via meaningful positive rescaling, the decision hyperplane can be set in order to have $\langle \mathbf{w}, \mathbf{x} \rangle = 1$ (we omit the bias term since it can be included in the feature vector $\mathbf{x}$), for the closest points on one side, and $\langle \mathbf{w}, \mathbf{x} \rangle = -1$ for the closest points on the other side. The hyperplanes identified by these formulations are called canonical hyperplanes, and have a confidence margin equal to 1. For all correctly classified points, it holds that $\langle \mathbf{w}, \mathbf{x} \rangle \geq 1$ (points over the -1 confidence margin canonical hyperplane have the -1 label). The margin band size is equal to two times the canonical hyperplane geometric margin, $2/\left\|\mathbf{w}\right\|^2$. Selecting the best separating hyperplane amounts to maximizing such margin, while correctly classifying all the training data. This amounts to solve the following constrained optimization problem:

$$\underset{w}{\text{minimize}}\ \tfrac{1}{2}\left\|\mathbf{w}\right\|^2$$

$$\text{subject to: } y_i\langle \mathbf{w}, \mathbf{x_i} \rangle \geq 1$$

Points laying on the canonical hyperplanes are support vectors, and they contribute to the decision function. All the other points are not part of the trained model, and are not used during inference. This version of SVM has hard margin, and does not allow points to be inside the margin band. For this reason, it is sensitive to outliers which have a significant effect on finding the decision function.

A more relaxed version of SVM uses soft margins. This formulation is more effective in the presence of outliers and noise. A training data point can lie inside the margin band, or even on the wrong side of the hyperplane. During learning, non-negative slack variables are introduced and the objective tries to minimize the sum of the errors, which in addition to $\left\|w\right\|^2$, includes the sum of the slack variables. Therefore, the optimization problem becomes:

$$\underset{w,\,\xi}{\text{minimize}}\ \tfrac{1}{2}\left\|\mathbf{w}\right\|^2 + C\sum_i \xi_i$$

$$\text{subject to: } y_i\langle \mathbf{w}, \mathbf{x_i} \rangle \geq 1 - \xi_i \qquad \xi_i \geq 0$$

If $\xi_i > 0$, there is a margin error, and if $\xi_i > 1$, there is a non-zero training error. A slack variable ξ_i represents the penalty associated with the example x_i, when the latter does not satisfy the margin constraint. The regularization parameter C trades-off the margin size and data fitting. For $C \to \infty$, the cost paid for each margin error is unaffordable, and we have again hard margins.

An SVM with soft margin prefers solutions (i.e. the hyperplane) with a wider margin, given that there is a reasonable cost to pay for training points falling inside the margin band, or on the wrong side of the hyperplane. This preference leads to better generalization and more robustness.

2.3.2 Dual Formulation

The SVM optimization problem can be expressed in the dual formulation, which enables the use of kernels in the algorithm. The Representer theorem [Kimeldorf and Wahba, 1970] tells us that the **w** parameter vector can be obtained by a linear combination of the training instances:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x_i} \tag{2.7}$$

If we substitute **w** in eq. 2.6, we have:

$$f_\alpha(\mathbf{x}) = \langle \sum_i \alpha_i y_i \mathbf{x_i}, \mathbf{x} \rangle = \sum_i \alpha_i y_i \langle \mathbf{x_i}, \mathbf{x} \rangle, \tag{2.8}$$

We omit the derivation of the dual formulation from the primal, and directly state the optimization problem here:

$$\underset{\alpha \geq 0}{\text{maximize}} \sum_i \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \mathbf{x_i}, \mathbf{x_j} \rangle$$

$$\text{subject to: } 0 \leq \alpha_i \leq C$$

Regarding the primal and dual formulation, we make few observations:

- there are d parameters (dimensionality of **w**) to learn for the primal, while there is a parameter α to learn for each of the N training examples;
- if $N << d$, it is more efficient to solve for α than **w**;
- examples with $\alpha_i \geq 0$ are support vectors and are used to classify new instances; typically, alphas are sparse and the SVM model will contain a number of support vectors smaller than the number of training instances;
- the dual formulation contains dot products only between input instances, which can be replaced with a kernel.

2.3.3 The Kernel Trick

Since the SVMs operations are expressed in terms of dot products between data points, it is possible to swap the bilinear operation with another potentially non-linear feature map $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^{d'}$:

$$f_\alpha(\mathbf{x}) = \sum_i \alpha_i y_i \langle \phi(\mathbf{x_i}), \phi(\mathbf{x}) \rangle \tag{2.9}$$

Interestingly, the non-linear feature map ϕ can be computed implicitly in a single operation. Therefore, as a result of the Mercer's Theorem (see Shawe-Taylor and Cristianini [2004a]), the dot product between the mapped instances can be computed with a kernel function, applying the so called "kernel trick":

$$K(\mathbf{x_i}, \mathbf{x_j}) = \langle \phi(\mathbf{x_i}), \phi(\mathbf{x_j}) \rangle \tag{2.10}$$

K is a kernel for some ϕ if the former is a proper inner product in a given space, or, more formally, if and only if $K(\mathbf{x_i}, \mathbf{x_j})$ is positive semidefinite, i.e.:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} k(\mathbf{x}_i, \mathbf{x}_j) c_i c_j \geq 0, \forall c$$

Every function that satisfies the aformentioned properties can be used to compute a notion of similarity between two instances, which may be not only vectors but also more complex objects such as strings, trees or graphs.

The final kernelized SVM classifier would be:

$$f_\alpha(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x_i}, \mathbf{x}) \tag{2.11}$$

Choosing the appropriate kernel for a given problem requires some expertise. Some popular kernels on vectors of fixed dimensionality are:

- The Linear Kernel: $K(\mathbf{x}, \mathbf{y}) = \langle \mathbf{x}, \mathbf{y} \rangle$
- The Polynomial kernel of degree d: $K(\mathbf{x}, \mathbf{y}) = (1 + \langle \mathbf{x}, \mathbf{y} \rangle)^d$, for any $d > 0$
- The Sigmoid kernel: $K(\mathbf{x}, \mathbf{y}) = \tanh(a\langle \mathbf{x}, \mathbf{y} \rangle)$
- The Gaussian RBF: $K(\mathbf{x}, \mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||^2/2\sigma^2)$, for $\sigma > 0$

In the next section, we describe a number of more complex kernels which operate on structured objects.

2.4 Structural Kernels

Structural kernels have been successfully applied in many domains and problems , since they allow the user to incorporate some knowledge about the structure of the data in the kernel similarity computation.

This is important especially for objects whose overall similarity is a function of the similarity of their subparts. In this section, we introduce general kernels which operate on strings and trees.

2.4.1 String Kernel

The String Kernel (SK) [Lodhi et al., 2002] computes the similarity between two strings s_1 and s_2 by counting the number of subsequences that are shared between them. Some symbols in the strings may be skipped, allowing the contribution of skipgrams to the final similarity. The SK is defined by the following equation:

$$K_{SK}(s_1, s_2) = \sum_{u \in \Sigma^*} \phi_u(s_1) \cdot \phi_u(s_2) = \sum_{u \in \Sigma^*} \sum_{\vec{I}_1 : u = s_1[\vec{I}_1]} \sum_{\vec{I}_2 : u = s_2[\vec{I}_2]} \lambda^{d(\vec{I}_1) + d(\vec{I}_2)} \tag{2.12}$$

Here, $\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$ is the set of all possible strings, $\vec{I}_1$ and $\vec{I}_2$ are the two sequences of indexes $\vec{I} = (i_1, ..., i_{|u|})$, with $1 \leq i_1 < .. < i_{|u|} \leq |s|$, such that $u = s_{i_1}..s_{i_{|u|}}$, $d(\vec{I}) = i_{|u|} - i_1 + 1$ and $\lambda \in [0, 1]$ is a decay factor. This means that the i indexes range from 1 to the length of substring u, and u is shorter than the string length. $d(\vec{I})$ is the distance between the first and last character of the substring.

2.4.2 Convolution Tree Kernels

Tree kernels are powerful functions that detect the similarity of tree structures by counting the number of common subparts. The difference between tree kernels is in how the tree fragments are generated.

Convolution Tree Kernels (TKs) compute the number of substructures between two trees T_1 and T_2 without explicitly enumerating all the possible tree fragments, which would be a very expensive operation.

Let $\mathcal{T} = \{t_1, ..., t_{|\mathcal{T}|}\}$ be the set of all possible trees in the space of structures, and $\chi_i(n)$ an indicator function, which is equal to 1 if the target t_i is rooted ad a node n, and equal to 0 otherwise. We can define a general tree kernel over T_1 and T_2 as:

$$K_{TK}(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) \tag{2.13}$$

N_{T_1} and N_{T_2} are the sets of nodes of the T_1 and T_2 trees, and

$$\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{T}|} \chi_i(n_1) \chi_i(n_2) \tag{2.14}$$

computes the number of common tree fragments rooted at the n_1 and n_2 nodes. In general, the specification of eq. 2.14 determines the TK expressivity. Indeed, how the fragments are defined and obtained gives rise to a number of different tree kernels.

Syntactic Tree Kernel

The Syntactic Tree Kernel (STK) Collins and Duffy [2002] evaluates the number of common substructures as follows:

1. if the productions at n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
2. if the productions at n_1 and n_2 are the same, and n_1 and n_2 have only leaf children (they are pre-terminals) then $\Delta(n_1, n_2) = 1$;
3. if the productions at n_1 and n_2 are the same, and n_1 and n_2 are not pre-terminals then:

$$\Delta(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (1 + \Delta(c_{n_1}^j, c_{n_2}^j)) \tag{2.15}$$

where $nc(\cdot)$ is a function that returns the number of children of the argument node, and c_n^j is the j-th child of the node n. A decay factor can be introduced by modifying the steps (2) and (3) as follows[1]:

2. $\Delta(n_1, n_2) = \lambda$;
3. $\Delta(n_1, n_2) = \lambda \prod_{j=1}^{nc(n_1)} (1 + \Delta(c_{n_1}^j, c_{n_2}^j))$

The running time of STK is $O(|N_{T_1}| \times |N_{T_2}|)$, but in pratice the computational complexity tends to be linear, i.e. $O(|N_{T_1}| + |N_{T_2}|)$ for syntactic trees Moschitti and Zanzotto [2007]. The main observation on the STK is that the production rules of the grammar used to generate the tree will not be broken, i.e., children of a node are not separated.

Partial Tree Kernel

The Partial Tree Kernel (PTK) differs from the STK in the fact that it allows the children of a node to be separated to form tree fragments. This means that the production rules of the grammar generating the tree can be broken. PTK produces a greater number of fragments, and therefore it is more expressive. The Δ function of the PTK is the following:

1. if the node labels n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$; else
2. $\Delta(n_1, n_2) = 1 + \sum_{\vec{I}_1, \vec{I}_2, l(\vec{I}_1) = l(\vec{I}_2)} \prod_{j=1}^{l(\vec{I}_1)} \Delta_\sigma(c_{n_1}(\vec{I}_{1j}), c_{n_2}(\vec{I}_{2j}))$

[1] For a similarity score between 0 and 1, a normalization in the kernel space, i.e. $\frac{TK(T_1,T_2)}{\sqrt{TK(T_1,T_1) \times TK(T_2,T_2)}}$ is applied.

where $\vec{I_1} = \langle h_1, h_2, h_3, ..\rangle$ and $\vec{I_2} = \langle k_1, k_2, k_3, ..\rangle$ are sequences of indexes synchronized with the ordered child sequences c_{n_1} of n_1 and c_{n_2} of n_2. $\vec{I}_{1j}$ and $\vec{I}_{2j}$ index the j-th child in the sequence, and $l(\cdot)$ returns the length of the index list, and therefore the number of children of a node.

In the following formulation, we add two decay factors: μ accounting for the tree depth, and λ for the length of the child subsequences with respect to the original sequence, which accounts also for gaps:

$$\Delta(n_1, n_2) = \mu\Big(\lambda^2 + \sum_{\vec{I_1},\vec{I_2},l(\vec{I_1})=l(\vec{I_2})} \lambda^{d(\vec{I_1})+d(\vec{I_2})} \prod_{j=1}^{l(\vec{I_1})} \Delta(c_{n_1}(\vec{I}_{1j}), c_{n_2}(\vec{I}_{2j}))\Big) \tag{2.16}$$

where $d(\vec{I_1}) = \vec{I}_{1l(\vec{I_1})} - \vec{I}_{11} + 1$ and $d(\vec{I_2}) = \vec{I}_{2l(\vec{I_2})} - \vec{I}_{21} + 1$. The decay factors penalize larger and deeper trees, and children that are far away from each other.

Smoothed Partial Tree Kernel

The Smoothed Partial Tree Kernel (SPTK) extends the PTK and enables the computation of lexical similarities. The Δ function is defined as follows:

1. if n_1 and n_2 are leaves then $\Delta_\sigma(n_1, n_2) = \mu\lambda\sigma(n_1, n_2)$; else
2. $\Delta_\sigma(n_1, n_2) = \mu\sigma(n_1, n_2) \times \Big(\lambda^2 + \sum_{\vec{I_1},\vec{I_2},l(\vec{I_1})=l(\vec{I_2})} \lambda^{d(\vec{I_1})+d(\vec{I_2})} \prod_{j=1}^{l(\vec{I_1})} \Delta_\sigma(c_{n_1}(\vec{I}_{1j}), c_{n_2}(\vec{I}_{2j}))\Big)$,

   where σ is any similarity between nodes, e.g., between their lexical labels, and $\mu, \lambda \in [0, 1]$ are two decay factors.

SPTK has been shown to be rather efficient in practice [Croce et al., 2011, 2012].

STK$_b$ and SubTree Kernels

We briefly mention two additional tree kernels which are variations on the STK.

The **STK$_b$** kernel extends STK by allowing leaf nodes in the feature space. Since most of the time leaves in our trees are words, we adopted the subscript b to indicate the bag-of-words.

The **Subtree Kernel** (SbtK) is one of the simplest tree kernels in terms of expressive power. The reason is that it only generates complete subtrees, namely every tree fragment rooted at a given node n contains all the descendants of n.

2.5 Summary

In this chapter, we have provided the concepts required for a clearer understanding of the contributions presented later in the thesis. We gave a high level overview of supervised learning, introducing the essential concepts for training discriminative classifiers. Then, we looked at the concrete NLP problems that we will tackle in the context of text classification and reranking. We described one of the main machine learning algorithms that power our models: Support Vector Machines. We explored their primal and dual formulations, the latter being essential for exploiting the "kernel trick", and therefore making practical the use of kernels. To conclude the chapter, we outlined popular kernels, with a particular focus on kernels on structures such as tree kernels. In the next chapter, we will provide a brief overview of neural networks, and present the first contribution of the thesis: a Siamese neural network architecture for question similarity.

Chapter 3

Neural Networks for Sentence Modeling

Neural networks are a family of powerful machine learning models. During the last decade, they have been successfully applied to NLP problems, often reaching state-of-the-art results. In this chapter, we discuss the basic ideas behind those models, and how words are represented as input to neural networks. Then, we describe the main network architectures that the reader will find in our contributions, from a simple feedforward network, to convolutional and recurrent networks. This discussion also includes the siamese network architecture, which is used in our model for question paraphrase detection [Nicosia and Moschitti, 2017a], and in Chapter 5, for training a sentence encoder for text classification [Nicosia and Moschitti, 2017b]. At the end of the chapter, we briefly touch neural sentence matching models, and describe our Hybrid Siamese Network for such task.

3.1 Neural Networks

In recent years, deep learning methods have enjoyed many successes in computer science areas ranging from computer vision to NLP [LeCun et al., 2015]. Neural networks are at the core of deep learning and have taken NLP by storm, excelling in tasks such as machine comprehension [Seo et al., 2017], machine translation [Bahdanau et al., 2015; Wu et al., 2016] and parsing [Chen and Manning, 2014].

In this section, we introduce the basic concepts and the useful notation for specifying a neural network. As an example, we consider the feedforward network in Figure 3.1, which is also called Multilayer Perceptron (MLP). This network is composed by four sets of units arranged in layers: the input layer, two hidden layers, and the output layer. Every unit in a layer is connected to every other unit in the next layer, except for the output

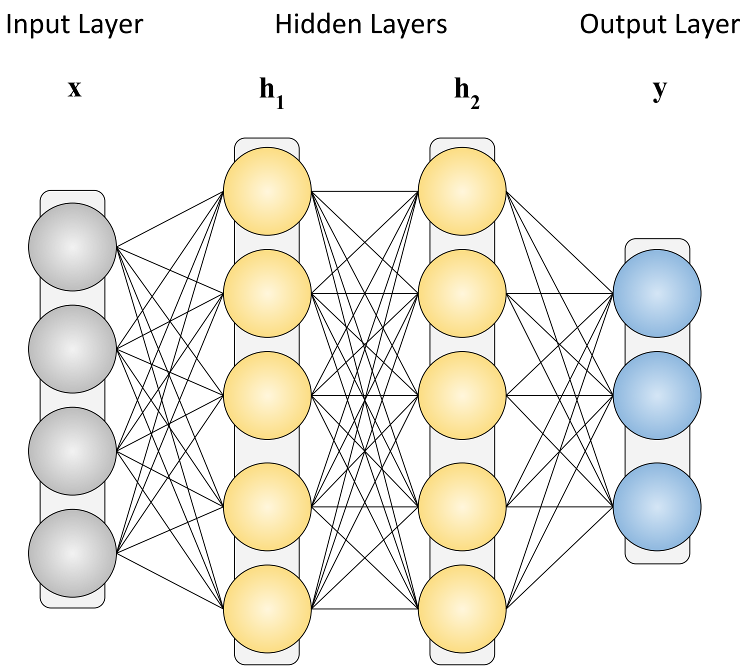

Figure 3.1: The Multilayer Perceptron (MLP), a feedforward network with two hidden layers.

layer, which represents the final output of the network. The MLP, denoted as $f(\mathbf{x})$, can be mathematically described with vector-matrix operations. The input layer coincides with the $\mathbf{x}$ vector, and the output layer is denoted as $\hat{\mathbf{y}}$. The hidden and output layer values are obtained by a linear transformation of the values from the previous layer. For example, the first hidden layer is computed by multiplying the input vector with a weight matrix, and adding a bias vector to the result. In our case, the full network is specified by the following equations:

$$\begin{aligned}\hat{\mathbf{y}} = f(\mathbf{x}) &= o(h_2(h_1(\mathbf{x}))\\ h_1(\mathbf{x}) &= \sigma(\mathbf{W_1}\mathbf{x} + \mathbf{b_1})\\ h_2(\mathbf{x}) &= \rho(\mathbf{W_2}\mathbf{x} + \mathbf{b_2})\\ o(\mathbf{x}) &= \mathbf{W_3}\mathbf{x} + \mathbf{b_3}\end{aligned}$$

$$\mathbf{x} \in \mathbb{R}^{d_{in}}, \hat{\mathbf{y}} \in \mathbb{R}^{d_{out}}, f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}},$$
$$\mathbf{W_1} \in \mathbb{R}^{d_1 \times d_{in}}, \mathbf{b_1} \in \mathbb{R}^{d_1}, \mathbf{W_2} \in \mathbb{R}^{d_2 \times d_1}, \mathbf{b_2} \in \mathbb{R}^{d_2}, \mathbf{W_3} \in \mathbb{R}^{d_{out} \times d_1}, \mathbf{b_3} \in \mathbb{R}^{d_{out}}$$

where d_{in} and d_{out} represent the input and output dimensions respectively, while σ and ρ are element-wise non-linear functions. These functions are applied after a linear transformation. Without them, the network would consist in a sequence of linear transformations,

Logistic (or sigmoid)	$f(x) = \frac{1}{1+\epsilon^{-x}}$
Hyperbolic Tangent (tanh)	$f(x) = \frac{2}{1+\epsilon^{-2x}} - 1$
Rectified Linear Unit (ReLU)	$f(x) = \max(0, x)$
Softmax	$f(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}}$ for $i = 1, ..., K$

Table 3.1: Common activation functions.

which is still equivalent to a linear transformation. Therefore, the network would not be able to perform non-linear computations and learn complex functions. The σ and ρ functions may be different, but often the same non-linear activation is used across the hidden layers. The output vector $\hat{\mathbf{y}}$ will contain unnormalized values also called logits. The specific nature of the learning problem usually guides the choice of some network details such as the number of outputs, the optional logit normalization function, and the loss to minimize.

3.1.1 Activation Functions

The choice of the activation function can have a substantial impact on the training process and on the final network performance. An activation function should be non-linear for allowing the network to learn complex functions — an MLP has sufficient power to act as an universal function approximator [Hornik et al., 1989]. A well behaved activation function should also be differentiable and have non-zero gradients almost everywhere.

Table 3.1 contains some common activation functions. The logistic function can be used to squash values between 0 and 1, and to model the probability of a single outcome in a network with one output unit. The tanh function outputs values between -1 and 1. The ReLU function [Nair and Hinton, 2010] keeps the positive part of the argument, and it is considered the default choice for hidden layer activation functions [Goodfellow et al., 2016]. The softmax function is often applied to the final output of a neural network in order to obtain a probability distribution out of a vector of logits. This is useful to model the probability of K different outcomes.

3.1.2 Training the Network

The output of a neural network, such as the MLP in Figure 3.1, depends on the input and on the parameters of the network, e.g., the weight matrices and bias vectors. These parameters are fixed during inference, but need to be tuned during training in order to find a configuration that minimizes the empirical risk on the training instances. The learning process happens in two steps: the forward and backward passes.

During the forward pass, the network output is evaluated for a given instance under the current parameter configuration. The result is compared with the desired output, usually determined by the gold labels associated with the training instances. The comparison produces an error measure called loss, which is used as feedback during the backward pass for applying small changes to the network parameters. The goal of this process is to reduce the loss at the next iteration, and thus make the network output closer to the desired output.

3.1.3 Loss Functions

During training, the network output is compared with some ground truth using a loss function that returns a numeric value representing the network error. It is of great importance to select the right loss for a given task. Such choice is certainly affected by the network output type, which could be a continuous value, or an outcome out of many. In the first case, a suitable loss function would be the Root Mean Squared Error (RMSE):

$$\mathrm{RMSE} = \sqrt{\frac{\sum_{i=1}^{n} (\hat{y}_i - y_i)^2}{n}}, \tag{3.1}$$

where $\hat{y}$ is the output of the network and y is the ground truth. In the second case, in which most NLP problems tackled in this thesis fall, an appropriate loss would be the categorical cross-entropy. This loss is used when we want to classify an input instance into one of possible classes. The cross-entropy measures the divergence between two probability distributions: the probability distribution produced by the network, and the ground truth probability distribution, which is usually encoded as a vector containing zeros, except for the value corresponding to the gold category, which is set to one. Such encoding is also known as one-hot label encoding. Therefore we have:

$$\mathrm{XENT} = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{m} y_{ij} \log(\hat{y}_{ij}), \tag{3.2}$$

where $\mathbf{y}$ is the one-hot encoded label vector and $\hat{\mathbf{y}}$ contains the result of transforming the network logits into a valid probability distribution with the softmax function.

3.1.4 Backpropagation

Once the network and the loss function are defined, the model parameters need to be updated in such a way that the error computed by the loss function is reduced. The backward propagation or backpropagation [Rumelhart et al., 1986] is an algorithm to iteratively adjust the parameters of the network while reducing the loss. Such algorithm is based on the computation of the gradient of the loss function with respect to the network parameters. Optimization algorithms based on gradient descent can use the gradient to update the network weights. Stochastic Gradient Descent (SGD) is a popular algorithm that draws a batch of examples from the training set, computes the gradients on that batch, and accordingly updates the weights [Bottou, 2010]. Adaptive optimization methods [Duchi et al., 2011; Kingma and Ba, 2014] are also popular, since they provide a faster convergence rate at the small cost of computing additional statistics.

3.2 Word Representations and the Sentence Matrix

In many machine learning algorithms, the words of a sentence are treated as symbols, and are encoded into features which signal the presence of the corresponding words or groups of words in that sentence. These features may be represented as boolean flags in the feature vector, or as continuous values capturing some occurrence statistic. These approaches for encoding words and sentences are not the best for deep learning models. Indeed, the standard way to feed textual data in neural networks is to map words into dense low dimensional representations, and build a sentence matrix. In turn, the word vectors or embeddings are obtained from the output of neural models trained with one of the following purposes: language modeling or word co-occurence prediction. Here we discuss how word embeddings are obtained, and how sentences are typically encoded.

3.2.1 Word Representations from Language Models

Language models are statistical models that try to accurately estimate the distribution of natural language sentences. Neural language model use neural networks to perform this estimation. In Bengio et al. [2003], the neural language model is trained by predicting the next word, given a sequence of previous words. Input words are represented as vectors having a fixed dimensionality. A feedforward neural network computes the joint probability function of word sequences, and it is trained by minimizing the negative log-likelihood of the training data. A feedforward network can condition its prediction on a fixed number of words. More powerful language models use recurrent networks [Mikolov et al., 2010], which can mantain a longer history of words seen so far. After the training converges, the tuned dense word vectors can be used in other models to represent the corresponding words.

3.2.2 Word Representations from Co-occurence Prediction

A different method for obtaining informative word embeddings consists in predicting word co-occurrence from large collections of documents. In this case, the task changes

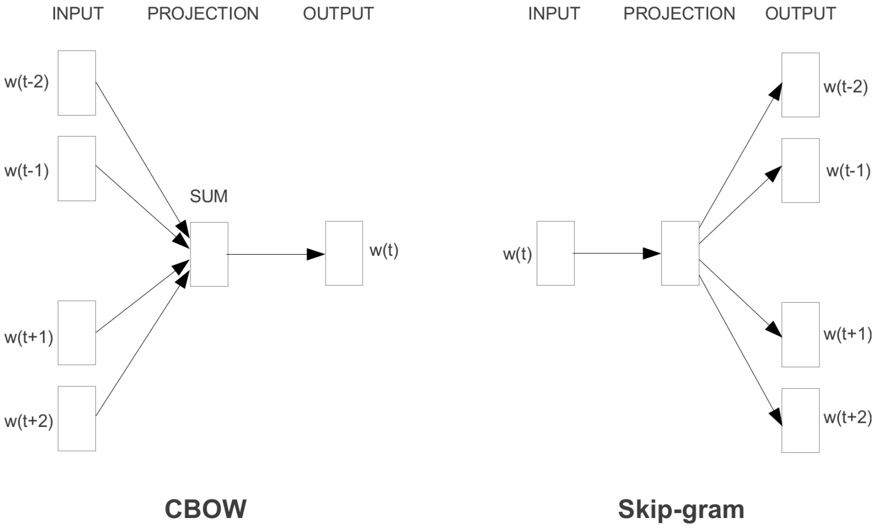

Figure 3.2: The CBOW and Skip-gram models [Mikolov et al., 2013b].

to predicting a word, given some of its adjacent words. Word2vec [Mikolov et al., 2013b] and GloVe [Pennington et al., 2014] are standard methods for learning word embeddings. Word2vec has two variants (Figure 3.2): the Continuous Bag-Of-Words (CBOW) model, where a word is predicted given a fixed window of adjacent words, and the Skip-gram model, where the words in a context are predicted given a word picked from that context. GloVe is a global log-bilinear regression model which exploits word co-occurrence statistics from the corpus in the optimization function.

3.2.3 Word Embedding and Sentence Matrices

Before training a neural model, a vocabulary V is induced from the textual data that will be processed by the network. The set of pretrained word vectors and the induced vocabulary V are then used to construct a word embedding matrix. This matrix is useful to lookup a word symbol into its corresponding dense representation, and it is built by concatenating the vectors associated with all the words contained in V. Words that are not in the vocabulary may be mapped to a special UNK token, and therefore to the same random vector. Alternatively, they can be mapped to different random vectors specifically allocated for each word.

When we do text classification or passage reranking, the input to our neural network is typically a sentence $\mathbf{s}$: a sequence of words $(w_i, .., w_{|s|})$ drawn from the vocabulary V. Each word in the sequence is looked up in the word embedding matrix $\mathbf{W} \in \mathbb{R}^{d \times |V|}$, and mapped to a distributional vector $\mathbf{w} \in \mathbb{R}^{1 \times d}$. The words in V are mapped to indices $1, \ldots, |V|$ to facilitate the lookup operation, i.e., an index j resolves to the j-th column of the W matrix. Therefore, each input sentence $\mathbf{s}$ is converted into a sentence matrix $\mathbf{S} \in \mathbb{R}^{d \times |s|}$, where each column i contains the embedding $\mathbf{w}_i$ associated with the i-th word in the sentence:

$$\mathbf{S} = \begin{bmatrix} | & | & | \\ \mathbf{w}_1 & \dots & \mathbf{w}_{|s|} \\ | & | & | \end{bmatrix}$$

After this, a neural network can apply a range of different transformations to the matrix in order to extract higher-level interactions between words and semantic features.

More complex sentence encoding may include additional information, such as syntactic features, subword and character-based word embeddings [Alexandrescu and Kirchhoff, 2006; Sperr et al., 2013; Botha and Blunsom, 2014; Dos Santos and Zadrozny, 2014].

3.3 Neural Bag-Of-Words

In many neural models there is a point where the sentence is mapped into a single vector of continuous values. A suitable and simple approach for such a mapping is to aggregate word embedding vectors across dimensions. In Kalchbrenner et al. [2014], the sum is selected as the aggregation operation. More recent empirical results show the superiority of the averaging operation (see Deep Averaging Networks [Iyyer et al., 2015]). In this case, the word embeddings from a sentence are averaged:

$$\mathbf{z} = \frac{1}{|s|} \sum_{i=0}^{|s|} \mathbf{w}_i, \tag{3.3}$$

and passed through a stack of non-linear layers before producing the final output. Sentence encodings which aggregate word vectors in this way do not preserve information about word order. For this analogy with bag-of-words models, they are also referred to as Neural Bag-Of-Words (NBOW) encodings.

3.4 Recurrent Networks

The NBOW encoding is fast and simple, but starts to be less effective when embedding long sentences or entire documents, since a high number of vectors is condensed into a single one. For longer pieces of text, other encoding mechanisms can be more appropriate. Given that sentences can be modeled as sequences of words, methods developed for sequential data can be considered.

Recurrent Neural Networks (RNNs) [Elman, 1990] are one of the main approaches for modeling sequences, and they have been widely adopted for NLP tasks. Vanilla RNNs consume a sequence of vectors one step at the time, and update their internal state as a function of the new input and their previous internal state. For this reason, at any given step, the internal state depends on the entire history of previous states. These networks may suffer from the vanishing gradient problem [Bengio et al., 1994], which is mitigated by a popular RNN variant, the Long Short Term Memory (LSTM) network [Hochreiter and Schmidhuber, 1997]. An LSTM can control the amount of information from the input that affects its internal state, the amount of information in the internal state that can be forgotten, and how the internal state affects the output of the network.

Given a sequence of word embeddings $(x_1, ..., x_T) \in \mathbb{R}^{d_{in}}$, a cell and state vector $c_t, h_t \in \mathbb{R}^{d_1}$ are computed for each element of the sequence by iteratively applying the following equations, where $h_0 = c_0 = 0$:

$$\begin{aligned} o_t &= \sigma(W_o x_t + V_o h_{t-1} + b_o) \\ f_t &= \sigma(W_f x_t + V_f h_{t-1} + b_f) \\ i_t &= \sigma(W_i x_t + V_i h_{t-1} + b_i) \\ g_t &= \tanh(W_g x_t + V_g h_{t-1} + b_g) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned}$$

The W, V matrices and the b vectors are parameters of the model. $\odot$ denotes element-wise (Hadamard) multiplication. σ is the logistic function and $tanh$ is the hyperbolic tangent function. o_t, f_t and i_t are referred to as output, forget and input gates respectively, since their values are bounded between 0 and 1, and control the rescaling of vectors.

The Gated Recurrent Unit (GRU) [Chung et al., 2014] is an LSTM variant with similar performance and less parameters, thus faster to train. In the following equations, s_t defines the state at timestep t. Given a sequence of input vectors, the GRU computes a sequence of states $(s_1, ..., s_T)$ according to:

$$\begin{aligned} z &= \sigma(x_t U^z + s_{t-1} W^z) \\ r &= \sigma(x_t U^r + s_{t-1} W^r) \\ h &= \tanh(x_t U^h + (s_{t-1} \odot r) W^h) \\ s_t &= (1 - z) \odot h + z \odot s_{t-1} \end{aligned}$$

This recurrent unit has an update, z, and reset gate, r, and does not have an internal memory beside the internal state. The U and W matrices are parameters of the model.

The last state of these recurrent networks can be considered as the vector which encodes the information contained in a sentence.

LSTMs and GRUs consume the input sequence in one direction, and thus earlier internal states do not have access to future steps. Bidirectional RNNs [Schuster and Paliwal, 1997] solve this issue by keeping a forward and backward internal states that are computed by consuming the input sequence in both directions. The state at any given step will be the concatenation of the forward and backward state at that step, and will contain useful information from both the left and right context of a word. Again, the last state of the bidirectional network will contain the encoding of the entire sentence.

3.5 Convolutional Neural Networks

For solving certain kinds of problems it may be important to model the spatial relationship between the elements of the input. For example, in image recognition, a classifier is much more effective if it is able to model not only the current pixel, but also its adjacent pixels.

Convolutional Neural Networks (CNNs) [LeCun et al., 1998] are explicitly designed to capture the spatial characteristics of an input. Due to their efficiency and properties they are very popular in computer vision [Krizhevsky et al., 2012; Lawrence et al., 1997; Karpathy et al., 2014]. At the same time they are also effective in modeling sentences [Kim, 2014; Kalchbrenner et al., 2014], and have been applied in many NLP problems.

A CNN that takes in input an image slides over the latter and repeatedly applies the same transformation over small image patches. While images are represented as two-dimensional matrices (per channel), text has only one temporal dimension, as it can be seen as a sequence of vectors. The convolution operation is therefore an affine transformation applied to a window of a fixed size, for all the words in a sequence. Such transformation can include a bias term and non-linearities. Furthermore, if we allow padding for the out-of-sequence tokens in a window, the convolution output resolution will be the same as the input. In the other case, the resolution will depend on the window size and sequence length.

3.5.1 Convolution Feature Maps

Here we describe how a single convolution filter is applied to the **S** sentence matrix. Let $\mathbf{w}_i \in \mathbb{R}^d$ be the d-dimensional word vector corresponding to the i-th column in S (and thus to the i-th word in the sentence). We can represent the sentence, which contains n words (and can be potentially padded), as a concatenation of vectors:

$$\mathbf{w}_{1:n} = \mathbf{w}_1 \oplus \mathbf{w}_2 \oplus \ldots \oplus \mathbf{w}_n, \tag{3.4}$$

where $\oplus$ is the concatenation operator. We refer to $\mathbf{w}_{i:i+j}$ as the concatenation of vectors $\mathbf{w}_i, \mathbf{w}_{i+1}, \ldots, \mathbf{w}_{i+j}$. Then, a convolution operation is the result of the dot product between a filter $\mathbf{U} \in \mathbb{R}^{hd}$, and a window of h words. For example, a feature c_i is computed from a window of words $\mathbf{w}_{i:i+h-1}$ by:

$$c_i = \sigma(\mathbf{U} \cdot \mathbf{w}_{i:i+h-1} + b). \tag{3.5}$$

$b \in \mathbb{R}$ is a bias term and σ is a non-linear activation function. The filter is applied to all the word windows from the sentence $\{\mathbf{w}_{1:h}, \mathbf{w}_{2:h+1}, \ldots, \mathbf{w}_{n-h+1:n}\}$ and the result is the following feature map:

$$\mathbf{c} = [c_1, c_2, \ldots, c_{n-h+1}], \tag{3.6}$$

with $\mathbf{c} \in \mathbb{R}^{n-h+1}$. The most important features from the map are then selected with a max-pooling operation [Collobert et al., 2011]. Typically, a single value per feature map contributes to the final pooled vectors, which contains features coming from multiple filters. This vector constitutes the fixed size sentence encoding which can be furtherly transformed, or used for classification.

3.6 Siamese Networks

CNNs and RNNs sentence encodings are usually the input of subsequent layers which furtherly transform them, and finally compute the output to improve in the context of a supervised task [Bowman et al., 2015]. Sentence encodings are tuned for this final task, and do not necessarily have a particular geometric meaning, because there are no constraints that explicitly enforce such property.

Siamese networks [Bromley et al., 1994] have precise architectural characteristics that enable metric learning, and therefore the mapping of objects into a geometric space where similar instances (according the selected metric) are close to each other, and dissimilar instances are separated by a certain distance. If during training siamese networks require pairs of similar and dissimilar objects, at inference any individual data point can be mapped into the geometric space using the trained encoder, and its neighbourhood can be easily explored.

In NLP, the siamese network is used to map sentences into a geometric space according to a similarity metric. Once the network is trained, the encoder can be used to model new sentences. Figure 3.3 shows the architecture from Chopra et al. [2005], where a siamese network is used for face verification. The X_1 and X_2 input objects are encoded using the same CNN, i.e. the same weights are reused across the two branches. After that, the similarity between the two object vector encodings is computed. The network weights are

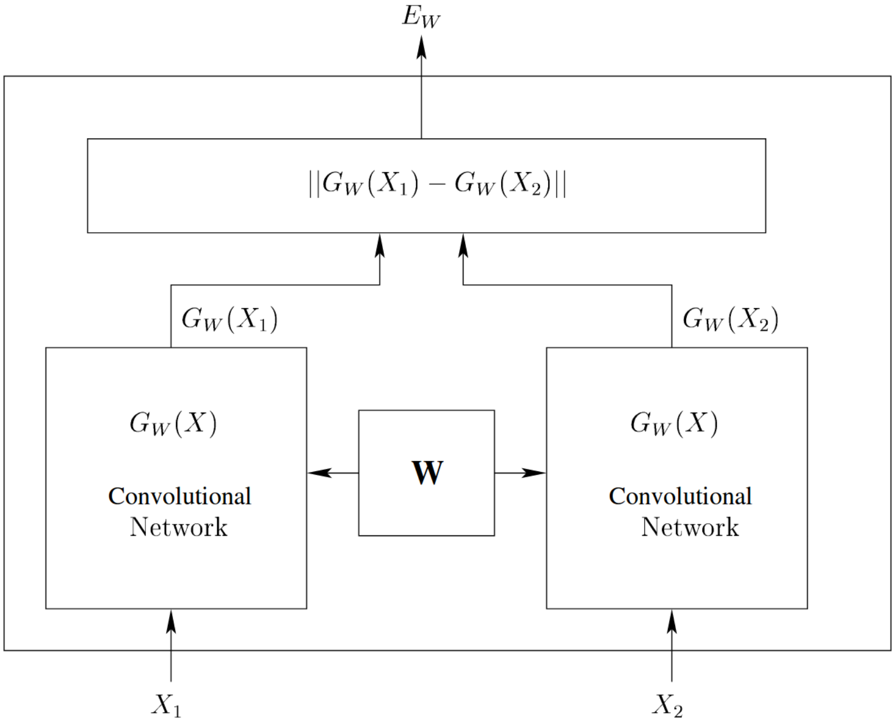

Figure 3.3: A Siamese network with CNN encoders [Chopra et al., 2005].

tuned to maximize this score for similar objects, while keeping it under a certain threshold for dissimilar objects.

3.7 Sentence Matching

Sentence matching can be defined as the general task of establishing if two sentences are related according to a given criterion. Many instantiations of this task can be found in Information Retrieval (IR), Question Answering (QA) [Ravichandran and Hovy, 2002], and more recently in cQA [Wang et al., 2009; Liu and Agichtein, 2008], as also shown by a recent workshop on semantic matching [Gonzalo et al., 2014].

Sentence matching models are also important for other tasks such as semantic textual similarity [Agirre et al., 2016], textual entailment [Bowman et al., 2015], and answer sentence selection [Rao et al., 2016], to name a few.

The problem has also been approached with deep learning models, e.g., CNNs [Severyn and Moschitti, 2015a] or LSTMs [Cohen and Croft, 2016], for passage reranking. A commonality of these methods is to build a representation for each sentence, and then compute their similarity with a feed-forward network. In these models, the sentence matching constraint is only enforced in the last layer, where a logistic loss is computed from the network output and the ground truth. However, it may be argued that this happens too late in the network, causing some of the properties of the sentence representation being neglected for learning the overall similarity. In contrast, siamese networks learn a distance metric between two sentences by mapping them into a geometric space. For this purpose, they apply a contrastive loss, which is based on the distance between two sentences, to learn the matching task. The main drawback is that their encoder produces representations that are independent from each other, i.e., the representation of a sentence is not conditioned on the other.

A basic network for sentence matching encodes each sentence into a pair of vectors, which are concatenated and passed through a number of hidden layers to make a prediction. Apart from this basic architecture, we can define three more sophisticated network types: the already mentioned Siamese architecture, Attentive models, and Compare-Aggregate networks.

Focusing on NLP tasks, we find *Siamese Networks* applications in textual similarity, paraphrase identification and mention normalization. MaLSTM [Mueller and Thyagarajan, 2016] encodes the two sentences with an LSTM and learns their Manhattan distance. After the network is trained on a similarity task, the parameters of the sentence encoder are fixed. The resulting feature extractor feeds an SVM classifier trained for recognizing textual entailment. Therefore, the siamese representations are not learned in an end-to-end fashion for the second task.

The SCQA network [Das et al., 2016] exploits data from cQA forums to learn the similarity between pairs of questions. First, questions and answers are mapped together in the same space using a CNN-based siamese network. This training phase with q/a pairs assumes that answers are semantically similar to questions, and thus they help to model question-question similarity. Indeed, the network shows improvements when fine-tuned on the question-question similarity task, due to such pretraining.

A two-layer bidirectional LSTM network on characters was used in Neculoiu et al. [2016] to normalize job titles. This siamese network was able to capture semantic differences, while being invariant to non-semantic string differences.

Attentive Networks [Parikh et al., 2016; Yin et al., 2016] use the attention mechanism to build the representation of a sentence in a pair by also conditioning on the other sentence. The computation has a relatively high cost, i.e., quadratic complexity.

Compare-Aggregate Networks [Wang and Jiang, 2017; Wang et al., 2017] extract multiple views from the two sentences, and then match them through a number of similarity functions. The results are aggregated and used to produce a final matching score.

3.8 Hybrid Siamese Network

Neural networks typically learn a sentence matching score by minimizing a logistic loss. Here we describe our work about learning sentence representations for sentence matching with a siamese network [Nicosia and Moschitti, 2017a] that: (i) uses parameter-sharing encoders; and (ii) compares two sentences using their euclidean distance by minimizing a contrastive loss. In addition, we extend the siamese network with an MLP, and simultaneously optimize the contrastive and the logistic losses. This way, our network can exploit a more informative feedback, given by the logistic loss, which is also quantified by the distance between the sentence representations in the euclidean space. We show that jointly minimizing the two losses yields a higher accuracy than minimizing them independently. We verify this finding by evaluating several baseline architectures on two tasks: question paraphrasing and textual entailment recognition. Our network approaches the state-of-the-art Bilateral Multi Perspective Matching (BiMPM) model [Wang et al., 2017], while being much simpler and faster to train, and with less parameters than its competitors.

3.8.1 Overview

In this section, we present our Hybrid Siamese Network for matching pairs of sentences. More specifically, we exploit the siamese network architecture to encode sentences and compare them in terms of their euclidean distance. In addition, an MLP furtherly transforms the concatenated sentence representations to enable the simultaneous optimization of a contrastive and a logistic loss. Therefore, our network uses (i) the contrastive loss to learn sentence representations based on their euclidean distance; and (ii) the informative feedback given by the logistic loss in the MLP to capture interdependencies between the two sentences.

We compare against several state-of-the-art networks for the question duplication and textual entaliment tasks, on the Quora and the Stanford Natural Language Inference datasets respectively. The results show that our network, also thanks to the optimization of our joint contrastive and logistic loss, approaches the the state of the art. It should be noted that the latter is achieved by much more complex architectures, which have a high parameter count and computational time for both training and testing.

The use of multiple or auxiliary losses is at the core of multitask learning. The same network can be trained to perform several tasks, by using multiple specialized final softmax layers. For example, a part-of-speech (POS) tagger can be trained to predict the tag and the binned log frequency of the next token [Plank et al., 2016]. Alternatively, a network can produce an output for a task at an earlier layer in the architecture, and a different output for another task in a later layer. Stack-propagation [Zhang and Weiss, 2016] uses this method to learn POS tags at a lower level, and dependency relations at the uppermost level. Our network does not address different tasks, but adopts multiple losses to extract different aspects of the input: the semantic traits of the sentences, captured by the distance of their representations, and the interactions between the latter, modeled by an MLP.

3.8.2 Sentence Matching with Hybrid Siamese Networks

In this section, we present our approach to sentence matching, define the siamese network architecture, and characterize the joint loss to optimize.

Model Architecture. The first module of our deep learning model is the sentence encoder, that we refer to as a generic function f, since its underlying architecture can vary. This function takes in input a sentence matrix, obtained as described in section 3.2.3, and transforms it into a vector. The same function f is used to encode both sentences in a pair, because weights are shared between the siamese network branches. This is typical of such a setting: the same network maps the two objects of a pair into a low dimensional space, where their distance is small if they are similar. The euclidean distance of two sentences s_1 and s_2 is computed as:

$$d(s_1, s_2) = \sqrt{\sum_{i=1}^{n} (f(s_1)_i - f(s_2)_i)^2} \tag{3.7}$$

Our hybrid siamese network furtherly models the interaction between the two sentences by feeding the outputs of the siamese encoder to an MLP. The latter takes in input the concatenated sentence representations and their distance, $c = [f(s_1); f(s_2); d(s_1, s_2)]$, and outputs the matching probability of the two sentences.

Joint Loss Optimization. The siamese network encoder is trained by optimizing a contrastive loss computed from the pairs of sentences in the dataset. Such loss compares the distance between two representations produced by the siamese encoder with the true label. Given the n examples in dataset $\mathcal{D} = \{(s_1^i, s_2^i, y^i)\}_{i=1}^{n}$, where y is equal to 1 if the two sentences match, and 0 otherwise, the total loss is:

$$\mathcal{L}_c = \sum_{i=1}^{n} L_c^i(s_1^i, s_2^i, y^i), \tag{3.8}$$

where L_c^i is the contrastive loss for an instance defined as:

$$L_c^i(s_1^i, s_2^i, y^i) = y^i d(s_1^i, s_2^i)^2 + (1 - y^i) \max(M - d(s_1^i, s_2^i), 0)^2 \tag{3.9}$$

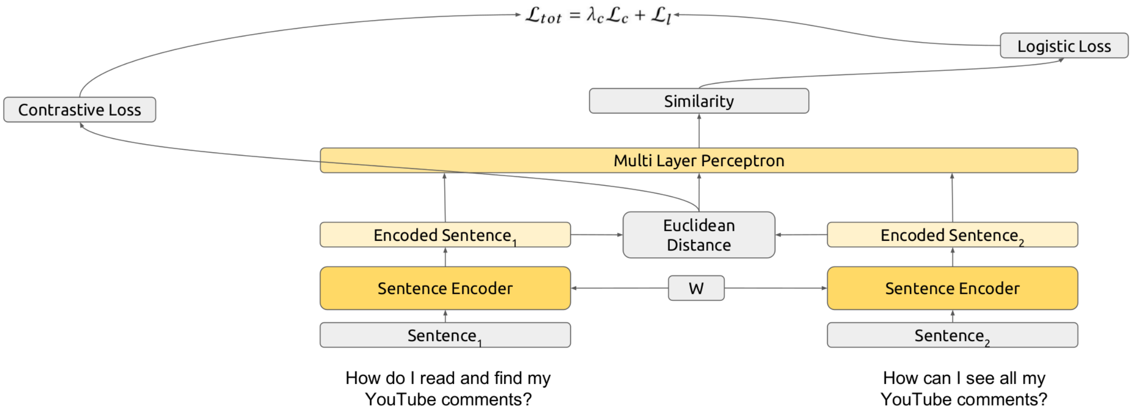

Figure 3.4: The Hybrid Siamese Network.

The loss for matching sentences is the square of their euclidean distance, while there is a loss in the opposite case, only when the distance between sentences that do not match is smaller than the margin M. In that case, the loss is equal to the square difference between the margin and the distance. Our final network optimizes the logistic loss computed on the output of the MLP with respect to the true labels:

$$\mathcal{L}_l = \sum_{i=1}^{n} y^i log(\tilde{y}^i) + (1 - y^i)log(1 - \tilde{y}^i) \tag{3.10}$$

The global loss of our full network is thus the sum of the two losses:

$$\mathcal{L}_{tot} = \lambda_c \mathcal{L}_c + \mathcal{L}_l, \tag{3.11}$$

where λ_c is an hyperparameter to tune the effect of the contrastive loss during training. A graphic representation of the network can be seen in Figure 3.4.

3.8.3 Experiments

We evaluate our model on two tasks: identification of duplicate questions in cQA, and textual entailment recognition.

Sentence Encoders and the MLP. We experiment with different network architectures for our sentence encoding function f:

- an LSTM network with 200 units;
- a GRU network with 200 units;

- a CNN network with 3 groups of 100 convolutional filters of size 1, 3 and 5;
- a stack of 2 bidirectional GRU (BiGRU) networks, with GRUs of 200 units.

The LSTM and GRU consume the input from left to right, and their last state is used to encode the sentence. The stacked BiGRU network has higher capacity and should better capture longer dependencies in a sentence. The BiGRU consumes token or states in both directions, producing a state for each input. Those states are fed to the upper BiGRU layer. The maximum values across the dimensions of the output states are selected, producing a sentence vector with 400 dimensions. We tuned the number of stacked layers on the development set. The two hidden layers of the MLP have 200 units.

We carry out three different experiments for each architecture, minimizing: (i) the contrastive loss, (ii) the logistic loss, (iii) the sum of the contrastive and logistic losses.

Network Training. We summarize here the settings common to all the experiments. Word embeddings are initialized with pretrained GloVe vectors of size 300. Sentences are truncated/padded to 50 words. Unknown word vectors are initialized sampling from the uniform distribution $U[-0.25, 0.25]$. Word embeddings are tuned for the task together with the network parameters.

The network is trained with the Adam optimizer [Kingma and Ba, 2014], setting the learning rate to .001. The contrastive loss margin M is set to 1. λ_c is tuned on the validation set: it is set to 1.0 for the paraphrase identification models, and to 0.01 for the textual entailment models. The smaller λ_c value for textual entailment may be due to the not strictly symmetric nature of the task. Training examples are shuffled and fed to the network in batches of size 128. All the models are trained for 20 epochs and the reported test accuracy corresponds to the best accuracy obtained on the validation set. In the contrastive only setting, labels are obtained by thresholding the distances at 0.5, value selected from the valdation set.

Paraphrase Identification

Dataset. The Quora dataset[1] contains pairs of questions from the Quora web site. In positive pairs, questions ask for the same thing. For the evaluation, we adopt the dataset splits and word embeddings[2] provided by Wang et al. [2017]. Their training split contains 384,348 pairs, and the balanced development and test sets contain 10,000 pairs each. The embeddings are a subset of the 300-dimensional GloVe word vectors pretrained on

[1] https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs

[2] https://github.com/zhiguowang/BiMPM

Siamese Encoder	Contrastive	Logistic	Joint loss
LSTM	85.26	81.98	85.71
GRU	86.72	83.00	**86.82**
CNN	83.1	83.04	83.95
Stacked BiGRU	85.74	84.38	86.06

Table 3.2: Test accuracies of our models for the paraphrase identification task (Quora), trained with the different losses.

Siamese Encoder	Contrastive	Logistic	Joint loss
LSTM	84.58	88.25	89.08
GRU	84.96	88.53	90.58
CNN	80.78	90.99	**92.00**
Stacked BiGRU	85.58	91.40	91.97

Table 3.3: Test accuracies of our models for the textual entailment task (SNLI), trained with the different losses.

the Common Crawl corpus[3], which cover a vocabulary induced from the Quora dataset. Questions are already tokenized, so we only lowercase them before splitting on spaces.

Results. Table 3.2 shows the results of our models for the Quora paraphrase identification task. Each model is trained using (i) the contrastive loss on the euclidean distance of the sentence representations from the siamese encoder; (ii) the logistic loss on the prediction of an MLP applied on the concatenated sentence representations; (iii) both losses, considering the MLP prediction as the final matching score. The last setup yields consistently higher accuracy with respect to separate loss minimization.

Recognizing Textual Entailment

Dataset. For this task, we use the SNLI dataset [Bowman et al., 2015], which contains 570,000 premise/hypothesis pairs labelled by humans. The labels are *entailment*, *contradiction* and *neutral*. We use the official training, development and test partitions to train, validate and test our models respectively. We keep only examples from the *entailment* and *contradiction* classes, experimenting with the binary task of understanding if the mean-

[3] https://nlp.stanford.edu/projects/glove/

ing of the hypothesis can be inferred from the premise or not. The resulting training, development and test partitions have 366,603/9,842/9,824 examples, respectively. We use GloVe vectors again, but we extract them from the distribution available on the GloVe project web site. The sentence are tokenized with SpaCy[4] and words are lowercased.

Results. Table 3.3 shows the performance of our models for the binary textual entailment task. We evaluated the models on this additional semantic task to corroborate the advantage of using the joint loss: the improvement is consistent. The lower accuracy in the contrastive loss setting may be due to the asymmetry of the task.

Discussion

The results in Table 3.2 for paraphrase identification shows that minimizing the joint loss is the better strategy for all the architectures used in the siamese sentence encoder.

Table 3.4 contains a comparison of our models with the BiMPM model [Wang et al., 2017], a sophisticated network from the Compare-Aggregate family. The authors implement siamese and multi perspective CNN and LSTM networks. Our models outperform their corresponding siamese baselines even when we do not use the joint losses, and, when we do, the improvement becomes more substantial. All the models use the same data and word embeddings, thus our improvement on siamese models just comes from a better hyper-parameter tuning and some architectural differences: we use 200 recurrent units instead of 100, we learn the euclidean distance between representations instead of the cosine distance, and we do not use character embeddings to learn word vectors.

Except for the CNN model, our other architectures are comparable or outperform the L.D.C. model [Wang et al., 2016c], even in the single contrastive loss optimization case. L.D.C. belongs to the compare-aggregate approach class, and it is very accurate on the answer selection task. It is possible that the CNN architecture may reach even higher accuracy with further hyper-parameter tuning. Our best model, the hybrid siamese GRU network achieves a test accuracy of 86.82%, i.e., only 1.35% less than the state-of-the-art BiMPM model. We stress the fact that the latter is much more complex: the two sentences in a pair are encoded with a BiRNN and every pair of encoded words from the two sentences are matched using multiple functions. One function even applies an expensive attention mechanism. For all these reasons, this model (i) is one order of magnitude slower to train than ours: 100 vs. 10 seconds per 100 training steps on a Tesla K40m GPU; (ii) it has many more trainable parameters: 6.5M vs. 300k, without considering word embeddings.

[4] https://spacy.io/

Model	Accuracy
Siamese-CNN [Wang et al., 2017]	79.60
Multi-Perspective-CNN [Wang et al., 2017]	81.38
Siamese-LSTM [Wang et al., 2017]	82.58
Multi-Perspective-LSTM [Wang et al., 2017]	83.21
L.D.C. [Wang et al., 2017]	85.55
BiMPM [Wang et al., 2017]	88.17
Hybrid Siamese LSTM	85.71
Hybrid Siamese GRU	86.82
Hybrid Siamese CNN	83.95
Hybrid Siamese Stacked BiGRU	86.06

Table 3.4: Comparison between the accuracies of the models in Wang et al. [2017] and our models (trained with the joint loss), for the paraphrase identification task (Quora).

The experiments on textual entailment (Table 3.3) also show a consistent improvement when using the joint loss. The contrastive loss incentives the sentence encoder to produce representations that can be semantically separated in a geometric space. We can think of the loss as a regularization mechanism imposing a structure on the intermediate network representations. Then, the MLP can refine the final network output both in terms of the interaction between the sentences and their distance. This way, it may be better informed regarding sentences that are mistakenly put close or far apart in the geometric space.

3.8.4 Conclusion

We presented a hybrid siamese network optimized with a joint loss, which is the sum of: (i) the contrastive loss used to optimize the sentence representations produced by a siamese sentence encoder; and (ii) the logistic loss used to optimize an MLP that models the interaction between the sentence representations.

The contrastive loss introduces an additional level of supervision when learning the sentence encodings, mapping them into an euclidean space. The joint loss consistently benefits the performance of our models across the different network architectures selected for the sentence encoder. This positive effect allows us to keep the last part of our network, i.e., the MLP, simple, and avoid computationally expensive matching and attention mechanisms. The resulting networks are competitive with state-of-the-art models, with lower complexity and number of parameters.

3.9 Summary

In this chapter, we introduced word and sentence representations for neural models, along with some basic neural architectures: feedforward, convolutional, recurrent, and siamese networks. The purpose of including the latter was to better prepare the reader for our first original contribution: an hybrid siamese network architecture for sentence matching.

After giving some context on this general NLP task and the family of models used to tackle it, we presented our neural network architecture that includes a siamese sentence encoder and an MLP. The model parameters are optimized by minimizing a joint loss which is a function of a contrastive and logistic loss. The joint loss setting shows greater performance with respect to the individual loss settings, in the tasks of question paraphrase identification and textual entailment.

Chapter 4

Structural Representations for Reranking Text Pairs

The problem of how to represent pairs of input text is central for many NLP tasks: good representations are required to let machine learning algorithms extract meaningful features that capture the relations between two text fragments. Traditional approaches rely on real valued vectors which signal the presence of a feature in one or both pieces of text, or which may contain a set of similarity features, computed with different algorithms and semantic resources. Such features may be useful, but also expensive to compute. More importantly, their effectiveness can be increased by incorporating structural syntactic and semantic information in the model.

In this chapter, we study the impact of relational and syntactic representations for an interesting and challenging task: the automatic resolution of crossword puzzles. An automatic system for solving crosswords typically includes the following modules: (i) a web search engine for retrieving relevant snippets given a target clue, and (ii) a database (DB) of clues from previously seen puzzles. We show that learning to rank models based on relational syntactic structures can improve such modules.

In particular, we present our work from Barlacchi et al. [2014b] and Nicosia et al. [2015]. We access the DB with a search engine and rerank its output by modeling the similarity of clues. This alone improves a previous system by about 25 absolute percent points in ranking answer candidates, and greatly improves the resolution accuracy of crossword puzzles. Given a list of clues, we also aggregate the clues that are associated with the same answer to furtherly refine the candidates. In addition, the web search module shows a clear benefit from our reranking approach, since the latter produces more accurate confidence scores for the search snippets that contain the clue answer.

4.1 Overview

Crossword Puzzles (CPs) are very popular language games and are played worldwide. They are very challenging, because solving them requires several skills: general knowledge, logical thinking, intuition, and the ability to deal with ambiguities and puns.

Researchers developed many automatic CP solvers, and some of the latter competed with human players in dedicated tournaments such as the American Crossword Puzzle Tournament. These automatic systems have been mainly developed by the Artificial Intelligence (AI) community, using AI techniques for filling the puzzle grid. The game objective is formalized as an optimization problem where the grid must be filled with the most probable answers, while satisfying some constraints.

Our effort for improving crossword solvers goes into the design of LTR models for reordering lists of objects related to the clue. Such lists are produced in two ways: (i) by issuing a query containing the clue to a Web search engine (e.g., Google or Bing), and (ii) by finding definitions similar to a target clue, from a DB of previously solved CPs.

We rerank the text snippets returned by the search engine using SVM preference ranking [Herbrich et al., 2000]. Our model exploits a syntactic representation of the clues to improve Web search. More specifically, we use structural kernels [Shawe-Taylor and Cristianini, 2004b] in SVMs applied to our syntactic representation of pairs, formed by clues with their candidate snippets.

Regarding the second method, we provide a completely novel solution by substituting the SQL and DB approach with a search engine for retrieving clues similar to the target one. Then, we rerank the retrieved clues by applying SVMs and structural kernels to the syntactic representation of clues. This way, SVMs learn to select the best candidate among the similar clues that are available in the DB. The syntactic representation can again capture clue paraphrasing properties.

In order to carry out our study, we created two different corpora, one for each task: (i) a Snippets Reranking Dataset and (ii) a Clue Similarity Dataset. The first includes 21,000 clues, each associated with 150 candidate snippets, whereas the latter comprises 794,190 clues. These datasets constitute interesting resources that we make available to the research community[1].

We compare our methods with one of the best systems for automatic CP resolution, WebCrow [Ernandes et al., 2005]. Regarding snippet reranking, our structural models improve on the basic approach of WebCrow based on Google by more than 4 absolute percent points in MRR, for a relative improvement of 23%. Concerning the similar clues retrieval, our methods improve on the one used by WebCrow, based on DBs, by 25 absolute

[1] `https://ikernels-portal.disi.unitn.it/projects/webcrow/`

percent points, i.e., about 53% of error reduction. Given such promising results, we integrated our clue reranking method in WebCrow, and obtained an average improvement of 15% in resolving complete CPs. This demonstrates that advanced QA methods such as those based on syntactic structures and LTR can help to win the CP resolution challenge.

4.2 Related Work

Proverb [Littman et al., 2002] was the first system for the automatic resolution of CPs. It includes several modules for generating multiple lists of candidate answers. These lists are merged and used to solve a Probabilistic-Constraint Satisfaction Problem. Proverb relies on a very large crossword database as well as several expert modules, each of them mainly based on domain-specific databases (e.g., movies, writers and geography). In addition, it employs generic-word list generators and clue-specific modules to find solutions for particular kinds of clues like ≪Tel ____ (4): *aviv* ≫. Proverb's modules use many knowledge sources: databases of clues, encyclopedias and Web documents. During the 1998 American Crossword Puzzle Tournament, Proverb placed 109th out of 251 contestant.

WebCrow [Ernandes et al., 2005] is based on Proverb, and participated in many international competitions. It incorporates additional knowledge sources, provides a solver for the Italian language, and improves the clues retrieval model from DB. In particular, it enables partial matching to retrieve clues that do not perfectly overlap with the query. WebCrow carries out basic linguistic analysis such as POS tagging and lemmatization. It takes advantage of semantic relations contained in WordNet, dictionaries and gazetteers. Its Web module can query a search engine, and thus retrieve text snippets or documents related to the clue. Answer candidates and answer confidence scores are generated from the retrieved content above. WebCrow uses a WA* algorithm [Pohl, 1970] for Probabilistic-Constraint Satisfaction Problems, adapted for CP resolution. The solver fills the grid entries for which no solution was found by the previous modules. It tries combinations of letters that satisfy the crossword constraints, where the letters are derived from words found in dictionaries or in the generated candidate lists.

Dr. Fill [Ginsberg, 2011] sees the crossword filling task as a Weighted-Constraint Satisfaction Problem. Constraint violations are weighted and can be tolerated. Dr. Fill heavily relies on huge databases of clues. It placed 92nd out of more than 600 opponents in the 2013 American Crossword Puzzle Tournament.

Regarding related work specific to QA using syntactic structures, a referring work for our research is the IBM Watson system [Ferrucci et al., 2010]. This is an advanced QA pipeline based on deep linguistic processing and semantic resources. It demonstrated that automatic methods can be more accurate than human experts in answering complex ques-

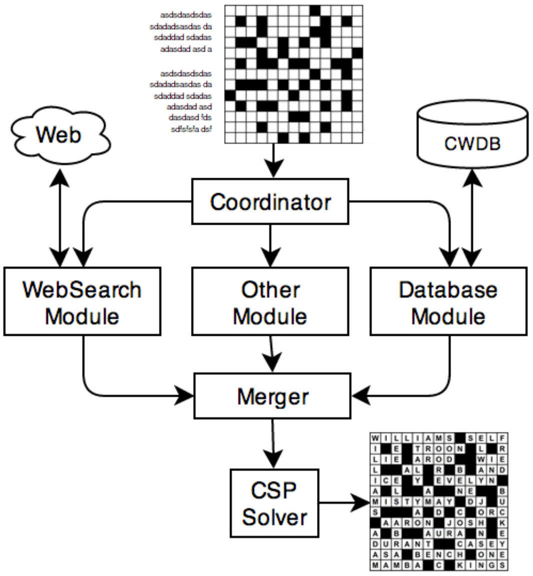

Figure 4.1: The architecture of WebCrow [Barlacchi et al., 2014b].

tions. More traditional studies on passage reranking, exploiting structural information, were carried out in Katz and Lin [2003], whereas other methods explored soft matching (i.e., lexical similarity) based on answer and named entity types [Aktolga et al., 2011]. Radlinski and Joachims [2006]; Jeon et al. [2005] applied question and answer classifiers for passage reranking. In this context, several approaches focused on reranking the answers to definition/description questions, e.g., Shen and Lapata [2007]; Moschitti et al. [2007]; Surdeanu et al. [2008]; Severyn et al. [2013a].

4.3 WebCrow

Our research focuses on the generation of accurate answer candidate lists, which, used in a CP resolution systems, can improve the overall accuracy of the solution. Therefore, the quality of our modules can be assessed by testing them within such systems. For this purpose, we selected WebCrow, as it is rather modular, it is one of the most competitive systems, and it was kindly made available by the authors. The architecture of the system is illustrated in Figure 4.1.

The solving process is divided in two phases. In the first phase, the coordinator module sends all the clues of an input CP to a set of modules that generate several candidate answer lists. Each module returns a list of possible solutions for a clue. All lists are then aggregated by the *Merger* module, which uses the list confidence values and the probabilities of correctness of each candidate. Eventually, a single list of candidate-probability pairs is generated for each input clue. During the second phase WebCrow fills the crossword grid by solving a constraint-satisfaction problem. WebCrow selects a single answer from each candidate answer list, trying to satisfy the imposed constraints. The goal of this phase is to find an admissible solution that maximizes the number of words inserted in the CP. We now focus on two essential modules of WebCrow: the Web and the DB modules.

4.3.1 WebSearch Module (WSM)

WSM carries out four different tasks: (i) the retrieval of useful text snippets (TS) and web documents, (ii) the extraction of the answer candidates from these documents, (iii) the scoring/filtering of the candidate lists, and (iv) the estimation of the list confidence.

The retrieval of TS is performed using the Google API, building a query with the clue. The word list generator extracts possible candidate answers from TS or Web documents by picking the terms (also multiwords) of the correct length. The generated lists are ranked by merging the confidence computed by two filters: the statistical filter and morphological filter. The score associated with each candidate word w is given by the heuristic formula:

$$p(w, C) = k(score_{sf}(w, C) \times score_{mf}(w, C)),$$

where C is the target clue, k is a constant tuned on a validation set that fulfills the probability requirement $\sum_{i=0}^{n} p(w_n, C) = 1$, $score_{sf}(w, C)$ is computed using statistical information extracted from the text, e.g., the classical TF*IDF, and $score_{mf}(w, C))$ is based on morphological features of w.

4.3.2 Database Module (CWDB)

The knowledge about previous CPs is essential for solving new games. Some clues inevitably repeat in different CPs, and the availability of a large DB of clue-answer pairs allows us to easily find the answers to such clues. WebCrow exploits the database of clue-answer pairs with three different modules:

- CWDB-EXACT, which simply checks the DB of clues for an exact match.
- CWDB-PARTIAL, which employs MySQL's partial match function, query expansion and positional term distances to compute clue-similarity scores.
- CWDB-DICTIO, which returns all the words having the same length of the expected answer length (which is known).

We improve the WSM and CWDB by applying LTR algorithms based on SVMs and tree kernels applied to structural representations. In the next section, we provide a detailed description of our models.

4.4 Learning to Rank with Kernels

Our reranking framework uses a preference kernel reranking approach (e.g., see Shen and Joshi [2005]). From an architectural perspective, it is rather similar to the QA framework based on kernels in Severyn et al. [2013a]. However, to tackle the novelty of the task, especially for clue DB retrieval, we modeled new kernels. Here, we first describe the general framework, and then we instantiate the latter for two reranking tasks.

4.4.1 Kernel Framework

The framework takes a textual query and retrieves a list of related text candidates using a search engine (applied to the Web or a DB), according to some similarity criteria. Then, the query and candidates are processed by an NLP pipeline. The pipeline is based on the UIMA framework [Ferrucci and Lally, 2004], and contains many text analysis components. For our specific problem we use: a tokenizer[2], a sentence detector[1], a lemmatizer[1], a POS tagger[1], a chunker[3], and a stopword marker[4]. The output annotations are used by additional components to produce structural models which represent clues and TS, i.e., text fragments are converted into trees.

We use trees and feature vectors to represent pairs of clues and TS. Then we train kernel-based rerankers for reordering the candidate lists from the search engine. Since the syntactic parsing accuracy can impact the quality of our structure and the accuracy of our LTR algorithms, we prefer shallow syntactic trees over full syntactic representations.

The UIMA pipeline for our experiments has been actively developed during the past years, and eventually evolved into **RelTextRank** [Tymoshenko et al., 2017]. RelTextRank is a highly-flexible pipeline (available on GitHub[5]) for the creation of structural kernel-based systems for relational learning from text.

In the next section, we describe the structures we use in our kernels, we briefly mention the kernels that are building blocks for our models and finally, we show the reranking models for both tasks: TS and clue reranking.

[2] http://nlp.stanford.edu/software/corenlp.shtml

[3] http://cogcomp.cs.illinois.edu/page/software_view/13

[4] Based on a standard stoplist.

[5] https://github.com/iKernels/RelTextRank

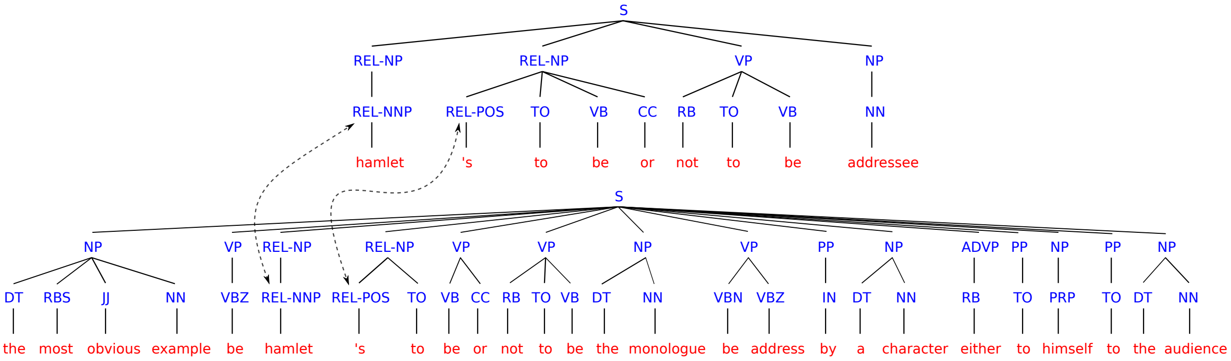

Figure 4.2: Shallow syntactic trees of clue (top) and snippet (bottom) and their relational links.

4.4.2 Relational Shallow Tree Representation

The structures we adopt are similar to those defined in our previous work on answer sentence selection [Severyn et al., 2013a]. They are shallow syntactic trees built from POS tags grouped into chunks. Each clue and its related text fragment (either a TS or clue) are encoded into a tree having word lemmas as leaves, and POS tags as preterminals. The nodes at the upper level group POS tags into chunks. For example, the tree at the top of Figure 4.2, shows a shallow tree for the clue: *Hamlet's "To be, or not to be" addressee*, whereas the lower tree represents a retrieved TS containing the answer, *himself*: *The most obvious example is Hamlet's "To be or not to be" ... the monologue is addressed by a character either to himself or to the audience.*

We use a special REL tag to link the clue/snippet trees above such that structural relations will be captured by tree fragments. The links are established as follows: words from a clue and a snippet that have a common lemma get their parents (POS tags) and grandparents, i.e., chunk labels, marked by a prepending REL tag. We build such structural representations for the two tasks of snippet and similar clue reranking.

4.4.3 Tree Kernels for Candidate Reranking

We experiment with different tree kernels in our classifiers. We use the **Syntactic Tree Kernel** (STK), the **STK**$_b$ which allows leaves (words) to be part of the feature space, the **Subtree Kernel** (SbtK) which only generates complete subtrees, and the **Partial Tree Kernel** (PTK) which generates all possible connected tree fragments. More details about these structural kernels can be found in Section 2.4.

4.4.4 Snippet Reranking

The task of snippet reranking consists in reordering the list of snippets retrieved from the search engine. such that those containing the correct answer are pushed to the top of the list. For this purpose, we transform the target clue into a search query, and retrieve candidate text snippets.

We rerank snippets using a preference reranking approach [Shen and Joshi, 2005]. This means that two snippets are compared to determine which one contains the answer with higher probability. Since we want to automatically learn this with kernel methods, we apply the following preference kernel:

$$P_K(\langle s_1, s_2\rangle, \langle s'_1, s'_2\rangle) = K(s_1, s'_1) + K(s_2, s'_2) - K(s_1, s'_2) - K(s_2, s'_1), \qquad (4.1)$$

where s_r and s'_r refer to two sets of candidates associated with two rankings and K is a kernel applied to pairs of candidates. We represent the latter as pairs of clue and snippet trees. More formally, given two candidates, $s_i = \langle s_i(c), s_i(s)\rangle$ and $s'_i = \langle s'_i(c), s'_i(s)\rangle$, whose members are the clue and snippet trees, we define

$$K(s_i, s'_i) = TK(s_i(c), s'_i(c)) + TK(s_i(s), s'_i(s)),$$

where TK can be any tree kernel function, e.g., STK or PTK. To conclude, it should be noted that, in order to add traditional feature vectors to the reranker, it is enough to add the product $(\vec{x}_{s_1} - \vec{x}_{s_2}) \cdot (\vec{x}_{s'_1} - \vec{x}_{s'_2})$ to the structural kernel P_K, where $\vec{x}_s$ is the feature vector associated with the snippet s.

4.4.5 Similar Clue Reranking

WebCrow creates answer lists by retrieving clues from the DB of previously solved crosswords. This is done using the classical SQL operator and full-text search. We instead verify the hypothesis that a search engine may be better suited for the task. Thus we index the DB clues and their answers with the open source search engine Lucene [McCandless et al., 2010], using the state-of-the-art BM25 retrieval model. This change alone significantly improves the quality of the retrieved clue list, which is further refined by applying reranking. The full procedure consists into: (i) retrieving a list of similar clues using a search engine, and (ii) moving the clues which are more similar to the query clue (and then share the same answer with high probability) at the top. For example, Table 4.1 shows the first five clues retrieved for the query clue: *Kind of connection for traveling computer users.* The search engine retrieves the wrong clue first (*Kind of connection for traveling computer users*) since it shares more words with the query.

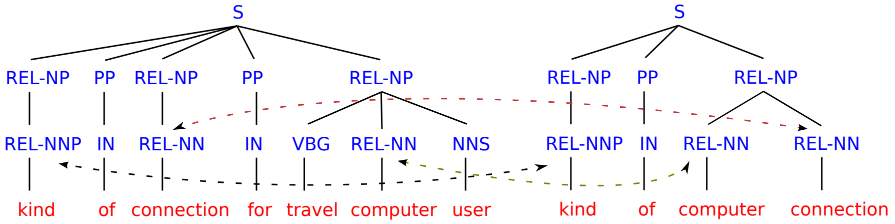

Figure 4.3: Two similar clues leading to the same answer.

Rank	Clue	Answer
1	Kind of support for a computer user	tech
2	Kind of computer connection	wifi
3	Computer connection	port
4	Comb users	bees
5	Traveling bag	grip

Table 4.1: Clue ranking for the query: *Kind of connection for traveling computer users (wifi).*

To solve these kinds of problems by also enhancing the generalization power of our reranking algorithm, we use a structural representation similar to the one that we illustrated for TS. The main difference is that now the reranking pair is only constituted by clues. For example, Figure 4.3 shows the representation of the pairs constituted by the query clue and the correct clue ranked in second position (see Table 4.1). The relational arrows suggest a syntactic transformation from `connection for * computer` to `computer connection`, which can be used by the reranker to prefer the correct clue to the wrong one. Note that such transformation corresponds to the pair of tree fragments:

`[S [REL-NP[REL-NN]][PP][NP[VBG][REL-NN]]]` → `[S [REL-NP[REL-NN][REL-NN]]]`,

where the `REL-NN` and `REL-NN` nodes define the arguments of the syntactic transformation. Such fragments can be generated by PTK, which can thus be used for learning clue paraphrasing. To build the reranking training set, we use the training clues as queries for the search engine, which draws candidates from the DB of indexed clues. We stress the fact that the DB of clues is disjoint from the clues in the training and test set. Thus, identical clues are not present across sets. At classification time, a new clue is used as a search query. Similar candidate clues are retrieved and paired woth the query clue.

4.4.6 Similarity Feature Vector

Structural representations are enhanced with features that capture the degrees of similarity between clues within a pair. Here we describe the main feature groups.

DKPro Similarity. These similarity features come from a top performing system in the Semantic Textual Similarity (STS) task, namely DKPro from the UKP Lab [Bär et al., 2013]. DKPro includes the following syntactic similarity metrics, operating on string sequences, and some advanced semantic similarities:

Longest common substring measure [Gusfield, 1997]. It determines the length of the longest substring shared by two text segments.

Longest common subsequence measure [Allison and Dix, 1986]. It extends the notion of substrings to word subsequences, by applying word insertions or deletions to the original input text pairs.

Running-Karp-Rabin Greedy String Tiling [Wise, 1996]. It provides a similarity between two sentences by counting the number of shuffles in their subparts.

Resnik similarity [Resnik, 1995]. The WordNet hypernymy hierarchy is used to compute a measure of semantic relatedness between concepts expressed in the text. The aggregation algorithm by Mihalcea et al. [Mihalcea et al., 2006] is applied to extend the measure from words to sentences.

Explicit Semantic Analysis (ESA) similarity [Gabrilovich and Markovitch, 2007]. It represents documents as weighted vectors of concepts learned from resources such as Wikipedia, WordNet and Wiktionary.

Lexical Substitution [Biemann, 2013]. A supervised word sense disambiguation system is used to substitute a wide selection of high-frequency English nouns with generalizations. Resnik and ESA features are then computed on the transformed text.

Reranking features. We designed the following features to especially capture the peculiarity of the CP reranking task:

Feasible Candidate. A binary feature signaling the presence or absence of words with the same length of the clue answer (only used for snippet reranking).

Term overlap features. Cosine similarity of text pairs encoded into different set of n-grams, e.g., from words, lemmas or POS tags, by also applying stop-words.

Kernel similarities. These are computed using string kernels applied to sentences, or PTK applied to structural representations with and without embedded relational information. This similarity is computed between the members of a $\langle clue, snippet\rangle$ or a $\langle clue, clue\rangle$ pair, thus, it is different from its use in the preference kernel.

4.5 Experiments on Snippet Reranking and Clue Retrieval

Our experiments aim at demonstrating the effectiveness of our models on two different tasks: (i) Snippet reranking and (ii) Similar Clue Retrieval (SCR). Additionally, we measure the impact of our best model for SCR in the WebCrow system. Our referring database of clues is composed by 1,158,202 clues, which belong to eight different crossword editors (downloaded from the Web[6]). We use the latter to create one dataset for snippet reranking and one dataset for clues retrieval.

4.5.1 Experimental Setup

To train our models, we use SVM-light-TK[7], which enables the use of structural kernels [Moschitti, 2006] in SVM-light [Joachims, 2002]. We use the default parameters, but for a polynomial kernel of degree 3 applied to the explicit feature vectors.

To measure the impact of the rerankers as well as the baselines, we track the most common metrics for assessing the accuracy of QA and retrieval systems, i.e.: Recall at rank 1 (R@1 and 5), Mean Reciprocal Rank (MRR), Mean Average Precision (MAP), the average Recall (AvgRec). R@k is the percentage of questions with a correct answer ranked at the first position. MRR is computed as follows: $MRR = \frac{1}{|Q|}\sum_{q=1}^{|Q|}\frac{1}{rank(q)}$, where *rank(q)* is the position of the first correct answer in the candidate list. For a set of queries Q MAP is the mean over the average precision scores for each query: $\frac{1}{Q}\sum_{q=1}^{Q} AveP(q)$. AvgRec and all the measures are evaluated on the first 10 retrieved snippets/clues. For training and testing the reranker, we use only the first 10 snippets/clues retrieved by the search engine.

4.5.2 Snippet Reranking

Querying the Web search engine is a very slow process, and this prevents us to retrieve and use entire documents. Therefore, we only consider the snippets in the result pages. In addition, since our reranking approach does not address special clues such as anagrams, linguistic games and *fill-in-the blank clues*, we remove them from our dataset. We crawl

[6] `http://www.crosswordgiant.com`

[7] `http://disi.unitn.it/moschitti/Tree-Kernel.htm`

Model	MAP	MRR	AvgRec	REC@1	REC@5
Google	16.00	18.09	69.00	12.50	24.80
V	18.00	19.88	76.00	14.20	26.10
SbtK	17.00	19.6	75.00	13.80	26.40
STK	18.00	20.44	76.00	15.10	27.00
STK$_b$	18.00	20.68	76.00	15.30	27.40
PTK	19.00	21.65	77.00	16.10	28.70
V+SbtK	**20.00**	**22.39**	**80.00**	**17.20**	29.10
V+STK	19.00	20.82	78.00	14.90	27.90
STK$_b$	19.00	21.20	79.00	15.60	28.40
V+PTK	19.00	21.68	79.00	16.00	**29.40**
V+DK	18.00	20.48	77.00	14.60	26.80
V+DK+SbtK	**20.00**	22.29	**80.00**	16.90	28.70
V+DK+STK	19.00	21.47	79.00	15.50	28.30
V+DK+STK$_b$	19.00	21.58	79.00	15.4	28.60
V+DK+PTK	**20.00**	22.24	**80.00**	16.80	29.30

Table 4.2: Snippet reranking.

all our data from the Web. We issue each clue as a Google query, and download the first 10 snippets in the result page. In order to reduce the dataset noise, we blacklist urls that may contain clues and answers, e.g., crossword websites. The training and test sets contain 20,000 and 6,000 clues respectively.

We implement and compare a number of models for reranking the correct snippets higher, i.e., containing the answer to the clue. They are listed in the first column of Table 4.2. V is the approach that uses only the reranking feature set (see Section 4.4.6); DK is the model using the DKPro features; the systems ending in TK are described in Section 4.4.3, and the plus operator indicates models that sum the specified kernels.

Results are reported in Table 4.2. Depending on the target measure they suggest slightly different findings. Hereafter, we comment on MRR as it is the most interesting performance metric from a ranking viewpoint. We note that: (i) Google is improved by the reranker based on the new feature vector by 2 absolute points; (ii) DK+V improves on V by just half point; (iii) PTK provides the highest result among individual systems; (vi) combinations improve on the individual systems; and (v) overall, our reranking improves on the ranking of paragraphs of Google by 3 points in MRR and 5 points in accuracy on the first candidate (REC@1).

Model	MAP	MRR	AvgRec	REC@1	REC@5
MB25	69.00	73.78	80.00	62.11	81.23
WebCrow	-	53.22	58.00	39.60	62.85
SbtK	52.00	54.72	69.00	36.50	64.05
STK	63.00	68.21	77.00	54.57	76.11
STK$_b$	63.00	67.68	77.00	53.85	75.63
PTK	65.00	70.12	78.00	57.39	77.65
V+SbtK	68.00	73.26	80.00	60.95	81.28
V+STK	71.00	76.01	82.00	64.58	83.95
V+STK$_b$	70.00	75.68	82.00	63.95	83.77
V+PTK	71.00	76.67	82.00	65.67	84.07
V+DK	71.00	76.76	81.00	65.55	84.29
V+DK+SbtK	72.00	76.91	82.00	65.87	84.51
V+DK+STK	**73.00**	**78.37**	**84.00**	**67.83**	**85.87**
V+DK+STK$_b$	**73.00**	78.29	**84.00**	67.71	85.77
V+DK+PTK	**73.00**	78.13	83.00	67.39	85.75

Table 4.3: Reranking of similar clues.

4.5.3 Similar clue retrieval

We compile a crossword database of 794,190 unique pairs of clue-answer. Using the clues of this set, we create three different sets: a training and a test set, plus a database of clues that we index for the retrieval of similar clues. The database contains 700,000 unique clue-answer pairs. The training set contains 39,504 clues whose answer may be found in database. Using the same approach, we create a test set containing 5,060 clues that (i) are not in the training set, and (ii) have at least an answer in the database.

We perform experiments with all the models from the snippet reranking section. Since WebCrow includes a database module, Table 4.3 contains an extra row indicating its accuracy. We note that: (i) BM25 shows a very accurate MRR, 73.78%. It largely improves on WebCrow by about 20.5 absolute percent points, demonstrating the superiority of an IR approach over DB methods. (ii) All TK types do not improve alone on BM25: this happens since they do not exploit the initial rank provided by BM25. (iii) All the feature vector and TK combinations achieve high MRR, up to 4.5 absolute percent points of improvement over BM25 and thus 25 points more than WebCrow. Such high result is promising in the light of improving WebCrow for the final task of solving complete CPs.

Model	MRR	REC@1	REC@5	REC@10
WebCrow	41.00	33.00	51.00	58.00
Our Model	46.00	39.00	56.00	59.00

Table 4.4: Performance on the word list candidates averaged over the clues of 10 entire CPs.

Model	%Correct words	%Correct letters
WebCrow	34.45	49.72
Our Model	39.69	54.30

Table 4.5: Performance given in terms of correct words and letters averaged on the 10 CPs.

4.5.4 Impact on WebCrow

In these experiments, we use our reranker for similar clues on 10 complete CPs (for a total of 760 clues) from the New York Times and the Washington Post. This way, we can measure the impact of our model on the full task carried out by WebCrow. More specifically, we give our reranked list of answers to WebCrow, in place of the list that would have been extracted by the CWDB module. It should be noted that for evaluating the impact of our list, we disabled the access of WebCrow to other lists, e.g., dictionaries. This means that the absolute resolution accuracy of WebCrow using our and its own lists can be higher (see Ernandes et al. [2008] for more details).

The first result that we derive is the accuracy of the answer lists produced for the 10 test CPs. The results are reported in Table 4.4. We note that the improvement on WebCrow is lower: this happens because a significant number of clues are not solved using just the clue DB. However, when we compute the accuracy on the complete CPs resolution, the impact is still remarkable as reported in Table 4.5. Indeed, the results show that when the lists produced by our reranker are used in WebCrow, the latter improves by more than 5 absolute percent points in both word and character accuracy.

4.6 Learning to Rank Aggregated Answers

In Barlacchi et al. [2014b], we showed that IR techniques can improve clue retrieval, but our approach was limited to providing better ranking of clues, whereas CP solvers require finding answers to fill the puzzle squares. In other words, given the list of similar clues retrieved by an IR system, a clue aggregation step and a further reranking process is needed to provide the list of answer candidates to the solver. More specifically, the clues in the rank generate a set of possible answer. A straightforward way to sort the answers is to consider the rank of theirs associated clues as a vote. However, this is subject to the problem that answers are relevant to the query with different probabilities. Even the LTR algorithm score is not calibrated on the entire list of clues and answers.

In this section, we study techniques for the aggregation of answers and their reranking, with the goal of solving the above problem. First of all, we calibrate the scores from the LTR model using logistic regression (LGR). This way, the voting approach uses calibrated probabilities and improves on previous results. Secondly, we combine the evidence provided by the clues associated with the same answer: we define a representation of each answer based on aggregate features extracted from corresponding clues, e.g., their average, maximum and minimum reranking score. We experiment with this new answer representation with LGR and SVM^{rank} [Joachims, 2002] models. Thirdly, we present an updated dataset for clue retrieval: this time it contains 2,131,034 clues and associated answers. We carried out two sets of experiments on two main tasks: (i) clue reranking, which focuses on improving the rank of clues similar to a target clue; and (ii) answer reranking, which targets the list of aggregated clue answers.

Our findings demonstrate that (i) the search engine greatly improves on DB methods for clue reranking, i.e., BM25 improves on SQL query by 6 absolute percent points; (ii) kernel-based rerankers improve SQL by more than 15 absolute percent points; and (iii) our answer aggregation procedure improves the Recall (Precision) at rank 1 by additional 2 points absolute over the best results.

As a starting point, we adopt the reranking framework applied to CPs described in Barlacchi et al. [2014b] and in the previous sections. In this work, we extend the feature sets for capturing the degrees of similarity between clues. In addition to the **DKPro Similarity (DKP)** features described in 4.4.6, we include the following features.

iKernels features (iK). Set of similarity features taking into account syntactic information captured by n-grams, and using kernels:

Syntactic similarities. Several cosine similarity measures are computed on n-grams (with $n = 1, 2, 3, 4$) of word lemmas and POS tags.

Kernel similarities. Computed using (i) string kernels applied to clues, and tree kernels applied to structural representations.

WebCrow features (WC). We included the similarity measures computed on the clue pairs by WebCrow and the search engine as features.

Lucene Score. BM25 score of the target candidate.

Clue distance. It quantifies how dissimilar the input clue and the retrieved clue are, with a formula which is mainly based on the well known Levenshtein distance.

4.6.1 Aggregation Models for Answer Reranking

Subsets of clues retrieved by the search engine may share the same answers. Since the reranker associates a score to each clue, a strategy to combine such scores is needed. In the following sections, we aggregate the evidence given by clues associated with the same answer, and we extract relevant features. We designed two different strategies: (i) the first applies LGR to the reranker scores, obtaining probabilities that can be aggregated for each unique answer; (ii) the second uses an answer candidate representation that contains features derived from all the clues associated with it, i.e., aggregated features using standard operators such average, min. and max.

Logistic Regression Model. The search engine and the reranker does not output probabilities for clues and answers. In contrast, the LGR outputs can be interpreted as a probabilities. The latter, learned using additional features, are more effective for aggregation. We apply the following formula: $Score(G) = \frac{1}{n}\sum_{c \in G} \frac{P^{LR}(y=1|\vec{x}_c)}{rank_c}$ to obtain a single score for each unique answer candidate, where c is the answer candidate, G is the set of clue answers equal to c, and n is the size of the answer candidate list. $\vec{x}_c$ is the feature vector associated with $c \in G$, $y \in \{0, 1\}$ is the binary class label ($y = 1$ when c is the correct answer). $rank_c$ is the rank assigned from the reranker to the word c. At the end, we divide the probability by the rank of the answer candidate to reduce the contribution of bottom candidates. The conditional probability computed by the linear model is the following: $P^{LR}(y = 1|c) = \frac{1}{1+e^{-y\vec{w}^T\vec{x}_c}}$, where $\vec{w} \in \mathbb{R}^n$ is a weight vector [Yu et al., 2011].

Learning to rank aggregated answers. We apply SVM^{rank} to the sets of clues associated to the same answer candidate. Aggregated features for each group are computed by averaging the features from the individual clues, which were fed to the first reranker. We call these features **FV**. Additionally, we compute the sum and the average of the scores, the maximum score, the minimum score and the term frequency of the word in the CPDB Dataset. We call them (**AVG**). Then, we model the occurrences of the answer instance in the list by means of positional features: we use n features, where n is the size of our candidate list (i.e., 10). Each feature corresponds to the position of the answer instance in the list. We call them (**POS**).

Model	MRR	SUC@1	SUC@5
WebCrow (WC)	64.65	57.14	74.98
BM25	75.17	63.78	90.40
RR (iK)	78.01	67.34	92.32
RR (iK+DKP)	80.89	71.62	93.14
RR (iK+DKP+WC)	**81.70**	**72.50**	**94.02**

Table 4.6: Similar Clue Reranking.

4.7 Experiments on Answer Aggregation

The experiments that follow compare a set of ranking models for the task of clue retrieval: the WebCrow automatic solver, the BM25 retrieval model, and several other rerankers. Most importantly, we evaluate our models for the aggregation and reranking of answers, based on LGR and SVM^{rank} models.

4.7.1 Database of Previously Solved CPs (CPDB)

We compile a crossword corpus combining (i) the CPs downloaded from the Web[8] and (ii) the clue database provided by Otsys[9]. We removed duplicates, fill-in-the-blank clues (which are better solved by using other strategies) and clues representing anagrams or linguistic games. The resulting compressed dataset, called CPDB, contains 2,131,034 unique and standard clues, with associated answers.

4.7.2 Experimental Setup

SVM-light-TK[10] is the software used to train the TK models. We used the default parameters, but for the polynomial kernel of degree 3 which is applied to the explicit feature vectors. To measure the impact of the rerankers as well as the baselines, we use: success at rank 1 (SUC@1), which is the percentage of questions with a correct answer in the first position; Mean Reciprocal Rank (MRR), which is computed by $\frac{1}{|Q|}\sum_{q=1}^{|Q|}\frac{1}{rank(q)}$, where *rank(q)* is the position of the first correct answer in the candidate list; and success at rank 5 (SUC@5), which is the percentage of questions with at least one correct answer in the first 5 reranked clues.

[8] http://www.crosswordgiant.com

[9] http://www.otsys.com/clue

[10] http://disi.unitn.it/moschitti/Tree-Kernel.htm

Model	MRR	SUC@1	SUC@5
Raw voting	41.33	17.44	78.48
LGR voting	83.16	73.18	96.68
SVM (AVG+POS)	83.49	73.82	96.78
SVM (AVG+POS+FV)	**83.95**	**74.60**	**96.78**
LGR (AVG+POS+FV)	81.70	73.54	96.74

Table 4.7: Answer reranking.

4.7.3 Ranking Results

The clues in CPDB are used to generate the training and test set for the reranking task. The CPDB clues are indexed, after reserving some input clues that are eventually used to query the index with a search engine. For each one of these input clues, the search engine returns a list of similar candidate clues. The training set is composed by 8,000 unique pairs of clue/answer which have at least one correct answer in the first 10 candidates retrieved by the search engine. We also create a test set containing 5,000 clues that are not contained in the training set.

We then evaluate (i) BM25 and (ii) reranking models (RR). We also report the accuracy of the database module in WebCrow. We use the BM25 implementation in Lucene [McCandless et al., 2010] as IR baseline. To rerank these lists, we adopt different combinations of features in the rerankers. The results in Table 4.6 show that: (i) BM25 produces an MRR of 75.17%, which improves on WebCrow by more than 6.5 absolute percent points, demonstrating the superiority of an IR approach over DB methods; (ii) RR (iK) achieves a higher MRR, up to 4 percent absolute of improvement over BM25 and thus about 10.5 points more than WebCrow. With respect to this model, the improvement on MRR of (iii) RR (iK+DKPro) is up to 1.2 percent points and finally, (iv) RR (iK+DKP+WC) improves the best results of another full percent point. Table 4.7 shows the results for answer reranking: (i) voting the answer using the raw score of the reranker is not effective; (ii) voting, after transforming scores into probabilities with LGR, improves on the best clue reranking model in terms of SUC@1 and MRR; (iii) the SVM^{rank} aggregation model using AVG and POS feature sets improves on the LGR voting model; (iv) when FV are added we notice a further increase in MRR and SUC@1; (v) LGR on the same best model AVG+POS+FV is not effective, showing that ranking methods are able to refine answer aggregation better than regression methods.

4.8 Conclusion

In this chapter, we presented work which improves the automatic CP resolution by modeling two innovative reranking tasks for: (i) CP answer list derived from Web search and (ii) CP clue retrieval from clue DB. Our rankers are based on SVMs and structural kernels, where the latter are applied to robust shallow syntactic structures. Our model applied to clue reranking allows us to learn clue paraphrasing by exploiting relational syntactic structures representing pairs of clues. A further step, which employs LTR aggregation methods, allows us to combine the evidence coming from multiple clues sharing the same answer, and refine the result lists. We also created two different corpora for snippets reranking and clue similarity, and made them available to the research community.

Chapter 5

Kernels Based on Neural Models

In this chapter, we connect two technologies, namely neural networks and tree kernels, to design models capable of performing automatic feature engineering.

We first encode pairs of text with the activations of a trained neural network, and use them as features in our kernel-based classifier. A kernel on such features is combined with a structural kernel, and the resulting model shows greater accuracy than single kernel models, and than a neural network in a scarce data setting.

Our second contribution in this chapter goes beyond using neural network as feature extractors. Indeed, we incorporate a similarity function between lexical items, directly into the kernel similarity computation. Such function is learned using a deep network which produces neural representations of words that contain information about the adjacent words and are optimized for the final classification task.

5.1 Tree and Deep Networks-based Kernels for Reranking

In this section, we present our work that compares and then intersects tree kernel and deep network models [Severyn et al., 2015]. The task is familiar: similar clue retrieval for the resolution of CPs. We create a dataset that enables us to train a Distributional Neural Network (DNN) for reranking clue pairs. Our DNN is computationally efficient, and can thus take advantage of such large datasets showing a large improvement over the TK approach, when the latter uses small training data. In contrast, when data is scarce, TKs outperform DNNs. The large-scale dataset for clue retrieval[1] contains 2,000,493 clues with their associated answers, and it is ideal for data-hungry learning models.

To assess the effectiveness of our DNN, we compare it with the state of the art model [Nicosia et al., 2015] presented in the previous chapter. The experimental results

[1] `http://ikernels-portal.disi.unitn.it/projects/webcrow/`

demonstrate that:

1. DNNs are efficient and shine when trained with large amounts of data;
2. when DNNs can learn from millions of examples, they largely outperform traditional feature-based rerankers as well as kernel-based models;
3. if training data is scarce, tree kernel-based models are more accurate than DNNs.

5.1.1 Clue Retrieval and Reranking

Given a target clue for which we want to find an answer, we use the clue as a query for a DB of previously solved CPs. The purpose is to retrieve similar definitions which may have the same answer.

In Chapter 4, we used the BM25 retrieval model to generate the lists of clues, which were further refined by our rerankers. The reranking step is important because search engines often do not retrieve the correct clues in the first positions.

5.1.2 Reranking with Kernels

We apply our reranking system for QA [Severyn et al., 2013a,b] adapted for similar clue retrieval, which we already presented in Section 4.4. The system first retrieves a list of related clues, by using the target clue as a query in a search engine (Web or DB). Then, the query and the candidate clues are represented as shallow syntactic structures (generated by running a set of NLP parsers) and traditional similarity features, which are fed to a kernel-based reranker.

5.1.3 Distributional Models for Clue Reranking

The architecture of our distributional matching model for measuring similarity between clues is presented in Fig. 5.1. Its main components are:

(i) sentence matrices $\mathbf{s}_{c_i} \in \mathbb{R}^{d \times |\mathbf{c}_i|}$, obtained by the concatenation of the word vectors $\mathbf{w}_j \in \mathbb{R}^d$ (with d being the size of the embeddings) of the corresponding words w_j from the input clues $\mathbf{c}_i$;

(ii) a distributional sentence model $f : \mathbb{R}^{d \times |\mathbf{c}_i|} \to \mathbb{R}^m$ that maps $\mathbf{s}_{c_i}$ to a fixed-size vector representations $\mathbf{x}_{c_i}$ of size m;

(iii) a layer for computing the similarity between the encoded input clues, using a similarity matrix $\mathbf{M} \in \mathbb{R}^{m \times m}$ – an intermediate vector representation $\mathbf{x}_{c_1}$ of a clue $\mathbf{c}_1$ is projected to a $\tilde{\mathbf{x}}_{c_1} = \mathbf{x}_{c_1}\mathbf{M}$, which is then matched with $\mathbf{x}_{c_2}$ [Bordes et al., 2014a], i.e., by computing a dot-product $\tilde{\mathbf{x}}_{c_1}\mathbf{x}_{c_2}$, that results in a single similarity score x_{sim};

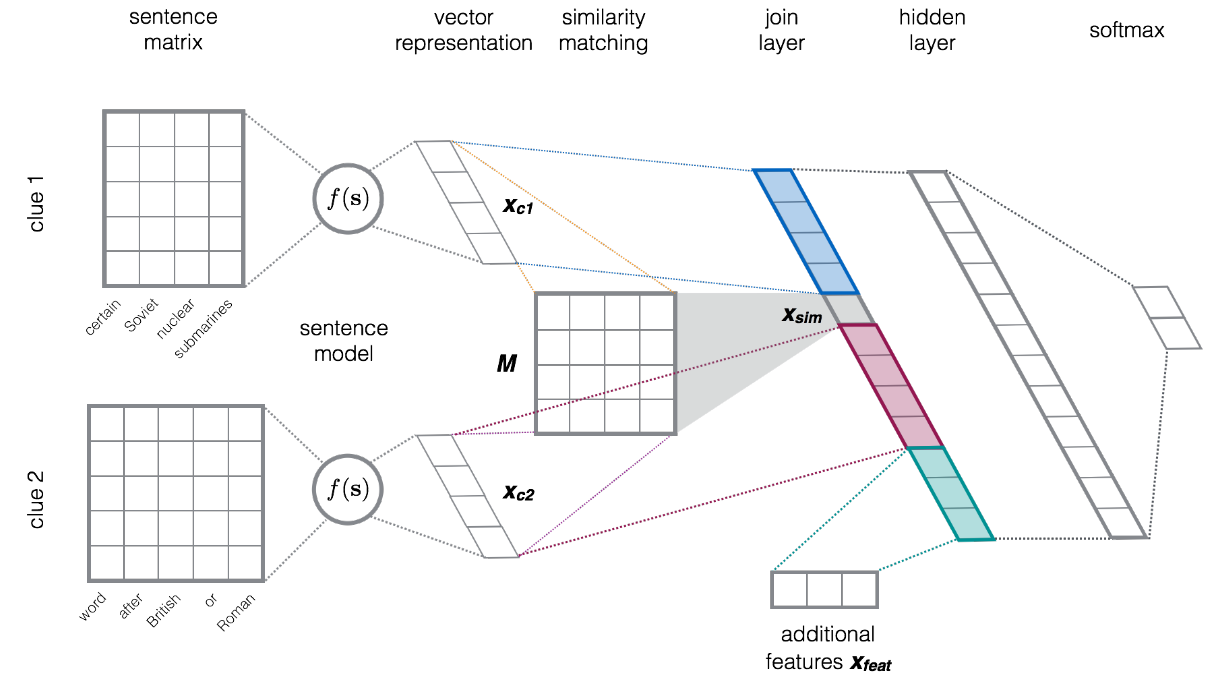

Figure 5.1: Distributional model for computing similarity between clues [Severyn et al., 2015].

(vi) a set of fully-connected hidden layers that models the similarity between clues using their vector representations produced by the sentence model (also integrating the single similarity score from the previous layer); and

(v) a *softmax* layer that outputs probability scores reflecting how well the clues match with each other.

In this model, we adopt an NBOW encoder for the clues, where $f(\mathbf{s}_{\mathbf{c}_i}) = \sum_i \mathbf{w}_i/|\mathbf{c}_i|$, i.e., the word vectors, are averaged in a single fixed-sized vector $\mathbf{x} \in \mathbb{R}^d$. Our preliminary experiments revealed that this simpler model works just as well as more complicated single or multi-layer convolutional architectures. We conjecture that this is largely due to the nature of the language used in clues, which is very dense and where the syntactic information plays a minor role.

Considering recent deep learning models for matching sentences, our network is most similar to the models in Hu et al. [2014] applied for computing sentence similarity and in [Yu et al., 2014] (answer sentence selection in QA) with the following differences:

(i) in contrast to the more complex convolutional sentence models in Hu et al. [2014]; Yu et al. [2014], our sentence model is composed of a single averaging operation;

(ii) our network computes the similarity between the vector representations of the clues with two methods: (i) computing the similarity score obtained by transforming one clue into another using a similarity matrix **M** (explored in Yu et al. [2014]), and (ii) directly modelling interactions between intermediate vector representations of the input clues via fully-connected hidden layers (used by Hu et al. [2014]).

5.1.4 Experimental Settings

Our experiments compare different ranking models: BM25 as the IR baseline, rerankers, and our distributional neural network (DNN) for the task of clue reranking.

Data. We compiled our crossword corpus combining (i) CPs downloaded from the Web[2] and (ii) the clue database provided by Otsys[3]. We removed duplicates, fill-in-the-blank clues (which are better solved by using other strategies) and clues representing anagrams or linguistic games. We collected over 6.3M pairs of clue/answer, and after duplicate removal, we obtained a dataset containing 2M unique and standard clues, with associated answers, which we called CPDB. We used these clues to build a Small Dataset (SD) and a Large Dataset (LD) for reranking. The two datasets are based on pairs of clues: query and retrieved clues. Such clues are retrieved using a BM25 model on CPDB.

For creating SD, we used 8k clues that (i) were randomly extracting from CPDB and (ii) satisfying the property that at least one correct clue (i.e., having the same answer of the query clue) is in the first retrieved 10 clues (of course the query clue is eliminated from the ranked list provided by BM25). In total we got about 120K examples, 84,040 negative and 35,960 positive clue[4].

For building LD, we collected 200k clues with the same property above. More precisely we obtained 1,999,756 pairs (10×200k minus few problematic examples) with 599,025 positive and 140,0731 negative pairs of queries with their retrieved clues. Given the large number of examples, we only used such dataset in classification modality, i.e., we did not form reranking examples (pairs of pairs).

Structural model. We use SVM-light-TK[5], which enables the use of structural kernels [Moschitti, 2006]. We applied structural kernels to shallow tree representations, and a polynomial kernel of degree 3 to feature vectors (FV).

[2] http://www.crosswordgiant.com

[3] http://www.otsys.com/clue

[4] A true reranker should be built using pairs of clue pairs, where the positive pairs are those having the correct pair as the first member. This led to form 127,109 reranking examples, with 66,011 positive and 61,098 negative pairs. However, in some experiments, which we do not report in this chapter, we observed that the performance of the simple classifier as well as the true reranker were similar, ans we decided to use the simpler classifier.

[5] `http://disi.unitn.it/moschitti/Tree-Kernel.htm`

Training classifiers with the Small Dataset (SD) (120K instances)

	BM25	SVMp	SVM(TK)	DNNM$_{SD}$	SVMp(DNNF$_{LD}$)	SVM(DNNF$_{LD}$,TK)
MRR	37.57	41.95	43.59	40.08	46.12	45.50
MAP	27.76	30.06	31.79	28.25	33.75	33.71

Training classifiers with the Large Dataset (LD) (2 million instances)

	BM25	SVMp	SVM(TK)	DNNM$_{LD}$	SVMp(DNNF$_{LD}$,−FV)	SVMp(DNNF$_{LD}$)
MRR	37.57	41.47	–	46.10	46.36	46.27
MAP	27.76	29.95	–	33.81	34.07	33.86

Table 5.1: SVM models and DNN trained on 120k (small dataset) and 2 millions (large dataset) examples. Feature vectors are used with all models except when indicated by −FV.

Distributional neural network model. We pre-initialize the word embeddings by running the `word2vec` tool Mikolov et al. [2013d] on the English Wikipedia dump. We opt for a skipgram model with window size 5 and filtering words with frequency less than 5. The dimensionality of the embeddings is set to 50. The input sentences are mapped to fixed-sized vectors by computing the average of their word embeddings. We use a single non-linear hidden layer (with the ReLU activation function), whose size is equal to the size of the previous layer.

The network is trained using SGD with shuffled mini-batches using the Adagrad update rule Duchi et al. [2011]. The batch size is set to 100 examples. We used 25 epochs with early stopping, i.e., we stop the training if there is no increase in accuracy on the dev set (a reserved 10% of the training set) in the last 5 epochs. The accuracy computed on the dev set is the Mean Average Precision (MAP) score. Fore extracting features from the DNN, we simply take the output of the hidden layer just before the softmax.

Evaluation. We used standard metrics widely used in QA: the Mean Reciprocal Rank (MRR) and Mean Average Precision (MAP).

5.1.5 Results

Table 5.1 summarizes the results of our different reranking models trained on a small dataset (SD) of 120k examples and a large dataset (LD) with 2M examples. The first column reports the BM25 result; the second column shows the performance of SVM perf

(SVM$_p$), which is a very fast variant of SVM, using FV; the third column reports the state-of-the-art model for crossword clue reranking [Nicosia et al., 2015], which uses FV vector and tree kernels, i.e., SVM(TK).

Regarding the other systems: DNNM$_{SD}$ is the DNN model trained on the small data (SD) of 120k training pairs; SVMp(DNNF$_{LD}$) is SVM perf trained with (i) the features derived from DNN trained on a large clue dataset *LD* and (ii) the FV; and finally, SVM(DNNF$_{LD}$,TK) is SVM using DNN features (generated from LD), FV and TK. It should be noted that:

(i) SVM$_p$ is largely improved by TK;

(ii) DNNM$_{SD}$ on relatively small data delivers an accuracy lower than FV;

(iii) if SVM$_p$ is trained with DNNM$_{LD}$, i.e., features derived from the dataset of 2M clues, the accuracy greatly increases; and

(iv) finally, the combination with TK, i.e., SVM(DNNF$_{LD}$,TK), does not significantly improve the previous results.

In summary, when a dataset is relatively small DNNM fails to deliver any noticeable improvement over the SE baseline even when combined with additional similarity features. SVM and TK models generalize much better on the smaller dataset. Additionally, it is interesting to see that training an SVM on a small number of examples enriched with the features produced by a DNN trained on large data gives us the same results of DNN trained on the large dataset. Hence, it is desired to use larger training collections to build an accurate distributional similarity matching model that can be then effectively combined with other feature-based or tree kernel models, although at the moment the combination does not significantly improve TK models. Regarding the LD training setting it can be observed that:

(i) the second column shows that adding more training examples to SVM$_p$ does not increase accuracy (compared with SD result);

(ii) DNNM$_{LD}$ delivers high accuracy suggesting that a large dataset is essential to its training; and

(iii) again SVM$_p$ using DNN features deliver state-of-the-art accuracy independently of using or not additional features (i.e., see −FV, which excludes the latter).

5.1.6 Summary and Future Work

In this section, we have explored various reranker models to improve automatic CP resolution. The most important finding is that our distributional neural network model is very effective in establishing similarity matching between clues. We combine the features produced by our DNN model with other rerankers to greatly improve on the previous state-of-the-art results. Finally, we collected a very large dataset composed of 2 millions clue/answer pairs that can be useful to the NLP community for developing semantic textual similarity models.

Future research will focus on increasing the number of links between entities in the text. In particular, our model which exploits Linked Open Data in QA [Tymoshenko et al., 2014] seems very promising to find correct answer to clues. This, as well as further research, will be integrated in our CP system described in Barlacchi et al. [2015].

5.2 Semantic Tree Kernels using Networks-based Similarities

As we have seen, structural kernels can automatically represent syntactic and semantic structures in terms of substructures, showing high accuracy in several tasks, e.g., relation extraction [Nguyen et al., 2009; Nguyen and Moschitti, 2011; Plank and Moschitti, 2013; Nguyen et al., 2015], and sentiment analysis [Nguyen and Shirai, 2015].

At the same time, deep learning has demonstrated its effectiveness on a plethora of NLP tasks such as Question Answering (QA) [Severyn and Moschitti, 2015a; Rao et al., 2016], and parsing [Andor et al., 2016], to name a few. Deep learning models (DLMs) usually do not include traditional features; they extract relevant signals from distributed representations of words, by applying a sequence of linear and non linear functions to the input. Word representations are learned from large corpora, or directly from the training data of the task at hand.

Clearly, joining the two approaches above would have the advantage of easily integrating structures with kernels, and lexical representations with embeddings into learning algorithms. In this respect, the Smoothed Partial Tree Kernel (SPTK) is a noticeable approach for using lexical similarity in tree structures [Croce et al., 2011]. SPTK can match different tree fragments, provided that they only differ in lexical nodes. Although the results were excellent, the used similarity did not consider the fact that words in context assume different meanings or weights for the final task, i.e., it does not consider the context. In contrast, SPTK would benefit to use specific word similarity when matching subtrees corresponding to different constituency. For example, the two questions:

- *What famous model was married to Billy Joel?*
- *What famous model of the Universe was proposed?*

are similar in terms of structures and words but clearly have different meaning and also different categories: the first asks for a human (the answer is Christie Brinkley) whereas the latter asks for an entity (an answer could be *the Expanding Universe*). To determine that such questions are not similar, SPTK would need different embeddings for the word *model* in the two contexts, i.e., those related to *person* and *science*, respectively.

In this section, we use distributed representations generated by neural approaches for computing the lexical similarity in TKs. We carry out an extensive comparison between different methods, i.e., word2vec, using CBOW and SkipGram, and Glove, in terms of their impact on convolution semantic TKs for question classification (QC). We experiment with composing word vectors and alternative embedding methods for bigger unit of text to obtain context specific vectors.

Unfortunately, the study above shows that standard ways to model the context are not effective. Thus, we propose a novel application of siamese networks to learn word vectors in context, i.e., a representation of a word conditioned on the other words in the sentence [Nicosia and Moschitti, 2017b]. Since a comprehensive and large enough corpus of disambiguated senses is not available, we approximate them with categorical information: we derive a classification task that consists in deciding if two words extracted from two sentences belong to the same sentence category. We use the obtained contextual word representations in TKs.

Our approach tested on two tasks, question and sentiment classification, shows that modeling the context further improves the semantic kernel accuracy compared to only using standard word embeddings.

5.2.1 Related Work

Distributed word representations are an effective and compact way to represent text and are widely used in neural network models for NLP. The research community has also studied them in the context of many other machine learning models, where they are typically used as features.

SPTK is an interesting kernel algorithm that can compute word to word similarity with embeddings [Croce et al., 2011; Filice et al., 2015, 2016]. In our work, we go beyond simple word similarity and improve the modeling power of SPTK using contextual information in word representations. Our approach mixes the syntactic and semantic features automatically extracted by the TK, with representations learned with DLMs.

Early attempts to incorporate syntactic information in DLMs use grammatical relations to guide the composition of word embeddings, and recursively compose the resulting substructural embeddings with parametrized functions. In Socher et al. [2012] and Socher et al. [2013], a parse tree is used to guide the composition of word embeddings, focusing on a single parametrized function for composing all words according to different grammatical relations. In Tai et al. [2015], several LSTM architectures that follow an order

Model	**Features**	**Accuracy**
SVM	Unigram, syntactic information, parser output, WordNet features, hand-coded features	95.0
DCNN	Unsupervised vectors	93.0
CNN$_{ns}$	CBOW fine-tuned vectors	93.6
DepCNN	Depencency guided filters	95.6
SPTK	SPTK and LSA word vectors	94.8

Table 5.2: QC accuracy (%) of SVM [Silva et al., 2010], DCNN [Kalchbrenner et al., 2014], CNN$_{ns}$ [Kim, 2014], DepCNN, [Ma et al., 2015] and SPTK [Croce et al., 2011] models.

determined by syntax are presented. Considering embeddings only, Levy and Goldberg [2014] proposed to learn word representations that incorporate syntax from dependency-based contexts. In contrast, we inject syntactic information by means of TKs, which establish a hard match between tree fragments, while the soft match is enabled by the similarities of distributed representations.

DLMs have been applied to the QC task. Convolutional neural neworks are explored in Kalchbrenner et al. [2014] and Kim [2014]. In Ma et al. [2015], convolutions are guided by dependencies linking question words, but it is not clear how the word vectors are initialized. In our case, we only use pre-trained word vectors and the output of a parser, avoiding intensive manual feature engineering, as in Silva et al. [2010]. The accuracy of these models are reported in Table 5.2, and can be compared to our QC results (Table 5.5) on the commonly used test set. In addition, we report our results in a cross-validation setting to better assess the generalization capabilities of the models.

To encode words in context, we employ a siamese network, a DLM that has been widely used to model sentence similarity. In a siamese setting, the same network is used to encode two sentences, and during learning, the distance between the representations of similar sentences is minimized. Therefore, the similarity is typically computed between pair of sentences. In our work, we compute the similarity of word representations extracted from the states of a recurrent network. Such representations still depend on the entire sentence, and thus encode contextual information.

5.2.2 Tree Kernels-based Lexical Similarity

TKs are powerful methods for computing the similarity between tree structures. They can effectively encode lexical, syntactic and semantic information in learning algorithms. For this purpose, they count the number of substructures shared by two trees. In most TKs, two tree fragments match if they are identical. In contrast, Croce et al. [2011] proposed the

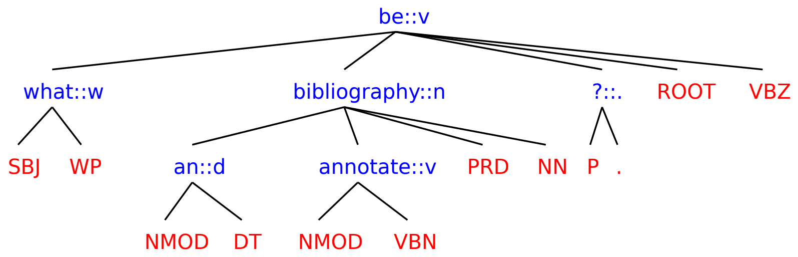

Figure 5.2: The Lexical Centered Tree (LCT) of the lemmatized sentence: "*What is an annotated bibliography?*".

Smoothed Partial Tree Kernel (SPTK), which can also match fragments differing in node labels. For example, consider two constituency tree fragments which differ only for one lexical node. SPTK can establish a soft match between the two fragments by associating the lexicals with vectors and by computing the cosine similarity between the latter. In previous work for QC, vectors were obtained by applying Latent Semantic Analysis (LSA) to a large corpus of textual documents. We use neural word embeddings as in Filice et al. [2015] to encode words. Differently from them, we explore specific embeddings by also deriving a vector representation for the context around each word. Finally, we define a new approach based on the category of the sentence of the target word.

Structural Representation for Text

Syntactic and semantic structures can play an important role in building effective representations for machine learning algorithms. In Croce et al. [2011], a wide array of representations derived from the parse tree of a sentence are evaluated. The Lexical Centered Tree (LCT) is shown to be the best performing tree layout for the QC task. An LCT, as shown in Figure 5.2, contains lexicals at the pre-terminal levels, and their grammatical functions and POS tags are added as left-most children. In addition, each lexical node is encoded as a word lemma, and has a suffix which is composed by a special :: symbol and the first letter of the POS tag of the word. These marked lexical nodes are then mapped to their corresponding numerical vectors, which are used in the kernel computation. Only lemmas sharing the same POS tag are compared in the semantic kernel similarity.

5.2.3 Context Word Embeddings for SPTK

We propose to compute the similarity function σ in SPTK as the cosine similarity of word embeddings obtained with neural networks. We experimented with the popular Continuous Bag-Of-Words (CBOW) and SkipGram models [Mikolov et al., 2013a], and GloVe [Pennington et al., 2014].

POS Tags in Word Embeddings

As in [Croce et al., 2011], we observed that embeddings learned from raw words are not the most effective in the TK computation. Thus, similarly to Trask et al. [2015], we attach a special :: suffix plus the first letter of POS to the word lemmas. This way, we differentiate words by their tags, and learn specific embedding vectors for each of them. This approach increases the performance of our models.

Modeling the Word Context

Although a word vector encodes some information about word co-occurrences, the context around a word, as also suggested in Iacobacci et al. [2016], can explicitly contribute to the word similarity, especially when the target words are infrequent. For this reason, we also represent each word as the concatenation of its embedding with a second vector, which is supposed to model the context around the word. We build this vector as (i) a simple average of the embeddings of the other words in the sentence, and (ii) with a method specifically designed to embed longer units of text, namely paragraph2vec [Le and Mikolov, 2014]. This is similar to word2vec: a network is trained to predict a word given its context, but it can access an additional vector specific for the paragraph, where the word and the context are sampled.

5.2.4 Recurrent Networks for Encoding Text

As described in Section 5.2.1, a siamese network encodes two inputs into a vectorial representation, reusing the sentence encoder parameters. In this section, we briefly describe the architecture used for our encoder.

5.2.5 Contextual Word Similarity Network

The methods to model the context described in Section 5.2.3 adds to the target word vector some dimensions which encodes the entire sentence. This provides some context that may increase the discriminative power of SPTK. The latter can then take advantage of a similarity which is computed between two words, but also depends on the sentences where such words come from. For example, when SPTK carries out a QC task, the sentences above have higher probability to share similar context if they belong to the same category. Still, this approach is rather shallow, as two words from the same sentence would be associated with almost the same context vector. That is, the approach does not really transform the embedding of a given word according to its context.

An alternative approach is to train the context embedding using neural networks on a sense annotated corpus, which can remap the word embeddings in a supervised fashion. However, since there are not enough large disambiguated corpora, we need to approximate the word senses with coarse-grained information, e.g., the category of the context. In other words, we can train a network to decide if two target words are sampled from sentences belonging to the same category. This way, the states of the trained network corresponding to each word, can be eventually used as word-in-context embeddings.

In the next sections, we present the classification task designed for this purpose, and then the architecture of the siamese network for learning contextual word embeddings.

Defining the Derived Classification Task

The end task that we consider is the categorization of a sentence $s \in D = \{s_1, ..., s_n\}$ into one class $c_i \in C = \{c_1, ..., c_m\}$, where D is our collection of n sentences, and C is the set of m sentence categories. Intuitively, we define the derived task as determining if two words extracted from two different sentences share the same sentence category or not. Our classifier learns word representations while accessing to the entire sentence. More formally, we sample a pair of labeled sentences $\langle s_i, c_i \rangle, \langle s_j, c_j \rangle$ from our training set, where $i \neq j$. Then, we sample a word from each sentence, $w_a \in s_i$ and $w_b \in s_j$, and we assign a label $y \in \{0, 1\}$ to the word pair. We set $y = 0$ if $c_i \neq c_j$, and $y = 1$ if $c_i = c_j$. Our goal is to learn a mapping f such that:

$$sim(f(s_i, w_a), f(s_j, w_b)) \in [0, 1], \tag{5.1}$$

where sim is a similarity function between two vectors that should output values close to 1 when $y = 1$, and values close to 0 when $y = 0$.

Data Construction for the Derived Task

To generate sentence pairs, we randomly sample sentences from different categories. Pairs labeled as positive are constructed by randomly sampling sentences from the same category, without replacement. Pairs labeled as negative are constructed by randomly sampling the first sentence from one category, and the second sentence from the remaining categories, again without replacement.

Bidirectional GRUs for Word Similarity

We model the function f that maps a sentence and one of its words into a fixed size representation as a neural network. We aim at using the f encoder to map different word/sentence pairs into the same embedding space. Since the two input sentences play a

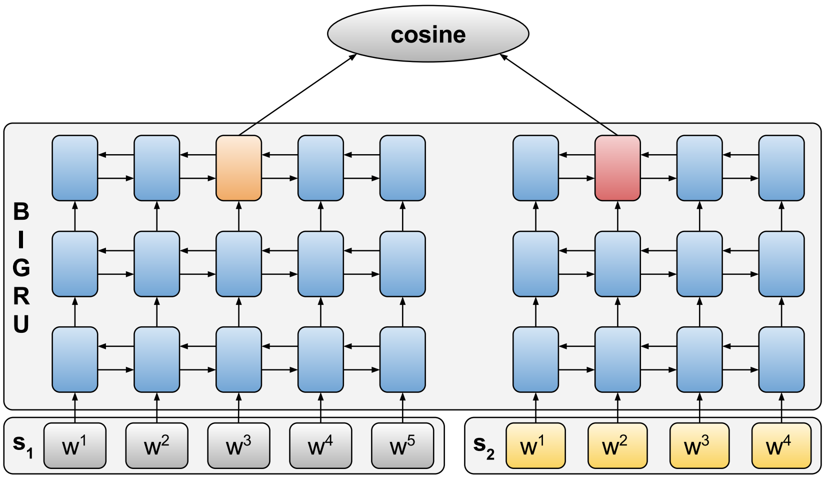

Figure 5.3: The architecture of the siamese network. The network computes $sim(f(s_1, 3), f(s_2, 2))$. The word embeddings of each sentence are consumed by a stack of 3 Bidirectional GRUs. The two branches of the network share the parameter weights.

symmetric role in our desired similarity and we need to use the same weights for encoding both of them, we opt for a siamese architecture [Chopra et al., 2005].

The optimization strategy is what differentiates our siamese network from others that compute textual similarity. We do not compute the similarity (and thus the loss) between two sentences. Instead, we compute the similarity between the contextual representations of two random words from the two sentences.

This is clearly depicted in Figure 5.3. The input words are mapped to integer ids, which are looked up in an embedding matrix to retrieve the corresponding embedding vectors. The sequence of vectors is then consumed by a 3-layer Bidirectional GRU (BiGRU). We selected a BiGRU for our experiments as they are more effective and accurate than LSTMs for our tasks. We tried other architectures, including convolutional networks, but RNNs gave us better results with less tuning effort. Note that the weights of the RNNs are shared between the two branches.

Each RNN layer produces a state for each word, which is consumed by the next RNN in the stack. From the top layer, the state corresponding to the word in the similarity pair is selected. This state encodes the word given its sentential context. Thus, the first layer, $BiGRU'$, maps the sequence of input vectors $(x_1, ..., x_T)$, into a sequence of states $(s'_1, ..., s'_T)$, the second, $BiGRU''$, transforms those states into $(s''_1, ..., s''_T)$, and the third, $BiGRU'''$, produces the final representations of the words in context $(s'''_1, ..., s'''_T)$.

Eventually, the network computes the similarity of a pair of encoded words, selected from the two sentences. We optimize the cosine similarity to match the similarity function used in SPTK. We rescale the output similarity in the $[0, 1]$ range and train the network to minimize the log loss between predictions and true labels.

5.2.6 Experimental Settings

We compare SPTK models with our tree kernel model using neural word embeddings (NSPTK) on question classification (QC), a central task for question answering, and on sentiment classfication (SC).

Data. The QC dataset [Li and Roth, 2006] contains a set of questions labelled according to a two-layered taxonomy, which describes their expected answer type. The coarse layer maps each question into one of 6 classes: Abbreviation, Description, Entity, Human, Location and Number. Our experimental setting mirrors the setting of the original study: we train on 5,452 questions and test on 500.

The SC dataset is the one of SemEval Twitter'13 for message-level polarity classification [Nakov et al., 2013]. The dataset is organized in a training, development and test sets containing 9,728, 1,654 and 3,813 tweets respectively. Each tweet is labeled as positive, neutral or negative. The only preprocessing step we perform on tweets is to replace user mentions and url with a <USER> and <URL> token, respectively.

In the cross-validation experiments, we use the training data to produce the training and test folds, whereas we use the original test set as our validation set for tuning the parameters of the network.

Word embeddings. Learning high quality word embeddings requires large textual corpora. We train all the vectors for QC on the ukWaC corpus [Ferraresi et al., 2008], also used in Croce et al. [2011] to obtain LSA vectors. The corpus includes an annotation layer produced with TreeTagger [6]. We process the documents by attaching the POS tag marker to each lemma. We trained paragraph2vec vectors using the Gensim[7] toolkit. Word embeddings for the SC task are learned on a corpus of 50M English tweets collected from the Twitter API over two months, using word2vec and setting the dimension to 100.

Neural model. We use GloVe word embeddings (300 dimensions), and we fix them during training. Embeddings for words that are not present in the embedding model are randomly initalized by sampling a vector of the same dimension from the uniform distribution $U[-0.25, 0.25]$.

The size of the forward and backward states of the BiGRUs is set to 100, so the resulting concatenated state has 200 dimensions. The number of stacked bidirectional

[6] http://www.cis.uni-muenchen.de/ schmid/tools/TreeTagger/

[7] https://radimrehurek.com/gensim/

networks is three and it was tuned on a development set. This allows the network to have high capacity, fit the data, and have the best generalization ability. The final layer learns higher order representations of the words in context. We did not use dropout as a regularization mechanism since it did not show a significant difference on the performance of the network. The network parameters are trained using the Adam optimizer [Kingma and Ba, 2014], with a learning rate of 0.001.

The training examples are fed to the network in mini-batches. The latter are balanced between positive and negative examples by picking 32 pairs of sentences sharing the same category, and 32 pairs of sentences from different categories. Batches of 64 sentences are fed to the network. The number of words sampled from each sentence is fixed to 4, and for this reason the final loss is computed over 256 pairs of words in context, for each mini-batch. The network is then trained for 5 epochs, storing the parameters corresponding to the best registered accuracy on the validation set. Those weights are later loaded and used to encode the words in a sentence by taking their corresponding output states from the last BiGRU unit.

Structural models. We trained the tree kernel models using SVM-Light-TK [Moschitti, 2004], an SVM-Light extension [Joachims, 1999] with tree kernel support. We modified the software to lookup specific vectors for each word in a sentence. We preprocessed each sentence with the LTH parser[8] and used its output to construct the LCT. We used the parameters for the QC classifiers from Croce et al. [2011], while we selected them on the Twitter'13 dev. set for the SC task.

5.2.7 Context Embedding Results

Table 5.3 shows the QC accuracy of NSPTK with CBOW, SkipGram and GloVe. The results are reported for vector dimensions (*dim*) ranging from 50 to 1000, with a fixed window size of 5.

The performance for the CBOW hierarchical softmax (*hs*) and negative sampling (*ns*), and for the SkipGram *hs* settings are similar. For the SkipGram *ns* settings, the accuracy is slightly lower for smaller dimension sizes. GloVe embeddings yield a lower accuracy, which steadily increases with the size of the embeddings. In general, a higher dimension size produces higher accuracy, but also makes the training more expensive. 500 dimensions seem a good trade-off between performance and computational cost.

To better validate the performance of NSPTK, and since the usual test set may have reached a saturation point, we cross-validate some models. We use the training set to perform a 5-fold stratified cross-validation (CV), such that the distribution of labels in

[8] http://nlp.cs.lth.se

	CBOW		SkipGram		GloVe
	hs	*ns*	*hs*	*ns*	-
dim					
50	89.8	89.8	91.0	91.6	89.8
100	93.0	93.6	94.2	92.8	91.6
150	94.2	94.0	94.2	93.8	92.4
200	94.6	93.6	93.2	94.2	93.2
250	94.4	94.4	94.2	94.2	93.6
300	94.2	94.0	94.4	94.0	93.8
500	95.2	95.0	94.8	93.8	94.4
750	94.8	94.6	95.0	94.4	94.2
1000	93.4	95.2	95.2	94.6	94.0

Table 5.3: QC test set accuracies (%) of NSPTK, given embeddings with window size equal to 5, and dimensionality ranging from 50 to 1,000.

	CBOW	SkipGram	GloVe	LSA
	hs	*hs*	-	-
dim				
100	84.47	84.63	82.92	-
250	85.75	85.85	85.04	85.39
500	86.48	86.32	85.73	-

Table 5.4: QC cross-validation accuracies (%) of NSPTK with the embeddings of specified dimensions.

each fold is similar. Table 5.4 shows the cross-validated results for a subset of word embedding models. Neural embeddings seem to give a slightly higher accuracy than LSA. A more substantial performance edge may come from modeling the context, thus we experimented with word embeddings concatenated to context embeddings.

Table 5.5 shows the results of NSPTK using different word encodings. The *word* and *context* columns refer to the model used for encoding the word and the context, respectively. These models are word2vec (*w2v*) and paragraph2vec (*p2v*). The word2vec vector for the context is produced by averaging the embedding vectors of the other words in the sentence, i.e., excluding the target word. The paragraph2vec model has its own procedure to embed the words in the context. CV results marked with † are significant with a p-value < 0.005. The cross-validation results reveal that word2vec embeddings without context are a tough baseline to beat, suggesting that standard ways to model the context are not effective.

word	context	QC test accuracy	QC CV accuracy
w2v	-	95.2	86.48
w2v	w2v	95.4	86.08†
w2v	p2v	95.0	86.46
p2v	-	92.8	82.65†
p2v	p2v	93.6	83.47†

Table 5.5: QC accuracies for the word embeddings (CBOW vectors with 500 dimensions, trained using hierarchical softmax) and paragraph2vec.

word	context	QC CV accuracy	Std. dev
w2v	-	86.48	.005
BiGRUs	-	84.61†	.027
w2v	BiGRUs	**88.32**†	.009

Table 5.6: QC accuracies for NSPTK, using the word-in-context vector produced by the stacked BiGRU encoder trained with the Siamese Network. Word vectors are trained with CBOW (hs) and have 500 dimensions.

5.2.8 Results of our Bidirectional GRU for Word Similarity

Table 5.6 shows the results of encoding the words in context using a more sophisticated approach: mapping the word to a representation learned with the Siamese Network that we optimize on the derived classification task presented in Section 5.2.5. The NSPTK operating on word vectors (best vectors from Table 5.4) concatenated with the word-in-context vectors produced by the stacked BiGRU encoder, registers a significant improvement over word vectors alone. In this case, the results marked with † are significant with a p-value $<$ 0.002. This indicates that the strong similarity contribution coming from word vectors is successfully affected by the word-in-context vectors from the network. The original similarities are thus modulated to be more effective for the final classification task. Another possible advantage of the model is that unknown words, which do not participate in the context average of simpler model, have a potentially more useful representation in the internal states of the network.

5.2.9 Sentiment Classification Results

Table 5.7 reports the results on the SC task. This experiment shows that incorporating the context in the similarity computation slightly improves the performance of the NSPTK. The real improvement, 12.31 absolute percent points over using word vectors alone, comes

word	context	SC F_1^{PN}
w2v	-	48.65
w2v	w2v	51.59
w2v	BiGRUs	**60.96**

SemEval system	SC F_1^{PN}
Castellucci et al. [2013]	58.27
Dong et al. [2015]	72.8

Table 5.7: SC results for NSPTK with word embeddings and the word-in-context embeddings. Runs of selected systems are also reported.

from modeling the words in context with the BiGRU encoder, confirming it as an effective strategy to improve the modeling capabilities of NSPTK.

Interestingly, our model with a single kernel function and without complex text normalization techniques outperforms a multikernel system [Castellucci et al., 2013], when the word-in-context embeddings are incorporated. The multikernel system is applied on preprocessed text and includes a BOW Kernel, a Lexical Semantic Kernel, and an SPTK. State-of-the-art systems [Dong et al., 2015; Severyn and Moschitti, 2015b] include many lexical and clustering features, sentiment lexicons, and distant supervision techniques. Our approach does not include any of the former.

5.2.10 Wins of the BiGRU model

An error analysis on the QC task reveals the *What* questions as the most ambiguous. Table 5.8 contains some of the successes of the BiGRU model with respect to the model using only word vectors. Those wins can be explained by the effect of the contextual word vectors on the kernel similarity. In Question 1, the meaning of *occupation* is affected by the presence of a person name. In Question 2, the word *level* loses its prevalent association with quantities. In questions 3 to 5, the underlined words are a strong indicator of locations/places, and the kernel similarity may be dominated by their corresponding word vectors. BiGRU vectors are instead able to effectively remodulate the kernel similarity and induce a correct classification.

5.2.11 Summary

In this section, we applied neural network models for learning representations with semantic convolution tree kernels. We evaluated the main distributional representation methods

Question	Wrong w2v	Correct BiGRU
1) What is the occupation of Nicholas Cage ?	enty	hum
2) What level of government (...) is responsible for dealing with racism?	num	hum
3) What is the Motto for the State of Maryland?	loc	desc
4) What is a virtual IP address?	loc	desc
5) What function does a community's water tower serve?	loc	desc

Table 5.8: Sample of sentences where NSPTK with word vectors fails, and the BiGRU model produces correct classifications.

for computing semantic similarity inside the kernel. In addition, we augmented the vectorial representations of words with information coming from the sentential content. Word vectors alone revealed to be difficult to improve upon. To better model the context, we proposed word-in-context representations extracted from the states of a recurrent neural network. Such network learns to decide if two words are sampled from sentences which share the same category label. The resulting embeddings are able to improve on the selected tasks when used in conjunction with the original word embeddings, by injecting more contextual information for the modulation of the kernel similarity. We show that our approach can improve the accuracy of the convolution semantic tree kernel.

5.3 Conclusion

In this chapter, we have explored some possible intersection points between neural network and tree kernel models. One of them sees the activations of the network used in the kernel-based classifier together with structural representations. The other consists in incorporating a contextual neural similarity between words, directly in the structural kernel. This similarity is learned with a siamese network, in such a way that information from the entire sentence is encoded into a word representation, and the latter is optimized for the end classification task.

In the next chapter, we describe our structural relational framework for QA based on tree kernels, and then present its new and fully neural counterpart.

Chapter 6

Semantic Linking in Convolutional Models

In this chapter, we propose passage reranking models that (i) do not require manual feature engineering, and (ii) effectively use the output of syntactic and semantic annotators to enhance the question and passage representations. The latter encode relations between questions and answer passages using information from automatic classifiers, i.e., question category and focus classifiers, and named entity recognizers. This way, effective relational patterns can be automatically learned with kernel machines and neural networks.

6.1 Question Analysis

Robust and production-level QA systems typically run several NLP components to extract syntactic and semantic annotations from text. The famous IBM Watson [Ferrucci et al., 2010] that beated top human *Jeopardy!* players employs more than one hundred techniques to analyze the text, generate and score potential answers.

Question analysis is the starting point of the overall process [Lally et al., 2012]. A careful analysis of the question can produce important syntactic and semantic clues that greatly help in scoring candidate answer passages, and in identifying the final answer. Useful annotations could be the question category, the lexical answer type (LAT), the question focus, and relations, in the form of syntactic subject-verb-object predicates or semantic links between entities. Leveraging a relatively small number of annotated examples, we can automatically train classifiers and extract question properties that may be exploited by a QA model to increase the accuracy of its answers.

6.1.1 Question Classification

Questions can be broadly classified into categories according to a given taxonomy. The taxonomy may include categories that signal the broad type of the answer, or answer complexity, e.g., a specific entity or a more descriptive answer, procedure or definition. When the category is indicative of the answer type, the latter can be furtherly characterized by the LAT, which according to Lally et al. [2012] is a word or noun phrase in the question that specifies the type of the answer without any attempt to understand its semantics.

The question category together with the LAT are very useful to establish a semantic link between the question and text passages that contain entities of compatible types. For example, if a question asks for the name of a president of a nation, a passage with named entities of type *Person* is more likely to contain a correct answer with respect to a passage that has a significant lexical overlap with the question, but contains only entities of type *Location*.

6.1.2 Question Focus Identification

In the literature there are multiple definitions of question focus. According to Ferrucci et al. [2010], the focus is the question part that substituted with the answer, renders the question a stand-alone statement. Bunescu and Huang [2010] says that the focus is the "set of all maximal noun phrases in the question that corefer with the answer". Their definition permits multiple focuses and an implicit focus. Additionally, this definition is more tied to the LAT and the focus can therefore be used to infer the answer type. We adopt their focus definition since we build our question focus identifier using the annotated data they provide. Note that we do not consider the case of multi-word or implicit focus.

6.2 Semantic Linking in Convolution Tree Kernels

As we have seen, a critical issue for implementing QA systems is the need to design answer search and extraction modules that capture the salient characteristics of pairs of text. These modules encode handcrafted rules based on syntactic patterns that detect the relations between a question and its candidate answers. Such rules are triggered when patterns in the question and the passage are found.

To reduce the burden of manual feature engineering for QA, we propose structural models based on kernel methods [Severyn et al., 2013a,b]. These systems are based on similar ideas from the frameworks in Chapter 4. The main adaptation for factoid QA is that we perform question analysis, and design a shallow syntactic representation of the text which contains relational tags.

In this section, we describe our structural relational model for QA, and then we show how to enrich the semantic representation of question/answer (q/a) pairs with the information provided by NLP annotators, i.e., question category (QC) and focus classifiers (FC) and Named Entity Recognizers (NERs). FC determines the constituent of the question to be linked to the named entities (NEs) of the answer passage. The target NEs are selected based on their compatibility with the category of the question, e.g., an NE of type PERSON is compatible with a category of a question asking for a human (HUM). This model greatly improves on IR baseline, e.g., BM25, by 40%, and on previous reranking models, up to 10%. We refer the interested reader to Severyn et al. [2013a] for details about all the experiments.

In this thesis, we only report the experiments on the TrecQA dataset from Severyn et al. [2013b], since we will evaluate our neural QA model from section 6.4 in the same answer sentence selection setting.

6.2.1 Learning to Rank with Kernels

Our reranking framework is similar to the one described in Chapter 4: given a query question, a search engine retrieves a list of candidate passages ranked by their relevancy. In alternative, a list of candidate passages can already be provided in the QA dataset. Then, various NLP components in a UIMA pipeline[1] analyze each question together with its candidate answers, e.g., POS tagging, chunking, named entity recognition, constituency and dependency parsing, etc. These annotations are used to produce structural models, which are enriched with the output of a question focus detector and question type classifiers, such as to establish relational links for a given q/a pair. The resulting tree pairs are used to train a kernel-based reranker, that refines the initial ordering of the answer passages.

We use tree structures as our base encoding since they provide sufficient flexibility in representation, and an easier feature extraction process than, for example, graph structures. We rely on the Partial Tree Kernel (PTK) [Moschitti, 2006] to handle feature engineering over the structural representations.

6.2.2 Relational Structural Models of Q/A Pairs

Our starting point for representing the text in a q/a pair is the shallow syntactic tree. In a shallow syntactic representation, first explored for QA in Severyn and Moschitti [2012], a question and its candidate answer are encoded into a tree where POS tags are located at the pre-terminal level, and word lemmas are at the leaf level. In the same model, a special **REL** tag is used to encode relationships between the question and the answer.

[1] http://uima.apache.org/

Dataset	STK	STK$_{bow}$	PTK
Mooney	81.9	81.5	80.5
SeCo-600	94.5	94.5	90.0
Bunescu	98.3	98.2	96.9

Table 6.1: Accuracy (%) of focus classifiers.

The adopted strategy is simple: lemmas shared between the question and the answer get their parents (POS tags) and grandparents (chunk labels) marked with a **REL** tag.

The idea of marking related fragments in the question and answer tree representations has been shown to yield more accurate relational models. However, such approach relies on a basic hard matching between word lemmas. Below, we propose a novel strategy to establish relational links using named entities extracted from the answer along with question focus and category classifiers. In particular, we use a question category to link the focus word of a question with the named entities extracted from the candidate answer.

Question Focus Classifier

For the identification of the focus, we train an SVM classifier with tree kernels applied to the constituency tree representation of the questions. To generate examples for training the reranker, we produce a number of trees where the parent (node with the POS tag) of each candidate word is annotated with a special `FOCUS` tag. Trees with the correctly tagged focus word constitute positive examples, while the others are negative examples. To detect the focus for a new question, we classify all possible trees obtained by tagging each single question word as a focus. The tree which receives the highest classification score reveals us the predicted focus word.

Our focus identifier is evaluated on three datasets: SeCo-600 [Quarteroni et al., 2012], Mooney GeoQuery [Damljanovic et al., 2010], and the dataset from Bunescu and Huang [2010]. The SeCo dataset contains 600 questions. We removed from that set multi-focus questions and non-interrogative queries. Mooney GeoQuery contains 250 questions mainly asking for U.S. geographical information. The first two datasets are very domain specific, so we also evaluate our classifier on the dataset from Bunescu and Huang [2010], which contains the first 2000 questions from the answer type dataset from Li and Roth [2006]. We discard questions with implicit and multiple focuses.

Table 6.1 displays the accuracies obtained by our question focus detector on the three datasets using different kernels: STK, STK$_{bow}$ (bag-of-words feature vector is added) and PTK. As we can see, using STK$_{bow}$ model yields the best accuracy, and we use it in our

Dataset	STK$_{bow}$	PTK
UIUIC [Li and Roth, 2006]	86.1	82.2
TREC 11-12	79.3	78.1

Table 6.2: Accuracy (%) of question classifiers.

QA pipeline to automatically detect the question focus.

Question Category Classifier

Previous work in question classification demonstrates the power of syntactic/semantic tree representations coupled with tree kernels to train state-of-the-art models [Bloehdorn and Moschitti, 2007]. Hence, we opt for an SVM multi-class classifier that uses tree kernels to extract the question class. We train the former according to a one-vs-all strategy.

Table 6.2 contains the accuracies of the question classifier on the UIUIC dataset Li and Roth [2006] and the TREC 11-12[2] questions that we also use for testing the reranker. We select the STK$_{bow}$ and PTK kernels.

Linking Focus Word with Named Entities using Question Class

We exploit the question category (automatically identified by a question type classifier) along with named entities found in the answer to establish relational links between the tree structures of a given q/a pair. In particular, once the question focus and question category are determined, we link the focus word w_{focus} in the question, with all the named entities whose type matches the question class. Table 6.3 shows the mapping between question classes and named entity types. We perform tagging at the chunk level, and use two types of relational tags: the plain **REL-FOCUS**, and a tag which is furtherly typed with a question class, e.g., **REL-FOCUS-HUM**. Figure 6.1 shows an example q/a pair where the typed relational tag is used in the shallow syntactic tree representation to link the chunk containing the question focus *name* to the named entities of the corresponding type *Person* (according to the mapping defined in Table 6.3), i.e., *samuel langhorne clemens* is linked to *mark twain*.

6.2.3 Feature Vector Representation

While the primary focus of our study is on the structural representations and relations between question and answer pairs, we also include basic features widely used in QA:

[2] We manually annotate 824 questions using the categories of UIUIC.

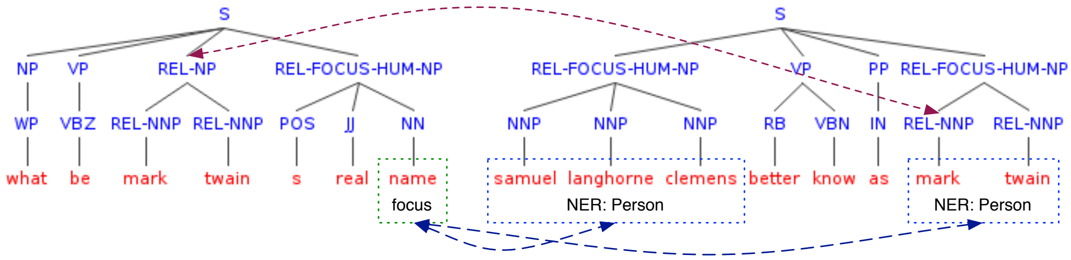

Figure 6.1: Shallow tree structure CH with a typed relation tag **REL-FOCUS-HUM** to link a question focus word *name* with the named entities of type *Person* corresponding to the question category (HUM).

Question Category	Named Entity types
HUM	Person
LOC	Location
NUM	Date, Time, Money, Percentage
ENTY	Organization, Person

Table 6.3: Mapping between question classes and named entity types.

Term-overlap features. Cosine similarity between question and answer: $sim_{COS}(q, a)$, where the input vectors are composed of n-grams (up to tri-grams) of word lemmas and POS tags, and dependency triplets. For the latter, we simply hash the string value of the predicate defining the triple together with its argument, e.g. *poss(name, twain)*.

PTK score. For the structural representations we also define a similarity based on the PTK score: $sim_{PTK}(q, a) = PTK(q, a)$, where the input trees can be both dependency trees and shallow chunk trees. Note that this similarity is computed between the members of a q/a pair.

NER relatedness represents a match between a question category and the related named entity types extracted from the candidate answer. It counts the proportion of named entities in the answer that correspond to the question type returned by the question classifier.

We also extend this feature vector to obtain an advanced feature vector V_{adv}, which contains 43 features in total. The latter include all the features already described in Section 4.4.6, and the similarity scores from a translation model (METEOR).

6.3 Comparison with State-Of-The-Art Systems

In this section, we compare our structural models with other systems that had state-of-the-art results at that time. We first briefly describe the experimental setup to replicate the settings of the previous work, and then we present our results.

6.3.1 Setup

We test our models on the manually curated TrecQA dataset[3] from Wang et al. [2007], which is a standard benchmark for answer sentence selection in the literature.

In Wang et al. [2007] setup, 100 manually judged questions from TREC 8-12 are used for training, while questions from TREC 13 are used for testing. Additionally, they provide a "noisy setting" experiment where 2,393 questions from the entire TREC 8-12 collection are used for training.

While we use the same test collection, our training data consists of only 824 questions from TREC 2002, 2003 (TREC 11-12).We extract candidate answer sentences from the previously retrieved paragraphs by keeping the ones that have at least one non-stopword in common with the question. We build examples for our reranker by pairing each positive candidate answer sentence with at most top 10 incorrect candidates, which results in a total of 8,730 examples.

6.3.2 Evaluation Measures

For the evaluation, we adopt Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR), common metrics for assessing Information Retrieval and QA systems. MAP is computed by averaging the precision over a set of questions Q as follows:

$$\frac{\sum_{q=1}^{Q} AveP(q)}{Q}$$

MRR is the average of the reciprocal ranks of a list of results for a set of questions Q:

$$\frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{rank_i},$$

where $rank_i$ is the position of the first correct result. When a result list contains more than one correct answer, MAP rewards systems which consistently group them as top results. MRR is equivalent to MAP in case there is a single correct answer in a list. We compute MAP and MRR using the official `trec_eval` scorer.

[3] `http://cs.stanford.edu/people/mengqiu/data/qg-emnlp07-data.tgz`

In few occations, people report also Precision at 1 (denoted as P@1). Such metric computes how many times the sentence at the top of a ranked list is relevant, out of the total number of lists. We can define it as:

$$\frac{1}{|Q|}\sum_{i=1}^{|Q|}\delta(rank(S^{+(i)})=1),$$

where $S^{+(i)}$ is the relevant sentence in the ranked list corresponding to question i, *rank* is a function returning the rank of that sentence, and δ is the indicator function.

6.3.3 Results

Table 6.4 shows the performance of the advanced feature vector V_{adv}, and the CH structural model that can also include the F relational linking strategy. We compare our methods with four other rerankers [Wang et al., 2007; Heilman and Smith, 2010; Wang and Manning, 2010; Xuchen Yao and Callison-Burch, 2013]. All of them report their performance on the same TrecQA evaluation setting as provided by Wang et al. [2007].

It can be noted that our combined set of basic and advanced features V_{adv} already is a rather strong baseline for the task. Furthermore, if we combine the feature vector with the structural representations, we obtain state-of-the-art performance, registering an improvement over the previous work by a large margin. Finally, if we also consider the F linking strategy in the CH representation, we further get two points in MAP, and one point in MRR. This proves the usefulness of our supervised components that annotate question focus and question category, together with NE information, to establish the semantic mapping between sentences in a q/a pair.

Our kernel-based LTR approach is conceptually simpler than the systems from previous work, as it relies on the structural kernels, e.g. PTK, to automatically extract salient syntactic patterns relating questions and answers.

6.3.4 Summary

This work shows a promising approach for QA. Its main characteristic is the use of structural kernel to automatically engineer features from structural semantic representations of question and answer passage pairs. The same technology is also used to construct question and focus classifiers, which are key for integrating more advanced relations in the tree structures.

Model	MAP	MRR
Wang et al. [2007]	0.6029	0.6852
Heilman and Smith [2010]	0.6091	0.6917
Wang and Manning [2010]	0.5951	0.6951
Xuchen Yao and Callison-Burch [2013]	0.6307	0.7477
$\mathrm{V_{adv}}$	0.5627	0.6294
$\mathrm{CH+V_{adv}}$	0.6611	0.7419
+F	**0.6829**†	**0.7520**†

Table 6.4: Answer sentence reranking on TrecQA. † indicates that the improvement delivered by adding F linking to $\mathrm{CH+V_{adv}}$ model is significant ($p < 0.05$).

6.4 Semantic Linking in CNNs

A wide array of NLP tasks requires to learn a notion of relevancy of a piece of text with respect to a query. Neural networks model syntactic and semantic similarity between two text fragments rather well. However, current models do not encode high-level semantic relations between question and answer passages.

In this section, we focus on LTR answer passages associated with a given question, by encoding high-level semantic features directly into word representations. In addition to lexical matches, we model a semantic relation between a question and an answer passage, by carrying out question analysis and linking the focus word in the question with relevant entities in the answer. We show that the resulting CNN-based system exploiting question type and focus automatically derived by neural classifiers, along with the entity category extracted with a NER, is fast to train, and more accurate than models using sophisticated attention mechanisms.

6.4.1 Overview

Transforming text pairs into representations which capture the salient parts of the text is at the core of many NLP tasks, and vital for machine learning algorithms. In recent years, traditional representations based on manual feature engineering have been shadowed by the advent of neural network. The latter are able to leverage unsupervised distributed word representations, compose them, and extract discriminative features.

The research community has firstly focused on neural network architectures that model the interaction of two pieces of text, to later shift towards more complex architectures that model word-to-word interactions across sentences, using sophisticated attention mechanisms [Parikh et al., 2016; Wang et al., 2016a; Santos et al., 2016] and compare-aggregate frameworks [He and Lin, 2016; Wang et al., 2017; Wang and Jiang, 2017].

The growth in architectural complexity is tied to longer training times, which can span hours or even half a day[4] on a single graphic processing unit. Meaningful relational and high-level semantic features may take long time to emerge by only leveraging word representations and the training data of the task at hand. This results in slow iteration cycles during the model development. Even more critical can be the case in which the scarcity of training data can prevent to learn meaningful word association. This problem happens in question answering tasks, e.g., answer sentence selection [Wang et al., 2007]. Thus effective word representations are crucial in neural network models to get state-of-the-art performance. It has been shown that, given two pieces of text, incorporating directly into the word representation a feature indicating that such word is contained in both text fragments, boosts the performance in classification/ranking tasks.

In the next part, we try to answer the following research questions:

- In addition to lexical links between words, can we incorporate a more semantic link between the words in a question and a candidate answer passage?

- After discerning the semantic information that relates question and answer in the form of the question category and question focus, and its mapping into named entity types, can we show that such information has an impact on the output quality of our model?

To answer the previous questions, we show that modeling semantic relations across two pieces of text can greatly improve the performance of a simple neural network. This can be done in terms of (i) a cheaper training time by leveraging the efficiency of specific neural network architectures, and (ii) a small annotation effort, because some semantic tasks can be carried out satisfactorily with models trained on small publicly available datasets, while for other tasks off-the-shelf tools exist. More specifically, we tackle the reranking of answer sentences related to a given question. This is a QA task that consists in producing a higher score for the sentences or passages with a correct answer, given a question and a list of candidate sentences [Wang et al., 2007; Yang et al., 2015]. We enhance the input representation of the sentence words of the question and the candidate answers by establishing, in addition to lexical links, semantic links between the question focus words and the entities in the candidate answers. The links are typed using the question category and the named entity types, where all these pieces of information are

[4] `http://dawn.cs.stanford.edu/benchmark/`

automatically derived with subparts of our system[5].

We show that this type of semantic link is more powerful when there is less lexical overlap between the words in the question and the candidate answers, registering improvement of about 1.5% and 2.7% absolute percent points in MAP over previous state-of-the-art systems, on the TrecQA and WikiQA datasets respectively. On YahooQA, a larger community question answering dataset, the lexical link brings an improvement of 23% absolute percent points in P@1, which is further improved by the semantic links.

6.4.2 Related Work

Learning relations between pieces of text is crucial for a number of NLP tasks. Traditional work on QA and ranking of search results makes heavily use of syntactic and semantic annotations to manually build rules and features for classification [Hickl et al., 2007; Ferrucci et al., 2010]. Such approaches tend to have a lot of different components and are difficult to replicate. A different direction consists in encoding text, syntactic and semantic relations in tree structures, and let structural kernels extract meaningful features for the problem at hand [Severyn et al., 2013a]. Recently, deep learning methods have demonstrated great effectiveness in many NLP tasks. Words are mapped into low dimensional representations and are composed to model sentences, with the goal of learning a supervised task. CNNs [Krizhevsky et al., 2012] and bidirectional recurrent networks [Schuster and Paliwal, 1997] are used to map the sentences in a pair into fixed vectors which are concatenated and typically fed to a feed-forward network. Apart from this simple architecture, there is active research on siamese networks, attentive networks and compare-aggregate networks.

Siamese networks [Chopra et al., 2005] map each input sentence into a fixed sized vector using the same neural component, i.e. sharing the weights, and then compute the similarity of such vectors. The advantages of such architecture are parameter saving and the symmetry of the network: the output is the same even if we swap the sentences in an input pair, since the former are mapped into the same space. In Tan et al. [2016], each branch of the siamese network encodes a sentence by using a bidirectional LSTM. The cosine similarity between the two sentence vectors is then computed. The latter are obtained by taking the last states of the bidirectional recurrent network using to encode the sentences, or by applying a pooling operation or a convolution on the intermediate states of the bidirectional network. The intermediate states contains contextual information for each word in the sentence. It is important to note that the models in Tan et al. [2016] which have separate weights for the query and passage encoders, or contain attentive mechanisms between the two input sentences, cannot be properly considered siamese.

[5] A network for named entity recognition is not included, but we will integrate it in the short-term future.

Attentive Networks [Parikh et al., 2016; Bahdanau et al., 2015; Yin et al., 2016] build a sentence representation by also considering the other sentence, weighting the contribution of its parts with the so-called attention mechanism. Such mechanism may compute a soft alignment between the words of two sentences, by outputting a similarity matrix. The scores in the matrix are then aggregated and used to weight the word embeddings of the sentences. Alternatively, the representation of a word in a sentence is computed with a quadratic operation by attending over all the words of the other sentence.

Compare-Aggregate Networks [Wang and Jiang, 2017] apply several decompositions to each sentence in a pair. The resulting vectors are compared or composed with multiple functions, and possibly some attention mechanism. All the intermediate results are then aggregated into a fixed size vector to quantify the final match. Common aggregation techniques include average or max-pooling.

The type of loss is another dimension in which approaches in the literature may vary. While many models are optimized using a loss computed on a single pair of sentences, there are others which employ a triplet ranking loss function [Rao et al., 2016]. This function learns to assign high scores to relevant query/result pairs, while simultaneously penalizing bad pairs with results not relevant for the query. The negative pairs can be selected from the top of the result list, which may be obtained using a retrieval model, or by focusing on the highest scoring negative pairs under the current neural model.

In this work, we take some elements of the traditional QA research, i.e., semantic features, and use them to model relationships between sentence pairs, in the context of a neural network which is less complex than attentive and compare-aggregate counterparts.

6.4.3 LTR Answer Passages with CNNs

The task of LTR a list of instances constituted by (i) a query or question q and (ii) a candidate answer passage a, can be defined as learning a function $f(q, a)$ that outputs a relevancy probability $s \in [0, 1]$. The output value s represents the probability that the candidate answer passage a contains the actual answer to the question q. The list of instances is potentially constructed with a search engine using the question q as search query. In other cases, a question may be posted on a web forum by a person, and answers can be provided by other users.

After producing the individual predictions, the list of instances is sorted in descending order by the score s. This ranking approach is also called pointwise because each prediction is made by observing a single question and answer pair, without accessing the other instances in the same list.

In this work, we perform pointwise LTR using neural networks. The common building blocks of all our neural modules are Convolutional Neural Networks (CNNs), an efficient way of modeling sentences [Kim, 2014; Kalchbrenner et al., 2014]. We use CNNs (i) to classify a question into a specified number of categories, (ii) to perform sequence labeling in order to identify the possible focus word in a question, and (iii) to build the question and answer representations for the answer reranking model.

Sentence Matrix Encoding

Each neural component in our architecture maps an input sentence into a matrix. A sentence s of length n is a sequence of words (w_1, ..., w_n), which are drawn from a vocabulary V. Each word is encoded with an integer id from 1 to $|V|$, and then represented as a vector, $\mathbf{w} \in \mathbb{R}^d$, looked up into an embedding matrix, $\mathbf{E} \in \mathbb{R}^{d \times |V|}$. The matrix $\mathbf{E}$ is obtained by concatenating all the embeddings of the words in V. The id 0 is used for padding and it is mapped to the zero vector. The i^{th} column in $\mathbf{E}$ corresponds to the word with integer id i to facilitate the lookup.

Question Category Network

The question category network takes in input a question and outputs a probability distribution over the possible question categories seen during the training phase.

The question q is transformed into a sentence matrix and passed through a convolution filter that operates only on windows falling inside the sequence. The output resolution can be determined by subtracting the filter size to the maximum length of the padded/truncated sequences and adding one. The output of the filter for each valid window is pooled to get the maximum value across each dimension. Filter of different widths are applied. Their pooled outputs are concatenated and the resulting question representation is fed to a single hidden layer with a ReLU activation function [Nair and Hinton, 2010], and finally to a softmax layer that outputs a probability value for each question category. The network is trained to minimize the negative log-likelihood of the m question classes over n examples:

$$-\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{m} y_{ij}\log(p_{ij}) \tag{6.1}$$

Question Focus Network

The question focus network takes in input a question and outputs a probability distribution over the tokens indicating how probable one of them is the question focus. The question q is transformed into a sentence matrix and passed through a convolution filter that operates on windows centered on each token. The position in a window that are outside of the sentence are padded and mapped to the zero vector. For this reason, the input and output resolutions are the same.

Drawing an analogy with the computer vision literature, we can define the receptive field of the convolution filter as the number of tokens that a filter sees each time. This number is equal to the filter width in case of a single filter operating on word vectors. A sequence tagging task may benefit of a wider receptive field and see more tokens before making a decision. For this reason, we stack a number of convolution filters on top of each other. This strategy increases the receptive field r linearly in the number l of stacked filters with fixed width w, such that $r = l(w-1)+1$. With a filter size of 10 and 4 stacked convolutions, the receptive field can be increased to 37 tokens, which is much higher than the average question length in the datasets studied in our experiments.

Every single output vector from the last convolution filter in the stack is passed through a hidden layer with a ReLU activation, and a final linear layer which produces a single scalar value.

All those values are normalized across each sentence with a softmax, to form a probability distribution over the sentence tokens. The network is trained to minimize the logarithmic loss where N is the total number of tokens in the questions, p_i is the model probability that the token i is the focus, and y_i is the corresponding gold label:

$$-\frac{1}{N}\sum_{i=1}^{N}[y_i \log p_i + (1-y_i)\log(1-p_i)] \tag{6.2}$$

Answer Sentence Reranking Network

In recent years, many neural models for ranking answer sentences have been proposed. We selected the model described in Severyn and Moschitti [2015a, 2016] (S&M from now on), to explore the contribution of semantic information. This model is fairly simple and well studied, fast to train, robust to the choice of hyperparameters, and it has been successfully reproduced in several papers [Rao et al., 2017; Chen et al., 2017; Sequiera et al., 2017].

We briefly describe the S&M model, highlighting and discussing its salient parts, considering also the findings from some recent work that investigated such model.

The S&M model, depicted in Figure 6.2, takes in input a question and a candidate answer passage, and outputs the probability that the passage contains a right answer. Each sentence is converted into a matrix by mapping its words into pretrained vectors. This matrix is processed by a convolutional filter and a max-pooling operation to obtain a fixed size representation of the corresponding sentence. The question and the answer candidate passage are processed by separate convolutional filters, i.e., the weights are

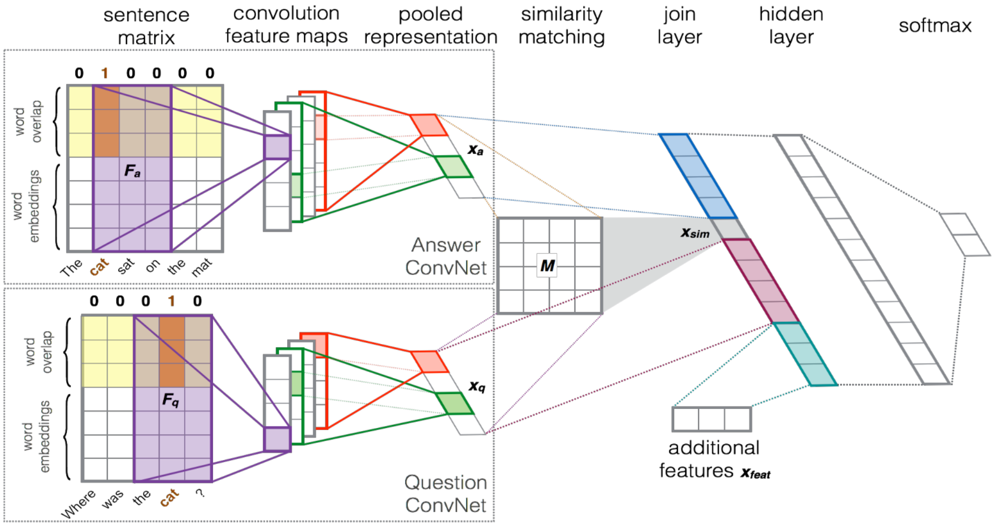

Figure 6.2: The S&M CNN model. Diagram from Severyn and Moschitti [2016].

different. A bilinear transformation, that follows the work of Bordes et al. [2014b], is used to obtain a scalar similarity value x_{sim} for the question and passage pair. The bilinear similarity matrix can be seen as a model of the noisy-channel approach from machine translation, used also in information retrieval and question answering [Echihabi and Marcu, 2003].

The similarity value, the fixed size question and passage representations pooled from the convolution filters, and a vector of additional real valued features x_{feat} are concatenated in the join layer. The latter is fed to a hidden layer with a non-linearity, and the final softmax layer outputs the matching probability.

The most important component of the S&M model, introduced in Severyn and Moschitti [2016], is the word overlap feature. The word vectors of the question and the answer are augmented with an additional binary feature, which is embedded in a small dimensional space. This feature signals if a word is contained in both question and answer or not. The feature embeddings are parameters of the model, and thus are learned during training. The addition of this feature alone produces a substantial increase of the CNN performance, rendering the inclusion of x_{feat} unnecessary. For this reason, we keep the word overlaps in our proposed model.

In Severyn and Moschitti [2015a], the model has access to a set of features which measure the lexical overlap between question and answer words, counting the common words. Those features are the Jaccard similarity and the Inverse Document Frequency Weighted Jaccard Similarity. Feature values are concatenated with the pooled representations of the question and the answer in the join layer. Results from Sequiera et al. [2017] and a

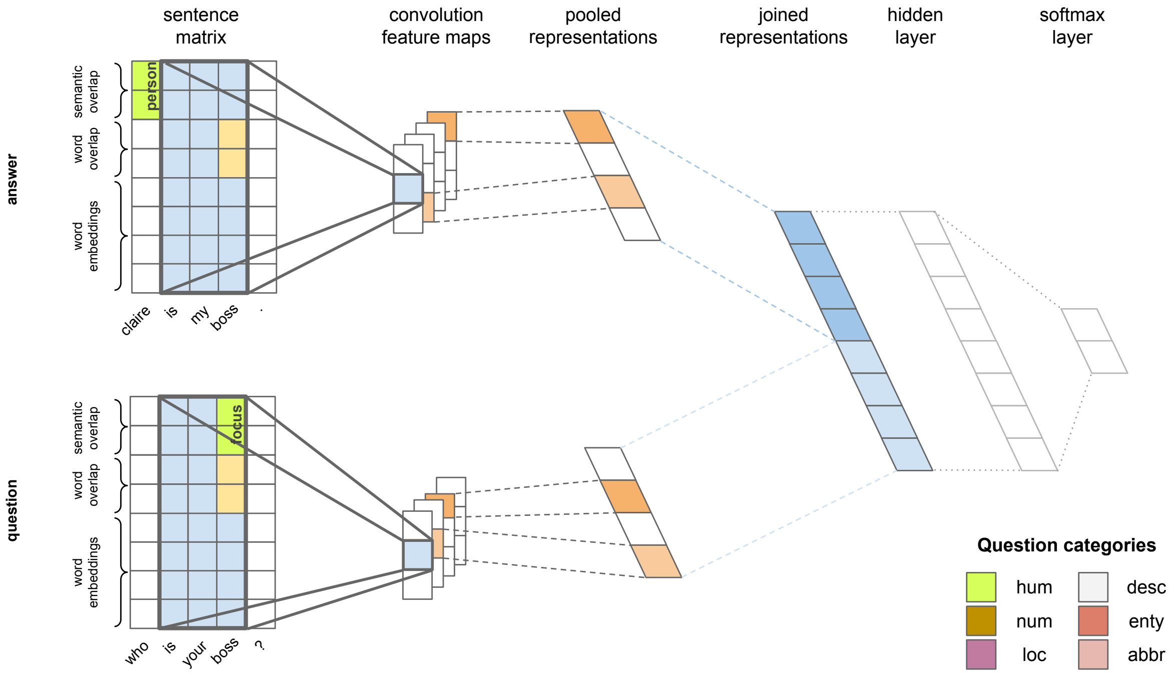

Figure 6.3: Our answer sentence reranking network with word and semantic overlap vectors. The convolution does not have padding, its size is 3 and the number of filters is set to 4. The semantic overlap vectors of the question focus word *boss*, and the answer word *claire* are the same, because the latter is an entity of type *Person*, and the question has been annotated with the HUM category. When we do not consider stopwords, the word *boss* appears in the question and the answer, which is also reflected in the word overlap embedding space.

replication study of the S&M model [Rao et al., 2017], show that the similarity features are beneficial when the lexical overlap between question and answer is high, but they are still less powerful than the embedded word overlap feature. Since the x_{feat} vector does not seem to help and make the training less stable, we do not include it in our final model.

The contribution of the bilinear transformation performed through a learned square matrix has been questioned by recent work [Rao et al., 2017; Sequiera et al., 2017]: it seems that this component does not really help, and according to the ablation tests performed in Rao et al. [2017], it may actually hurt the overall performance of the network. We have been able to replicate the results of the S&M model without the similarity matrix, and therefore we eliminate it from our proposed architecture, in favour of a simpler model and less parameters.

Our Answer Sentence Reranking Network with Semantic Overlap

We propose to add semantic features to the sentence matrix, in order to establish links between words that go beyond lexical matching. Figure 6.3 describes our answer sentence reranking network. The key addition to the S&M model is the semantic overlap vectors. Each word is represented by its word embedding, the word overlap vector, and the semantic overlap vector. The latter is the result of embedding a feature $so \in \{0, ..., C\}$, where C is the number of question classes. Each feature value can be looked up into an embedding matrix $\mathbf{W}_{so} \in \mathbb{R}^{s \times |C|+1}$, where s is the dimensionality of the semantic overlap vector. Analogously, the word overlap binary feature is looked up into an embedding matrix $\mathbf{W}_{wo} \in \mathbb{R}^{r \times |2|}$. The final word representation $\mathbf{w}^{'}$ will be the concatenation of the following vectors: $\mathbf{w}^{'} = [\mathbf{w}; \mathbf{w_{wo}}; \mathbf{w_{so}}]$, resulting in a vector of dimension $d + r + t$. The semantic overlap feature is computed thanks to the output of the CNNs that perform question analysis. The question focus CNN is used to determine which word in the question is the focus, while the question category CNN assigns the question to a class. After this, the so feature is attached to each word according to the following strategy:

- each word in the question is mapped to 0, with the exception of the question focus word, which is mapped to the id associated with the question category (this id is used to look up a vector into the $\mathbf{W}_{so}$ matrix);

- each word in the answer is mapped to 0, with the exeception of words covered by named entities: these words are mapped to the id of the question category which is compatible with their entity type, according to a mapping rule defined in table 6.5.

The $\mathbf{W}_{wo}$ and $\mathbf{W}_{so}$ matrices are parameters of the model, and they are learned during training. The question category and question focus annotations for the pairs from the question answering datasets are produced by our neural network classifiers. The named entities are obtained by processing the questions and answers with an off-the-shelf processor, trained on OntoNotes [Weischedel et al., 2012].

6.4.4 Experimental Settings

In this section, we first describe the details in common to all our neural classifiers and experiments. After that, we describe the training process of each question analysis component, and to conclude, we present the experimental settings for the final candidate answer sentence reranker.

Category	Named Entity Type
HUM	Person
LOC	Loc, Gpe
NUM	Date, Time, Percent, Quantity, Ordinal, Cardinal
ENTY	Norp, Org, Facility, Product, Event, Work of art, Law, Language
DESC	Norp, Org, Facility, Product, Event, Work of art, Law, Language, Date, Time, Percent, Quantity, Ordinal, Cardinal
ABBR	Norp, Org, Facility, Product, Event, Work of art, Law, Language

Table 6.5: Mapping between question categories and OntoNotes entity types.

Shared Settings. Some details of preprocessing, model construction, training, validation and evaluation phases are shared between all the experiments on the different datasets. For this reason, we describe them in the following sections.

Preprocessing. All the textual input to our models is preprocessed using SpaCy[6], an off-the-shelf NLP pipeline for tagging, parsing and named entity recognition. Questions are tokenized and lowercased. Answer passages are also annotated with named entities. The maximum sentence length is set to 60. Longer sentences are truncated, while shorter sentences are padded with zeros. The zero id reserved for padding will be mapped to the zero vector during the embedding operation.

Word Embeddings. The vectors used to initialize our word embeddings are the same from Severyn and Moschitti [2015a] and are publicly available[7]. They were computed by running the word2vec tool [Mikolov et al., 2013c] on a corpus obtained by joining the English Wikipedia dump and the AQUAINT corpus[8], which contains about 375 million words. The 50-dimensional embeddings were trained using the skipgram model with window size 5, and not considering words occurring less than 5 times. The embeddings of words not included in this set of vectors are randomly sampled from the uniform distribution $U[-0.25, 0.25]$.

The word embedding matrix is kept fixed during training given the relatively small datasets, which do not allow us to benefit from tuning the matrix weights directly on the data. On the other hand, this reduces the number of parameters and updates, resulting

[6] `https://spacy.io/`

[7] `https://github.com/aseveryn/deep-qa`

[8] `https://catalog.ldc.upenn.edu/LDC2002T31`

in a faster training.

Training, Model Selection and Evaluation Metrics. The networks are trained using Adam [Kingma and Ba, 2014], setting β_1 to 0.9 and ϵ to $1e-5$. During the question classifier and question focus identifier training, we monitor their accuracy on some percentage of heldout examples. The QA model is trained for 30 epochs. In this case, model selection is done by monitoring MAP on the development set. The model with the highest MAP is used to compute predictions on the test data. In addition to MAP, we also report MRR. For details on these metrics see Section 6.3.2.

6.4.5 Question Classification

In this part, we describe how we classify questions into a taxonomy using a CNN.

Dataset. The UIUC question classification dataset [Li and Roth, 2006] is composed by 5,452 training and 500 test questions, organized into a two-layered taxonomy with coarse and fine classes. The coarse layer maps each question into one of 6 classes: Abbreviation, Description, Entity, Human, Location and Number. We use the coarse classes to train our question classifier.

Training and Hyperparameters. The CNN network for question classification applies convolution filters of width 1, 2 and 3 and size 300 to the question, followed by max-pooling. The three pooled representations are concatenated and fed to a hidden layer with 100 dimensions, followed by a ReLU activation function and the final softmax layer. A dropout rate of 0.3 is applied to hidden layer, and L2 regularization is applied to all the network parameters, scaling the regularization term added to the loss by $\beta = 1e-5$. We use the Adam optimizer to train the network. The learning rate is set to $5e-4$, and the batch size to 32. We reserve 5% of the examples for monitoring the accuracy and selecting the best model.

Results. The performance of our classifier are displayed in table 6.6. Our goal is not to have a state-of-the-art classifier, but to be able to annotate unseen questions with a reasonable accuracy. Since the model has a good convergence, we annotate the questions in the question answering datasets after learning the model on the full training set, using the test set to pick the best model.

Question Classifier	Accuracy
CNN static	92.8
Our CNN	91.2

Table 6.6: Comparison between a CNN model with static embeddings [Kim, 2014] and our model. The difference in performance can be quantified in 8 misclassified instances.

Question Focus Classifier	CV Accuracy
PTK	96.9
Our CNN	92.3

Table 6.7: Comparison of the cross-validation accuracies between PTK, a tree kernel based classifier [Severyn and Moschitti, 2013], and our CNN model.

6.4.6 Question Focus Identification

In this section, we detail the training of the second component for question analysis, namely the question focus identifier.

Dataset. The question focus identification dataset [Bunescu and Huang, 2010] contains the first 2,000 UIUC questions annotated with focus information, but three questions do not have focus annotations. After removing them and the questions with implicit and multi-focus, we end up with 1,030 questions for training the model. In future work, we will also include the implicit and multi-focus cases in our model.

Training and Hyperparameters. The focus identification network applies a stack of 4 convolution filters. The filters have width equal to 10 and size equal to 100. Each filter output, corresponding to a sentence token, is passed through the same hidden ReLU layer of size 100, and a final linear layer outputs the focus score. L2 regularization is applied on all the network parameters, scaling by $\beta = 1e - 4$. The Adam learning rate is set to $1e - 4$, and the batch size is set to 16.

Results. The cross-validation accuracy of the classifier is reported in Table 6.7. The Partial Tree Kernel (PTK) model [Severyn and Moschitti, 2013] is an SVM kernel classifier on structures which counts the number of common substructures between two trees. Sentences are encoded with trees that contain POS tags, syntactic chunks and lemmas.

To obtain such annotations an NLP pipeline with a POS tagger, a chunker and a lemmatizer is required. In comparison, we use only word vectors as input for the question focus identifier. The accuracy of our network on the development set converges after 300 epochs. After that number of epochs, we use the network to annotate the focus words in the question answering datasets.

6.4.7 TrecQA

The first experiment is on newswire data from the TREC campaign.

Dataset. We test our answer sentence reranking model on the TrecQA dataset [Wang et al., 2007], one of the most popular benchmark for answer reranking. The dataset contains factoid questions and candidate answer sentences. The questions are collected from the TrecQA tracks 8-13. The entire TREC 13 and the first 100 questions from TREC 8-12 are manually judged. This effort is made to improve the quality of the labeling, originally done with regular expressions.

We use the same train, dev., and test splits of the original data, but we run our experiments using the larger provided training set (TRAIN-ALL), which includes 1,229 questions from TREC 8-12 with automatic judgements. These judgements increase the dataset noise, but give us many more examples for training. We remove from the dev. and test sets questions with no answers, and questions with only correct or incorrect answer sentence candidates. The resulting dev. and test set contain 65 and 68 questions respectively. This evaluation setting is also referred to as TrecQA Clean[9].

Training and Hyperparameters. The CNN network for ranking question/answer pairs applies two separate convolution filters of width 5 and size 100 to the question and the answer, followed by a max-pooling operation. The two pooled representations are concatenated and passed through a hidden layer with 200 dimensions, followed by a ReLU activation function and the final softmax layer. A dropout rate of 0.5 is applied to the hidden layer, and L2 regularization is applied on all the parameters. The L2 loss is scaled by $\beta = 1e - 5$ for the filters, and by $\beta = 1e - 4$ for the fully connected layers. The dimensionality of the word and semantic overlap vectors is set to 5. We train the network using Adam, with a learning rate of $5e - 5$, and a batch size of 50. We report mean and standard deviation of 10 random restarts of our models.

[9] `https://aclweb.org/aclwiki/Question_Answering_(State_of_the_art)`

System	MAP	MRR
Tan et al. [2015]	72.79	82.40
Wang and Ittycheriah [2015]	74.60	82.00
Santos et al. [2016]	75.30	85.11
He and Lin [2016]	75.88	82.19
Severyn and Moschitti [2016]	76.54	81.86
Wang et al. [2016b]	77.14	84.47
Rao et al. [2016]	80.10	87.70
Wang et al. [2017]	80.20	87.50
Shen et al. [2017]	82.20	88.90
CNN$_{WO}$ TRAIN-ALL	76.49 (0.4)	84.22 (0.5)
CNN$_{WO+SO}$ TRAIN-ALL	77.93 (0.7)	84.89 (0.9)

Table 6.8: MAP and MRR (%) on the TrecQA Clean dataset.

Results. Table 6.8 contains a survey of results from previous work, and the performance of our models. CNN$_{WO}$ is our variant of the S&M model with word overlap embeddings, and without the bilinear matrix and the additional count features. It has comparable performance in terms of MAP, but it is 2.4% points higher in MRR. Our model CNN$_{WO+SO}$ that integrates the semantic overlap improves over CNN$_{WO}$ by 1.44% points in MAP, and 0.67% points in MRR. It approaches the model by Rao et al. [2016], which uses a triplet ranking loss, and several strategies to build training instances with difficult negative examples. Our system beats several others which use word alignments and attention mechanisms. The better systems employ expensive bidirectional networks, sophisticated attention mechanisms, and extract multiple views of questions and answers for comparing and aggregating them.

6.4.8 WikiQA

In addition to TrecQA, we evaluate our model on a bigger dataset with questions that share less words with the answer passages.

Dataset. TrecQA is a small dataset with a small evaluation set, so results may be unstable. There is also a significant lexical overlap between questions and answer can-

System	MAP	MRR
Yang et al. [2015]	65.20	66.52
Santos et al. [2016]	68.86	69.57
Miao et al. [2016]	68.86	70.69
Yin et al. [2016]	69.21	71.08
Severyn and Moschitti [2016]	69.51	71.07
Chen et al. [2017]	70.10	71.80
Rao et al. [2016]	70.90	72.30
Tymoshenko et al. [2016]	71.25	72.30
Guo et al. [2017]	71.71	73.36
Wang et al. [2017]	71.80	73.10
Shen et al. [2017]	73.30	75.00
Wang et al. [2016a]	73.41	74.18
Wang and Jiang [2017]	74.33	75.40
CNN$_{WO}$	69.53 (0.5)	71.35 (0.5)
CNN$_{WO+SO}$	72.24 (0.5)	73.91 (0.5)

Table 6.9: MAP and MRR (%) on the WikiQA dataset.

didates [Yih et al., 2013]. This means that simple lexical similarity features have high discriminative power for the task. To better assess the quality of our models, we also experiment with the WikiQA dataset [Yang et al., 2015], which is an order of magnitude larger than TrecQA. Questions are sampled from Bing query logs, associated with Wikipedia pages based on user clicks. The sentences in the page are manually annotated by crowdworkers. Following Yin et al. [2016], we remove the questions without any correct answer from our evaluation.

Training and Hyperparameters. We use the same procedures applied for TrecQA.

Results. Table 6.9 contains the results on WikiQA. Again the MAP score is comparable with the S&M model, while the MRR is slightly higher. Our model CNN$_{WO+SO}$ improves over CNN$_{WO}$ by 2.71% points in MAP, and 2.56% points in MRR, with a higher margin with respect to TrecQA. Interestingly, our approach improves the structural model

System	P@1	MRR
Random Guess	22.50	49.27
CNN	41.25	63.23
CNTN	46.54	66.87
LSTM	48.75	68.29
NTN-LSTM	54.48	73.09
HD-LSTM	55.69	73.47
BM25	57.9	72.6
LSTM-RNN	69.0	82.2
MV-LSTM-Cosine	73.9	85.2
MV-LSTM-Bilinear	75.1	86.0
MV-LSTM-Tensor	76.6	86.9
CNN	57.71 (0.1)	73.46 (0.1)
CNN$_{WO}$	80.91 (0.1)	88.21 (0.1)
CNN$_{WO+SO}$	81.44 (0.1)	88.52 (0.1)

Table 6.10: P@1 and MRR (%) on YahooQA. The first group of results is reported in Tay et al. [2017]. The second group of results is reported in Wan et al. [2016]. The third and final group of results is related to our convolutional models: basic, with word overlap, and with word and semantic overlap.

by Tymoshenko et al. [2016] by 1 MAP point. Similarly to our work here and in Section 6.2, that model includes relational information in terms of question focus, entities and question categories but uses much more syntactic annotations. Our network is able to make better use of the provided semantic clues. Surprisingly, CNN$_{WO+SO}$ has also a higher MAP score than Wang et al. [2017], one of the state-of-the-art complex model mixing attention and interaction factors of multiple sentence perspectives.

6.4.9 YahooQA

The final experiment is carried out on q/a provided by users on web forums.

Dataset. To conclude our experiments, we evaluate our model in a community Question Answering (cQA) setting. In cQA, a user asks a question on a forum, and other users provide answers. This setup differs from other QA tasks where the answer passages come from formal text (newswire or Wikipedia), by the higher noise due to user provided text. At word level, misspellings are very frequent, and it is easy to encounter slang, elongated words and typos. At sentence level, users may type totally unrelated answers, with the intent of being sarcastic or spammy.

We select the YahooQA dataset[10], which is a curated subset of a 2007 dump of the *Yahoo! Answers* web site. The dataset contains 142,627 questions with their answers. It is important to note that all the question starts with one of the following patterns: "how {to — do — did — does — can — would — could — should}". For this reason, they are in general more complex and longer than purely factoid questions.

We adopt the dataset preprocessing steps and the train/dev/test splits proposed in Wan et al. [2016], followed and published[11] in Tay et al. [2017]: answers are constrained to have 5 to 50 tokens, and non-alphanumeric characteres are filtered. The final number of q/a pairs is 63,360. The training, development and test ratio is 80/10/10, resulting in 253,440, 31,680 and 31,680 classification instances respectively.

Training and Hyperparameters. Again, we use the same procedures applied for the experiments on the TrecQA and WikiQA datasets.

Results. Table 6.10 contains the results on the YahooQA dataset. The first group of measures are reported by Tay et al. [2017], and it seems that the performance of the models are very low compared to the baselines and models reported in Wan et al. [2016]. Both papers share the same experimental settings, but in the second case, the BM25 and LSTM baselines are much stronger. Our results are more aligned with Wan et al. [2016]. For completeness, we also report the performance of our CNN model without word and semantic overlap features.

The word overlap feature yields a very strong contribution on this dataset, reducing the performance difference with the semantic overlap feature. Nonetheless, the latter still gives an improvement in $P@1$ and MRR. The smaller effect may be attributed to the nature of the questions in YahooQA: in general they are *how* questions that require more complex descriptive answers with respect to factoid questions, which usually seek an entity of some kind.

Figure 6.4 shows the distributions of the category predictions from our question classifier on each training instance contained in the YahooQA, WikiQA and TrecQA datasets.

[10] `http://webscope.sandbox.yahoo.com/catalog.php?datatype=l&did=10`

[11] `https://github.com/vanzytay/YahooQA_Splits`

Query Question: *How many people were killed in the Oklahoma City bombing ?*				
GS	**Answer Sentence Candidates**	**CNN**$_{WO+SO}$	**CNN**$_{WO}$	**CNN**
I	The Oklahoma City bombing was a domestic terrorist bomb attack on the Alfred P. Murrah Federal Building in downtown Oklahoma City on April 19, 1995.	0.318	0.408	0.425
I	It would remain the most destructive act of terrorism on American soil until the September 11, 2001 attacks	0.025	0.071	0.067
R	The Oklahoma blast claimed 168 lives, including 19 children under the age of 6, and injured more than 680 people.	0.582	0.367	0.335
I	The blast destroyed or damaged 324 buildings within a sixteen - block radius, destroyed or burned 86 cars, and shattered glass in 258 nearby buildings.	0.045	0.058	0.074
I	The bomb was estimated to have caused at least $ 652 million worth of damage.	0.013	0.018	0.063
I	Extensive rescue efforts were undertaken by local, state, federal, and worldwide agencies in the wake of the bombing, and substantial donations were received from across the country.	0.052	0.071	0.104

Table 6.11: A question along with different answer sentence candidates from the WikiQA dataset. The first column report the gold standard annotation, i.e., Relevant (R) or Irrelevant (I) labels whereas the last three columns report the CNN score using word and semantic links, only word links and no link.

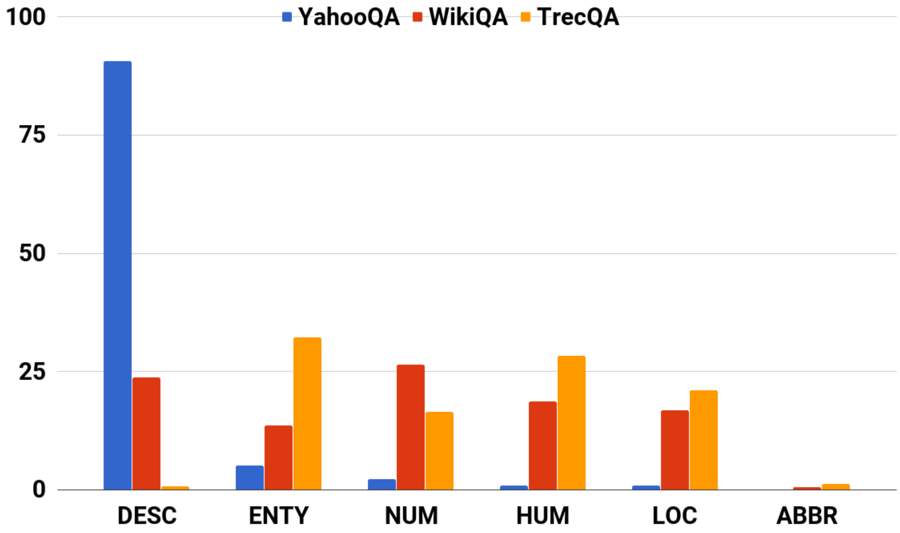

Figure 6.4: Percentage of training question/answer pairs (y-axis) where the question is classified with the corresponding question type (x-axis) by our question classifier.

As expected, the vast majority of the questions in YahooQA are classified as seeking a description for an answer (DESC). The question category distributions for WikiQA and TrecQA are different. WikiQA has a relatively more uniform distribution of categories. TrecQA has very few DESC questions, being a mainly factoid QA dataset. The YahooQA skewness may explain the modest improvement coming from the semantic overlap feature.

We can certainly conclude that the word overlap is very helpful and enables a simple CNN model to outperform more sophisticated systems. For example, the MV-LSTM models run a bidirectional LSTM to obtain contextual word representations, and then compute an interaction matrix between two sentences. The reported variants employ a cosine function, a bilinear function, or a tensor layer. At the end, interactions are aggregated using a k-max pooling layer, and refined by a multilayer perceptron.

6.4.10 Discussion and Error Analysis

The results with the CNN$_{WO+SO}$ model suggest that the semantic overlap vectors are an effective way of establishing a semantic link between questions and answers. This is especially true, as suggested by the higher performance on WikiQA, where the questions and answers have little lexical overlap. With the additional semantic information, the convolutional network is able to better model the relevancy of candidate passages. It also surpasses the accuracy of more complex systems, which have higher training time. We conducted error analysis to better understand the effectiveness of the semantic links.

Table 6.11 shows a question along with six candidates. Interestingly, the first candidate, which receives a high score (0.318) contains the phrase *The Oklahoma City bombing*, which exactly matches the same phrase in the question. Additionally, it also contains other three named entities, i.e., *Alfred P. Murrah Federal Building*, *Oklahoma City* and *April 19, 1995*, where the repetition of *Oklahoma City* clearly strengths the contribution of the matching with the same entity in the question. In these conditions, any model that does not exploit relational semantic links (or other forms of relational information) cannot correctly classify the sentence. For example, even more sophisticated neural models matching entities would be misled. Indeed the only entity of the question is of type *Location*, which has a match with the category of *Oklahoma City* appearing in the answer sentence.

In contrast, our model, CNN_{WO+SO}, is aware that the focus of the question is of type *Number* and does not match with any category of the entities of the sentence. Differently, the correct candidate (in the third row) contains several numbers, which are of course annotated with the category *Number* and thus compatible with the type of the focus. This is why it receives a score from the network, 0.582, which is twice the one the first candidate got.

Finally, we note that the annotation networks and the answer reranking networks take little time to train: from 10 to 20 minutes, depending on the number of question/answer pairs. CNNs are faster at training and inference time with respect to RNNs, especially when the latter incorporate attention mechanisms, which increase the number of computations. We argue that annotating a relatively small number of examples to model some semantic traits of a text, could be cost well spent to increase the accuracy of a model, while reducing its complexity.

6.4.11 Summary and Future Work

In this section, we presented a neural architecture that models semantic links between questions and answers, in addition to lexical links. The semantic annotations required to establish such links are produced by a set of fast neural components for question analysis. We train these components on public datasets. The evaluation on three answer sentence ranking datasets shows that our final model is able to achieve state-of-the-art performance for a simple CNN, as well as low complexity and training time.

Chapter 7

Conclusion and Future Work

Convolution tree kernels and neural networks are effective methods for automatic feature engineering. Merging the two approaches is an interesting research direction. Ideally, we would like to exploit (i) the power of tree kernels on syntactic and semantic structures, especially in scarce data settings, and (ii) the capability of neural networks to learn useful word representations that may also contain information from the context in which the word appears, up to information from the entire sentence.

To explore these research lines and ideas, we develop accurate tree kernel and neural network based models, and we apply them to QA and text classification tasks. First, we study such methods in isolation, and then we explore two approaches that merge tree kernel and neural network models. In one case, we use the features computed by a deep neural network in an SVM classifier, therefore by combining a tree kernel on structures, and a polynomial kernel on the features. In the second case, we move the deep neural network directly in the computation of the kernel function by learning a similarity between words, which takes into account not only the sentence in which the words appear, but also the categorical labels related to the text classification task.

In Chapters 2 and 3, we give background information on the two core technologies presented in this thesis: kernels and neural networks. In Chapter 3, we also introduce the neural network architecture at the core of one of our main contributions, and related to that, we describe our hybrid siamese network and multiloss training setup for paraphrase identification and textual entailment tasks.

In Chapter 4, we demonstrate the power of structural kernels, in the context of the automatic resolution of crossword puzzles. We present two LTR approaches that incorporate structural similarities, and improve an automatic crossword solver, achieving state-of-the-art results in the resolution of challenging puzzles.

Chapter 5 contains the two main contribution of the thesis. First, we propose a neural network model for the identification of crossword clues which have the same solution. A kernel on the features from the trained network is combined with a structural kernel, to obtain improved results when the training data is scarce. Finally, we design a novel tree kernel that combines structural similarity with a deep learned similarity between words. The key to train such model is a siamese network that produces word representations and a similarity optimized for the end classification task. In the future, we would like to study the sensitiveness of the latter method to a higher number of classes, experiment with other sampling mechanisms for training the siamese encoder, and explore the effect of language modeling representations [Peters et al., 2018].

In Chapter 6, we present our previous work on structural kernels and representations for factoid QA. An advanced semantic linking mechanism for question and candidate answers is obtained by enhancing their structural representations with special tags that signal a relation between tagged nouns and entities. In addition to that, we present our recent fully neural system that beats its structural based counterpart. Such system uses CNNs to annotate the question with its focus and category, and a neural sentence encoding to link question words and entitites in the answers. Our approach is an interesting first step towards a future architecture, in which we will jointly optimize the semantic annotators and the QA model in a purely end-to-end fashion. This would let the neural annotators learn better representations with the data from the QA task, improving their outputs, and ultimately refine the output of the QA module. In addition, automatically learning a better mapping between NE types and question categories could be an interesting research direction.

Bibliography

Agirre, Eneko; Banea, Carmen; Cer, Daniel; Diab, Mona; Gonzalez-Agirre, Aitor; Mihalcea, Rada; Rigau, German, and Wiebe, Janyce. Semeval-2016 task 1: Semantic textual similarity, monolingual and cross-lingual evaluation. In *Proc. of SemEval-2016*, June 2016.

Aktolga, Elif; Allan, James, and Smith, David A. Passage reranking for question answering using syntactic structures and answer types. In *ECIR*, 2011.

Alexandrescu, Andrei and Kirchhoff, Katrin. Factored neural language models. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, NAACL-Short '06, pages 1–4, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1614049.1614050`.

Allison, L and Dix, T I. A bit-string longest-common-subsequence algorithm. *Inf. Process. Lett.*, 23(6):305–310, December 1986. ISSN 0020-0190.

Andor, Daniel; Alberti, Chris; Weiss, David; Severyn, Aliaksei; Presta, Alessandro; Ganchev, Kuzman; Petrov, Slav, and Collins, Michael. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany, August 2016. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P16-1231`.

Bahdanau, Dzmitry; Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.

Bär, Daniel; Zesch, Torsten, and Gurevych, Iryna. DKPro similarity: An open source framework for text similarity. In *Proceedings of ACL (System Demonstrations)*, 2013.

Barlacchi, Gianni; Nicosia, Massimo, and Moschitti, Alessandro. A retrieval model for automatic resolution of crossword puzzles in italian language. In *The First Italian Conference on Computational Linguistics CLiC-it 2014*, 2014a.

Barlacchi, Gianni; Nicosia, Massimo, and Moschitti, Alessandro. Learning to rank answer candidates for automatic resolution of crossword puzzles. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, 2014b.

Barlacchi, Gianni; Nicosia, Massimo, and Moschitti, Alessandro. SACRY: Syntax-based automatic crossword puzzle resolution system. In *Proceedings of 53nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, Beijing, China, July 2015. Association for Computational Linguistics.

Bengio, Y.; Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, March 1994. ISSN 1045-9227. doi: 10.1109/72.279181. URL `http://dx.doi.org/10.1109/72.279181`.

Bengio, Yoshua; Ducharme, Réjean; Vincent, Pascal, and Jauvin, Christian. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

Biemann, Chris. Creating a system for lexical substitutions from scratch using crowdsourcing. *Lang. Resour. Eval.*, 47(1):97–122, March 2013. ISSN 1574-020X. doi: 10.1007/s10579-012-9180-5.

Bloehdorn, Stephan and Moschitti, Alessandro. Combined syntactic and semantic kernels for text classification. In *ECIR*, 2007.

Bordes, Antoine; Chopra, Sumit, and Weston, Jason. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 615–620, Doha, Qatar, October 2014a. Association for Computational Linguistics.

Bordes, Antoine; Weston, Jason, and Usunier, Nicolas. Open question answering with weakly supervised embedding models. In *Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 8724*, ECML PKDD 2014, pages 165–180, New York, NY, USA, 2014b. Springer-Verlag New York, Inc. ISBN 978-3-662-44847-2. doi: 10.1007/978-3-662-44848-9_11. URL `http://dx.doi.org/10.1007/978-3-662-44848-9_11`.

Botha, Jan A. and Blunsom, Phil. Compositional morphology for word representations and language modelling. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1899–II–1907. JMLR.org, 2014. URL `http://dl.acm.org/citation.cfm?id=3044805.3045104`.

Bottou, Léon. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Bowman, Samuel R.; Angeli, Gabor; Potts, Christopher, and Manning, Christopher D. A large annotated corpus for learning natural language inference. In *Proc. of EMNLP*, 2015.

Bromley, Jane; Guyon, Isabelle; Lecun, Yann; Sckinger, Eduard, and Shah, Roopak. Signature verification using a "siamese" time delay neural network. In *Proc. of NIPS*, 1994.

Bunescu, Razvan and Huang, Yunfeng. Towards a general model of answer typing: Question focus identification. In *CICLing*, 2010.

Castellucci, Giuseppe; Filice, Simone; Croce, Danilo, and Basili, Roberto. Unitor: Combining syntactic and semantic kernels for twitter sentiment analysis. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 369–374, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S13-2060`.

Chen, Danqi and Manning, Christopher. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750. Association for Computational Linguistics, 2014. doi: 10.3115/v1/D14-1082. URL `http://www.aclweb.org/anthology/D14-1082`.

Chen, Ruey-Cheng; Yulianti, Evi; Sanderson, Mark, and Croft, W. Bruce. On the benefit of incorporating external features in a neural architecture for answer sentence selection. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 1017–1020, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5022-8. doi: 10.1145/3077136.3080705.

Chopra, Sumit; Hadsell, Raia, and LeCun, Yann. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 539–546, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2372-2. doi: 10.1109/CVPR.2005.202. URL `http://dx.doi.org/10.1109/CVPR.2005.202`.

Chung, Junyoung; Gulcehre, Caglar; Cho, KyungHyun, and Bengio, Yoshua. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Cohen, Daniel and Croft, W. Bruce. End to end long short term memory networks for non-factoid question answering. In *Proc. of ICTIR*, 2016. ISBN 978-1-4503-4497-5.

Collins, Michael and Duffy, Nigel. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 263–270, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics. doi: 10.3115/1073083.1073128. URL `http://dx.doi.org/10.3115/1073083.1073128`.

Collobert, Ronan; Weston, Jason; Bottou, Léon; Karlen, Michael; Kavukcuoglu, Koray, and Kuksa, Pavel. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12(Aug):2493–2537, 2011.

Croce, Danilo; Moschitti, Alessandro, and Basili, Roberto. Structured lexical similarity via convolution kernels on dependency trees. In *In EMNLP*, Edinburgh, Scotland, UK., 2011. URL `http://www.aclweb.org/anthology/D11-1096`.

Croce, Danilo; Moschitti, Alessandro; Basili, Roberto, and Palmer, Martha. Verb classification using distributional similarity in syntactic and semantic structures. In *ACL (1)*, pages 263–272. The Association for Computer Linguistics, 2012. ISBN 978-1-937284-24-4.

Damljanovic, Danica; Agatonovic, Milan, and Cunningham, Hamish. Identification of the question focus: Combining syntactic analysis and ontology-based lookup through the user interaction. In *LREC*, 2010.

Das, Arpita; Yenala, Harish; Chinnakotla, Manoj, and Shrivastava, Manish. Together we stand: Siamese networks for similar question retrieval. In *Proc. of ACL*, August 2016.

Dong, Li; Wei, Furu; Yin, Yichun; Zhou, Ming, and Xu, Ke. Splusplus: A feature-rich two-stage classifier for sentiment analysis of tweets. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 515–519, Denver, Colorado, June 2015. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S15-2086`.

Dos Santos, Cícero Nogueira and Zadrozny, Bianca. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1818–II–1826. JMLR.org, 2014. URL `http://dl.acm.org/citation.cfm?id=3044805.3045095`.

Duchi, John; Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, 2011.

Echihabi, Abdessamad and Marcu, Daniel. A noisy-channel approach to question answering. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, ACL '03, pages 16–23, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics. doi: 10.3115/1075096.1075099. URL `https://doi.org/10.3115/1075096.1075099`.

Elman, Jeffrey L. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. doi: 10.1207/s15516709cog1402_1.

Ernandes, Marco; Angelini, Giovanni, and Gori, Marco. Webcrow: A web-based system for crossword solving. In *In Proc. of AAAI 05*, pages 1412–1417. Menlo Park, Calif., AAAI Press, 2005.

Ernandes, Marco; Angelini, Giovanni, and Gori, Marco. A web-based agent challenges human experts on crosswords. *AI Magazine*, 29(1), 2008. ISSN 0738-4602.

Ferraresi, Adriano; Zanchetta, Eros; Baroni, Marco, and Bernardini, Silvia. Introducing and evaluating ukwac, a very large web-derived corpus of english. In *Proceedings of the 4th Web as Corpus Workshop (WAC-4) Can we beat Google?*, page 47, 2008.

Ferrucci, David and Lally, Adam. Uima: An architectural approach to unstructured information processing in the corporate research environment. *Nat. Lang. Eng.*, 10(3-4):327–348, September 2004. ISSN 1351-3249. doi: 10.1017/S1351324904003523.

Ferrucci, David A.; Brown, Eric W.; Chu-Carroll, Jennifer; Fan, James; Gondek, David; Kalyanpur, Aditya; Lally, Adam; Murdock, J. William; Nyberg, Eric; Prager, John M.; Schlaefer, Nico, and Welty, Christopher A. Building watson: An overview of the deepqa project. *AI Magazine*, 31(3):59–79, 2010.

Filice, Simone; Da San Martino, Giovanni, and Moschitti, Alessandro. Structural representations for learning relations between pairs of texts. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1003–1013, Beijing, China, July 2015. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P15-1097`.

Filice, Simone; Croce, Danilo; Moschitti, Alessandro, and Basili, Roberto. Kelp at semeval-2016 task 3: Learning semantic relations between questions and answers. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1116–1123, San Diego, California, June 2016. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S16-1172`.

Gabrilovich, Evgeniy and Markovitch, Shaul. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 1606–1611, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

Ginsberg, Matthew L. Dr.fill: Crosswords and an implemented solver for singly weighted csps. *J. Artif. Int. Res.*, 42(1):851–886, September 2011. ISSN 1076-9757.

Gonzalo, Julio; Li, Hang; Moschitti, Alessandro, and Xu, Jun. Sigir 2014 workshop on semantic matching in information retrieval. In *Proc. of SIGIR*, 2014. ISBN 978-1-4503-2257-7.

Goodfellow, Ian; Bengio, Yoshua, and Courville, Aaron. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

Guo, Jiahui; Yue, Bin; Xu, Guandong; Yang, Zhenglu, and Wei, Jin-Mao. An enhanced convolutional neural network model for answer selection. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, pages 789–790, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4914-7.

Gusfield, Dan. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, New York, NY, USA, 1997. ISBN 0-521-58519-8.

Hanbury, Allan; Kazai, Gabriella; Rauber, Andreas, and Fuhr, Norbert, editors. *Advances in Information Retrieval - 37th European Conference on IR Research, ECIR 2015, Vienna, Austria, March 29 - April 2, 2015. Proceedings*, volume 9022 of *Lecture Notes in Computer Science*, 2015. ISBN 978-3-319-16353-6. doi: 10.1007/978-3-319-16354-3. URL `http://dx.doi.org/10.1007/978-3-319-16354-3`.

He, Hua and Lin, Jimmy. Pairwise word interaction modeling with deep neural networks for semantic similarity measurement. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 937–948, San Diego, California, June 2016. Association for Computational Linguistics.

Heilman, Michael and Smith, Noah A. Tree edit models for recognizing textual entailments, paraphrases, and answers to questions. In *NAACL*, 2010.

Herbrich, R; Graepel, T, and Obermayer, K. Large margin rank boundaries for ordinal regression. In Smola, A.J.; Bartlett, P.L.; Schölkopf, B., and Schuurmans, D., editors, *Advances in Large Margin Classifiers*, pages 115–132, Cambridge, MA, 2000. MIT Press.

Hickl, Andrew; Williams, John; Bensley, Jeremy; Roberts, Kirk; Shi, Ying, and Rink, Bryan. Question answering with lcc's chaucer at trec 2006. In *In Proceedings of the Text REtrieval Conference*, pages 283–292, 2007.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Hornik, Kurt; Stinchcombe, Maxwell, and White, Halbert. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. URL `http://www.sciencedirect.com/science/article/pii/0893608089900208`.

Hu, Baotian; Lu, Zhengdong; Li, Hang, and Chen, Qingcai. Convolutional neural network architectures for matching natural language sentences. In *NIPS*. 2014.

Iacobacci, Ignacio; Pilehvar, Mohammad Taher, and Navigli, Roberto. Embeddings for word sense disambiguation: An evaluation study. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 897–907, Berlin, Germany, August 2016. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P16-1085`.

Iyyer, Mohit; Manjunatha, Varun; Boyd-Graber, Jordan, and Daumé III, Hal. Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1681–1691, 2015.

Jeon, Jiwoon; Croft, W. Bruce, and Lee, Joon Ho. Finding similar questions in large question and answer archives. In *CIKM*, 2005.

Joachims, T. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of ECML-98*, 1998.

Joachims, Thorsten. Making large-scale support vector machine learning practical. In Schölkopf, Bernhard; Burges, Christopher J. C., and Smola, Alexander J., editors, *Advances in Kernel Methods*, pages 169–184. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-19416-3. URL `http://dl.acm.org/citation.cfm?id=299094.299104`.

Joachims, Thorsten. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X. doi: 10.1145/775047.775067.

Kalchbrenner, Nal; Grefenstette, Edward, and Blunsom, Phil. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

Karpathy, Andrej; Toderici, George; Shetty, Sanketh; Leung, Thomas; Sukthankar, Rahul, and Fei-Fei, Li. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

Katz, Boris and Lin, Jimmy. Selectively using relations to improve precision in question answering, 2003.

Kim, Yoon. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, Doha, Qatar, October 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D14-1181`.

Kimeldorf, George S and Wahba, Grace. A correspondence between bayesian estimation on stochastic processes and smoothing by splines. *The Annals of Mathematical Statistics*, 41(2):495–502, 1970.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2014.

Krizhevsky, Alex; Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012.

Lally, A.; Prager, JM; McCord, MC; Boguraev, BK; Patwardhan, S.; Fan, J.; Fodor, P., and Chu-Carroll, J. Question analysis: How watson reads a clue. *IBM Journal of Research and Development*, 56(3.4), 2012.

Lawrence, Steve; Giles, C Lee; Tsoi, Ah Chung, and Back, Andrew D. Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113, 1997.

Le, Quoc V. and Mikolov, Tomas. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014. URL `http://arxiv.org/abs/1405.4053`.

LeCun, Yann; Bottou, Léon; Bengio, Yoshua, and Haffner, Patrick. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

LeCun, Yann; Bengio, Yoshua, and Hinton, Geoffrey. Deep learning. *nature*, 521(7553):436, 2015.

Levy, Omer and Goldberg, Yoav. Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 302–308, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P14-2050`.

Li, Xin and Roth, Dan. Learning question classifiers: the role of semantic information. *Natural Language Engineering*, 12(3):229–249, 2006.

Littman, Michael L.; Keim, Greg A., and Shazeer, Noam. A probabilistic approach to solving crossword puzzles. *Artificial Intelligence*, 134:23 – 55, 2002. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(01)00114-X.

Liu, Yandong and Agichtein, Eugene. On the evolution of the yahoo! answers qa community. In *Proc. of SIGIR*, 2008. ISBN 978-1-60558-164-4.

Lodhi, Huma; Saunders, Craig; Shawe-Taylor, John; Cristianini, Nello, and Watkins, Chris. Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444, 2002.

Ma, Mingbo; Huang, Liang; Zhou, Bowen, and Xiang, Bing. Dependency-based convolutional neural networks for sentence embedding. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 174–179, Beijing, China, July 2015. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P15-2029`.

McCandless, Michael; Hatcher, Erik, and Gospodnetic, Otis. *Lucene in Action, Second Edition: Covers Apache Lucene 3.0.* Manning Publications Co., Greenwich, CT, USA, 2010. ISBN 1933988177, 9781933988177.

Miao, Yishu; Yu, Lei, and Blunsom, Phil. Neural variational inference for text processing. In *International Conference on Machine Learning*, pages 1727–1736, 2016.

Mihalcea, Rada; Corley, Courtney, and Strapparava, Carlo. Corpus-based and knowledge-based measures of text semantic similarity. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pages 775–780. AAAI Press, 2006. ISBN 978-1-57735-281-5.

Mikolov, Tomáš; Karafiát, Martin; Burget, Lukáš; Černockỳ, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

Mikolov, Tomas; Chen, Kai; Corrado, Greg, and Dean, Jeffrey. Efficient estimation of word representations in vector space. *ICLR Workshop*, 2013a.

Mikolov, Tomas; Chen, Kai; Corrado, Greg, and Dean, Jeffrey. Efficient estimation of word representations in vector space. *International Conference on Learning Representations (2013)*, 2013b.

Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg S, and Dean, Jeff. Distributed representations of words and phrases and their compositionality. In Burges, C. J. C.; Bottou, L.; Welling, M.; Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013c.

Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado, Greg S, and Dean, Jeff. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119, 2013d.

Moschitti, Alessandro. A study on convolution kernels for shallow semantic parsing. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, ACL '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

Moschitti, Alessandro. Efficient convolution kernels for dependency and constituent syntactic trees. In *ECML*, pages 318–329. Springer, 2006.

Moschitti, Alessandro and Zanzotto, Fabio Massimo. Fast and effective kernels for relational learning from texts. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 649–656, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273578. URL `http://doi.acm.org/10.1145/1273496.1273578`.

Moschitti, Alessandro; Quarteroni, Silvia; Basili, Roberto, and Manandhar, Suresh. Exploiting syntactic and shallow semantic kernels for question/answer classification. In *ACL*, 2007.

Mueller, Jonas and Thyagarajan, Aditya. Siamese recurrent architectures for learning sentence similarity. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2786–2792. AAAI Press, 2016. URL `http://dl.acm.org/citation.cfm?id=3016100.3016291`.

Nair, Vinod and Hinton, Geoffrey E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Nakov, Preslav; Rosenthal, Sara; Kozareva, Zornitsa; Stoyanov, Veselin; Ritter, Alan, and Wilson, Theresa. Semeval-2013 task 2: Sentiment analysis in twitter. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 312–320, Atlanta, Georgia, USA, June 2013. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S13-2052`.

Neculoiu, Paul; Versteegh, Maarten, and Rotaru, Mihai. Learning text similarity with siamese recurrent networks. In *Proceedings of the 1st Workshop on Representation Learning for NLP*, pages 148–157, Berlin, Germany, August 2016. Association for Computational Linguistics. URL `http://anthology.aclweb.org/W16-1617`.

Nguyen, Thien Hai and Shirai, Kiyoaki. Aspect-based sentiment analysis using tree kernel based relation extraction. In Gelbukh, Alexander, editor, *Computational Linguistics and Intelligent Text Processing: 16th International Conference, CICLing 2015*, pages 114–125, Cairo, Egypt, 2015. Springer International Publishing. ISBN 978-3-319-18117-2.

Nguyen, Thien Huu; Plank, Barbara, and Grishman, Ralph. Semantic representations for domain adaptation: A case study on the tree kernel-based method for relation extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 635–644, Beijing, China, July 2015. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P15-1062`.

Nguyen, Truc Vien T. and Moschitti, Alessandro. End-to-end relation extraction using distant supervision from external semantic repositories. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 277–282, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P11-2048`.

Nguyen, Truc-Vien T.; Moschitti, Alessandro, and Riccardi, Giuseppe. Convolution kernels on constituent, dependency and sequential structures for relation extraction. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1378–1387, Singapore, August 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D/D09/D09-1143`.

Nicosia, Massimo and Moschitti, Alessandro. Accurate sentence matching with hybrid siamese networks. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 2235–2238, New York, NY, USA, 2017a. ACM. ISBN 978-1-4503-4918-5. doi: 10.1145/3132847.3133156. URL `http://doi.acm.org/10.1145/3132847.3133156`.

Nicosia, Massimo and Moschitti, Alessandro. Learning contextual embeddings for structural semantic similarity using categorical information. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 260–270, Vancouver, Canada, August 2017b. Association for Computational Linguistics. URL `http://aclweb.org/anthology/K17-1027`.

Nicosia, Massimo; Barlacchi, Gianni, and Moschitti, Alessandro. Learning to rank aggregated answers for crossword puzzles. In Hanbury et al. [2015], pages 556–561. ISBN 978-3-319-16353-6. doi: 10.1007/978-3-319-16354-3_61. URL `http://dx.doi.org/10.1007/978-3-319-16354-3_61`.

Parikh, Ankur; Täckström, Oscar; Das, Dipanjan, and Uszkoreit, Jakob. A decomposable attention model for natural language inference. In *Proc. of EMNLP*, November 2016.

Pennington, Jeffrey; Socher, Richard, and Manning, Christopher D. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

Peters, Matthew E; Neumann, Mark; Iyyer, Mohit; Gardner, Matt; Clark, Christopher; Lee, Kenton, and Zettlemoyer, Luke. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

Plank, Barbara and Moschitti, Alessandro. Embedding semantic similarity in tree kernels for domain adaptation of relation extraction. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1498–1507, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P13-1147`.

Plank, Barbara; Søgaard, Anders, and Goldberg, Yoav. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proc. of ACL*, August 2016.

Pohl, Ira. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(34):193 – 204, 1970. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/0004-3702(70)90007-X.

Quarteroni, Silvia; Guerrisi, Vincenzo, and Torre, Pietro La. Evaluating multi-focus natural language queries over data services. In *LREC*, 2012.

Radlinski, Filip and Joachims, Thorsten. Query chains: Learning to rank from implicit feedback. *CoRR*, 2006.

Rao, Jinfeng; He, Hua, and Lin, Jimmy. Noise-contrastive estimation for answer selection with deep neural networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, CIKM '16, pages 1913–1916, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4073-1.

Rao, Jinfeng; He, Hua, and Lin, Jimmy. Experiments with convolutional neural network models for answer selection. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1217–1220. ACM, 2017.

Ravichandran, Deepak and Hovy, Eduard. Learning surface text patterns for a question answering system. In *Proc. of ACL*, July 2002.

Resnik, Philip. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'95, pages 448–453, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1-55860-363-8, 978-1-558-60363-9.

Rumelhart, David E; Hinton, Geoffrey E, and Williams, Ronald J. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

Santos, Cicero dos; Tan, Ming; Xiang, Bing, and Zhou, Bowen. Attentive pooling networks. *arXiv preprint arXiv:1602.03609*, 2016.

Schuster, Mike and Paliwal, Kuldip K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Seo, Min Joon; Kembhavi, Aniruddha; Farhadi, Ali, and Hajishirzi, Hannaneh. Bidirectional attention flow for machine comprehension. *ICLR*, 2017.

Sequiera, Royal; Baruah, Gaurav; Tu, Zhucheng; Mohammed, Salman; Rao, Jinfeng; Zhang, Haotian, and Lin, Jimmy J. Exploring the effectiveness of convolutional neural networks for answer selection in end-to-end question answering. In *SIGIR 2017 Workshop on Neural Information Retrieval (Neu-IR'17)*, 2017.

Severyn, Aliaksei and Moschitti, Alessandro. Structural relationships for large-scale learning of answer re-ranking. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval (SIGIR)*, pages 741–750. ACM, 2012.

Severyn, Aliaksei and Moschitti, Alessandro. Automatic feature engineering for answer selection and extraction. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 458–467, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

Severyn, Aliaksei and Moschitti, Alessandro. Learning to rank short text pairs with convolutional deep neural networks. In *Proc. of SIGIR*, 2015a. ISBN 978-1-4503-3621-5.

Severyn, Aliaksei and Moschitti, Alessandro. Unitn: Training deep convolutional neural network for twitter sentiment classification. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 464–469, Denver, Colorado, June 2015b. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/S15-2079`.

Severyn, Aliaksei and Moschitti, Alessandro. Modeling relational information in question-answer pairs with convolutional neural networks. *CoRR*, abs/1604.01178, 2016.

Severyn, Aliaksei; Nicosia, Massimo, and Moschitti, Alessandro. Learning adaptable patterns for passage reranking. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 75–83, Sofia, Bulgaria, August 2013a. Association for Computational Linguistics.

Severyn, Aliaksei; Nicosia, Massimo, and Moschitti, Alessandro. Learning semantic textual similarity with structural representations. In *Proceedings of the 51st Annual Meeting of the ACL (Volume 2: Short Papers)*, pages 714–718. ACL, 2013b. URL `http://aclweb.org/anthology/P13-2125`.

Severyn, Aliaksei; Nicosia, Massimo; Barlacchi, Gianni, and Moschitti, Alessandro. Distributional neural networks for automatic resolution of crossword puzzles. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 199–204, Beijing, China, July 2015. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P15-2033`.

Shawe-Taylor, John and Cristianini, Nello. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004a. ISBN 0521813972.

Shawe-Taylor, John and Cristianini, Nello. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004b. ISBN 0521813972.

Shen, D. and Lapata, M. Using semantic roles to improve question answering. In *EMNLP-CoNLL*, 2007.

Shen, Gehui; Yang, Yunlun, and Deng, Zhi-Hong. Inter-weighted alignment network for sentence pair modeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1179–1189, Copenhagen, Denmark, September 2017. Association for Computational Linguistics.

Shen, Libin and Joshi, Aravind K. Ranking and reranking with perceptron. *Machine Learning*, 60(1-3):73–96, 2005.

Silva, João; Coheur, Luísa; Mendes, Ana Cristina, and Wichert, Andreas. From symbolic to sub-symbolic information in question classification. *Artificial Intelligence Review*, 35(2):137–154, 2010. ISSN 1573-7462.

Socher, Richard; Huval, Brody; Manning, Christopher D., and Ng, Andrew Y. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1201–1211, Jeju Island, Korea, July 2012. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D12-1110`.

Socher, Richard; Perelygin, Alex; Wu, Jean; Chuang, Jason; Manning, Christopher D.; Ng, Andrew, and Potts, Christopher. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/D13-1170`.

Sperr, Henning; Niehues, Jan, and Waibel, Alex. Letter n-gram-based input encoding for continuous space language models. In *Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality*, pages 30–39, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/W13-3204`.

Surdeanu, M.; Ciaramita, M., and Zaragoza, H. Learning to rank answers on large online QA collections. In *Proceedings of ACL-HLT*, 2008.

Tai, Kai Sheng; Socher, Richard, and Manning, Christopher D. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China, July 2015. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P15-1150`.

Tan, Ming; Xiang, Bing, and Zhou, Bowen. Lstm-based deep learning models for non-factoid answer selection. *CoRR*, abs/1511.04108, 2015. URL `http://arxiv.org/abs/1511.04108`.

Tan, Ming; dos Santos, Cicero; Xiang, Bing, and Zhou, Bowen. Improved representation learning for question answer matching. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 464–473, Berlin, Germany, August 2016. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P16-1044`.

Tay, Yi; Phan, Minh C.; Tuan, Luu Anh, and Hui, Siu Cheung. Learning to rank question answer pairs with holographic dual lstm architecture. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 695–704, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5022-8.

Trask, Andrew; Michalak, Phil, and Liu, John. sense2vec - A fast and accurate method for word sense disambiguation in neural word embeddings. *CoRR*, abs/1511.06388, 2015. URL `http://arxiv.org/abs/1511.06388`.

Tymoshenko, Kateryna; Moschitti, Alessandro, and Severyn, Aliaksei. Encoding semantic resources in syntactic structures for passage reranking. In *Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics*, pages 664–672, Gothenburg, Sweden, April 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/E14-1070`.

Tymoshenko, Kateryna; Bonadiman, Daniele, and Moschitti, Alessandro. Convolutional neural networks vs. convolution kernels: Feature engineering for answer sentence reranking. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1268–1278, San Diego, California, June 2016. Association for Computational Linguistics.

Tymoshenko, Kateryna; Moschitti, Alessandro; Nicosia, Massimo, and Severyn, Aliaksei. Reltextrank: An open source framework for building relational syntactic-semantic text pair representations. In *Proceedings of ACL 2017, System Demonstrations*, pages 79–84, Vancouver, Canada, July 2017. Association for Computational Linguistics. URL `http://aclweb.org/anthology/P17-4014`.

Wan, Shengxian; Lan, Yanyan; Guo, Jiafeng; Xu, Jun; Pang, Liang, and Cheng, Xueqi. A deep architecture for semantic matching with multiple positional sentence representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 2835–2841. AAAI Press, 2016. URL `http://dl.acm.org/citation.cfm?id=3016100.3016298`.

Wang, Bingning; Liu, Kang, and Zhao, Jun. Inner attention based recurrent neural networks for answer selection. In *ACL*, 2016a.

Wang, Kai; Ming, Zhaoyan, and Chua, Tat-Seng. A syntactic tree matching approach to finding similar questions in community-based qa services. In *Proc. of SIGIR*, 2009. ISBN 978-1-60558-483-6.

Wang, Mengqiu and Manning, Christopher D. Probabilistic tree-edit models with structured latent variables for textual entailment and question answer- ing. In *ACL*, 2010.

Wang, Mengqiu; Smith, Noah A., and Mitamura, Teruko. What is the Jeopardy model? a quasi-synchronous grammar for QA. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 22–32, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

Wang, Shuohang and Jiang, Jing. A compare-aggregate model for matching text sequences. In *Proc. of ICLR*, 2017.

Wang, Zhiguo and Ittycheriah, Abraham. Faq-based question answering via word alignment. *arXiv preprint arXiv:1507.02628*, 2015.

Wang, Zhiguo; Mi, Haitao, and Ittycheriah, Abraham. Sentence similarity learning by lexical decomposition and composition. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1340–1349. The COLING 2016 Organizing Committee, 2016b.

Wang, Zhiguo; Mi, Haitao, and Ittycheriah, Abraham. Sentence similarity learning by lexical decomposition and composition. In *Proc. of COLING*, December 2016c.

Wang, Zhiguo; Hamza, Wael, and Florian, Radu. Bilateral multi-perspective matching for natural language sentences. In *Proceedings of IJCAI*, 2017.

Weischedel, Ralph; Pradhan, Sameer; Ramshaw, Lance, and others, . Ontonotes release 4.0. 2012.

Wise, Michael J. Yap3: Improved detection of similarities in computer program and other texts. In *Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '96, pages 130–134, New York, NY, USA, 1996. ACM. ISBN 0-89791-757-X. doi: 10.1145/236452.236525.

Wu, Yonghui; Schuster, Mike; Chen, Zhifeng; Le, Quoc V.; Norouzi, Mohammad; Macherey, Wolfgang; Krikun, Maxim; Cao, Yuan; Gao, Qin; Macherey, Klaus; Klingner, Jeff; Shah, Apurva; Johnson, Melvin; Liu, Xiaobing; Kaiser, Lukasz; Gouws, Stephan; Kato, Yoshikiyo; Kudo, Taku; Kazawa, Hideto; Stevens, Keith; Kurian, George; Patil, Nishant; Wang, Wei; Young, Cliff; Smith, Jason; Riesa, Jason; Rudnick, Alex; Vinyals, Oriol; Corrado, Gregory S.; Hughes, Macduff, and Dean, Jeffrey. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

Xia, Fen; Liu, Tie-Yan; Wang, Jue; Zhang, Wensheng, and Li, Hang. Listwise approach to learning to rank: Theory and algorithm. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 1192–1199, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4. doi: 10.1145/1390156.1390306. URL `http://doi.acm.org/10.1145/1390156.1390306`.

Benjamin Van DurmeXuchen Yao, Peter Clark and Callison-Burch, Chris. Answer extraction as sequence tagging with tree edit distance. In *NAACL*, 2013.

Yang, Y. An evaluation of statistical approaches to text categorization. *Information Retrieval Journal*, 1999.

Yang, Yi; Yih, Wen-tau, and Meek, Christopher. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

Yih, Wen-tau; Chang, Ming-Wei; Meek, Christopher, and Pastusiak, Andrzej. Question answering using enhanced lexical semantic models. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1744–1753, Sofia, Bulgaria, August 2013. Association for Computational Linguistics.

Yin, Wenpeng; Schtze, Hinrich; Xiang, Bing, and Zhou, Bowen. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *Transactions of the Association for Computational Linguistics*, 4:259–272, 2016. ISSN 2307-387X.

Yu, Hsiang-Fu; Huang, Fang-Lan, and Lin, Chih-Jen. Dual coordinate descent methods for logistic regression and maximum entropy models. *Mach. Learn.*, 85(1-2):41–75, October 2011. ISSN 0885-6125. doi: 10.1007/s10994-010-5221-8.

Yu, Lei; Hermann, Karl Moritz; Blunsom, Phil, and Pulman, Stephen. Deep learning for answer sentence selection. *CoRR*, 2014.

Zhang, Yuan and Weiss, David. Stack-propagation: Improved representation learning for syntax. In *Proc. of ACL*, August 2016.