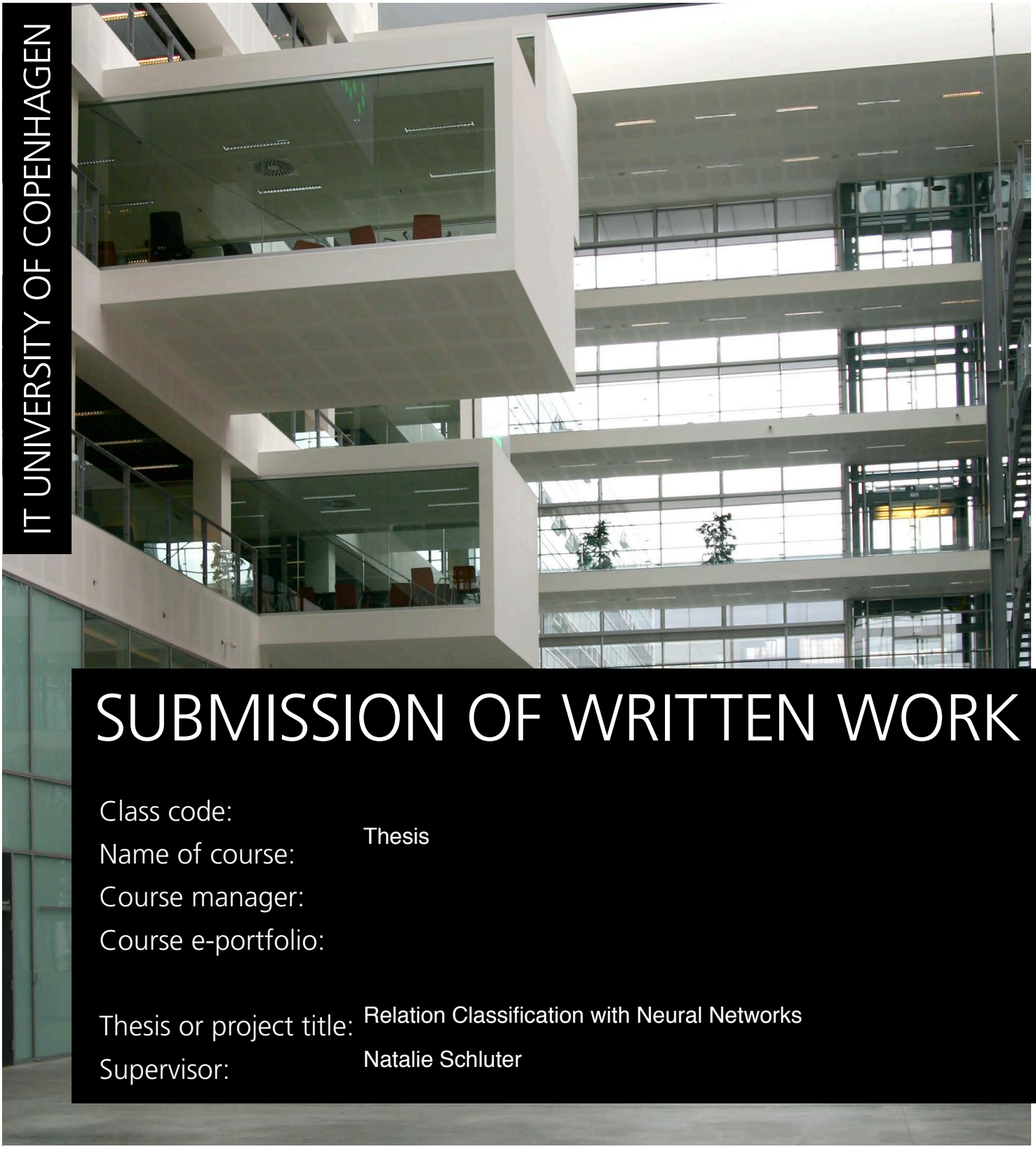
IT UNIVERSITY OF COPENHAGEN



SUBMISSION OF WRITTEN WORK

Class code:

Name of course: Thesis

Course manager:

Course e-portfolio:

Thesis or project title: Relation Classification with Neural Networks

Supervisor: Natalie Schluter

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Magnus Stahl Jacobsen	26/02-1991	msta@itu.dk
2.			@itu.dk
3.			@itu.dk
4.			@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk

Relation Classification with Neural Networks

Magnus Stahl Jacobsen

Supervised by Natalie Schluter

A thesis presented for a Master of Science in
Software Development: Advanced Computing

Denmark
01-06-2017

Abstract

This thesis investigates a modern approach to solving semantic relation classification by using convolutional neural networks. Semantic relation classification is the task of finding semantic relations between word pairs in a sentence. The task is investigated by the use of the SemEval 2010 task 8 dataset. The thesis uses a supervised baseline model and a semi-supervised self-training approach. A number of reproduceability issues occurs when implementing the baseline, and solutions are discussed in the thesis. Solutions are better score reporting and better specification of the baseline. The thesis finds that the baseline uses unnecessary l2-based constraining and large parameter space for positional embeddings which can be removed. The thesis adds early stopping to the network, which yields a small performance improvement. Class weights are also added and yields significant improvement. The robustness of the baseline is tested on unlabeled data from the W2C Wiki corpus, which significantly degrades performance. The failure of the semi-supervised approach is mainly attributed to the poor generilzation power of the SemEval dataset. The thesis suggests that further work on relation classification discontinues the use of the SemEval dataset. Finally, suggestions are made to the relation classification task itself which makes it more suitable for general usage in NLP.

Contents

Chapter 1

Introduction

Software that can understand and process natural language is in tremendous demand. It is embedded in personal computers, smartphones and physical objects that humans interact with from elevators to spaceships. Understanding human language as well as humans do is arguably one of the most complex contemporary tasks for a computer. Even when we take a static view of language it is incredibly difficult, and we know that's not true - it is a changing illusive substance that is influenced by culture, context and a very different ruleset from one language to another.

Building language systems requires gathering facts from texts that can be understood by the computer. One of the tasks that does this is the *semantic relation extraction and classification* task. Examples of such systems are question answering machines, chatbots and information extraction. Semantic relation classification is the task of identifying semantic relations between words in a sentence.

The modern approach to semantic relation classification is to use *neural networks*, which is a machine learning model that has been the root of a surge of good results for several language understanding tasks the last 15-20 years. Semantic relation classification is one of these tasks. The neural networks are attractive because of their simple core concepts, but applications quickly become very complex to manage and understand because of their layered nature.

It is the main goal of this thesis to provide insight into the modern approach to solving semantic relation classification problem with neural networks. The focus will particularly be on modern implementations of *convolutional neural networks*, which is a network structure that have been especially popular for semantic relation classification. It is also the goal to investigate if these existing implementations can be simplified, partly by implementing them with a high-level framework, and partly by experimentally remove components which might not necessary. If components can be removed, it also leaves room for potentially enhancing the model by introducing a few simple techniques.

One of the key goals of a relation classification is that it should be robust - because the structure of language and sentences is extremely varied. It is therefore appropriate to test if the network can be improved by the inclusion of data from the wild. The datasets which we use for relation classification is all-important for how the task is defined and how well we can solve it. The final goal of this thesis is therefore also to analyze and discuss the most important dataset for semantic relation classification, the SemEval 2010 task 8 (Hendrickx et al., 2009). The thesis intends to accomplish these goals by the following list of objectives:

1 To provide anyone with an understanding of computer programming and basic math a starting point for solving semantic relation classification with convolutional neural networks in a supervised and basic semi-supervised setting.

2 To provide an insight into how recent work on semantic relation classification is done, with particular emphasis on neural networks.

3 To implement, detail and provide an open-source implementation of a baseline convolutional neural network for relation classification.

4 To investigate if the baseline model can be simplified without loss of performance.

5 To investigate if the baseline methods can be enhanced with simple, general learning concepts.

6 To test the robustness of the baseline, trained on SemEval data, by using the *self-training* approach for gathering new training data and testing it on the SemEval data.

7 To discuss the definition and execution of semantic relation classification as a task with a particular focus on using the SemEval dataset as the only data resource.

The thesis is structured by these objectives: the preliminaries is as a basic introduction to semantic relation classification, supervised and semi-supervised machine-learning with self-training, neural networks and convolutional neural networks. The related works provides an overview of recent work on relation classification with traditional feature-based methods and neural networks. The research chapter is split into three parts. The first details the structure of the baseline model: how the model is constructed, which components to remove and which components to add. The second section presents the semi-supervised approach to getting more data while simultaneously testing the model robustness. Finally, an analysis and discussion of the SemEval dataset and the semantic relation classification task is presented.

Chapter 2

Preliminaries

This chapter provides the background information necessary to understand the semantic relation classification task and how to solve it. It is the intention of this chapter to accomplish objective 1. We begin the chapter by defining semantic relation classification. Solving semantic relation classification effectively and accurately requires using *machine learning*, and we shall specifically focus on machine learning with *neural networks*, which are presented in this section. A sub-class of neural networks are the *convolutional neural networks*, which have shown promising results for semantic relation classification. Both the general neural network and the convolutional neural network will be covered in this chapter. The baseline model used in chapter 4 is a convolutional neural network.

2.1 Definitions and Dataset

We begin the definition of semantic relation classification as a task by placing it in the necessary context. *Natural Language Processing* (NLP) is a research area which combines linguistics and computer science with the goal of understanding human language in various forms such as spoken language, written language and recorded audio.

Understanding all human language in one holistic system is a daunting task, which is why many research areas in NLP are focuses instead on creating models for different subcomponents of language. These subcomponents, or *tasks*, operate on different levels of abstractions and input types. For example, *sentence classification tasks* are systems that take in individual sentences as input and tries to output a particular label of meaning for that sentence.

2.1.1 Semantic Relation Classification

Sentence classification tasks have the end goal of extracting a form of knowledge from the sentence by placing a label on it. The *semantic relation* between pairs of words is a form of knowledge we can extract from a sentence. The semantic relation stands opposed to syntactic relations, which models how words are structurally and gramatically related in the sentence. The semantic relation is a high level of *meaning* we would like to understand between a pair of words. We illustrate a semantic relation with an example:

(1) *She soon had a stable$_{e_1}$ of her own rescued hounds$_{e_2}$.*

Here, the words of interest are marked with e_1 and e_2. One label we could put on the relation between the marked words is the `Member-Collection(e2,e1)` label. The word *stable* can mean a physical building which houses animals, but it can also mean a set of animals which are grouped together as a concept. The `Member-Collection` label conveys a specific abstract meaning for *stable* because of the relation to the other word in the sentence.

In the *semantic relation classification* task (RC), the goal of the task is to output a label which describes the semantic relation between two words.

Notice that in this example, the marked words are common nominals. These marked words, or *entities*, can be any pair of words (or even groups of words), but different variants of RC place different constraints on the form and position of the words which are then eligible for classification. In this thesis, entity words are constrained to be common nominals because of the SemEval dataset discussed in subsection 2.1.2.

The label for the above sentence might be different depending on who defines the task, and what the labels might be used for. Usually, the labels are designed by linguists to represent general classes of semantic meaning (Girju et al., 2007; Hendrickx et al., 2009). The available labels for a semantic relation classification task forms the *inventory*. In the example, The difference between general relation classification and semantic relation classification in academic literature is not well defined. In this thesis, the term "relation classification" means semantic relation classification unless specified otherwise.

We can define relation classification more precisely:
Let L be the sentence vocabulary for a language and S any sentence where $S \in L$. In this thesis L will always be English. The input x for a RC task is any triple (S, e_1, e_2) where e_1, e_2 are entities $e_i \in S$. The input space is $\mathcal{X}$ where $x \in \mathcal{X}$. Each $x \in \mathcal{X}$ has a corresponding semantic relation $y \in \mathcal{Y}$, where $\mathcal{Y}$ is the inventory for the task. The goal of RC is to find the "true" mapping $h* : \mathcal{X} \mapsto \mathcal{Y}$ for all $x \in \mathcal{X}$. Unfortunately, because of the inherent fluidity of natural language, it is probably impossible to even define $h*$, so instead we search for a $h \approx h*$, which maximizes the number of x for which $h(x) = h*(x)$. Generally, the requirement for h is that $h(x_i) = y_i$ for a set of samples (x_i, y_i) which a human annotator has produced.

Closely related to RC is the *relation extraction* task (RE). The actual difference between the two tasks is not well-defined in the litterature. Here, the difference is the following: The relation extraction task includes an open label `No-Relation` which, x, define *if* a (semantic) relation exists between e_1 and e_2. For most word pairs in a sentence, an RE function should output `No-Relation`. For e_1 and e_2 in the example above, an RE function outputs the same relation as RC. Datasets which investigate relation extraction are usually unbalanced in the number of samples for each labels, because the majority of words in a sentence have no semantic relation. An example of this is the ACE 2005 dataset (Walker et al., 2005).

It is important to notice that RC assumes that entities are always correctly identified, and a semantic relation exists between them. To achieve this, we usually need a human annotator to *pre-select* sentences for which these assumptions can hold. We could include the entity detection step to be part of the RC task, which changes it to *end-to-end relation classification*. In end-to-end classification, $x = S$ and y contains e_1 and e_2 along with the semantic label.

The RC task is heavily influenced by the datasets which define it. There might be might be several different definitions in the academic literature which does not not agree with the one provided here.

2.1.2 SemEval 2010 Task 8 Dataset

Perhaps the biggest problem with RC is the lack of datasets that are high-quality and open-source. We need datasets that possess these qualities so we can develop models and measure their performance and robustness under the assumption that the models perform well in other circumstances. Semantic relation inventories are also hard to construct, and the relations which are interesting to extract can be very specific to the context in which it is extracted. Inventories are usually constructed for a specific target dataset (Culotta et al., 2006; Gurulingappa et al., 2012). And because of the immense variety of language, we generally need a large amount of samples are labeled before the dataset is useful.

The most commonly used dataset for semantic relation classification is the SemEval 2010 task 8 (Hendrickx et al., 2009) dataset. The dataset was produced for the *Semantic*

Evaluation workshop in 2010. The data set fits the definition from the previous section well, because entities in the training set are always correctly identified and possess a semantic relation. There is no `No-Relation` label in the SemEval inventory. The inventory is *exclusive*, which means that any input triple may only have one relation label. An example of a relation in SemEval is the `Cause-Effect` relation, which means: *"When an event or object leads to an effect "* (Hendrickx et al., 2009). The inventory contains 9 distinct relation classes plus the `Other` label which is loosely defined as: *"any semantic relation that cannot be annotated as one of the other 9"*. Each label also have directionality between entities so that, for example, `Cause-Effect` becomes `Cause-Effect(e1,e2)` and `Cause-Effect(e2,e1)`. directionality is applied to all classes except the `Other` label. The total amount of labels 19. The distribution of the inventory is shown in Figure 2.1.2 (excluding directionality).

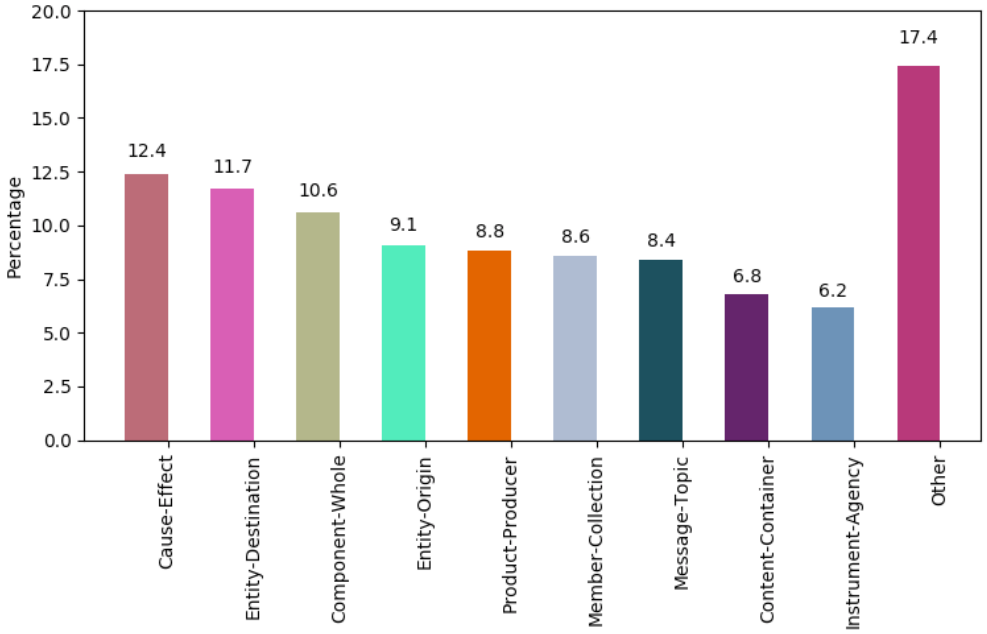


Fig. 2.1
Distribution of relation classes in the SemEval training set.

If we take the directionality into account in the distribution, the distribution becomes extremely skewed. For example, the `Instrument-Agency(e2,e1)` relation only has a single sample in the entire training set.

The dataset is divided in a fixed train and test set. The split is determined by the original creators of the dataset to create a fair evaluation for different classifiers who tried the dataset at the time of the 2010 workshop. The tradition of using the fixed split has since carried on in recent work (chapter 3). The training and test sets are of size 8000 and 2717, respectively, which brings the total amount of samples to 10717.

Official Scoring

An official scoring tool is available for the dataset[1]. The intention of the scoring tool was to allow teams from the original workshop to compare their classifiers on the official test set. Using the scoring tool on the fixed test split is also a trend that has continued for RC studies. The official scorer uses the $F1_{macro}$ metric defined in subsection 2.2.2. The

[1] `https://github.com/vzhong/semeval/tree/master/dataset/SemEval2010_task8_scorer-v1.2`

directionality of relations is ignored when calculating precision and recall for each label, but included when determining if a sample is correctly classified. The F1 score for the `Other` label is also excluded from the $F1_{macro}$.

Choosing this metric for evaluation of RC carries significant weight. $F1_{macro}$ represents a subjective weighting that all relation classes are equally important to detect, no matter the frequency with which they truly appear.

Annotation

It is important to notice that the SemEval dataset is - in the academic community - *the* dataset for semantic relation classification. It is an important advantage of the dataset that it is freely available, and such datasets are hard to come by. chapter 3 shows several examples of studies that primarily use this dataset for benchmarking their models. For this reason it is necessary to describe the process of how samples in the dataset were gathered and annotated. Candidate sentences for the dataset were collected through a pattern-based search Web search. From these searches, the annotators selected samples that were suitable for the dataset by filtering out sentences from a list of criterias:

- Entities must be simple common nouns. For example, *the engine of a lawn mower* is a complex noun phrase which cannot be used as an entity. The simple noun phrase *lawn mower* can be used.
- Entities must exist in the same sentential clause. This limits the distance between entities to be relatively short and sentences to be simple.
- Annotaters must agree on the label of the sentence. When agreement for a sentence-label pair cannot be reached, the sentence is removed from the dataset.

The goal of the annotation process was to use 90% of the results from the pattern-based searches. The real result turned out to be closer to 75% (Hendrickx et al., 2009). In other words, one out of four samples that were preselected by a pattern search were filtered out because they were too difficult to annotate. We might reasonably expect that this can affect how a relation classifier trained on this data reacts to new, unfiltered data. These issues are investigated in section 4.4.

Alternatives

There are other datasets available for relation classification, but all of them suffer from issues which arguably makes them less suited for RC. An example is the *ACE 2005* dataset (Walker et al., 2005). ACE is an annotated dataset which primarily consists of transcribed text. The distribution of `Relation` and `No-Relation` samples are greatly skewed , which makes it more suited for the relation extraction task. ACE is also not freely available. In other languages and domains we can find datasets. In medical studies, sentences that describe protein interactions is a popular target for RC, but the application is very specific and hard to transfer to other domains. For a comprehensive listing of semantic relationship datasets for different languages and domains, see Batista (2017).

We briefly summarize. In order to solve RC, we are required find a function h which organizes sentences according to the inventory. To find it, we can use the SemEval data which already contains labels for a set of sentences, which means we can learn from it.

2.2 Machine Learning

Modern solutions to the RC problem is based on *machine learning*. Machine learning allow computers to solve problems by learning rules and patterns from data. The historical alternative to machine learning is to use hand-written rule-based systems called *grammars*. Hand-written rules for languages are notoriously hard to engineer and prone to errors. A semantic relation can be constructed with a wide variety of words and sentence structures,

and words have different meaning depending on where in the sentence they appear. With machine learning, we present a generic algorithm with a set of data that contains information from which we wish to generalize to new data. By this approach, the patterns which emerge from the data have a statistical root in the data from which they are learnt and consequently more robust. In this section I will describe different concepts in machine learning and relate it to the RC task. This section defines the types of machine learning which will be used to solve RC with the SemEval dataset. We will examine the basic definition of classification problems and how to measure the performance of h.

2.2.1 Types of learning

We can define the RC problem as a learning problem. Let $h* : X \mapsto Y$ be the correct semantic mapping function for all sentences. Because $h*$ is unknown we have to learn an approximate function $h(x) \approx h*(x)$ instead from a set of training points D_{train}. The mapping h provides the semantic relation and is learned from a set of sentences which contains the semantic relations.

The information given in D_{train} defines the setting of the machine learning task. Let D_{train} be a set of points $X = (x_0 \dots x_n)$. If all labels $\mathcal{Y} = (x_0 \dots y_n)$ are known, we call the setting *supervised learning*. This is the most common approach in machine learning. The objective of h is easy to define because the output space is defined by $\mathcal{Y}$. If we have no labels for X the algorithm must infer the output space of h by itself. We call this approach *unsupervised learning*. We may also have the situation where X is in two parts: $X = (x_0 \dots x_i)$ for which we know the labels $Y = (y_0 \dots y_i)$ and the unlabeled part $X = (x_i + 1 \dots x_n)$ for which we have no labels. We can still use the unlabeled data points, but we have to make certain assumptions. We call this setting *semi-supervised learning*.

Supervised Relation Classification

With the above definitions, we can now the RC task with the SemEval dataset as a *supervised classification problem*. It is supervised because D_{train} is labeled and $\mathcal{Y}$ consists of discrete labels for the relations which are known for all the samples. We want to automatically infer patterns in the data which allows as many samples as possible to be classified correctly.

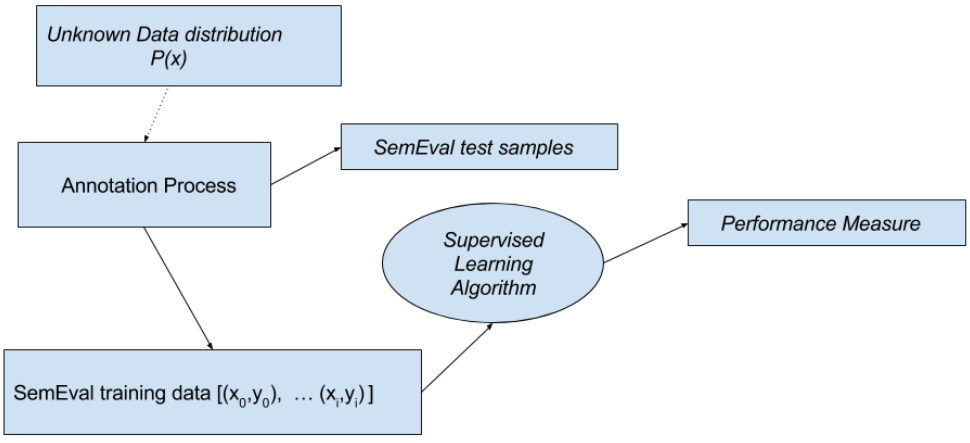


Fig. 2.2
The basic supervised setting with the SemEval dataset and relation classification.

Semi-Supervised Assumptions

We discussed in subsection 2.1.2 how quality labeled data is hard to obtain. Raw unstructured text, however, is in abundance on the Internet. This data is *unlabeled* because it has not been annotated yet. It is worth investigating if we can leverage the massive amount of unlabeled data available on the Internet to improve an otherwise purely supervised approach. The intended effect of the unlabeled data is, in some way, to inform the learning algorithm about the labeled data. The immediate problem is then the relation between the unlabeled and labeled data. Intuitively, it is not a good idea to use images as unlabeled data when trying to classify sentences. Consequently, we have to make an assumption about the distribution of our data before we can reasonably expect the semi-supervised approach to work. The assumption is that the unlabeled ddata must provide information about the distribution of the labeled data. Ideally, we would like the process that generates the data to be the same for the labeled and unlabeled data. The assumption is then trivially met. With datasets such as the SemEval, the situation is different. The entity sample selection shifts the distribution of labeled data away from any other raw text data we can gather. We will examine this problem in section 4.5.

Assumptions about the labeled data is not exclusive to semi-supervised approaches. In a purely supervised learning, we also have to rely on the *smoothness assumption*: if two labeled samples x_1 and x_2 are close in the input space, their labels y_1 and y_2 should be the same with high probability. A generalization of this assumption is used in the *semi-supervised smoothness assumption*. If a labeled sample (x_1, y_1) and an unlabeled sample (x_2) are close in the input space and in a high-density region, the latent class (y_2) should be the same as (y_1) (Chapelle et al., 2009).

In other words, since the labeled unlabeled data share the same input space, the classes for the unlabeled data should look like the labeled. An important consequence of this assumption is that if an unlabeled example "looks like" a labeled example but actually have a different label, using the unlabeled data might be worse than just using the supervised approach by itself.

The most common semi-supervised approach for NLP is the *word embeddings*. In fact, they are so fundamental in recent RC applications that they are usually not defined as an semi-supervised approach. For clarity, they are omitted from this section and presented in section 2.4 instead, which is closer to the application of neural networks to RC.

Self-Training

Under the semi-supervised smoothness assumption we can leverage unlabeled data by applying the *self-training* algorithm (Abney, 2007, 2). Self-training is a basic form of semi-supervised learning which follows a simple idea: use a classifier trained purely supervised to predict the labels of new data and use this newly predicted data as training data for a new classifier. Like other semi-supervised approaches, self-training hinges on crucial assumptions:

- Self-training relies on the distribution of the unlabeled data being similar or equal to the labeled data. For the SemEval dataset, where samples are pre-selected and filtered, this might not be the case.
- Even if these two distributions are equal, the way the newly trained classifier samples the unlabeled data will also shift the distribution.
- Taking new samples in as training data requires that the classifier is sufficiently confident in the samples being correct. Assuming that our classifier h estimates the distribution of the unlabeled data $P(D_{new})$ with an error less than some rate ϵ. The function $select(D_{all})$ of h chooses newly labeled data from D_{all} which are added to D_{train}. In order for self-training to work, *select* can only take in samples that are labeled with a confidence higher than $1 - \epsilon$. Unfortunately, ϵ is unknown to us and must be tuned experimentally.

- While we can use virtually any classifier h for self-training, h must output confidence values for predictions on new samples. Luckily, subsection 2.3.2 will show that we can design neural networks that output the conditional probability $P(y|x)$ of sample x belonging to a class y.

Self-training has seen usage in NLP and its simplicity warrants using it for investigating the possibility of using unlabeled data with the SemEval dataset (Kate and Mooney, 2007; McClosky et al., 2006). The pseudo-code for self-training is shown in algorithm 1.

Result: A classifier h
$h \leftarrow \text{train}(D_{train})$
while *stopping criterion not met* **do**
  $D_{train} \leftarrow D_{train} + \text{select}(\text{label}(D_{new}, h))$
  $h \leftarrow \text{train}(D_{train})$
end
return h

Algorithm 1: Basic self-training

The stopping criterion and selection process are two *hyperparameters* of the algorithm which allows for some variation. A hyperparameter is a setting for a learning algorithm which cannot be learned from the data, but must be chosen as a design choice by the developer of the learner.

The selection process typically includes *throttling, balancing and thresholding*. Throttling limits the number of new samples that can be taken in at every iteration while balancing limits the number of samples for each class label based on some heuristic. Here, the heuristic will be the distribution of labels in the training set. The stopping criterion is usually a lower bound on samples that can be added to the training set in an iteration. These parameters are discussed in the experiments in section 4.4. Finally, the threshold value is chosen as an approximation of ϵ. While we can measure ϵ on the training set, we cannot know how h performs on D_{new}. Therefore, we must experiment with different threshold values and see which value is restrictive enough to not corrupt the classifier with error-ridden samples, while still taking in new data from D_{new}.

2.2.2 Metrics

To evaluate the performance of a RC function h produced either purely supervise or with self-training, we apply the function on a set of data $D_{test} = [x_0, x_1 \dots x_n]$ for which the labels $h^*(D_{test}) = [y_1 \dots y_n]$ are known. Since the labels of the relations are discrete and exclusive, x_i is correct if $h(x_i) = y_i$. It is also useful to measure what label is predicted instead of the correct label when the classifier is wrong. By observing the predictions for all samples we can construct a *confusion matrix* which shows predictions on one axis and the actual labels on the other. An example of a confusion matrix for the SemEval training set is shown in Table 2.1.

The diagonals in the confusion matrix are correct predictions, and every other placement is wrong. From the positions in the confusion matrix predictions for each class Y_i are measured and put into four categories:

- A *true positive* (tp) is a sample $x_i \in D_{test}$ where $h^*(x_i) = Y_i$ and $h(x_i) = Y_i$.
- If $h^*(x_i) \neq Y_i$ and $h(x_i) = Y_i$, x_i is a *false positive* (fp).
- Conversely, if $h^*(x_i) \neq Y_i$ and $h(x_i) \neq Y_i$, x_i is a *true negative* (tn).
- Finally, if $h^*(x_i) = Y_i$ and $h(x_i) \neq Y_i$ x_i is a *false negative* (fn).

	C-E	*C-W*	*C-C*	*E-D*	*E-O*	*I-A*	*M-C*	*M-T*	*P-P*	*O*	**SUM**
C-E	285	1	0	0	6	0	0	0	2	34	328
C-W	0	233	3	1	0	11	9	3	0	52	312
C-C	0	1	172	3	1	0	0	0	0	15	192
E-D	0	2	12	240	0	0	0	1	0	37	292
E-O	3	4	2	2	201	2	0	1	1	42	258
I-A	0	5	1	2	0	92	0	0	3	53	156
M-C	0	4	0	1	1	1	157	1	0	68	233
M-T	0	1	0	1	1	0	1	203	0	54	261
P-P	5	6	1	1	4	6	0	1	141	66	231
O	10	18	25	18	21	15	17	21	15	294	454
SUM	303	275	216	269	235	127	184	231	162	715	2717

Table 2.1
Example confusion matrix for the SemEval test set

F1 Measure

The standard notion of *accuracy* for a multi-class problem is defined by the number of true positives:

$$\frac{\sum_i^C tp(Y_i)}{n} \tag{2.1}$$

where C the number of labels in $\mathcal{Y}$. This might not be a useful measure when the distributions of labels is imbalanced, because it does not take the distribution of classes into account. If a dataset contained 2 labels where one of them had a 99% probability of occuring, a classifier could trivially achieve 99% accuracy by labeling all samples as that class.
Instead, it would be useful to describe how well the classifier performs on each individual class label in the dataset, which can then tell us how "balanced" the classifier is. *Precision* (P) and *recall* (R) is two examples of metrics that replaces general accuracy. Precision is defined as $\frac{tp}{tp+fp}$ and shows how confident a classifier is with predictions for a certain class Y_i. A high precision value for Y_i means that few samples which were labeled as Y_i were not actually Y_i. The opposite value of precision is recall, which is defined as $\frac{tp}{tp+fn}$ and shows how well a classifier finds all samples which belongs to Y_i. High recall means that very few Y_i samples are not found and put in Y_i. Precision and recall are ends of a spectrum - a classifier can achieve maximum precision for a label by never predicting that label, but at the cost of recall. Likewise recall can trivially be obtained for Y_i by labeling all samples as Y_i. By balancing them together we obtain the $F1 = \frac{2*p*r}{p+r}$. Example precision, recall and F1 values from the previous confusion matrix are shown in Table 2.2.

The full F1 score for a dataset summarizes the performance. This score comes in two variants which is determined by how individual F1 values for each class are averaged. The simple average over all F1's obtains *macro* F1:

$$F1_{macro} = \frac{\sum_i^C \frac{2*p_{Y_i}*r_{Y_i}}{p_{Y_i}+r_{Y_i}}}{C}. \tag{2.2}$$

Alternatively, we can sum the individual precision and recall values of each class:

$$p_{micro} = \frac{\sum_i^C tp_{Y_i}}{\sum_i^C tp_{Y_i} + \sum_i^C fp_{Y_i}} \qquad r_{micro} = \frac{\sum_i^C tp_{Y_i}}{\sum_i^C tp_{Y_i} + \sum_i^C fn_{Y_i}} \tag{2.3}$$

Label	Precision	Recall	F1
Cause-Effect	283 /(301 + 2) = 93.40%	238/328 = 86.28%	89.70%
Component-Whole	227/(269 + 6) = 82.55%	227/312 = 72.76%	77.34%
Content-Container	170/(214 + 2) = 78.70%	170/192 = 88.54%	83.33%
Entity-Destination	239/(268 + 1) = 88.85%	239/292 = 81.85%	85.20%
Entity-Origin	201/(235 + 0) = 85.53%	201/258 = 77.91%	81.54%
Instrument-Agency	91 /(126 + 1) = 71.65%	091/156 = 58.33%	64.31%
Member-Collection	156/(183 + 1) = 84.78%	156/233 = 66.95%	74.82%
Message-Topic	200/(228 + 3) = 86.58%	200/261 = 76.63%	81.30%
Product-Producer	140/(161 + 1) = 86.42%	140/231 = 60.61%	71.25%
Other	294/(715 + 0) = 41.12%	294/254 = 86.28%	64.76%

Table 2.2
Example F1 macro calculations for the SemEval test set.

and then obtain the *micro* F1:

$$F1_{micro} = \frac{2 * p_{micro} * r_{micro}}{p_{micro} + r_{micro}} \tag{2.4}$$

The $F1_{micro}$ places less importance on classes with low frequency, where the macro considers each class equally important. Which metric to pick depends on the intended performance of the classifier. For example, if we do RE instead of RC, the semantic relations will have a low frequency compared to the `No-Relation` label, but they are probably more important to classify correctly.

2.2.3 Validation

The section above described that the F1 metric is applied to a dataset D_{test}. Choosing D_{test} correctly is important for getting a good estimate of how well h will perform on unseen data. Unfortunately, just measuring how well h is doing on D_{train} does not guarantee that it will do generalize well.

Overfitting

Learned models are subject to *overfitting*. Overfitting happens when the difference between the training performance and general performance of a model is high, particularly when the training performance is high but the general performance is not. The usual problem is that we cannot know the general performance of our model. Why does overfitting occur? Here are the main reasons:

- *Stochastic Noise*: Real data is noisy. Text have grammar and spelling errors, and annotators who label the datasets might make mistakes. These types of errors introduce *stochastic* noise to the training data on which we learn h. Encoding errors are another common cause of errors for text. This stochastic noise cannot be reasonably approximated by a function, exactly because it is stochastic. Unfortunately, we cannot expect h to recognize the stochastic noise and avoid learning from it. A more complex model have more parameters to fit the stochastic noise, which then leads to overfitting.
- *Deterministic Noise*: Because h is ultimately an approximation of $h*$, there might be some data D in $\mathcal{X}$ which cannot be modeled correctly with the complexity of h, because the part of $h*$ that describes D is outside the approximation. D is not stochastic noise but the same issues apply. Even though the data cannot be modeled

correctly, a complex function can still use its parameters to model this data, which is then overfitting. Because this problem is a consequence of the choice of h, we call this *deterministic noise* (Abu-Mostafa et al., 2012).

- *Target complexity*: The function h is learned only from the data. Due to noise, however, the training data can mislead h towards a more complex function because it fits the training data better. The more complex function then produces better results on D_{train}, but loses some of the generality needed for accurately predicting new data. In other words, when the *capacity* of a model that finds h is high, the risk of overfitting increases because the model have more ways of choosing incorrectly (Goodfellow et al., 2016, p. 107). For the neural networks detailed in section 2.3, the capacity is usually huge because the network have many layers and parameters.

Choosing the right capacity/complexity for h is hard - even the detailed definition of capacity as a measurable quantity is beyond the scope of this presentation. Instead, to prevent overfitting, we can apply a series of techniques that dynamically reduces the capacity of h. These techniques are called *regularization* and are detailed for neural networks in subsection 2.3.3.

Validation Set

The problem with choosing a model based on its performance on D_{train} is that the estimate is not reliable. If the model we have produced is overfitted, we can expect it to report a very good performance, but since D_{train} is only a sample of the entire space $\mathcal{X}$, h may not produce the same results on a new D_{test}. So to obtain a more reliable estimate, we must measure h on a dataset which has not been part of D_{train}. We create a test set D_{test} from D_{all} which must not be used *at all* to select h. For some datasets, the test set will be a preselected split. This is the case for the SemEval dataset which will be used in chapter 4. This also means that we cannot use D_{test} in a process where we select a model, run the training algorithm, estimate the error, then adjust the algorithm and repeat the process. By doing so we are still implicitly overfitting on the model because we are, however indirectly, letting specific information about the sample from $\mathcal{X}$ (D_{test}) leak into h. If we want to estimate the effectiveness of h during the learning process, we split the remaining data $D_{all} - D_{test}$ into D_{train} and D_{val}.

We can now use our metrics on D_{val} to estimate how well h will perform on D_{test} without ever actually touching D_{test}.

Cross-Validation

Using D_{val} as an intermediate estimate poses another problem: now h is now longer trained on all the available data and we might lose some generalization power. The idea behind leaving D_{val} out in the first place is to obtain a reliable estimate of performance when designing the model that learns h. It's a bit of a dilemma - we want a lot of training data to get a good generalization power from our model, but a lot of samples in our validation set to reduce the variance of the estimate. In domains where the amount of labeled data is small (NLP), particular attention must be given to balance these conflicting estimates. For example, the SemEval dataset has 8.000 training samples - in comparison the canonical MNIST dataset for digit recognition has 10.000 samples for testing alone(LeCun and Cortes, 2010). We can reduce the variance empirically for our estimated performance by using *cross-validation*. With cross-validation, D_{all} is partitioned a number of times into different training and validation splits. In the extreme case, given N samples, the set is divided N times and the validation is a set of size $K = 1$ for which the performance is then estimated. The cross-validation estimate is then the average of all N estimations. This approach quickly becomes very computationally expensive, so in practice D_{all} is split into K exclusive *folds* of size N/K. Each fold is used as a validation set once, while the other $K - 1$ folds are used for training. The resulting technique is called

k-fold cross-validation. In practical applications a good value for K is 5 or 10 (Abu-Mostafa et al., 2012, 4.3).

One convenient usage of cross-validation is that we can use it to select specific settings of h that achieves the best performance without examining D_{test}, since it provides a good estimate of how h will perform on new data.

2.3 Deep Feedforward Neural Networks

Learning a relation classifier can be done with many different classification algorithms. The focus of this thesis is the *neural network*, which have been used extensively in recent years with significant performance improvements in several research areas including NLP. Recent work on neural networks for NLP, and specifically RC, is covered in chapter 3. This section covers the definition of a deep feedforward neural network and how it is trained to approximate h through derivatives. The deep feedforward neural network is the foundation for convolutional neural networks, which is covered in section 2.5. Choosing the correct structure of the network that approximates h is a problem that often leads to the concept of *overfitting* (Goodfellow et al., 2016, sec. 5.2). The final part of this section covers a group of *regularization* techniques that constrain the model capacity and reduce overfitting.

2.3.1 Definition

A deep feedforward neural network is a datastructure which can be used to learn and represent a function $h : X \mapsto Y$ from a set of training data D_{train}. Usually, the term neural network also encompasses the algorithms which are used to build the datastructure and approximate the function h. A common way to describe neural networks is to describe each word and why it is used:

The word *network* is used because the datastructure represents $h(x) = y$ as a series of simpler functions which are chained together to form a network. For example, we might construct a simple h with three layer functions $h(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. The first layer $f^{(1)}(x) \wedge x \in \mathcal{X}$ is called the *input layer* and the last layer $f^{(3)} = y \wedge y \in \mathcal{Y}$ is the *output layer*. The layers in between the input and output layers are called *hidden layers*. In the example, there is a single hidden layer $f^{(2)}$. The number of layers (here 3) is called the *depth* of the network and is denoted d. The layers are called hidden because they are not defined by the training data and must be learned by the individual network (Goodfellow et al., 2016, 6.1).

The word *feedforward* is used because the network has no connections that leads back to previous layers. The information flows in one direction through the network when it is activated. A network where the output of an intermediate layer can flow back to a previous layer is called a *recurrent neural net*. Recurrent nets and their usage for RC is detailed briefly in subsection 3.1.5.

The word *deep* is used when the number of hidden layers are more than zero. A network with no hidden layers - so that the input layer is directly connected to the output layer - is commonly called a *perceptron*. Why neural networks works differently and more efficiently when they have many hidden layers is an open research question. It has been shown that a network with a single hidden layer can be used to approximate any function (Hornik, 1991). But it has also been shown that increasing the depth of a network greatly reduces the number of neurons needed in the network, compared to a network with single hidden layer (Delalleau and Bengio, 2011).

Finally, the word *neural* is used because the components of a layer in the network is loosely modeled after models of biological brains, as we shall see in the following section. In this presentation, the analogy to the brain is not important. The reason for the analogy here is that it provides a convenient naming scheme for the components in the network.

Consider the l'th layer in a network. The output of this layer is the application of $f^{(l)}(x^{(l)})$, which is also the input for the next layer x^{l+1}:

$$x^{(l+1)} = f^{(l)}(x_l) \tag{2.5}$$

In this notation, x is a vector and $f^{(l)}$ is a vector-to-vector function. We can also deconstruct $x^{(l+1)}$ to its individual components:

$$x^{(l+1)} = [f_0^{(l)}; f_1^{(l)} \dots; f_n^{(l)}]^T \tag{2.6}$$

Each component $f_i^{(l)}$ is a vector-to-scalar function that can be applied in parallel on the input vector. These individual components are called *neurons* or *units*. The length $n_l = |f(l)|$ is the *width* and is also the number of neurons in the layer. For clarity we denote the width of the previous layer as $m = n_{l-1} = |x|$. The input for each neuron is determined by how much information flows from a neuron in the previous layer $f_i^{(l-1)}$. The edges that connect the neurons are called *synapses*. The strength of the synapses are controlled by real-valued *weights* $[w_{i,0}^{(l)}, w_{i,1}^{(l)} \dots w_{i,m}^{(l)}]$, which determines how much information flows from one neuron to another. The incoming weights for a particular neuron sums to form the *activation* of the neuron. The activation $f_i^{(l)}$ is then a linear combination of the inputs determined by the weights:

$$f_i^l = \sum_{j=0}^{m} f_j^{l-1} * w_{ij}^l \tag{2.7}$$

The definitions above show how an individual output neuron is calculated. Together, they form a layer which is calcuated by a matrix constructed from the weight vector of each neuron. With n neurons, the total amount of weights in a layer is $n * m$, and the total output of a layer is then defined by the matrix $W^l \in \mathbb{R}^{n \times m}$ and the input from the previous layer $x^{(l)}$ as a matrix-vector product:

$$f^{(l)} = W^{(l)} * x^{(l)} \tag{2.8}$$

Activation Function and Bias

A neuron in a biological brain contains an *axon*, which is a transmitter that measures a potential in another component called the *soma*. The soma is modeled loosely with (2.7). When the potential in the soma is large enough, the axon delivers a signal to neurons it is connected to. We can model this thresholding effect by introducing a non-linear *activation function*, usually denoted σ, which "fires" when the input is large enough to pass the threshold. Because we want each threshold value to be learned specifically for each neuron, we also add a *bias parameter* b to the activation. In the matrix-vector notation, $b \in \mathbb{R}^n$ is a vector consisting of all the bias parameters for a layer. We can now write the full output of a layer $f^{(l)}$ with the input multiplied by the weights, adjusted by the bias and passed through the activation function:

$$f^{(l)} = \sigma(W^{(l)}x^{(l)} + b) \tag{2.9}$$

where σ and b is applied element-wise to each neuron in the l'th layer.

As we will see in subsection 2.3.2, the derivative of the activation function is important because of the way neural networks learn from data. This restricts activation functions to be differentiable, continuous functions. The traditional activation function is the *logistic sigmoid* function, which is a smooth "S"-shaped function which centers around zero. The derivative is simple and easy to compute. The sigmoid is the traditional activation function but is not centered around zero, which is a problem for gradient-based optimization.(Goodfellow et al., 2016, p. 66). To compensate for this problem, the *hyperbolic tangent* function have been used instead, and is centered around zero. Both

functions suffer from *saturation* which is also a problem for optimization because of *vanishing gradients* (Glorot and Bengio, 2010). A third, modern option is the *rectified linear unit* (ReLU), which does not saturate. All three functions and their derivatives are shown in Figure 2.3.1.

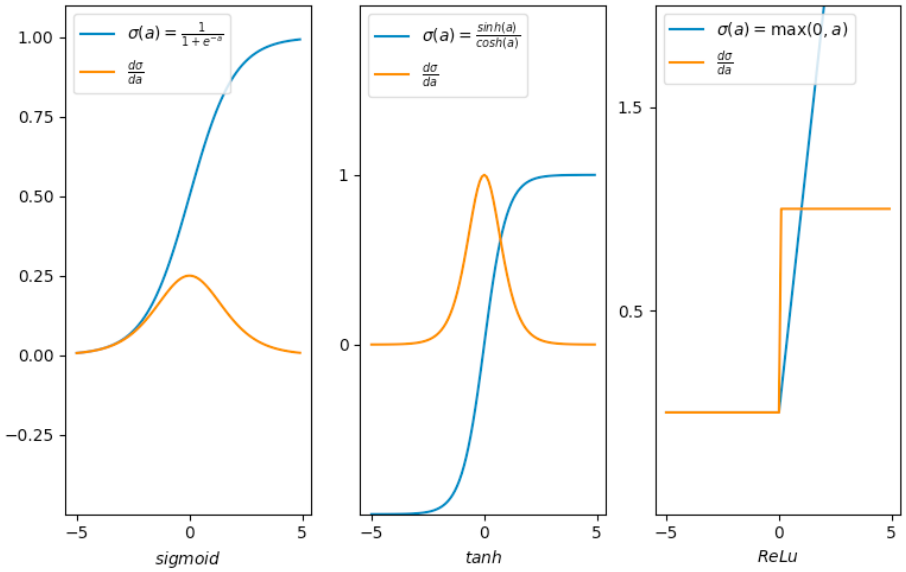


Fig. 2.3

Activation functions and their derivatives. The sigmoid is a smooth function that suffers from saturation issues. Tanh produces a stronger gradient have the same problems. ReLU does not saturate.

Constructing a feedforward network now requires defining the depth, width and activation function. These three attributes together define the *hypothesis set* $\mathcal{H}$ from which we can choose h. Each unique assignment of the weight values in the network produces a $h \in \mathcal{H}$. We use the weights to learn the specific RC function h, but we are constrained in the design of the network architecture. To explicitly describe the configuration of the weights, we parameterize h to include the weights $w = [w^i \dots w^d]$ by writing $h(x, w)$, where x is the input to the first layer in the network.

2.3.2 Optimization

The question remains: how do we use the network $h(x, w)$ to approximate $h : \mathcal{X} \mapsto Y$? To do this, we need to construct a way of measuring of how well $h(x, w)$ predicts the samples in D_{train}. The measures shown in section 2.2.2 cannot reveal any information about *why* the chosen setting of w predicts the labels that it does. We need a general function that depends on w and, when minimized, maximizes the probability of D_{train}. This measure is usually called an *objective function* or *loss function*.

Objective function

Consider a supervised learning problem where we have a dataset D_{train} where we know y for each $x \in D_{train}$ We can construct the output layer $y(x)$ of the network $h(x, w)$ such that each label in $\mathcal{Y}$ has a corresponding output neuron. To find out which y_i we should choose from the network predictions we inspect the activations of the neurons. One way of interpreting the predictions is as a conditional probability distribution over the classes in $\mathcal{Y}$ given x: $h(w, x) = P(Y = y|X = x)$. We achieve this by using an activation function

on the output neurons which transforms the output into a probability distribution. An example of such a function is the *softmax*. Given the output layer described above and classes $y \in C$, the softmax is defined:

$$\sigma(y_i) = \frac{e^{y_i}}{\sum_{j=0}^{C} e^{y_j}} \tag{2.10}$$

Since we know the label for x in advance because x is in D_{train}, we can also construct the "true" probability distribution y over Y by setting the probability for the label to 1 and all other probabilities to 0. This representation of class labels is called a *one-hot encoding*. For a single sample (x_i, y_i) in D_{train} we can measure how well the predicted distribution aligns with the true distribution of D_{train} with the *cross-entropy loss*:

$$Ł(h, x_i, y_i) = -\sum_{j}^{C} h(x_i) * log(y_i) \tag{2.11}$$

The cross-entropy is a concept from information theory which measures the difference in information between two distributions. Depending on the nature of the logarithmic term, we measure a number of *nats* needed to send a message from the distribution from which $h(x_i)$ is drawn, but using the the one-hot encoding of y_i (Goodfellow et al., 2016, 3.13). These units are not important for the machine learning here, but the idea of using this measure is that we can minimize it to make the output of $h(x_i)$ look like the encoding of y_i. For the entire dataset of N training samples D_{train}, we simply average the error:

$$Ł(h, D_{train}) = -\frac{1}{N} \sum_{i}^{N} \sum_{j}^{C} h(x_i) * log(y_i) \tag{2.12}$$

The cross-entropy is equivalent to the *average negative log-likelihood*. Log-likelihood is a statistical estimation principle for producing the best estimator for predicting the samples in D_{train} (Abu-Mostafa et al., 2012). While the loss functions shows the error of h in predicting the samples in D_{train}, we cannot guarantee that h will produce the same results on new samples that is not in D_{train}.

Optimization

With (2.12) we now have a function subject to minimization with differential calculus. Because h is dependent on the weights x we can compute the gradient of Ł with respect to x. We cannot compute an analytical solution to find the minimum of Ł, since the layers of the network output non-linear activations. But we can search for a minima iteratively by applying the *gradient descent* algorithm. The gradient descent algorithm calculates the error for the entire D_{train}, so instead we use *stochastic gradient descent* which uses a small number of samples in D_{train} in a *batch* to approximate the error. And finally, in order to calculate each partial derivative through the network, we use the *backpropagation algorithm* (Nielsen, 2017).

AdaDelta

A modern variant of the gradient descent algorithm is to apply the *AdaDelta* update rule, which adapts the *learning rate* in which the weights change by accumulating a weighted sum of the past gradients. The rule also modifies how gradients are computed by approximating the second-order derivative of Ł with respect to the weights (Zeiler, 2012).

2.3.3 Regularization

In section 2.2.3 it is described how h may overfit on the available training data, particularly when the capacity of h is large. The capacity is a property of the hypothesis space, which

we choose from the activation function, depth and width. The problem is that guidelines for designing the network is mainly empirical not well understood theoretically. This makes controlling the capacity for neural networks a hard problem. What we do instead is encourage that the network to learn a simpler $h \in \mathcal{H}$ by reducing the capacity dynamically. These techniques are called regularization. This section shows three regularization techniques: the *l_2-based weight constraint*, *early stopping* and *dropout*.

l_2-based weight constrainting

Reducing the capacity of h by adding a term to objective loss function which penalizes large weight parameters is a technique that has been used for decades, both for neural networks and simpler regression techniques (Goodfellow et al., 2016, 7.1). For RC, a variant called l_2-based weight constraining is typically used (Hinton et al., 2012). When computing the gradients, a proposed update to a weight vector w might entail that $||w||_2 > s$ for a hyperparameter s. With weight constraining, weights are never allowed to be above s, and the constraint rescales $||w||_2 = s$ by division if it happens. The reasoning for using this variant instead of a norm penalty is that it theoretically allows a large initial learning rate for a training algorithm without risking weights become too large. However, modern update rules adapt the learning rate and RC datasets are not large, which motivates an investigation. We investigate the effects of the weight constraint in section 4.2.

Early Stopping

A non-intrusive way of reducing overfitting is to continually measure the performance of h on a separate validation set, which is not used for training. When the network begins to overfit, a decrease in training error can be observed, while the performance on the validation set starts to degrade. *Early stopping* terminates the learning process when this is observed. It is called early stopping because it terminates earlier than an upper bound of iterations which is usually determined empirically by the developer.

The validation set is partitioned from the training data and has no overlap on the actual test set, which is used at the end of the experiment to estimate the generalization error. A disadvantage of early stopping is that the validation data is not used for training. We can modify early stopping and use the validation data to find the number of training iterations for the reduced training set, and then assume that this number of iterations is a good place to stop for the full training set as well. We then re-train the network on all the available data. This variant of early stopping is equivalent to Goodfellow et al. (2016, Algorithm 7.2).

Early stopping introduces two hyperparameters, the validation set size and *patience*, which determines how many iterations with decreasing validation set performance should be observed before the learning process is terminated. The specific size of the validation set is not of great importance as long as it is large enough to be a sound estimate. Patience can be increased arbitrarily without loss of performance because we always select the best iterations for the validation set from the history of all iterations.

Dropout

Dropout is a simple but powerful regularization technique for neural networks. With dropout, we constrain a network layer by randomly setting the output of neurons in that layer to zero ("dropping out") with a probability p during training. p is a hyperparameter but is usually $p = 0.5$. At test time, no connections are dropped but their weights are multiplied with p. The effect of dropping out units is that a layer with n units can be interpreted as 2^n smaller networks with parameter sharing that functions as an ensemble classifier (Srivastava et al., 2014). As I will show in chapter 3, dropout is universally used with convolutional neural networks and the RC task.

2.4 Word Embeddings

The neural networks described so far only deal with real valued vectors or matrices. Some input types are easy to represent with real values. Images are directly representable as pixels and color channels, but for written text we need an encoding which retains the salient information in characters and words. A *language model* is a probabilistic model which captures the statistical characteristics of sequences of words, which in turn allow the model to predict new words in a sentence given previous ones. Probabilistic predictions are numbers which can be input to a neural network. The traditional approach to language modeling is the *N-gram model*, which convert each word into a probability of the word occuring given a previous sequence of $N-1$ words. This approach suffers from the *curse of dimensionality* (Jurafsky and Martin, 2000, 4.).

A major advance in NLP was to learn a better representation for words, the *distributed representation* (Bengio et al., 2003). Distributed representations, or *word embeddings* can be learned efficiently with a neural network on huge amounts of unlabeled data, partly because of advances in parallelized computations on GPUs. One such neural network which is used to learn word embeddings is the *word2vec* network and algorithm (Mikolov et al., 2013). The details of the algorithm is not relevant for this thesis, but the main idea is: raw text can be used to let a network learn to predict words given their context. The representation which a neural network learns internally to make these predictions is a many-dimensional representation of the word which retains semantic information about the word, since semantically similar words tend to appear in similar contexts. Word embeddings are an example of using neural networks for *representation learning*.

Word embeddings are commonly used in NLP tasks by training a word embedding on huge amounts of unlabeled data, extracting the embedding, inserting them into a new neural network and then fine-tuning them for the specific task. If the specific task is a supervised learning problem, using word-embeddings like this is actually a semi-supervised approach. The most common way to describe this approach is by the term *layer-wise pre-training*, where only the embedding layer is pre-trained (Goodfellow et al., 2016, 15.). The word embedding layer for the specific task can also be initialized randomly and trained exclusively for the specific task.

We shall create a notation for a distributed representation of a sentence which is converted to a distributed representation so we can refer to it later. Let $T = [t_1, t_2 \dots t_n]$ be a sentence of words of length N, and let E be a word embedding matrix $E \in \mathbb{R}^{w^d \times l}$, where w^d is the dimension of the word representation and l is the number of words in the vocabulary for T. E can be indexed into by t_i by some temporary encoding of t_i (for example a hash) and produces a word-embedding $e_i \in R^{w^d}$. Now we can produce the sentence matrix $S \in R^{w^d \times N}$ from T by looking up all words in T and using them as columns in S:

$$S = [e_1, e_2 \dots e_n] \tag{2.13}$$

2.4.1 Positional Embeddings

Sometimes we would like S to use other embeddings for T, which are trained in different ways and encode different information. A popular auxiliary embedding for relation classification is the *positional embedding* (Zeng et al., 2014). The positional embedding d_i calculates the distance from each word t_i to a marked entity and embeds it in a vector $d_i \in \mathbb{R}^{p^d}$, where p^d is the embedding dimension (usually around 50). We can still produce S by concatenating each embedding together. For example, for the word t_i, we can produce the full word representation s_i with 2 positional embeddings d_{i1}, d_{i2} and a word vector e_i:

$$s_i = [e_i, d_{i1}, d_{i2}]^\intercal \tag{2.14}$$

And finally create $S \in \mathbb{R}^{(w^d+2*p^d)\times N}$ with s_i instead of e_i:

$$S = [s_1, s_2 \dots s_n] \tag{2.15}$$

2.5 Convolutional Neural Networks

This section presents a general introduction to Convolutional Neural Networks (CNN) and how they are applied to sentence classification tasks such as RC. A CNN, is a particular neural network structure that can be used to process grid-like input such as fixed-length sentences, images, signals or bounded time-series (LeCun et al., 1995).

Throughout the section we will construct an example which motivates the use of convolutions. The section presents the two central components of a CNN, the *convolutional kernel* and the *pooling layer*. The section is specifically restricted to define convolutions in the context of sentence input, and therefore skips the case where it is multidimensional but the height is not 1.

There is difference between the terms convolution and CNN. CNN is used to describe any neural network which includes a convolutional layer, but the convolution itself is a mathematical operation and only a part of a CNN.

2.5.1 Convolutional Kernel

A *convolution* is a mathematical operator which takes two functions $s(x)$, the *input function* and $f(x)$, the *kernel function*, and produces a third function $s * f$:

$$s * f(x) = \int_{-\infty}^{\infty} s(X) f(x - X) dX \tag{2.16}$$

We will restrict $s(x)$ and $f(x)$ to be the functional representation of a finite matrix $\mathbb{Z} \mapsto \mathbb{R}$, where x indexes into the matrices f and s. f and s are then zero when x is not equal to a valid index. The integral in (2.16) then becomes a finite sum. The convolution can then be rewritten as:

$$s * (x) = \sum_X s(X) f(x - X) \tag{2.17}$$

Unfortunately, "convolutional networks" actually uses another function, but it is very similar to a convolution. Instead of decreasing the indexing into the kernel when x increases, we increase it instead:

$$s * (x) = \sum_X s(X) f(x + X) \tag{2.18}$$

(2.18) is called *cross-correlation*. The effect of using it in a neural network will be the same. In the machine learning tradition, we will name both operations convolutions, but primarily use cross-correlation (Goodfellow et al., 2016, 9).

Motivational Example

We will make the convolution operable in a neural network by providing a running example and use (2.13), where S is the output of a word embedding layer. Now the challenge is, given S, to produce additional network layers that helps predict a label for S.

Using a regular fully-connected layer on top of a word embedding layer is problematic because of the very high dimensionality of the N-words sentence matrix $S \in \mathbb{R}^{d^w \times N}$. For example, if S consists of 15 words that are embedded in a 300-dimensional space, the number of weights required for a 50-neuron dense layer is $15 * 300 * 50 = 225000$, and 50 is a rather small number of neurons.

Instead, we can use a convolution which *convolves* over a *window* W of words $[s_i, s_{i+1} \dots s_{i+W}]$ in S. Let f be a kernel matrix which is randomly initialized and part of the weights in $h(x, w)$, which are learned by optimization. The number of weights in the kernel is much smaller than in a regular neural network layer because the weights are reused across the windows. Given W, the weights in the kernel is the matrix $f \in \mathbb{R}^{W*d^w}$ In the example, if we choose W to be 2, the kernel has weights $2 * 300 = 600$. In general, the word window W over the sentence representation $S \in \mathbb{R}^{d^w \times N}$ requires a convolutional kernel $f \in \mathbb{R}^{W \times d^w}$.

The convolution in (2.17) produces a value which substitute the output of a neuron in a regular neural network layer. The analogue to a neuron is that the kernel f is equal to a set of weights for a single neuron. Instead of having each neuron having its own set of weights, f is shared across several neurons. For the same reasons as regular neurons have, we would like to have the ability to approximate non-linear functions by adding the same family of activation functions and the bias parameter from section 2.3.1. Now, s_i is the score for a single word w_i, and the product of the convolution $(s * f)$:

$$s*_i = \sigma(\sum_{k=0}^{W-1} f_k^T S_{i+k}^T + b) \tag{2.19}$$

Note that the kernel f and the individual words in S are vectors, and $s*_i$ is a scalar. We use $s*_i$ to distinct the score of a convolution from the output of a neuron. $s*_i$ represents the score for a single word, and convolution for an entire sentence produces the *feature* or *filter* $S* \in \mathbb{R}^{n-W}$:

$$S* = [s_0, s_1 \dots s_{n-W}] \tag{2.20}$$

We cannot use the convenient matrix-to-vector notation in (2.9) anymore because $s*_i$ only multiplies a part of f. A single feature corresponds to a regular layer of neurons in a neural network, which produce an output vector. We would like to detect many features and compose a *feature map* from S, so the convolutional layer produces a matrix instead of a vector. Instead of a single feature kernel f we define d^c kernels $[f_1, f_2 \dots f_{d^c}]$. Finally, we change $s*_i$ to index the kernel f_j for the specific feature:

$$S*_{i,j} = \sigma(\sum_{k=0}^{W-1} f_{j,k}^T S_{i+k}^T + b) \tag{2.21}$$

For convenience, the notation for the entire feature map is still the same but now $S* \in \mathbb{R}^{(n-W)\times d^c}$:

The convolutional layer employs the concept of *local connectivity*, because ach word score $s*_{i,j}$ is only connected to words in its immediate context. Local connectivity assumes that input far from each other spatially is less relevant. With this assumption we can avoid a large number of unnecessary weights. The other upside of $S*$ is that each $s*_{i,j}$ is computed with the same set of weights. This concept of *parameter sharing* enables the convolutional layer to detect features invariant of where they occur in the sentence. Returning to the above example, a 50-feature kernel will now consist of $2 * 300 * 50 = 30000$ weights, which is several times smaller than the number of weights in the fully connected layer.

The idea of applying the convolution kernel to neural networks is directly inspired from biological principles (Goodfellow et al., 2016, 9.10). Specifically, "simple cells" in the human brain are the features f_i which are then composed into higher-level features by subsequent layers in the network. The simple cells are believed to be close to linear functions of small locally connected sections of visual input (Srinivas et al., 2016).

Variants

A number of parameters can be changed in (2.19) to make the convolution different. The running example is constructed to portray how networks used for relation classification

usually do. Convolutional layers are applied to a wide selection of input types and can be used differently. We shall briefly examine some common variations in this section.

The number of filters d^c determines how many features we are looking to produce from the input. The way the convolution "scans" the sentence can also be changed. In (2.19), this changes the summation to jump a number of j's at each step. This is the *stride* of the convolution and has the value of 1 in the running example. Input words are usually equally important when they enter the convolution and sentences are usually short, so the changing the stride is unlikely. Even though S is a two-dimensional structure, convolutions over a sentence usually convolves over the entire embedding. When the input is images, we might only target a small region of the second dimension at a time. The definition of multidimensional convolution is omitted from this presentation because they are not used for sentence classification, but they are an important part of the convolution in general. Finally, we have only implicitly defined what happens to n-grams at the border of the sentence. In the above definitions, the convolution uses a *valid padding*, which means that words in the beginning and end of the sentence are less connected than words in the middle. This is because they will never be included in convolutions to words left and right of them, respectively. The alternative is to use *same padding* which uses zero-tokens at the borders of the sentence.

Multiple Windows

The window size W represent *n-grams* of words that are examined by the convolutional kernel. Changing the window size changes the scope of the context for which a word is examined. Instead of only choosing one W we can examine several W and produces several $S*$, which would allow different patterns of word contexts to emerge. In the example above, we can change the configuration of $W = 2$ to instead be $W = [2, 3, 4, 5]$. The individual $S*$ does not change but the entire output of the convolutional layer is now the list $S*_2, S*_3, S*_4, S*_5$. Of course, this also increases the number of weights substantially. Now the total amount of weights in the example is $\sum_{j=2}^{5} 300 * 50 * j = 210000$, but still less than a fully connected layer. Using *multiple windows* allows the convolution to recognize a wider array of patterns and is shown in chapter 3 to have a significant impact on sentence classification tasks including relation classification.

2.5.2 Pooling Layer

The second central component a convolutional neural network is the *pooling layer*. The goal of the pooling layer is to make the output for a specific feature *translationally invariant*. In relation classification, we would usually like to detect the presence of a certain n-gram which is a strong indicator of a particular semantic relation. For example, the tri-gram "is caused by" is likely a strong indicator of the relation `Cause-Effect`. The presence of this tri-gram might be marked in a feature for each word $s * i, j$ for some j. We can summarize the all the $s * i$'s for a particular feature by using the *max-pooling* operation on $S*$:

$$p_j(S*) = max(S *_{0,j} \dots S*_{n-W,j}) \tag{2.22}$$

This convertion of $S*$ is useful if we want to detect the presence of a feature rather than where the feature is located. The translational invariance property means that changing some words in S will not change the outcome of the pooling operation if the important feature is not changed. And finally, the outputs which are now connected to an upstream classification layer is drastically lower than using the direct output of the convolutional layer. In the running example, the number of outputs for a multiple window configuration with 50 features is $\sum_{j=2}^{5}(n - W) * 50 = 2500$, but the pooling layer produces $50 * 4 = 200$ outputs instead at the cost of 0 weights. Each p is a single score. With multiple regions we gather all p's to form a final feature vector z. Given the configuration of multiple

$W = [x, y, z]$, the output of the pooling layer is:

$$z = [p_0(S*_x) \dots p_{d^c}(S*_x); p_0 S*_y \dots p_{d^c}(S*_z)] \tag{2.23}$$

Variants

The max pooling operation is not the only way to downsample the outputs of the convolutional layers. Additional convolutional layers can be applied, or other pooling operations such as *average-pooling or l2-pooling* can be used (Goodfellow et al., 2016, 9.3). These variants are omitted from the presentation because the majority of convolutional networks used for sentence classification in chapter 3 use max-pooling only.

The pooling operation is likewise inspired by neuroscience. The V1 part of the human brain contains *complex cells*, which are feature detectors like the simple cells that is modeled by the convolution itself. Complex cells are also invariant to changes in input such as rotation of images. Quiroga et al. (2005) showed the presence of neuron in the brain that fire when specific high-level features are detected in visual input.

2.5.3 Summary

We have examined the general definition of convolutions and shown how we can use it instead of a regular fully-connected layer to form a convolutional layer. We have also shown how convolutions are done multiple times in a single layer with different window sizes to produce several augmented sentence representations. To reduce the number of outputs a convolutional layer produces, we apply a pooling operation that summarizes a feature into a single score. And finally, the pooling scores are concatenated to form a feature vector which can be used for classification later in the network.

Chapter 3

Related Work

Applications of neural networks to NLP have been a very active area of research in the past two decades. In relation classification, an neural networks have been used to significantly improve results on common datasets such as the SemEval 2010 task 8 and the ACE 2005.

This chapter presents related work in the area of relation extraction and classification. The focus of the chapter is the shift in tendency from heavily hand-crafted *feature-engineered systems* to neural networks with unsupervised pre-training which require considerably less task-specific feature-engineering. These neural networks are commonly called *feature-learning systems* or *representation-learning systems*. Additionally, the chapter covers approaches that uses semi-supervised learning and unlabeled data during or after the training process.

It is important to notice that the definition *feature-based system* - which is used in many RE and RC papers such as Bach and Badaskar (2007); Miwa and Bansal (2016); Nguyen and Grishman (2015) - is kind of a misnomer. What it usually specifically means is that the features which are used in the learning system are handcrafted with linguistic knowledge, and that they are not learned by another system. Such features might detail the shape of the word, context words, synonym or antonym groups and to which sentential clause the word belongs. Neural networks also use features. These features may not be handcrafted, or they may be called a representation instead, but they are still features. Word vectors, which are the common input structure used for neural networks in NLP, is both a representation and a set of features for each word which is learnt from an unsupervised task. In neural network terminology, the words features and representation are often used interchangeably. The systems also overlap: for example, Zeng et al. (2014) combines a word-vector with "traditionally" extracted features as input for a CNN.

3.1 Supervised Approaches

Using labeled data to learn a classifier in a supervised manner is the dominant approach to RC. Work on RC and RE are often divided into feature-based and *kernel-based systems* (Nguyen and Grishman, 2015; Zeng et al., 2014). This division can be confusing since the supervised learner *Support Vector Machines* (SVMs) are often used in both types of systems. In common software libraries SVMs always uses a kernel even when treating the input as a linearly separable problem.[1]. For the sake of clarity, I will define kernel-based systems to only include SVMs which uses non-linear kernels, and by doing so does not explicitly compute the feature space.

[1]Here are two major frameworks who do just that - LIBSVM (`https://www.csie.ntu.edu.tw/~cjlin/libsvm/`) and SKLearn (`http://scikit-learn.org/stable/modules/svm.html#svm-kernels`)

3.1.1 Feature-based systems

The quintessential model of feature engineered systems in RC is the SVM. With an SVM, the input sentence is represented by a series of linguistic features in a vector. The goal of the SVM is to maximize the margin between these training vectors in the dimensionality defined by the vectors.

For the SemEval training set, an extensively cited SVM had the highest accuracy for the 2010 SemEval conference (Rink and Harabagiu, 2010). The SVM uses a total of 45 feature types for each word. Here is a description of a few selected:

1. *Wordnet Hypernyms*, which are generic classes, or hypernyms, for groups of words, defined by a Princeton research group[2].
2. *Part-Of-Speech Tags*, which describe how each word relates to different parts of speech and grammatical properties.
3. *Dependencies*, which describe how each word are related to the syntactic structure of the tree. Dependencies are analyzed by using a dependency parser on the sentence.

Each individual feature is represented as a boolean value and the feature space is treated as linearly separable.

3.1.2 Kernel-based systems

Because the dimensionality of the feature vectors can be very high, kernel-based systems use the *kernel trick* to implicitly compute the distance in space between the vectors. Collins et al. (2001) is a seminal paper on using kernels in NLP. Notice that the input to a kernel may be raw text accompanied by some extracted features, which then forms a new representation. The full feature space is usually an exponential combination of the representation, and it is this space that the kernel avoids. The most commonly kernel-based classifiers are SVMs and the *Voted Perceptron* (Freund and Schapire, 1999). Essential for all kernels described in the following sections is that they exploit the structural nature of text by either creating parse-trees or sub-sequences that can be analyzed.

Subsequence Kernels

Kernels that exploit the sequential nature of text are called *(sub)sequence kernels*. Bunescu and Mooney (2005a) show a general subsequence kernel and use it for RE to extract protein interactions from a biomedical corpus, and on the ACE 2005 dataset. The used kernel has specific measures for penalizing longer subsequences between relation heads, entity words which are important for RE and RC.

Sequence kernels can also be applied to raw text alone to but is often used in tasks where more text are availaible such as document classification (Lodhi et al., 2002).

Tree Kernels

Parsed texts are represented as tree structures which define the relations between words in a sentence. Kernels can be designed that use these structures to compute the vector product. Bunescu and Mooney (2005b); Plank and Moschitti (2013); Qian et al. (2008); Zelenko et al. (2003) all use different variants of parse trees in their kernels. The tree-kernel encapsulates a wide array of parsing techniques such as constituent parsing, shortest path dependencies and shallow parsing. Additionally, each kernel is highly specific to the RC task. The kernels are used either in a voted perceptron or a SVM.

[2] `https://wordnet.princeton.edu/`

Feature-Kernel Hybrids

Several systems have augmented the tree-kernel with extensive linguistic features. Such system creates a parse-tree from the input and subsequently annotates each node with a variety of features. Some features which can be added to the tree are:

- *Instance features*, which are added to the root node of the parse-tree. These features can specify information about the marked entities such as their syntactico-semantic structure or their common word type (Chan and Roth, 2011).
- *Phrase features*, which annotates each phrase node with lexical and contextual information (GuoDong et al., 2005).
- *Context Information features*, which annotates each node with information relative to the parse tree such as path to the marked to the entities or the context path from a sub-tree root to the annotated node (Zhou et al., 2007).

(Sun and Han, 2014) defines a *feature-enriched tree kernel* for relation extraction which uses all the above defined enhancements to the kernel and achieves (at the time) state-of-the-art performance on the ACE2004 dataset, which is closely related to ACE2005.

3.1.3 Feature-learning systems

Recently, work on the RC and RE task have moved from the above mentioned systems to systems that use less linguistic knowledge as input for the classifier. A sub-goal of these systems is to use an input representation which is as close to the original input as possible. The system should then automatically learn and derive features from the input instead of being input by a human expert. In NLP, these systems are almost exclusively neural networks.

In RC, neural network studies is roughly divided into networks that utilize recurrent structures and work that uses convolutional layers as their respective core component. Common to all networks in the following sections is that they use word embeddings as their initial layer, which definitely shows how important they are to advances in RC.

3.1.4 Convolutional Neural Networks

CNNs have been applied to a wide range of sentence classification and modelling tasks. Collobert and Weston (2008) presents the basic CNN which is used for six different NLP tasks. The paper defines the basic structure of a CNN for NLP which is the structure used in many subsequent papers:

1. Input sentences are transformed word-by-word to word embeddings.
2. The word embeddings are convolved over with one or more window size.
3. The convolved output is pooled and fed into one or more fully connected layers.
4. The last layer is fed into a softmax classifier designed for the specific task.

This structure modeled the convolution as a linear operation. Kalchbrenner et al. (2014) introduced the non-linearity to the convolution stage and introduced variable-sized downsampling, but is otherwise built directly on top of the previous network. Building on top of recently published CNNs is a noticeable trend in recent years. Specifically for the SemEval 2010 dataset the last five years have been a series of increasing state-of-the-art performances by CNNs (among other networks). Nguyen and Grishman (2015); Santos et al. (2015); Wang et al. (2016); Xu et al. (2015a); Zeng et al. (2014) have all expanded on the core idea described above and pushed the accuracy on the dataset. Each CNN achieves state-of-the-art performance for the SemEval test set. Below is a list of networks sorted chronologically and by increasing reported F1 scores on the SemEval test set:

- Zeng et al. (2014) use a CNN enriched with hypernyms and other lexical features and introduced many-dimensional positional embeddings.
- Nguyen and Grishman (2015) removes the lexical features but use several window sizes for the convolution. The network is also trained with the AdaDelta update rule instead of classic gradient descent.
- Santos et al. (2015) changes the loss function to a *ranking loss* function which prevents the network from updating the weights as a result of the `Other` relation being incorrectly classified. They also learn the representation of the class labels instead of encoding them as a one-hot vector.
- Xu et al. (2015a) uses dependency paths between the entities as input to the CNN instead of the full sentences given in the training data. Additionally, by using dependency paths it is easy to model the directionality of the predicted relation by switching the direction of the dependencies and feeding it to the network as a negative sample.
- Wang et al. (2016) uses *attention-mechanisms* to weigh the semantic meaning of words and entities together. They also use a class embedding similar to Santos et al. (2015) while introducing a margin-based loss function that maximizes the distance between the embeddings while minimizing the negative log-likelihood of the dataset and the network output.

At the time of writing this thesis, Wang et al. (2016) have the highest published results for the SemEval test set. These are only a few of the CNNs which have been used in the past two years for RC, but they are highlighted since they are built upon the same baseline which will be used in chapter 4.

3.1.5 Recursive Neural Networks

The other class of networks which have been used extensively in the past years for NLP and RC is the *recursive neural network* (Pollack, 1990). Recursive networks are a generalization of *recurrent networks* which can operate on tree-structures (Elman, 1990). The idea of a RNN is to learn a function h which for a temporal sequence, of input outputs a state vector and an output for each step. The RNN is an abstract model for recursive computations, and choosing a suitable h is essential for implementing an effective learning model. The most commonly used model is the *long short term memory network* (LSTM), which computes the state for each timestep by learning a series of memory gates. The serves as controllers that filter input, filter output and update the state (Schmidhuber, 2003). The described networks in this section all use a LSTM or a variant of the original algorithm.

In NLP, a sentence can be seen as a temporal sequence where individual words are timesteps. For RC specifically, sentences are usually dependency parsed and a RNN is applied to the parse tree to produce an output. Socher et al. (2012) applies an RNN to a dependency tree with the goal of learning composite vectors for phrases and sentences. This paper focuses mostly on this representation, which involves creating matrices for every word and learning functions which can combine them. However, the representation were then applied to different end tasks, including the SemEval 2010 task 8 with reported state-of-the-art performance. The above mentioned strategy of using a parse tree as input is key to several recent results for the RC task. Additionally, most of these networks also use other NLP tools such as POS-tagging or WordNet hypernyms. Ebrahimi and Dou (2015); Xu et al. (2015b) both uses a *dependency tree* to find the most relevant path of words between entities in a sentence. These paths are used as input to a RNN with added linguistic features about each word.

While a sentence is usually read once from left to right in a CNN or a simple RNN, RNNs such as the LSTM can also be modified to read the sentence backwards. A *bidirectional LSTM* (BI-LSTM) records the state at each output as a concatenation of the states

achieved by reading the sentence in both directions. Miwa and Bansal (2016) uses a BI-LSTM to both extract entities and classify them, and achieves state-of-the-art results on the ACE 2005 dataset. Their model uses as input both a sequence for the entity detecting and a parse tree for the relation classification.

Finally, RNNs can be stacked to achieve depth in the model. While it is not theoretically well understood why stacked RNNs can perform better than single RNNs, they have shown promising results. Goldberg (2016) mentions several areas of NLP that have shown increases in performance with stacked RNNs. On the SemEval 2010 dataset, Xu et al. (2016) uses a deep RNN which achieves state-of-the-art performance albeit with a heavy training cost.

When comparing the RNNs to CNNs, it is important to notice that all of the recent results on RNNs use auxillary input such as parse trees, POS-taggers and WordNet Hypernyms. When these features are excluded, the RNNs in general do not perform significantly better or worse on the SemEval 2010 and ACE 2005 datasets.

3.2 Semi-supervised Approaches

The remainder of this chapter is a brief description of semi-supervised approaches to relation classification. Comparing semi-supervised approaches in sentence classification to the purely supervised is somewhat difficult because these approaches usually use part of the original training data as unlabeled data. The question asked in these studies are then: given less data, can we still achieve comparable results to a purely supervised approach that uses all the data? Investigating semi-supervised approaches this way has the nice benefit of the unlabeled data being from the same data generation process as the unlabeled. Perhaps that is all related work I have found use this approach.

3.2.1 Semi-Supervised as Unsupervised Pre-training

Semi-supervised approaches are hard categorize when applied to relation classification because the term is used for any approach that have some unsupervised process as part of the learning strategy. For example, Sun et al. (2011) uses a clustering method to generate word clusters on known labeled training data. The clustering method is unsupervised and is used to generate features, but the learning process itself is supervised. In a sense, all the networks detailed in section 3.1 that use pre-trained word embeddings are semi-supervised, because the algorithm used to train the embeddings is unsupervised. Instead, when a specific resource is constructed before the actual learning process takes place, it is more useful to refer to them as examples of unsupervised pre-training. This distinction makes the research detailed in the following sections different than the previous.

3.2.2 Distant Supervision

Closely related to semi-supervised learning is *distant supervision*, which usually generates all training data automatically from a knowledge base. The knowledge base is not usually training data in itself, but can generate labeled data with a specific error rate. Hoffmann et al. (2011); Surdeanu et al. (2012) uses the *Freebase* dataset to generate training sentences for relation extraction. The distantly supervised relation extraction problem is usually different because the number of relation classes is unbounded.

3.2.3 Relation Classification/Extraction

As far as I know, only a single study uses the SemEval 2010 task 8 with a semi-supervised approach. Chen et al. (2010) uses several unlabeled datasets to help learn a representation for relation classification, and reports very promising results. However, they modify the SemEval problem to be binary, which makes the results uncomparable to the other work in this chapter. The self-training, label-adding approach described in in section 2.2.1 have also been investigated for RC. Qian et al. (2009) uses self-training but names it

self-bootstrapping, mainly because they are using a more complicated sampling strategy to select the seed data. The results are evaluated on the ACE data, but again does not compare with fully supervised approaches.

The presentation of this chapter is intended to cover objective 2 and provide an understanding of where this thesis is placed in the RC field.

Chapter 4

Research

As shown in chapter 3, there are many solutions for the RC problem based on neural networks which have been developed in recent years. Models are frequently built upon each other because they provide comparable baselines for futher work on the SemEval dataset. Different experimental techniques may overlap each other in terms of their effect on performance. Restrictions on computational power and time to execute experiments may limit the degree to which results are certain, and neural networks have lots of randomness in them which influences the test scores. Since they usually also consist of many different layers with many different implementation details, small deviations can make small changes in the reported results which makes published results harder to reproduce. The transition to neural networks in NLP necessitate a change in how experimental results should be reported, but this change has not arrived in the academic literature (yet).

This chapter intends to answer objectives 3-7 of the thesis objectives. It presents a series of experiments and discussions which highlights central problems in the common approach to relation classification. We shall examine a common baseline model based on convolutional neural networks and highlight implementation problems that is caused by reproduceability issues. Then, we look at simple reductions that can be made to the baseline without reducing the general performance of the classifier. Common non-invasive machine learning techniques are applied, and their effect investigated, to try and improve the performance and robustness of the classifier. A semi-supervised approach, based on self-training, is presented with the main focus of getting more training data for the classifier. Finally, the results of the semi-supervised approach prompts a general discussion of the relation classification task and the SemEval dataset.

4.1 Baseline Experiments

Based on the investigation of recent neural networks from chapter 3, the CNN is currently the most promising structure for RC. In this section the process of constructing a baseline CNN for RC is detailed. The section serves two purposes: first, it is intended to detail the model architecture so potential future work can accurately re-implement the model. Additionally, the process of implementing the baseline developed by Nguyen and Grishman (2015) is intended to highlight common reproduceability issues and propose two solutions to the issue.

In the baseline and all subsequent experiments the SemEval dataset is used because of the reasons detailed in subsection 2.1.2.

4.1.1 Model Architecture

The baseline model is based on the CNN developed by Nguyen and Grishman (2015). It is a rather simple architecture which only uses the input sentences and pre-trained word

embeddings as a linguistic resource. No additional linguistic resources such as dependency paths or WordNet hypernyms are used to train the classifier. All hyperparameters in the model architecture are tuned with 5-fold cross-validation on the training set. The model consists of five sequential layers:

- **Preprocessing**: Each input sentence is sanitized and tokenized. All words are converted to lower case and special characters are filtered out. The index of each entity and the distance from each word to the entity words are calculated. Sentences which have a token entity distance greater than a fixed bound $N = 15$ are removed from the training set to reduce noise. The tokenizer includes common hyphenations as tokens, but removes more unusual symbols. The preprocessing steps produce a series of tokens $T = [t_0, t_1 \dots t_N]$.

- **Embedding layer**: The tokenized words T are embedded with a `word2vec` word embedding and two positional embedding matrices, which maps the distance from each $t_i \in T$ to the marked entity words. The sentence matrix S is constructed from these embeddings as in (2.15). The word embeddings are publicly available, pre-trained 300-dimensional word vectors[1]. Words in the training set that do not have a matching pre-trained embedding are mapped to a randomly and uniformly initialized vector of the same size in the range $[-0.25; 0.25]$. This range approximates the variance of the pre-trained word embeddings. The positional embedding matrices are 50-dimensional and randomly initialized.

- **Convolutional layer**: The third layer applies the convolution described in (2.21) on the sentence S . The activation function in the convolutional layer is the *ReLu* function described in section 2.3.1. The window size configuration $[2, 3, 4, 5]$ produces a list of matrices $[S*_2, S*_3, S*_4, S*_5]$, where each element is $S*_W \in \mathbb{R}^{(N-W+1)\times d^c}$. $d^c = 150$ is the number of features for each window.

- **Pooling layer**: Each convolved feature for each window $S*_W$ is max-pooled and concatenated together to form the feature vector z as described in (2.23). Given 4 windows, the number of features in z is $d^c * 4 = 600$.

- **Dropout layer**: Regularization is performed by using dropout on z, which outputs a new vector z_p of same size but with the activation of dropped neurons set to zero. The dropout rate $p = 0.5$.

- **Fully-connected output layer with dropout**: z_p is input to a regular fully connected layer which learns the contribution of each pooled feature to the target labels. The activation of this output layer is the softmax function defined in (2.10), which outputs a discrete probability distribution for the sentence over the target labels.

Training

The network is trained with stochastic gradient descent using shuffled batches of size 50. The gradients are computed with backpropagation and the weights are updated with the AdaDelta update rule described in subsection 2.3.2. A l_2-based weight constraint as described in section 2.3.3 is applied to weights in the layer with the maximum norm value $s = 3$. The number of full iterations through the dataset is 9.

General Hyperparameters

All weights that are not pre-trained are initialized with *xavier initialization*, which is values drawn randomly $U\big[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\big]$, where $U[-l, l]$ is the uniform distribution in the range $(-l, l)$ and n is the number of neurons in the previous layer. This initialization is used because there is strong empirical evidence that the variance of the weights should be

[1] `https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit`

connected to the number of neurons (Glorot and Bengio, 2010). All bias parameters is initialized to zero. The entire architecture is modeled in Figure 4.1.1.

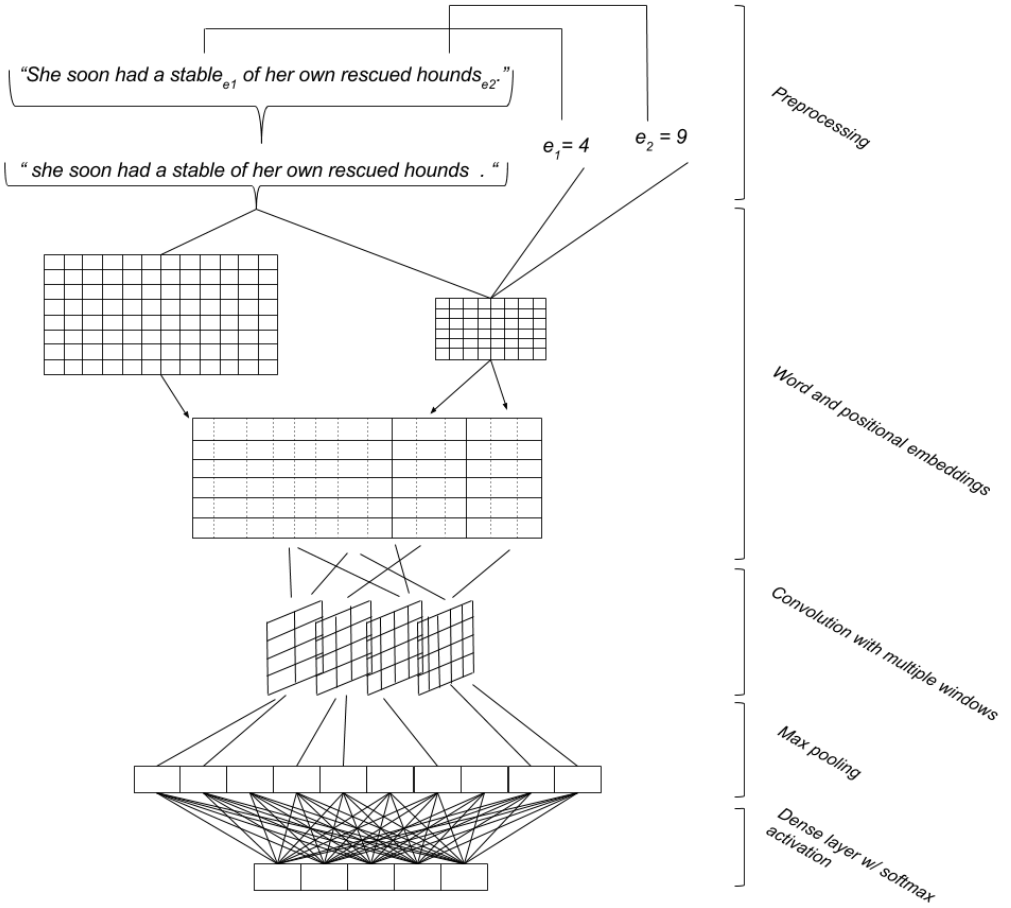


Fig. 4.1
The complete CNN model architecture. The number of cells drawn at each layer do not strictly correspond to the number of neurons in the layer.

4.1.2 Reproduceability

objective 4 requires a baseline model with performance close or equal to the reported scores for the baseline. Getting the same results from the baseline model as reported by Nguyen and Grishman (2015), however, proved to be very difficult. Whenever a model is published, it should be easy to reproduce the empirical results of the model, especially on a dataset such as SemEval where the testsplit is pre-designated. Reproduceability is arguably an issue in NLP in general (Pedersen, 2008).

The baseline model suffers from reproduceability problems which are highlighted here, but other networks in chapter 3 suffer from similar issues.

Network Details

Neural networks are layered structures with many different hyperparameters and initialization schemes. Here is a list of network details that is missing from the original description of the baseline model:

- Initialization schemes for weights in the network can have significant effect on the results due to the iterative nature of the training algorithms (Goodfellow et al., 2016, 8.4). However, the initialization scheme for the weights in each layer is undocumented.
- The preprocessing from the raw sentence to the embedding layer is largely undocumented, yet this step is extremely important for performance. It is clear that the network should accept an input sentence of fixed length n, which is the maximum distance between marked entities. How is n chosen? It could be either computed as the maximum n in the training set (which degrades the performance of the system significantly), or it could be fixed at a lower distance to reduce noise. Tokenization may filters out non-word characters such as punctuation, which impacts how n is calculated. If n includes these tokens the sentences are significantly shorter than if n is only counted with words.
- Pre-trained embeddings have a large impact on performance in sentence classification tasks (Kim, 2014). Many details in the training procedures for these embeddings are undocumented in RC studies. As an example, Santos et al. (2015) reports a state-of-the-art performance at the time but does not document whether the embeddings are pre-trained for the specific experiment, or imported from a public repository.

It is important to notice that these issues are especially important because we are trying to investigate if the CNN architecture is a robust model for RC. This becomes difficult if we cannot build existing architectures that perform at the reported level. To accomodate the listed issues, the baseline model in this thesis is accompanied by a thorough specification of the network and its hyperparameters. The code for the network and the preprocessing steps are also included and will be kept online in a public repository[2]. Finally, reproducing the results of a particular experiment will, given the correct installation of dependencies and word embeddings, be reproduceable with a command-line script. The full specification is given in section A.1.

Measuring the Results

The SemEval dataset is measured with the official scorer on the designated test set as described in subsection 2.1.2. This leads to models being compared on very narrow margins for test scores (Nguyen and Grishman, 2015; Zeng et al., 2014). Using neural networks makes this approach even more problematic since they have several sources of randomness. Weights are initialized randomly, and training occurs in batches which are shuffled, making the gradient descent different from experiment to experiment. When an empirical result is reported for a network, it should be specified how robust the score is compared to the average performance of the network. Normally, this robustness is redundant because of cross-validation. But cross-validation reduces the size of the dataset and that in turn produces lower scores. Since the tradition for RC and SemEval is networks being compared on the designated test set, the cross-validation scores are never used as baselines for performance.

I therefore propose as an accomodating middle-ground that scores from the SemEval test set are reported as a small statistic over 100 experiments. The statistic should as a minimum include the min, max, mean and standard deviation/variance of these runs. The statistic will show if the network is robust and how much of an outlier a potential "high-score" can be considered. Additionally, all the experiments in this thesis includes the 5-fold validation estimate as well for future baseline reference. The score will be shown with a **C-V** header in all result tables. For the baseline model, the result statistic is shown in Table 4.1.

[2] `https://github.com/msta/cnn_rc`

Min	Max	Mean	σ	C-V
79.84	82.42	81.13	0.70	80.10

Table 4.1
CNN baseline scores.

The table show that the implemented baseline has a maximum score that is comparable to the network developed by Nguyen and Grishman (2015), but a mean value that is considerably lower. The standard deviation shows the robustness of the model across a number of runs. And finally, the cross-validation score provides a robust and reliable estimate of how the model performs, but it is unfortunately not very comparable to other scores in recent studies.

4.2 Network Simplification

Armed with the baseline model we move to objective 4 of the thesis objectives. We want to investigate if the chain of ideas and networks that leads up to the baseline might have introduced redundant network structure. We can reduce the complexity of a neural network by examining the regularization techniques for potential overlap. Removing overlapping techniques eases the transition from an experimental academic network to a production quality implementation. The number of parameters in different layers should also be examined because simpler models tend overfit less as explained in section 2.2.3. The components which should be examined are especially those components that introduce real-valued hyperparameters.

The baseline model have two properties which will be investigated: the l_2-norm based weight constraining and the dimension of the positional embeddings. Each property is examined and a simplification is proposed. The results are shown in Table 4.2.

l_2-based constraints

An unusual property of the CNNs that we have examined in chapter 3 is they apply constraints on the l_2-norms of the weight vectors. The original intended use for these constraints is training very large networks with dropout (Hinton et al., 2012). This intended use is arguably a rather weak connection to using dropout in sentence classification CNNs, as RC CNNs are much smaller and use dynamic learning rate update rules such as the AdaDelta, which eliminates the need for worrying about the learning rate. The weight constraint unfortunately also introduces a real-valued hyperparameter s. Real-valued hyperparameters can be harder to tune than discrete ones because the parameter space is infinitely wider. Finally, Zhang and Wallace (2015) shows that l_2 constraints in general have little to no effect on sentence classification tasks. Some tasks shows a marginal improvement while it has a degrading effect on other tasks. For these reasons, the constraints will be removed and the effect investigated here.

Positional Embeddings

The positional embeddings are designed to provide important information about the distance from each word to the entities in the sentence. With sentences of n length this introduces $(2 * n - 2) * d$ new weights at the embedding layer, and d is usually 50 or more (Nguyen and Grishman, 2015; Santos et al., 2015; Wang et al., 2016). In the convolution layer, the number of extra weights needed upstream is multiplied, since multiple window sizes requires a new set of weights for each window. A critical view of the positional embedding p_i is as a many-dimensional way of informing the network whether the word t_i is an entity or not. With this view, reducing their dimension could be possible without loss of performance.

Configuration	Min	Max	Mean	σ	C-V
Baseline	**79.84**	82.42	81.13	**0.70**	80.10
1-dim embedding	79.32	82.56	81.11	1.03	80.03
No l_2-constraint	78.21	82.63	80.99	0.89	79.81
1-dim / No l_2-constraint	78.48	**82.76**	**81.25**	1.07	**80.45**

Table 4.2
Complexity reduction scores.

The results show that the above modifications to the network can safely be made without performance loss. The highest score achieved with any setting is made with 1-dimensional positional embeddings and no l_2 weight constraints, however it is not a significant increase. Reducing overfitting is a strong argument for reducing the positional embedding, and keeping the dimensionality low is consistent with the original idea of the embedding (Zeng et al., 2014). Removing the weight constraints is a strong argument for making the model easier to reproduce and less complicated.

4.3 Network Improvement

The results from the previous sections suggests that the baseline can be simplified without performance loss. This section will propose a two improvements to the baseline model, which are both rooted in general purpose machine learning principles. Ideally, a general purpose method can be applied to other network structures and other sentence classification tasks if they deliver promising results.

4.3.1 Early Stopping

The first suggested improvement to the network is the early stopping technique shown in section 2.3.3. Early stopping is supported by many open-source machine learning frameworks and can be implemented for any iterative learning algorithm. A strong argument for using early stopping for a CNN baseline is that it simulates the effects of the popular technique l_2 *weight decay* regularization (Goodfellow et al., 2016, 7.6). Weight decay is often used for neural networks to control overfitting but haven't seen usage in recent work on RC. The hyperparameters introduced by early stopping is not as important for performance as choosing the right l_2 value either, which makes it easier to apply. The effect of applying early stopping shown in table Table 4.3.

	Min	Max	Mean	σ	C-V
Baseline	**79.84**	82.42	81.13	0.70	80.10
Early Stopping	79.63	**82.66**	**81.24**	**0.55**	**80.21**

Table 4.3
Early stopping scores.

As the table shows, using early stopping yields a nice small improvement on all statistics except the minimum, which is slightly lower. The results suggests that early stopping generalizes better than tuning the number of iterations manually. It is worth noting that the early stopping can be fine-tuned even more at the cost of computational expense. The early stopping implementation used here measures the validation error after each full pass through the training set. This can be changed to measure the validation

error after each batch, which makes the algorithm even more sensitive to small changes to the generalization estimate.

4.3.2 Class Weighting

The second suggested improvement to the baseline implementation is rooted in an analysis of the requirements for the SemEval dataset. The dataset is relatively small and has a high number of classes compared to its size. This means that some labes, like the `Instrument-Agency` label, have a low number of samples. The $F1_{macro}$ metric of the official scoring tool suggests that relation classes are equally important to classify. We can exploit this metric by changing how the classifier observes each training sample. We add *class-weighting*, which multiplies the loss function when it is calculated for a sample (x_i, y_i) with a weight related to y_i. Because of the $F1_{macro}$, the loss function is changed to include the inverse of the probability of the label occuring in the training set. Given the distribution of the relation classes $P(y)$ in the training set and the training pair (x_i, y_i), the cross-entropy loss in (2.11) becomes:

$$Ł_{weighted}(h, x, y) = Ł(h, x, y) * \frac{1}{P(y)} \tag{4.1}$$

The effect of applying the class weights are shown in Table 4.4.

Configuration	Min	Max	Mean	σ	C-V
Baseline	79.84	82.42	81.13	0.70	80.10
Class-weights	**80.53**	**83.39**	**82.36**	**0.55**	**81.83**

Table 4.4
Class-weighted loss function scores

The intended effect of the class weighting is to move the gradients more when important samples are being learnt and conversely move the gradients less on common classes. It is observed in the table that the class weights yields significant improvements in all statistics. They are conveniently implemented in modern software libraries with a few lines of code. There is a strong theoretical connection to the maximum likelihood estimate: the class weights trades bias of the model by constructing a "fake" distribution of the training set where the minority classes have a higher probability. Implementing class weighting also illustrates the trade-off involved with using the $F1_{macro}$ as the primary metric. We might not be building a classifier which is robust for any other datasets than the SemEval, since the relation classes are likely to be distributed differently in other datasets.

Summary

The above sections show that it is possible to simplify several properties of the baseline model without hurting performance, which answers objective 4. It is also possible to introduce general approaches such as class weighting and early stopping increase the performance of the classifier. With these augmentations we have investigated objective 5. We will now examine whether it is possible to get better performance from the baseline model by adding unlabeled data to the training process.

4.4 Semi-supervised training

With objective 6, it is the goal to test if the classifier, which is trained purely supervised on SemEval training data, is robust enough to apply to unlabeled data. Using unlabeled data to help a supervised classifier is a very general purpose method and does not require

structural changes to the neural network. A simple way to reduce overfitting and improve the generalization ability of a complex model like a CNN is to introduce more data (Abu-Mostafa et al., 2012, 5.). We can observe the effect of adding more data by feeding the training set to the model in small chunks and observe the performance metric at each chunk. This observation is called the *learning curve* and is shown in Figure 4.4.

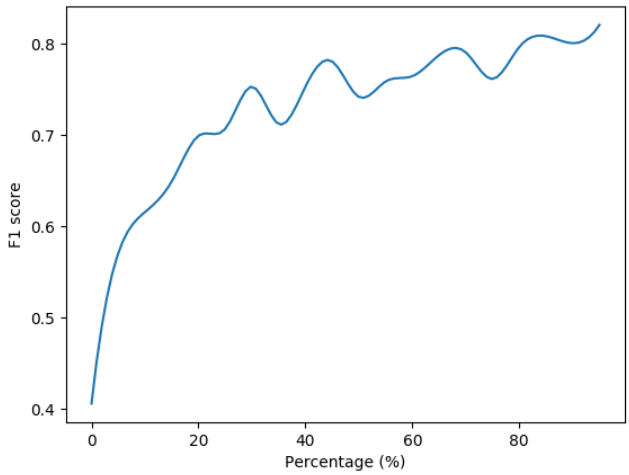


Fig. 4.2
Learning curve which show the test set F1 value on increasing sizes of the SemEval training set.

The curve shows that more data improves the classifier, and the shape of the curve does not indicate that it will stop increasing if we could increase the size of the training set. Unfortunately, annotated semantic relation data is hard to come by, and there is no other dataset that defines the same inventory as SemEval. Unlabeled text data, however, is plentiful on the internet. We leverage unlabeled data along with the labeled data by using the semi-supervised learning paradigm described in subsection 2.2.1.

Self-Training

We apply the self-training algorithm shown in section 2.2.1 to the CNN with the intention of adding more labeled data to the training set. The classifier is trained on the original training data and then the unlabeled data is classified. The stopping criterion is implemented with the following tests:

- The number of samples at each iteration is lower than 50.
- The loss of the new classifier, trained on the augmented dataset, is higher than the previous classifier. The loss is measured on a validation set which is partitioned purely from SemEval training data.

Terminating if the loss of the new classifier is higher than the previous is unusual for self-training, which would normally only terminate by the first stopping criterion. The reasoning behind the criterion is this: since the self-training algorithm is compared to a fully supervised model - the baseline - that uses the entire training set, the scores produced by the baseline can be used as a stopping criterion. The assumption is that degrading performance at one iteration of self-training will never help to increase the performance later. If newly classified samples "ruins" the prediction of the validation set, they will continue to do so because they are part of the training set.

Hyperparameters

The throttling parameter is set to 2% of the original data $8000/50 = 160$ and the confidence levels are > 0.99. The stopping parameter is set to 50.

Choosing unlabeled data for the experiment requires selecting a text corpus. For these experiments the W2C Wiki Corpus is a suitable corpus because it is reasonably formatted and available (Majlis and Zabokrtskỳ, 2012). The W2C Wiki Corpus is a text corpus extracted from Wikipedia. The corpus is split into sentences by PunktTokenization (Kiss and Strunk, 2006). After splitting the corpous into sentences, candidate sentences have to be selected by a selection process. The selection process is based on finding sentences with words that could be entities suitable for classification. Three different approaches to obtaining candidate sentences are used:

- **Simple Entity-Pairs Only**: This first approach strictly chooses entity pairs in the text corpus which are already present in the SemEval training set. The argument for this strategy is that the same two entities are likely to have the same relation even though they are in a different context.
- **WordNet/Antonym Entity Selection**: The second and third approach tries to obtain new candidate entity pairs (and sentences) by marking entities in the unlabeled corpus which are similar to the entities in the training set, but not exactly the same. Similarity is measured with two different strategies, which yields two distinct candidate sentence datasets:
  - The second approach looks up synonyms of the SemEval training entities through the lexical database *WordNet* (Miller, 1995).
  - The third approach looks up antonyms of the SemEval training entities by the method developed by Zhang et al. (2014).

The self-training algorithm is applied to the SemEval training set and the unlabeled candidate sentence sets, and the results are shown in Table 4.5.

Configuration	Min	Max	Mean	σ	Iterations	CV
Baseline	**79.84**	82.42	81.13	0.70	-	80.10
Entity-Pairs Only	79.45	**82.74**	**81.41**	**0.61**	**1.42**	**80.34**
WordNet Synonyms	75.92	82.14	80.22	2.44	0.83	79.39
Antonym Word Embeddings	65.68	81.62	79.52	6.07	0.21	76.32

Table 4.5
Semi-supervised learning results

As Table 4.5 shows, the effect of applying self-training is mostly negative. The reported scores have higher variance and little performance gain except for the first approach. The first approach produces the best results but is also the most restrictive in the entity selection process. The other two results degrade the performance of the classifier. Since they both take in new entities, we will place both processes under the assumption that they suffer from the same issues. The behaviour of the classifier is similar with these two strategies even though the marking strategy is different.

Result Analysis

The first approach carries the least amount of risk and yields a small increase in performance. Here is an example of how selecting the same entity pair can produce a new sentence with a relevant context:

(2) *this is a short film$_{e1}$ with 12 minutes duration on the life$_{e2}$ of a live statue , a mimo - on the streets of florence , italy .*
label=`Message-Topic`

(3) *in 1965 , he directed the film$_{e1}$ andrei rublev about the life$_{e2}$ of andrei rublev , the 15th century russian icon painter*
label=`Message-Topic` with confidence=.998

The first example is a SemEval training sample and the other example is an unlabeled sample with the same entity pair. The new sentence provides a new context around the relation of the two entities. By using a CNN we can expect that the classification vector p_z is translationally invariant to variations in the input due to the pooling layer. In other words, the salient connections between the two entities that mark the selected `Message-Topic` relation seems captured for both examples. The smoothness assumption, which is required for semi-supervised learning, holds in the sense that these similar examples have the same label. The disadvantage of this appropach is that a low number of samples are taken in. By only selecting matching entity pairs, we might also increase the risk of overfitting on the SemEval training set since we only produce new samples that are directly matching the training set.

Applying self-training to the second and third dataset causes the performance of the network to degrade significantly. An interesting observation is the high increase in variance, particularly for the Antonym dataset. In many of the experiment runs, the network breaks down completely once too many incorrect samples are taken in. The main reasoning behind self-training is that the confidence level on new labels must be reliable and higher than the error rate of the seed classifier. When the algorithm breaks down, one reason can then be that the confidence levels are not a reliable estimate of the generalization error because the classifier does not generalize well.

Another interesting observation is the low number of extra iterations the self-training algorithm can run before it is terminated. Because we have a high expectation of the gain at each self-training iterations, there is no reason to wait many iterations before measuring how the classifier performs on the validation set.

We can investigate what the classifier is doing wrong more closely by trying to qualitatively understand why new samples are being labeled incorrectly. Here is a candidate sentence from the WordNet dataset, which is mislabeled:

(4) *a common urban$_{e1}$ legend claims that twinkies have an infinite shelf life$_{e2}$*
label=`Content-Container` with confidence=.9935

The first entity word *urban* is derived from the word *city*, which occurs as an entity in the training set. The problem with deriving the word *urban* from *city* in this context is that it occurs in another part-of-speech than the original word. In this sentence, we cannot expect the classifier to correctly classify the sentence since the relation classification invariant is broken. The RC invariant assumes that entity identification is done correctly. Additionally, with the SemEval dataset, the classifier is only trained to classify relations between nominals. The word urban is not a simple noun phrase in this sentence. We might expect that new entity words are harder to correctly identify because we have to derive them in some way from the original words. How big can the problem be? The number of new entities in Antonym and WordNet datasets are shown in Figure 4.

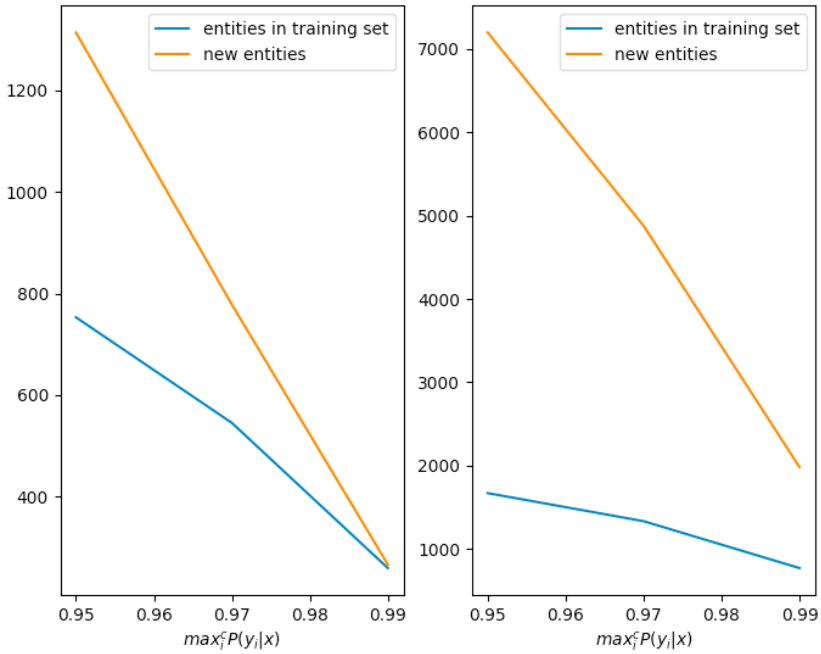


Fig. 4.3
Entities taken in at different confidence levels for the two entity-selection processes. Left: the WordNet approach. Right: the Antonym approach. The number of new entities taken in fall rapidly, showing that the classifier is much more confident with original entity pairs.

Here is another example which shows another interesting observation:

(5) *it allows the person to give tasks to other users*$_{e1}$ *and use the software*$_{e2}$ *for communication*
label=`Instrument-Agency` with confidence=.9943

In this sentence, both entities are nouns in the same sentential clause. The semantic relation `Instrument-Agency` actually appears in the sentence, and is recognized correctly by the classifier. Unfortunately, the relation is between the wrong pair of entity words *person* and *software*. This presents a peculiar problem. Even though the label is correctly recognized, the candidate sentence should not be added to the training set because it adds incorrect information (the entities). However, Wang et al. (2016) showed that the word *use* is a strong indicator of the `Instrument-Agency` relation, and we cannot expect the classifier to classify the entire sentence differently because the entities are not correct.

It seems part of the problem is the absence of the `No-Relation` label in the SemEval dataset. The label is not part of the inventory because of the sample selection process. If it was available, RC would instead be relation extraction which the SemEval dataset is not designed for. So instead, the classifier might choose the most likely relation to appear in the sentence. If we change the marked entities and re-classify the example, the same result is produced but with higher confidence:

(6) *it allows the person*$_{e1}$ *to give tasks to other users and use the software*$_{e2}$ *for communication*
label=`Instrument-Agency` with confidence=.9982

Self-Training Conclusions

The observations from the examples provides suggests that the entity marking process is faulty, since it marks words as entities that fall outside the SemEval specification of target entities. They also suggest that a semi-supervised approach with the SemEval

dataset as the seed dataset is unlikely to work, because the classifier have no possibility of abstaining, even when the sample has no correct answer to choose from the inventory.

The way neural networks outputs a label may also affect the self-training algorithm negatively. The number of neurons in the output layer will determine the available classes. Softmax classification always yields a distribution over the available outputs, which means that confidence values must be interpreted in relation to the other possible labels. When the network produces a 99.43% probability that the `Instrument-Agency` is the correct relation even when the entities is wrong, it is effectively comparing this label to the other available labels. In contrast, a rule-based classifier does not need to interpret its value conditionally to other labels and can abstain if it does not have a high enough precision. Finally, the conclusion to objective 6 is that a classifier trained on the SemEval data is not robust to self-training.

These conclusions solicits two discussions on the semantic relation classification task and the usage of the SemEval dataset. These will be presented in the following sections.

4.5 SemEval discussion

This section presents a retrospective investigation and discussion of the SemEval dataset as a response to the results in the previous section. SemEval is the most popular dataset for relation classification and studies will in many cases value their performance exclusively on this dataset. A Google Scholar search turns up 158 citations on the description of the dataset[3]. All machine learning approaches hinges on assumptions on the distribution of data we are examining. In the semi-supervised setting, we discussed in section 2.2.1 how the distribution of the unlabeled data must be similar to the labeled data. Since we cannot measure the distribution of labels in the unlabeled directly - since it is unlabeled - we must rely on measuring the input and approximating the labels.

4.5.1 Perplexity

One way we can compare the input distributions of the labeled SemEval data with unlabeled data is by measuring *perplexity* (Jurafsky and Martin, 2000, 4.4). The perplexity $P_e(m)$ of a model on the training set is the normalized log probability of a test set of words $t = w_1 \dots w_n$:

$$P_e(m, t) = \sqrt[n]{\frac{1}{m(t)}} \tag{4.2}$$

where $m(t)$ is the probability estimate of the sentence given a language model m. In other words, a high perplexity value for a model means that the language model poorly predicts the test set from the training set. To use this approach to compare the distribution of two datasets, we fix the parameters for a specific language model and measure how well it performs on two different sets. Here, the $m(t)$ we will use is the n-gram model, which is a model that estimates the conditional probability of each word in t given the previous $n - 1$ words in the sentence. For the purpose of this examination, a bi-gram ($n = 2$) model is created. The model is used to compare the perplexity scores between the SemEval train set and test set. The process is repeated with train and test sets from W2C Web and W2C Wiki corpora. The W2C Web corpus is used here instead of the W2C Wiki corpus to get as close to the data generation process of the SemEval sets as possible. All candidate sentences in the SemEval set are found by Web searches as detailed in subsection 2.1.2. While using the W2C Web corpus could distort the analysis because it is not the corpus used to obtain candidate unlabeled data, the W2C Wiki corpus provides approximately the same results. The training sets are also cross-tested on the test sets from the other corpora, and the results are shown in Table 4.6.

[3]Here is a shortlink for the search: `goo.gl/63r2Uc`

	W2C-Web-test	**SemEval-test**	*(W2C-Wiki-test)*
W2C-Web-train	2693	2020	2285
SemEval-train	3540	**1463**	2183
(W2C-Wiki-train)	3139	1958	1912

Table 4.6
Perplexity scores for different datasplits of W2C and SemEval.

From Table 4.6 we observe that given a SemEval training set, it is considerably easier to predict a SemEval test set than doing the same with the W2C Web corpus. Pedicting a W2C Web test set from a SemEval training set is considerably harder, which suggests that the SemEval training set does not generalize well.

An interesting observation is that predicting the SemEval test set from the W2C Web training set is roughly as difficult as predicting the W2C test set. This suggests that the W2C Web training set is more aligned with the "true" distribution of random text. From the "outlook" of model generated on the W2C training set, the SemEval test set looks just as random as the W2C Web test set. Conversely, the SemEval training set is "overfitted" to the SemEval test set. This is possibly because the sample selection process is the same for the SemEval training and test sets, but it is heavily biased.

4.5.2 Sample Selection Process

The original intention of the dataset was to include equal amounts of samples for each semantic relation (Hendrickx et al., 2009). However, it was reported that finding enough samples for specific semantic relations such as the `Instrument-Agency` proved too difficult which resulted in a skewed distribution. The annotation process additionally began with a pattern-based pre-selection. Pre-selection creates a new data distribution which is very different from randomly selecting sentences from the Web and including them in the dataset. When it is hard to balance classes equally in the pre-selection process, it probably means that the classes are not equally balanced at all in general. If this is the case, artificially constructing another distribution in the training set hurts the robustness of the classifier when applied to less restricted, unlabeled data.

Entity Distribution

It is hard to directly quantify how biased the sample selection process makes the SemEval data. Another way of approximating the effect of the sample selection process is to look at the entity pairs in the SemEval training set grouped by their relation class. We can compare the distribution of entities in the SemEval training set, for which we know their corresponding label, with the same entities when they occur in the W2C Web corpus. If the assumption that entity pairs *usually* appears in the same label is true, this comparison can approximate the distribution of labels in the W2C Web corpus. The distributions are shown in Figure 4.5.2.

The figure show a large skewness for some classes when compared to the W2C Web corpus. The classes which are most problematic is the `Content-Container` relation, which is present in 6.8% of the training data but only 1.9% in new examples, and the `Instrument-Agency` relation, which have 3.2% entity occurences in W2C Web and 6.3% in SemEval.
While the approach above is probably not optimal to measure how classes are truly distributed in the W2C Web, but it actually approximates the distribution of classes in the SemEval very well. Using the entity pair distribution as a class distribution estimator yields an average error of 1.13%. If the approximation has any merit for the unlabeled data, it can contribute to the explanation of why the semi-supervised approach does not work here.

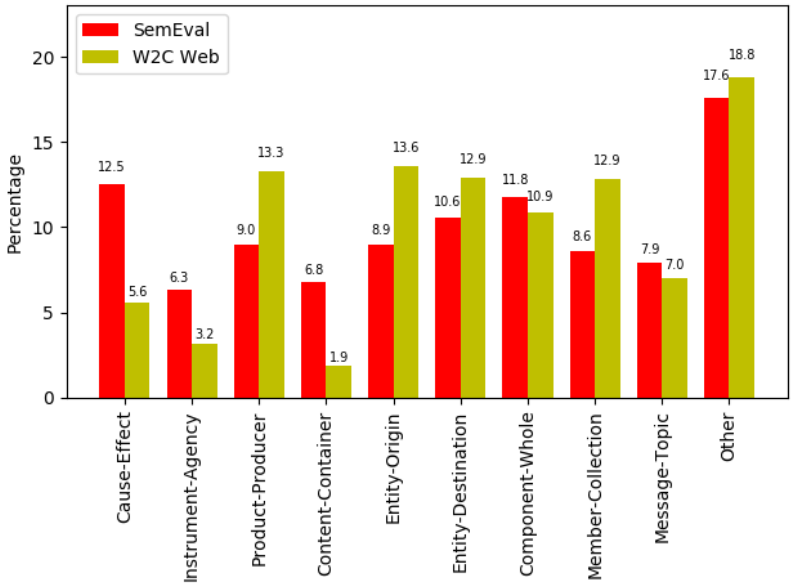


Fig. 4.4

The distribution of entities for each class. Left: Percentage of entities that belong to each label in the full SemEval training set. Right: The same entities occuring in the W2C Web corpus.

The discussion does not conclusively show that the SemEval data is biased, but it strongly suggests that semi-supervised approaches that augments SemEval by using the W2C Wiki and W2C Web corpora will not work. The classifier which is trained on the SemEval training set will be much more biased towards the SemEval test set, and estimating the generalization power on the test set yields an exaggerated result.

4.5.3 Dataset Recommendations

My suggestion, based on the empirical evidence in the previous section, is to avoid using the SemEval dataset in the future to investigate robust relation classification. The problem is then which alternative to use instead. A suggestion to an alternative is to create a new, modified version of the SemEval dataset which have more unfiltered samples and also extends the relation to include the `No-Relation` label. The quality of the samples in SemEval is not poor, but the process in which they were selected is probably too biased and should be generalized. The focus of any new relation classification dataset should be to make it generally robust by avoiding complicated entity indentification and pre-selection of candidate sentences.

Any continued use of the SemEval dataset should also be modified. Here is a list of suggested modifications:

- **Dont use the official test-train split**: Keeping a test set of such a small size (2717 samples) will invariably leak implicit information from the test set into how networks are constructed. When a model produces on a high score on the SemEval test set, it will naturally be argued that it is a strong model architecture. When a series of papers are published - each with marginally better results than the previous - it is hard to distinct what is statistically significant results and what is *overfitting as a community* on the test set. Overfitting as a community happens when many different groups of researchers works on a specific test set and compete for scores. While the literature is sparse on the subject, the effect have been reported in machine-learning competitions (Park, 2012). The traditional approach in machine learning, which should be used instead, is to mix all the official test data with the training data, and create new tests sets for each experiment. If the model is so sensitive that splits have

considerable effect on the performance of the model, it is biased and should not be used as a measure of the generalization error. Creating a new test-set for each experiment run is a central part of cross-validation, which should be the standard for how baseline models are measured.

- **Dont use the official scorer**: The official scorer has been designed to fairly compare models that use the dataset, but the assumptions that it place on the model and have significant impact on how models are designed. The obvious example is the class weights from subsection 4.3.2. The intrinsic measure used for the model should depend on the scenario in which the model should be used. Probably some relation labels, like `Cause-Effect`, are much more common in natural language than others, like the `Instrument-Agency`. It might then be more appropriate to use the $F1$-micro variant instead because it places a different weighting on the distribution of the relations.

The suggestions in this section are based on the empirical evidence that the generalization ability of a classifier trained on SemEval is poor. An entirely unexplored angle in this thesis is the validity of the actualy semantic inventory designed for the SemEval dataset. Santos et al. (2016) provides a recent critique of the semantic inventory used in the SemEval, which overlaps the issues detailed here, because it also questions the sample selection process and the generalization properties of the dataset.

4.6 Task Discussion

The second response to the results in section 4.4 is a retrospective discussion of the relation classification task itself. The reason why the task is unsuited for semi-supervised learning is because getting unlabeled data which upholds the necessary invariants for relation classification is hard to get. Candidate entity selection seems a hard task to do correctly on unlabeled data without manual intervention. The two approaches listed in section 4.4 for getting new entities is not accurate enough. Even if the entities could be marked more accurately, it is still hard to believe that relation classifiers, when trained on the SemEval dataset, can be robust. Relation classifiers cannot abstain in the face of a `No-Relation` sample. This central property of relation classification undermines the validity of using it in a semi-supervised setting or on any unlabeled data in general that is not pre-selected.

The focus on the SemEval dataset in recent years also show a lack of extrinsic measuring of how useful relation classification is in general. To the best of my knowledge, there are no documented studies that show the usefulness of relation classifiers (trained on SemEval data) in a practical setting. The assumption that increased performance on the SemEval dataset yields an increased performance on another extrinsic measure is undocumented.

4.6.1 Modifications to Relation Classification

While intrinsic measures such as the $F1$ are easy to quantify and measure on the test set, they are also much harder to verify the usefulness of. Semantic relation classification is particularly vulnerable as a task to this problem because downstream applications are not discussed in the studies that design the neural network architectures.
In retrospect of the research carried out in this thesis, here are some suggestions for how research on relation classification could be done in a better way:

- **Use Extrinsic Validation**: The most immediate change is to always include an extrinsic validation procedure for relation classification systems. This suggestion have been highlighted for NLP subtasks before, and it is possible to create automated procedures that better establish the link between improvements in intrinsic measures and the actual usefulness of the task (Belz, 2009). For example, a basic question

answering system could be constructed which uses the semantic relation classification system as input. Integrating a downstream application into future research is a time-consuming task, but the specificity of relation classification requires it.

- **Always do Relation Extraction**: To make the semi-supervised setting more viable, the classifier should always be trained to detect the presence of relations between entities. This changes the RC task to the relation extraction task, which is more inclusive and then probably a more natural choice for semi-supervised learning. The semantic inventory can still be the same as in RC. The common dataset RE is the subsection 2.1.2, but that dataset does not annotate semantic relations with the exclusivity that the inventory in SemEval does. If extrinsic validation could show that the SemEval inventory was useful, it could be used in a semi-supervised setting but with an added `No-Relation` class. The dataset for relation extraction should include samples with `No-Relation`proportional to how it occurs in unlabeled data without pre-selection.
- **RC as pure Sentence Classification**: Depending on the extrinsic validation of RC or RE, detecting the presence of a specific semantic relation in a sentence - without knowing the entities beforehand - might be what we actually want from the task. RC could be redefined as the following sentence classification problem: Given the semantic relation labels $\mathcal{Y}$ and a sentence S, which semantic relation $y \in \mathcal{Y}$ is the most "dominant" in S? This definition of RC removes the need for entitiy detection. The complicated entity selection process can be removed from the semi-supervised setting, which increases the probability of the semi-supervised smoothness assumption holding for new, unlabeled data. The SemEval data could be modified and extended by simply removing the marked entities from the training set, and checking if that modifies the semantic relation.

The list of suggestions above is different from the list of future work in section 4.7, because it is part of objective 7. The suggestions are intended to be general advice for everybody who considers investigating model architectures for semantic relation classification. It is imperative that downstream applications be discussed and the appropriate datasets and measurements chosen in accordance. With the current state of relation classification the link between the task and the usefulness is weak and should be strengthened.

4.7 Future Work

The research chapter concludes with a list of suggestions for future work in the context of the thesis.

- An downstream task could be implemented and the connection between relation classification and the task monitored. The intrinsic measures are then accompanied by scoring related to the downstream task.
- Convolutional neural networks are still at a rather simple stage for sentence classification. There are several options for improving the CNN architecture for RC that can be investigated. For example, attention mechanisms is a neural network mechanism which has been applied for RC but needs more substantial investigation.(Wang et al., 2016). Another topic of investigation is the depth of the convolutions, which until now have been mostly restricted to 1. Deep convolutions have consistently performed at state-of-the-art for image recognition tasks (Szegedy et al., 2015).
- More semi-supervised approaches to RE in general can be investigated. Self-training is a very basic approach with many variants, but there is also several families of semi-supervised learning algorithms which could be relevant for RC. Some examples are *co-training, expectation maximization* and *label propagation* (Chapelle et al., 2009; Yu et al., 2011, 3,5,11).

Chapter 5

Conclusion

This thesis has investigated relation classification with convolutional neural networks in a supervised and semi-supervised setting. The theoretical grounding for machine learning and different neural network structures was presented, and experiments were done to demonstrate the effectiveness of the convolutional neural network as a baseline model for relation classification. When reporting the scores for the baseline, a new statistic have been suggested which allows users of the SemEval dataset to use the same designated test set but with more robust reporting, which allows for better comparisons. We have also discussed the problems with reproducing the baseline and, as a measure to prevent the same thing from happening with the experiments in this thesis, provided an open-source implementation that makes results easy to reproduce.

The effect of simplifying the baseline model, by removing the l_2-based weight norm constraint from the model and reducing the dimensions of the positional embedding to 1, have been investigated. Both experiments showed that the simplifications can safely be made without loss of performance.

We have investigated how common techniques such as early stopping and class weighting provides small but significant performance improvements when applied to the baseline model. Multiple approaches to gathering new candidate sentences, which could be used in a semi-supervised self-training setting, have been investigated. For the current definition of relation classification, the SemEval dataset, and the entity selection processes which were investigated, the performance degrades significantly. We have investigated examples of why this is - because the entity selection and candidate sentence selection in SemEval is very strict, and because unlabeled data probably cannot be reliably sampled so that marked entities always have a semantic relation.

Finally, we have investigated the SemEval dataset generally and questioned its usefulness. The lack of reported downstream applications raises a doubt whether classifiers trained on this data could ever be used for production quality applications. The input distribution of the SemEval dataset is nothing alike raw unlabeled data, and the design of the semantic inventory may have influenced the construction of the dataset so that some classes are appearing at an unnatural frequency compared to other text corpora. The semantic relation classification task needs redefining or work that proves that it is useful in other applications. While the argument for using SemEval might be: ”it is a general-purpose academic dataset that can compare model architectures ”, we have observed that the scores on the designated testset is not a reliable estimate of generalization. The current usage of the dataset might lead to implicit overfitting on the testset, which consequently builds unreliable faith in specific model architectures because they are not tested on other datasets and tasks.

Relation classification is a fundamental task in NLP with a *theoretically* sound connection to language modeling and applications in various business domains. It is the intention of this thesis to help demonstrate some fundamental problems in how the research community investigate it and to propose changes which can improve the state of the task.

Appendices

Appendix A

Appendix

A.1 Network Specification

The code can be run be following the instructions at the GitHub repository[1]

Preprocessing Preprocessing converts sentences to integer mappings. The preprocessing uses a custom tokenizer which filters the following tokens:

```
'!"#$%&()*+,-.:;=?@[\\]^_`{|}~\t\n'
```

The tags surrounding entites in the SemEval dataset are removed so they are not a part of the input sentence. Each input sentence is clipped to have a maximum length after tokenization of 15. Tokens that are not filtered with the above pattern are taken as words. Each word is translated to lower case. Entities that are more than one word in length are pasted together with a dash "-".

Embeddings The word-embeddings are 300-dimensional vectors which are pre-trained and publicly available. They can be downloaded here[2]. They are trained using the `word2vec` algorithm. The positional embeddings are 50-dimensional and initialized with xavier initalization. Out-of-vocabulary vectors are initialized uniformly randomly in the range $[-0.25, .25.]$ during training. At test time, unknown words are translated to one specific OOV-vector which is the same for all unknown words.

Convolutional Layers The window configuration is: $[2, 3, 4, 5]$. The filter size is: 150. The kernels uses *valid padding* and height 1 for the convolution. The stride is also 1. The activation function is the "ReLu" function. The kernels are initialized using xavier uniform initialization. Each convolutional kernel has a bias vector which are initialized to zeros.

The l_2-norm is $s = 3$

Max-Pooling Each window in the convolution is maxpooled individually and concatenated to form z. They have no hyperparameters since they downsample the entire window.

Fully-Connected Dropout The dropout rate p is: 0.5. The weights in fully-connected layer are initialized with xavier uniform initialization. Bias weights are initialized to zero

[1] `https://github.com/msta/cnn_rc`
[2] word-vector

Training The l_2-norm is $s = 3$ for the weight constraints. The loss function used when training is categorical crossentropy.

The AdaDelta rule uses the following parameters:

- Initial learning rate: 1.0.
- p-value which controls momentum decay: 0.95
- learning rate decay is: 0.0.

Evaluation Is done by writing all scores for the test set and using the official scoring tool. No test samples are excluded from prediction.

Early Stopping Patience is 4, validation set size is 30%.

Class Weights The class weights directly proportional to the distribution of samples in the training set.

List of Tables

List of Figures

Bibliography

Abney, S. (2007). *Semisupervised learning for computational linguistics*. CRC Press.

Abu-Mostafa, Y. S., Magdon-Ismail, M., and Lin, H.-T. (2012). *Learning from data*, volume 4. AMLBook New York, NY, USA:.

Bach, N. and Badaskar, S. (2007). A review of relation extraction. *Literature review for Language and Statistics II*.

Batista, D. (2017). Datasets of annotated semantic relationships.

Belz, A. (2009). That's nice what can you do with it? *Computational Linguistics*, 35(1):111–118.

Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155.

Bunescu, R. and Mooney, R. J. (2005a). Subsequence kernels for relation extraction. In *NIPS*, pages 171–178.

Bunescu, R. C. and Mooney, R. J. (2005b). A shortest path dependency kernel for relation extraction. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 724–731. Association for Computational Linguistics.

Chan, Y. S. and Roth, D. (2011). Exploiting syntactico-semantic structures for relation extraction. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 551–560. Association for Computational Linguistics.

Chapelle, O., Scholkopf, B., and Zien, A. (2009). Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. *IEEE Transactions on Neural Networks*, 20(3):542–542.

Chen, Y., Lu, Y., Lan, M., Su, J., and Niu, Z. (2010). A semi-supervised method for classification of semantic relation between nominals. In *Asian Language Processing (IALP), 2010 International Conference on*, pages 146–149. IEEE.

Collins, M., Duffy, N., et al. (2001). Convolution kernels for natural language. In *NIPS*, volume 14, pages 625–632.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Culotta, A., McCallum, A., and Betz, J. (2006). Integrating probabilistic extraction models and data mining to discover relations and patterns in text. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 296–303. Association for Computational Linguistics.

Delalleau, O. and Bengio, Y. (2011). Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674.

Ebrahimi, J. and Dou, D. (2015). Chain based rnn for relation classification. In *HLT-NAACL*, pages 1244–1249.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, 14(2):179–211.

Freund, Y. and Schapire, R. E. (1999). Large margin classification using the perceptron algorithm. *Machine learning*, 37(3):277–296.

Girju, R., Nakov, P., Nastase, V., Szpakowicz, S., Turney, P., and Yuret, D. (2007). Semeval-2007 task 04: Classification of semantic relations between nominals. In *Proceedings of the 4th International Workshop on Semantic Evaluations*, pages 13–18. Association for Computational Linguistics.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256.

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

GuoDong, Z., Jian, S., Jie, Z., and Min, Z. (2005). Exploring various knowledge in relation extraction. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 427–434. Association for Computational Linguistics.

Gurulingappa, H., Rajput, A. M., Roberts, A., Fluck, J., Hofmann-Apitius, M., and Toldo, L. (2012). Development of a benchmark corpus to support the automatic extraction of drug-related adverse effects from medical case reports. *Journal of biomedical informatics*, 45(5):885–892.

Hendrickx, I., Kim, S. N., Kozareva, Z., Nakov, P., Ó Séaghdha, D., Padó, S., Pennacchiotti, M., Romano, L., and Szpakowicz, S. (2009). Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions*, pages 94–99. Association for Computational Linguistics.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hoffmann, R., Zhang, C., Ling, X., Zettlemoyer, L., and Weld, D. S. (2011). Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 541–550. Association for Computational Linguistics.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257.

Jurafsky, D. and Martin, J. H. (2000). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*.

Kate, R. J. and Mooney, R. J. (2007). Semi-supervised learning for semantic parsing using support vector machines. In *Human Language Technologies 2007: the Conference of the North American Chapter of the Association for Computational Linguistics; Companion Volume, Short Papers*, pages 81–84. Association for Computational Linguistics.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Kiss, T. and Strunk, J. (2006). Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525.

LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.

LeCun, Y. and Cortes, C. (2010). MNIST handwritten digit database.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, 2(Feb):419–444.

Majlis, M. and Zabokrtskỳ, Z. (2012). Language richness of the web. In *LREC*, pages 2927–2934.

McClosky, D., Charniak, E., and Johnson, M. (2006). Effective self-training for parsing. In *Proceedings of the main conference on human language technology conference of the North American Chapter of the Association of Computational Linguistics*, pages 152–159. Association for Computational Linguistics.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

Miwa, M. and Bansal, M. (2016). End-to-end relation extraction using lstms on sequences and tree structures. *arXiv preprint arXiv:1601.00770*.

Nguyen, T. H. and Grishman, R. (2015). Relation extraction: Perspective from convolutional neural networks. In *Proceedings of NAACL-HLT*, pages 39–48.

Nielsen, M. (2017). How the backpropagation algorithm works. `http://neuralnetworksanddeeplearning.com/chap2.html`.

Park, G. (2012). The dangers of overfitting or how to drop 50 spots in 1 minute.

Pedersen, T. (2008). Empiricism is not a matter of faith. *Computational Linguistics*, 34(3):465–470.

Plank, B. and Moschitti, A. (2013). Embedding semantic similarity in tree kernels for domain adaptation of relation extraction. In *ACL (1)*, pages 1498–1507.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, 46(1):77–105.

Qian, L., Zhou, G., Kong, F., and Zhu, Q. (2009). Semi-supervised learning for semantic relation classification using stratified sampling strategy. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 3-Volume 3*, pages 1437–1445. Association for Computational Linguistics.

Qian, L., Zhou, G., Kong, F., Zhu, Q., and Qian, P. (2008). Exploiting constituent dependencies for tree kernel-based semantic relation extraction. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 697–704. Association for Computational Linguistics.

Quiroga, R. Q., Reddy, L., Kreiman, G., Koch, C., and Fried, I. (2005). Invariant visual representation by single neurons in the human brain. *Nature*, 435(7045):1102–1107.

Rink, B. and Harabagiu, S. (2010). Utd: Classifying semantic relations by combining lexical and semantic resources. In *Proceedings of the 5th International Workshop on Semantic Evaluation*, pages 256–259. Association for Computational Linguistics.

Santos, C. N. d., Xiang, B., and Zhou, B. (2015). Classifying relations by ranking with convolutional neural networks. *arXiv preprint arXiv:1504.06580*.

Santos, V., Huerliman, M., Davis, B., Handschuh, S., and Freitas, A. (2016). Semantic relation classification: Task formalisation and refinement. In *Proceedings of the 5th Workshop on Cognitive Aspects of the Lexicon (CogALex - V)*, pages 30–39, Osaka, Japan. The COLING 2016 Organizing Committee.

Schmidhuber, J. (2003). Long short-term memory: 2003 tutorial on lstm recurrent nets.

Socher, R., Huval, B., Manning, C. D., and Ng, A. Y. (2012). 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. Association for Computational Linguistics.

Srinivas, S., Sarvadevabhatla, R. K., Mopuri, K. R., Prabhu, N., Kruthiventi, S. S., and Babu, R. V. (2016). A taxonomy of deep convolutional neural nets for computer vision. *arXiv preprint arXiv:1601.06615*.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958.

Sun, A., Grishman, R., and Sekine, S. (2011). Semi-supervised relation extraction with large-scale word clustering.

Sun, L. and Han, X. (2014). A feature-enriched tree kernel for relation extraction.

Surdeanu, M., Tibshirani, J., Nallapati, R., and Manning, C. D. (2012). Multi-instance multi-label learning for relation extraction. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 455–465. Association for Computational Linguistics.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9.

Walker, C., Strassel, S., Medero, J., and Maeda, K. (2005). Ace 2005 multilingual training corpus. `https://catalog.ldc.upenn.edu/LDC2006T06`.

Wang, L., Cao, Z., de Melo, G., and Liu, Z. (2016). Relation classification via multi-level attention cnns. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics. Association for Computational Linguistics*.

Xu, K., Feng, Y., Huang, S., and Zhao, D. (2015a). Semantic relation classification via convolutional neural networks with simple negative sampling. *arXiv preprint arXiv:1506.07650*.

Xu, Y., Jia, R., Mou, L., Li, G., Chen, Y., Lu, Y., and Jin, Z. (2016). Improved relation classification by deep recurrent neural networks with data augmentation. *arXiv preprint arXiv:1601.03651*.

Xu, Y., Mou, L., Li, G., Chen, Y., Peng, H., and Jin, Z. (2015b). Classifying relations via long short term memory networks along shortest dependency paths. In *EMNLP*, pages 1785–1794.

Yu, S., Krishnapuram, B., Rosales, R., and Rao, R. B. (2011). Bayesian co-training. *Journal of Machine Learning Research*, 12(Sep):2649–2680.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zelenko, D., Aone, C., and Richardella, A. (2003). Kernel methods for relation extraction. *Journal of machine learning research*, 3(Feb):1083–1106.

Zeng, D., Liu, K., Lai, S., Zhou, G., Zhao, J., et al. (2014). Relation classification via convolutional deep neural network. In *COLING*, pages 2335–2344.

Zhang, J., Salwen, J., Glass, M., and Gliozzo, A. (2014). Word semantic representations using bayesian probabilistic tensor factorization.

Zhang, Y. and Wallace, B. (2015). A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*.

Zhou, G., Zhang, M., Ji, D.-H., and Zhu, Q. (2007). Tree kernel-based relation extraction with context-sensitive structured parse tree information. In *EMNLP-CoNLL*, volume 2007, pages 728–736. Citeseer.