An Experimental Analysis of Multi-Perspective Convolutional Neural Networks

by

Zhucheng Tu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2018

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Modelling the similarity of sentence pairs is an important problem in natural language processing and information retrieval, with applications in tasks such as paraphrase identification and answer selection in question answering. The Multi-Perspective Convolutional Neural Network (MP-CNN) is a model that improved previous state-of-the-art models in 2015 and has remained a popular model for sentence similarity tasks. However, until now, there has not been a rigorous study of how the model actually achieves competitive accuracy. In this thesis, we report on a series of detailed experiments that break down the contribution of each component of MP-CNN towards its statistical accuracy and how they affect model robustness. We find that two key components of MP-CNN are non-essential to achieve competitive accuracy and they make the model less robust to changes in hyperparameters. Furthermore, we suggest simple changes to the architecture and experimentally show that we improve the accuracy of MP-CNN when we remove these two major components of MP-CNN and incorporate these small changes, pushing its scores closer to more recent works on competitive semantic textual similarity and answer selection datasets, while using eight times fewer parameters.

Acknowledgements

First and foremost, I would like to thank my advisor, Professor Jimmy Lin, for his continuing guidance and support during the past two and a half years since I knew him, spanning both my undergraduate and Master's studies. Without Jimmy, I would not be writing this thesis today and getting the opportunity to work with so many amazing peers and exposure to interesting research areas.

I would like to thank the readers of my thesis, Professor Ming Li and Professor Mei Nagappan, for reviewing my work.

Special thanks to Laetitia, Ralph, Meng, and Youngbin for the special roles they have played in my life in grad school. I am also extremely grateful to Bai, Alice, Phoebe, and Frank for their friendship and for continuing to motivate and inspire me in grad school.

I was lucky to start my grad school journey just when the newly renovated Data Systems lab was about to open. I will miss the company of amazing colleagues like Royal, Haotian, and Mina who made the lab a lively, energetic, and collaborative place to work. I will also miss the late nights spent in the lab with Peng and Victor.

Finally, I would like to thank my parents for their unwavering encouragement and support through all these years of school.

Dedication

This is dedicated my sister, Emily.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Modelling pairs of sentences to determine their semantic relatedness is a critical problem in natural language processing and information retrieval. In the literature, this is sometimes also referred to as sentence matching. Sentence similarity modelling is important for many tasks, such as paraphrase identification for detecting plagiarism, answer selection for selecting the most relevant answer to a question, and detecting textual entailment or contradiction in natural language inference. In this chapter, we introduce the problem of sentence pair similarity modelling and highlight the contributions and organization of this thesis.

1.1 Sentence Pair Modelling

1.1.1 Paraphrase Identification and Semantic Textual Similarity

In paraphrase identification (PI), the goal is to determine if two sentences express the same or very similar meaning (Figure 1.1). Semantic textual similarity (STS) is similar to paraphrase identification, but instead of a binary target, paraphrase or not paraphrase, semantic textual similarity has the goal of judging the level of semantic equivalence based on a graded scale. For example, a score of 0 could mean there is no relation between two sentences, whereas a score of 5 would mean complete semantic equivalence, with scores 1-4 indicating graded levels in-between.

Paraphrase identification and semantic textual similarity are foundational problems that have importance in many language-related tasks. A few of these include detecting

Charles O. Prince, 53, was named as Mr. Weills successor.

Mr. Weills longtime confidant, Charles O. Prince, 53, was named as his successor.

Figure 1.1: A pair of sentences that are paraphrases of each other, from the Microsoft Research Paraphrase Corpus (MSRP) [9].

plagiarism, detecting semantically-equivalent queries, and machine translation evaluation. Plagiarism detection can be complicated when words are replaced by their synonyms, which might elude methods that use n-gram matching. Likewise, word overlap features are insufficient for identifying semantically equivalent queries due to the variability of linguistic expression. Quora, the English question and answer website, recently released a dataset for a Kaggle competition to ask the community to develop methods that help them identify duplicate questions on their website, to improve the quality of service.[1] In machine translation, the BLEU score is commonly used as an evaluation metric for the translation quality, which measures how many n-grams overlap between a human-generated reference translation and a machine-generated translation, but this evaluation is quite brittle to different ways of expressing the same meaning [5] - determining the semantic equivalence of the two translations could be a better alternative.

1.1.2 Answer Selection

Question: *How many planets is Jupiter away from the sun?*
Candidate Answers:
(+) *Jupiter is the fifth planet from the sun and the largest planet in the solar system.*
(−) *Together, these four planets are sometimes referred to as the outer planets.*
(−) *Mars can briefly match Jupiter's brightness at certain points in its orbit.*

Figure 1.2: An example of answer selection from candidate sentences, from the WikiQA dataset [51]. The relevant answer is indicated by (+) and non-relevant answers are indicated by (−).

[1] https://www.kaggle.com/c/quora-question-pairs

There are two main paradigms in question answering. In question answering over knowledge bases, the approach is to query the answer on structured knowledge bases such as DBpedia and Freebase. In question answering over free text, one can directly work with the free text corpora. Although structured knowledge bases may make it easier for computer to process, they require constructing the knowledge base which arguably contains only a subset of the information found in the text they are constructed from. On the other hand, resources like Wikipedia are rich in information and are constantly updated with new information. Directly enabling question answering over free text is arguably better suited for open-domain questions [7] and detecting textual similarity plays an important role.

A typical free text question answering architecture often contains two phases, a document retrieval phase and answer selection phase [10]. In the document retrieval phase, an information retrieval system retrieves a few relevant documents to a query from a large corpus to reduce the search space, focusing on efficiency and recall. Once the relevant documents are retrieved, the answer is selected from some sentence or passage in the returned documents [37]. Often, the sentences from the documents are ranked using a simple approach, and then the top sentences are re-ranked using a more complex model. The best answers should have high semantic relevance with the question. An example of answer selection is shown in Figure 1.2.

1.2 Contributions

The Multi-Perspective Convolutional Neural Network (MP-CNN) [15] is a neural network based-model that rivals or exceeds the state-of-the-art on paraphrase identification tasks and semantic textual similarity tasks as of 2015, without using any external resources such as syntactic parsers. However, it is not clear what makes the model so effective. There has not been a detailed analysis of to what extent each component of the model contributes to its overall effectiveness.

This thesis makes the following contributions:

- A detailed experimental analysis of MP-CNN, showing the contribution of each choice for each component of the model towards the overall effectiveness, with model robustness in consideration.
- Demonstrate that two key components of the original MP-CNN are not necessary for good accuracy and in fact actually hurt model robustness. A greatly simplified version of the model can actually yield similar or better accuracy as MP-CNN.

- Propose a modified model based on a lightweight MP-CNN model that yields competitive accuracy compared to popular models proposed in 2017 for answer selection and semantic textual similarity with eight times fewer parameters as MP-CNN.

1.3 Thesis Organization

In Chapter 2, we go over related work and background concepts for understanding MP-CNN. In Chapter 3, we go over the model and the choices for each component in detail. Chapter 4 describes the datasets used for evaluation. Chapter 5 goes over the experimental methodology and results for all of the model choices we make. Chapter 6 concludes the thesis.

Chapter 2

Related Work and Background

A general trend of sentence pair modelling over the past decade is the shift from methods that rely on manually engineered features to automatically learned features using distributed representations and neural networks. This chapter provides an overview of sentence pair modelling literature over the years and covers some concepts for understanding MP-CNN, including convolutional neural networks, word vectors, and neural network training.

2.1 Non-Neural Network Methods

Some of the early work use string overlap features as means of identifying sentence similarity. These include features such as bag-of-words overlap, and unigram, bigram, and trigram overlap which are used for computing the BLEU score [48]. However, these approachs suffer from variability of expression, that is, different ways of express the same meaning, and they cannot capture linguistic and semantic constraints (the question "*Who did Charlie call?*" is not answered by "*Bob called Charlie*" although they have high word overlap). In another example adapted from [10], consider the question "*Who is the leader of France?*". Althought the sentence "*Henri Hadjenberg, who is the leader of France's Jewish community, endorsed confronting the specter of the Vichy past*" contains all of the words in question, it does not answer the question. Meanwhile, the sentence "*Bush later met with French President Jacques Chirac*" does answer the question, but it does not contain any of the words in the question. Hence bag-of-words features can be poor for sentence similarity related tasks [42] when used alone.

To help mitigate the problem of the variability of expression, various approaches use lexical resources like WordNet [28] beyond simple lexical matching to take into account words that are different but that are synonyms of each other [11]. A simple lexical match cannot match a word like *consequence* with its synonym *result*, or *incursion* with *invasion*.

In addition, many approach utilize syntactic and semantic structure to capture more of the linguistic and semantic constraints. For example, in question answering, one can build a dependency tree for the question and each of the candidate answers. The dependency tree contains semantic information like named entities. The answers can be ranked based on increasing order of edit distance between the question dependency tree and each answer dependency tree [32]. Alternatively, Wang et al. [49] match dependency trees using the quasi-synchronous grammar formalism developed in [40] to learn a generative probabilistic model that can generate the question from the answer using syntactic and semantic transformations.

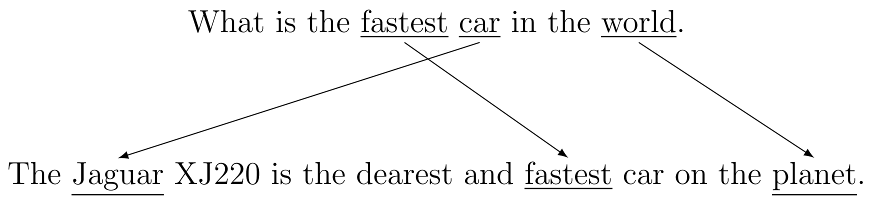

What is the fastest car in the world.
The Jaguar XJ220 is the dearest and fastest car on the planet.

Figure 2.1: There exists an alignment between the pair of sentences that is not explicitly provided. Lines indicate words that are synonyms or hypernyms/hyponyms of each other. Adapted and modified from [53].

One can observe that there is a hidden alignment between the pair of sentences being compared - that is, words and syntax between the two sentences can be matched with each other, as shown in Figure 2.1. However, this alignment is not explicitly provided and deriving the correct alignment is not a trivial task. Structures such as dependency trees are used by previous work to derive the latent mapping structure between the pair of sentences. However, it is not necessary to use trees to model the alignment and relationship between the words in the pair of sentences. Yih et al. devises a latent word-alignment structure model that use lexical semantics to significantly improve the accuracy over the state-of-the-art at the time [53]. Lexical semantics go beyond surface-form matching and considers synonymy and antonymy, hypernymy and hyponymy (IS-A relationship), and semantic word similarity.

Another family of approaches use distributional methods, such as latent semantic analysis (LSA). Guo et al. argues that although techniques such as latent semantic analysis can model word and sentence semantics jointly in low-dimensional latent space, its lackluster accuracy compared with high-dimensional word similarity based models is rooted in the sparsity in the semantic space [13]. Sentences are often short and not suitable for deriving latent semantics. They propose to solve this problem by considering words in the sentence and words not in the sentence - modelling using words not in the sentence can tell what the sentence is not about. Finally, they use weighted matrix factorization to build latent vectors for sentences. This partially inspired Ji et al. to build upon their work by reweighting the individual distributional features and learning the importance of each latent dimension, ultimately obtaining a sentence distributional representation directly instead of starting out from the meaning of individual words [18].

Neural network methods aim to remove the need for manual feature engineering using distributed representations through word vectors and neural network architectures, which are described in the following sections.

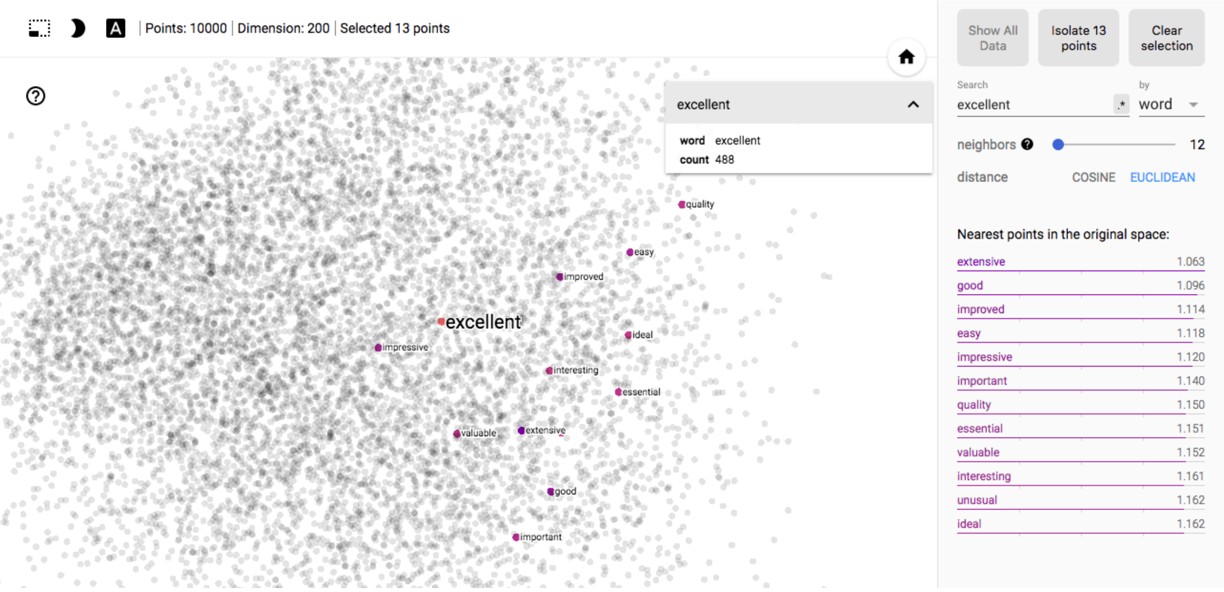
2.2 Word Vectors

Word vectors are dense vectors that can capture word usage via co-occurrence statistics,[1] so that points in the word vector space close to each other co-occur more often together (Figure 2.2). A natural way of representing a word might be using a one-hot representation, which is using a vector having the length equal to the size of the vocabulary with all elements set to zero except a one at the position of the word being represented. However, this representation cannot capture any notion of similarity, since any two vectors are orthogonal to each other. We would like the word representation to intrinsically capture the notion of similarity without resorting to some other resource.

There is an old saying "You shall know a word by the company it keeps". We can use algorithms to automatically create word vectors based on the neighbours of words in a corpus, in contrast with a taxonomy like WordNet which requires manual effort to curate and update. One of the most popular ways of building word vectors is using `word2vec` [27]. There are two algorithms in `word2vec`, one is the skip-grams model which builds a probability model to predict words in a window surrounding some target center word. The words surrounding the target are known as the context words. In the continuous bag of words model we predict the target center word given the context words.

As a brief illustration of how this works, suppose we use the skip-grams method to

[1] https://twitter.com/zacharylipton/status/973379818032128000



Points: 10000 | Dimension: 200 | Selected 13 points

excellent

word excellent
count 488

Search: excellent, by: word

neighbors 12

distance COSINE EUCLIDEAN

Nearest points in the original space:

extensive	1.063
good	1.096
improved	1.114
easy	1.118
impressive	1.120
important	1.140
quality	1.150
essential	1.151
valuable	1.152
interesting	1.161
unusual	1.162
ideal	1.162

Figure 2.2: A projection of the top 10,000 English `word2vec` vectors projected onto 2D space using PCA as visualized on Google Embedding Projector (`http://projector.tensorflow.org/`). The 12 closest neighbours to the word *excellent* is shown, by increasing order of Euclidean distance in the original space. One can see that they are semantically similar to *excellent*.

generate word vectors.[2] We define an objective function that we want to maximize: the probability of observing the context words surrounding a target word w_t for all words in the corpus $w_1, w_2, ..., w_T$. The context of a word is given by the m words that precede the word and m words that follow a word. Mathematically, this objective function is given by:

$$J(\theta) = \prod_{t=1}^{T} \prod_{-m \leq j \leq m, j \neq 0} P(w_{t+j}|w_t; \theta) \tag{2.1}$$

where the words are represented by vectors parameterized by θ. One way of formulating the probability $P(w_{t+j}|w_t;\theta)$ is by using a softmax. Suppose we have two vectors for representing each word, where one is used when it is the center target word, denoted by v_c, and the other is used when it is a context word, denoted by u_o (o for outside). Then, the softmax formulation is:

[2] We use the same notation as that in `http://web.stanford.edu/class/cs224n/index.html`, which has a much more in-depth discussion of word vectors.

$$P(w_o|w_c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)} \tag{2.2}$$

Intuitively, to maximize this probability, we want the vectors u_o and v_c to be similar to each other, so their dot product is bigger. The denominator takes the sum of probabilities of dot products, or similarity, between the center word vector and every context word vector. We are encouraging the center word vector and context word vector to be similar to each other to maximize the probability/objective.

Since we have large corpora of texts, we have a lot of "free" training data in the sense that we do not need any labels for training word vectors - the center words and their surrounding context words can be picked directly from the text itself. We can use the formulation we have established so far to predict the surrounding words of some center word, and calculate the difference from the true surrounding words to obtain a loss. Then, we can use backpropagation to optimize the parameters of the word vectors to minimize the loss. Note that the formula presented above is a simplified model to get an intuition of how word vectors are trained. In practice, more efficient methods such as negative sampling is used instead of a simple softmax approach.

There exist pre-trained word vectors on extremely large corpora shared with the community to use. Popular pre-trained word vectors include `word2vec` trained on Google News, GloVe [31] trained on Common Crawl/Wikipedia/Twitter, and fastText [19] trained on Common Crawl/Wikipedia. Whereas `word2vec` captures the co-occurrence of words with words in their surround context implicitly by using the target word to predict its surrounding words, GloVe is motivated by capturing the co-occurrence counts directly. Typically, a count-based approach is to build a large co-occurrence matrix and then factorizing it using a technique like Singular Value Decomposition (SVD), except this does not scale well. GloVe combines the advantages of prediction based approaches, such as scalability to large corpora and capturing extra information in addition to word similarity, with advantages of count-based approaches such as faster training and efficient usage of statistics. For fastText, one of its main features is that it treats words as character n-grams, enabling the creation of better word vectors for rare words and even enabling the creation of word vectors for out-of-vocabulary words.

2.3 Sentence Representation

Using dense vectors to represent words naturally leads to us to the question of whether we can also represent longer pieces of text such as sentences and documents as vectors. To effectively compare the similarity of sentences, it is important to be able to represent sentences in a way that is rich and capture their subtleties.

A simple idea to represent a sentence is to use a bag-of-words approach and simply add up all the word vectors of words in the sentence. Arora et al. [1] recently showed that simply taking the weighted sum of word vectors where the weight is a smoothed version of the estimated inverse word frequency and then removing a special direction of the weighted sum (denoising) yields an extremely strong baseline for sentence similarity tasks that can beat complicated approaches such as Long-Short Term Memory (LSTM) models. This is despite the fact that taking a weighted sum of word vectors completely ignores the order of words in the sentence. However, it should be noted that the relative position of words in some local region of the sentence can be important, such as negation. Although bag-of-words approaches may pick up individual words in some sentence such as "not" and "forgiving", the meaning of the phrase "not forgiving" is lost when the word vectors are summed up over words in the entire sentence.

As a potential solution to this problem, researchers explored using bag of n-grams instead of bag-of-words. Instead of taking a sum over individual word vectors in a sentence, we now take the sum over vectors for each n-gram. This way, the locality of combination features such as negation is captured and this can deal with phrases such as "not forgiving". This means we need a vector representation for each n-gram. A naive approach is to build for example a bigram or trigram embedding instead of word embedding, but this will cause an explosion in the embedding size. There will be more and more combinations of words in a n-gram as n increases, and this cannot scale [12]. Furthermore, this does not capture any relationships between similar n-grams - "perfect display" and "excellent display" may be similar in meaning, yet their embeddings are completely independent of each other. Ideally, we want some parameter sharing between n-grams are similar to each other to reduce the number of parameters, and this is where convolutional neural networks can help.

2.4 Convolutional Neural Networks

Convolutional neural networks (CNNs) can be thought of feature detectors that are good at detecting indicative local patterns with some freedom with respect to the position of the local patterns in the input, and they can combine multiple local patterns to create fixed-size feature representations. In the context of NLP, this means that convolutional neural networks are able to detect indicative n-grams in the input for the task it is trained on. CNNs eliminate the need to use separate parameters for different n-grams and are able to deal with n-grams of different words that are similar in meaning effectively. The following section gives a previous overview of the fundamental operations and terminologies of convolutional neural networks in NLP.

CNNs consist of two types of operations: convolution and pooling. Imagine a sliding window of width k that slides along the words in a sentence, where the sentence has length $|s|$, which is the number of words in the sentence. Then there are $|s| - k + 1$ possible width-k windows in the sentence. The main idea of convolution is we want to apply a non-linear, learned function to each of the windows, where the function returns a scalar value that captures some information about the particular window. Each function corresponds to a **filter**, also known as a kernel, in a CNN. Applying the filter to each position in the sentence yields a length $|s| - k + 1$ vector, also known as a convolutional feature map. There are multiple functions or filters that are used to detect different kinds of n-gram patterns, hence if we have l filters then we will have l feature maps, which are vectors of $|s| - k + 1$ in length each.

In this section we will keep the discussion of convolution at a fairly high-level, and describe the mathematical details in the next chapter when introducing MP-CNN. Essentially, each sentence can be represented by a matrix, also known as a sentence embedding, by concatenating the word vectors for each token in the sentence together following the same order as the tokens themselves. Hence, if the word vectors are d dimensions long, we have a $d \times |s|$ sentence embedding. Each filter is parameterized by a matrix of size $d \times k$, where $k < |s|$, and a scalar bias. Hence, the filter can be slid across the sentence embedding (in the row direction), creating an output at each window. At each position, the dot product between the filter and the sentence embedding region it overlaps with is taken and added to the bias, and the result is passed into a non-linearity function such as hyperbolic tangent (`tanh`) or the rectified linear unit (`ReLU`).

Recall that CNNs return a fixed-size feature representation. The size of each convolutional feature map depends on the size of the input. Generally for each feature map, we only want to keep the most important or indicative information about the input. Pooling in convolutional neural networks is a function applied to convolutional feature maps and it has several goals. It preserves the most important information from the convolutional feature maps and discards the other information which are irrelevant, making the representation compact while eliminating the noise. An effect of this is that it produces a fixed-size output when it follows a convolution layer, regardless of size of the input to the convolution layer, since the pooling function returns a fixed-size output regardless of its input. The two most popular types of pooling functions are **max pooling** and **average pooling**, also known as mean pooling. Max pooling returns the maximum element in every convolutional feature map. One can see that regardless of the position of the maximum element in the feature map, max pooling returns the same result. Hence, max pooling allows a feature to appear anywhere in the input. Average pooling returns the average of the elements in the convolutional feature map, hence indicating the prevalence of a feature detected by the filter over the entire input rather than anywhere in the input, unlike max pooling. With max or average pooling, if there are l filters, each detecting a different kind of pattern, then the output is a length l vector, which doesn't depend on the length of the input sentences.

Convolutional neural networks are only feature detectors and not particularly useful when used alone. They are typically used together with fully-connected networks. We can concatenate the results obtained by pooling different feature maps into a feature vector. The feature vector is fed into a fully-connected network (multi-layer perceptron), and a loss can be computed based on the difference between the final layer of the network and the labels in the training set. The gradients of the loss with respect to each neuron input can be backprogated through the network to the filter parameters. Over time, the filter parameters are tuned to minimize the loss, and are tuned to detect different kinds of indicative patterns about the input, although they may usually not be interpretable to humans. For examples of visual interpretation of convolution filters trained on a sentiment analysis task, please refer to our work [24].

Convolutional neural networks were initially used in the computer vision community, but has since been adopted by the NLP community starting with Collobert et al. [8]. Kim applied CNNs to sentence classification and improved the state-of-the-art on four out of seven tasks [21]. Kim's architecture is very similar to the general CNN architecture for NLP described above, with the exception of two small but important simple modifications. First, Kim uses filters of multiple widths, capturing n-gram patterns of different granularities. Second, Kim uses two separate channels for the sentence embedding. In one channel, the embedding is kept static, meaning that the word vectors are fixed and not updated in the training process. In the other channel, the embedding is non-static, which means that the word vectors are parameters of the model and are updated when the gradients are backpropagated.

One of most notable works that use convolutional neural networks for textual similarity tasks, specifically answer selection, is the CNN of Severyn and Moschitti [38], hereafter referred to as SM-CNN. SM-CNN has a Siamese architecture [4], meaning there are two convolutional neural networks with shared parameters that operate in parallel, one on each input sentence. The feature vectors of the two sentences are concatenated, with additional

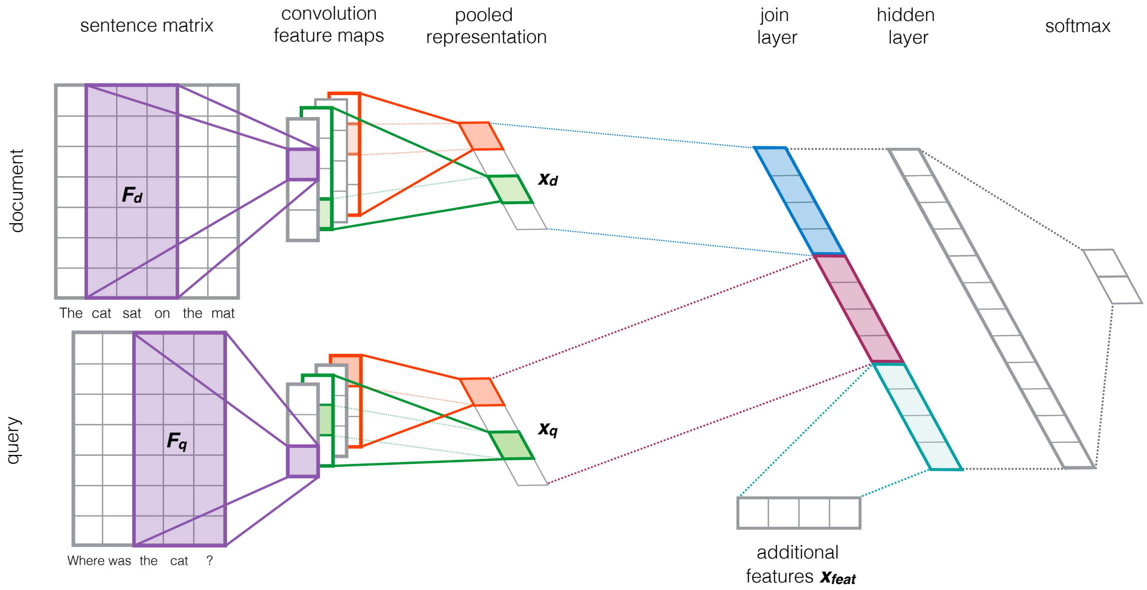


Figure 2.3: A modified version of the Siamese convolutional neural network of Severyn and Moschitti [38]. Figure is adapted from [37].

features about the corpus, before being fed into a fully-connected network for classification. These additional features include the proportion of word overlap between the two sentences, including and excluding stopwords, and *idf*-weighted and non-*idf*-weighted. However, in an experimental analysis by Rao et al. [34], it was found that the additional features contributed more to the model accuracy than the convolution neural network itself, and the model performed poorly without the additional features.

Yin and Schütze developed two Siamese convolutional neural networks for sentence similarity modelling tasks. The first [54] is developed on the insight that a successful model need to take into consideration multiple levels of granularity. To achieve this, they stack multiple convolutional layers on top of each other following [20] since later layers have a larger receptive field, encapsulating more words for each stride of the convolution filter. We will see that MP-CNN captures multiple levels of granularity instead by using multiple widths of filters, similar to Kim's CNN. The best results of the first models relies on a unsupervised pretraining step, which is not required for MP-CNN. Their second model [55] uses attention to compute the interaction between sentence embeddings before convolution and on the convolution output after convolution. However, this attention mechanism significantly increases the number of parameters required. In the next section we'll see an alternative attention mechanism proposal that does not increase the number of model parameters required and beat their performance on the common answer selection task.

2.5 Other Neural Network Methods

There are numerous other works that apply neural network methods to sentence similarity tasks, except they use recurrent or recursive architectures instead of the convolutional architecture. Socher et al. used a recursive neural network for paraphrase detection [41]. Their model consists of two components, the unfolding recursive autoencoder, a recursive neural network, and dynamic pooling. The recursive autoencoder builds feature representation for each node in an unlabeled parse tree, which enables word vectors below each node to be recursively constructed using the autoencoder. The dynamic pooling layer ensures that a fixed-size representation can be outputted from the similarity matrix which compares single word features and higher-level multi-word features in non-leaf nodes.

Tai et al. [43] generalized LSTM architectures, which are known for their represention power and strength in captureing long-term dependencies for sequence modeling, to tree-structured topologies, which seems natural since sentences can have a tree representation from a syntactic point-of-view. The main difference between standard LSTMs and tree-based LSTMs is that while standard LSTMs' current hidden state is influenced by the hidden state of the previous time step and the input at the current time step, tree-based LSTMs' hidden state from the hidden states of child nodes and an input vector. However, this approach depends on dependency parsers. The MP-CNN model investigated in this work does not rely on external resources other than pre-trained word vectors.

Chapter 3

Multi-Perspective Convolutional Neural Networks

Multi-Perspective Convolutional Neural Networks (MP-CNN) are convolutional neural networks that use a multiplicity of perspectives for comparing the semantic similarity of two sentences. It consists of two main modules, the **sentence modelling** module that converts the input sentences into a sentence representation for similarity measurement and a **similarity measurement layer** that compares corresponding regions of the sentence representation to generate a feature vector to feed into a two layer fully-connected network for classification, as shown in Figure 3.1. This chapter describes each component of MP-CNN in detail.

3.1 Sentence Input Representation

First, the raw sentences need to be converted to sentence embeddings in order to be processed by the CNN. Suppose each sentence is given as a sequence words w_1, w_2, ..., $w_{|s|}$. We can retrieve a word vector of dimension d from a pre-trained embedding using popular techniques such as GloVe or `word2vec` for each word that is in the vocabulary of the pre-trained embedding corpus. If a word is not in the vocabulary, it is typically represented by an `<unk>` symbol and each element of the `<unk>` token word vector is drawn from a distribution, such as a uniform distribution $U[-0.25, 0.25]$ as in [38].

With a word vector $\mathbf{w_i}$ for each word at hand, we can concatenate the word vectors for the sequence of words to build a sentence embedding or sentence matrix $\mathbf{S} \in \mathbb{R}^{\mathbf{d}\times|\mathbf{s}|}$:

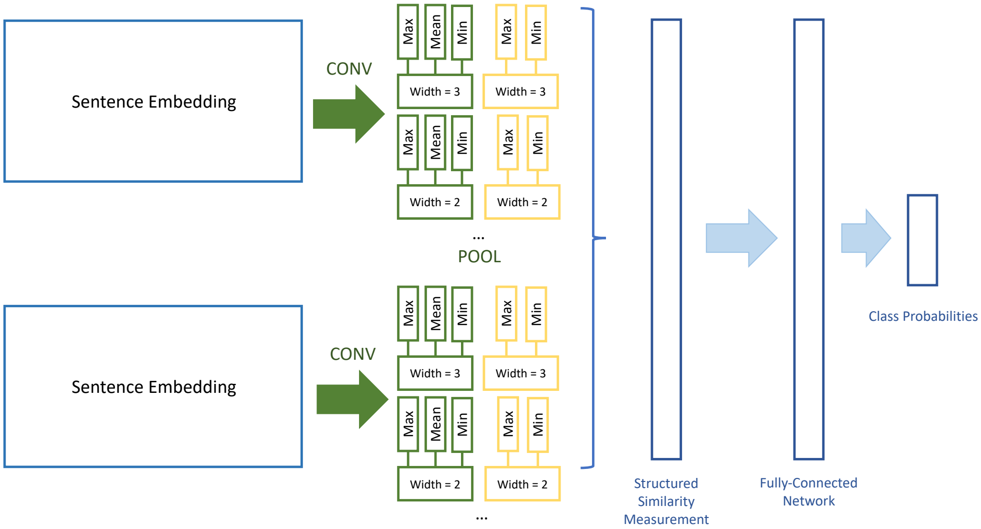


Figure 3.1: Architecture overview of MP-CNN. The sentence modelling module uses multiple types of convolution, producing multiple types of building blocks (green and yellow). Each convolution method has multiple filter widths and multiple types of pooling. The structured similarity measurement layer uses multiple types of comparisons over local regions of the building blocks before feeding the feature vector into a fully-connected network.

$$\mathbf{S} = \begin{bmatrix} | & | & & | \\ \mathbf{w_1} & \mathbf{w_2} & \dots & \mathbf{w}_{|\mathbf{s}|} \\ | & | & & | \end{bmatrix}$$

3.2 Sentence Modelling Module

3.2.1 Multiple Convolution Types

MP-CNN follows a Siamese network structure. Two networks each convert one sentence to its sentence representation in parallel, and they share the weights of the convolution filters. The first type of multiplicity of perspectives comes from multiple types of convolution, consisting of **holistic convolution**, which uses filters that span across all dimensions of the word vectors, and **per-dimensional convolution** which uses filters that are restricted to single dimension of the word vectors. See Figure 3.2 for an illustration.

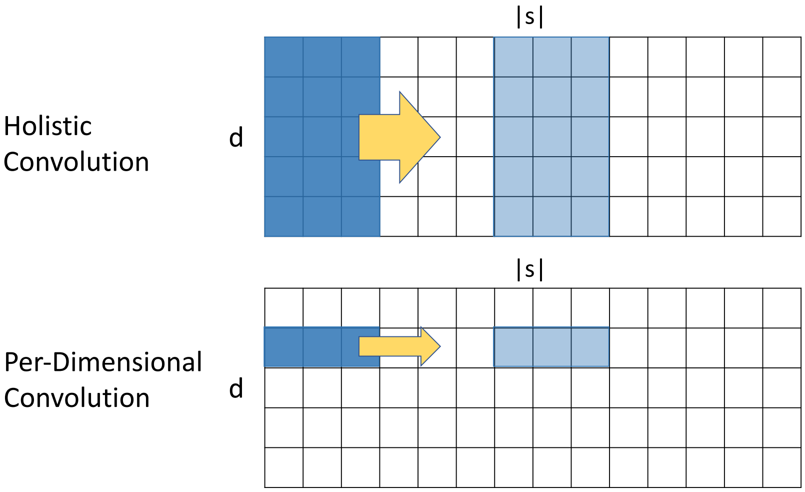


Figure 3.2: Holistic convolution and per-dimensional convolution.

Holistic convolution can be seen as "temporal" convolution, since 1D convolution is performed over the sequence of tokens in the order they are seen, or over time. The reason this type of convolution is called holistic convolution and each filter is called a holistic filter is that each filter spans all dimensions of the word vector. Each filter F can be parameterized by a tuple $< w_s, \mathbf{W}_F, b_F, h_F >$ where w_s is the width of the window, $\mathbf{W}_F \in \mathbb{R}^{d \times w_s}$ is the filter weight matrix, b is the bias, and h_F is the activation function such as `tanh`. The filter is slided along the sequence of tokens in the sentence embedding. At each possible position, the dot product of $\mathbf{W}_F$ and the sentence embedding submatrix the filter overlaps with is taken, the bias is added, and the result is passed through the activation function. A sample visualization of the activations can be seen in Figure 3.3. Mathematically, a vector $\mathbf{out_F} \in \mathbb{R}^{|s|-w_s+1}$ is generated, in which each entry i is

$$\mathbf{out_F}[i] = h_F(\mathbf{W}_F \cdot \mathbf{S}_{i:i+w_s-1} + b_F) \tag{3.1}$$

where $\mathbf{S}_{i:i+w_s-1}$ denotes the i-th to $i + w_s - 1$-th columns of $\mathbf{S}$ inclusive and $1 \leq i \leq |s| - w_s + 1$.

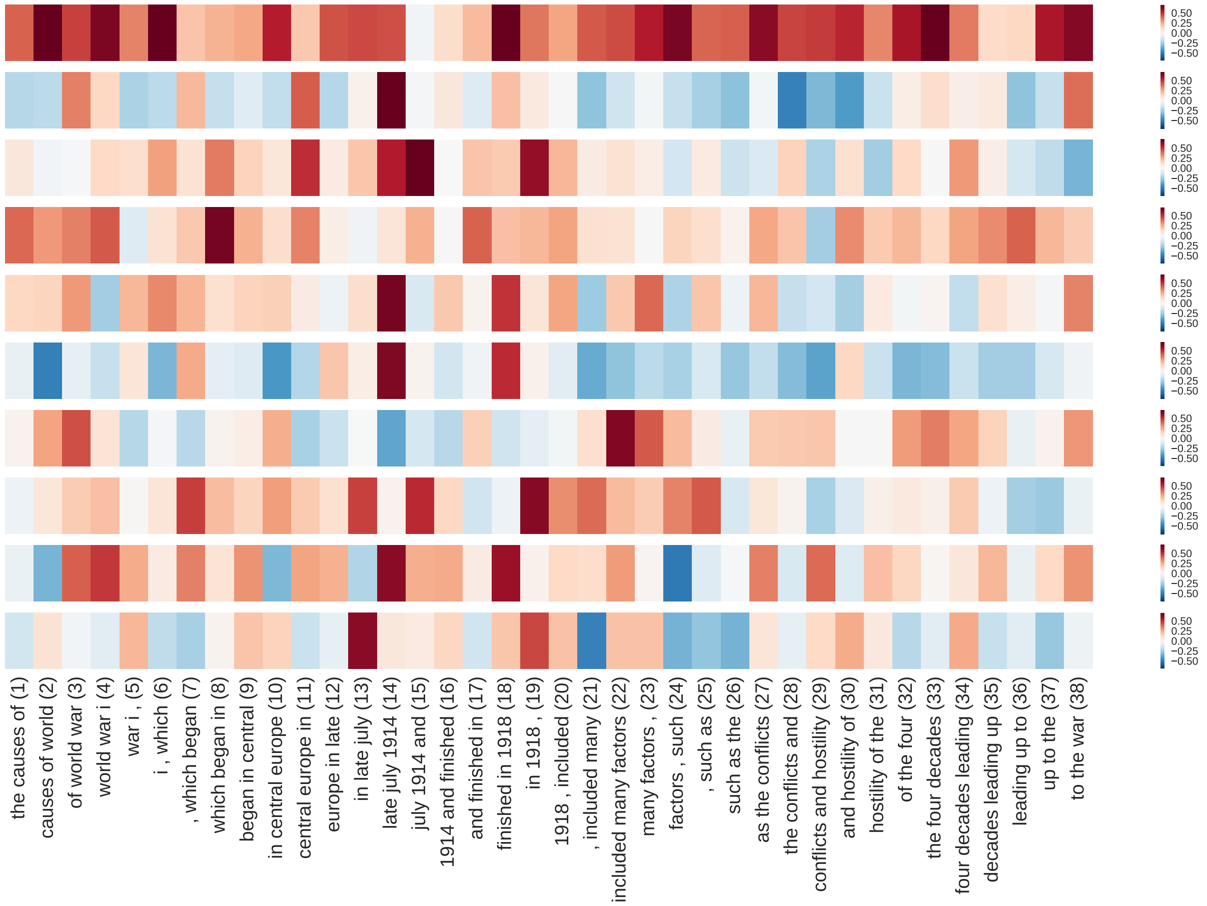


Figure 3.3: Feature maps for 10 holistic filters of $w_s = 3$ that have the highest maximum neuron activation. Many filters seem to have the maximum activation upon seeing "late July 1914" while the first filter has high activation in many regions.

The second type of convolution is per-dimensional convolution, which can be seen as "spatial" convolution restricted to a single word vector dimension, consists of separate filters for each dimension that only slide in that particular dimension across the sentence token sequence. Each filter is called a per-dimensional filter. A per-dimensional filter $F^{[k]}$, for dimension k, can be parameterized by the tuple $< w_s, \mathbf{W}_F^{[k]}, b_F^{[k]}, h_F^{[k]} >$, where w_s is the width of the window, $\mathbf{W}_F^{[k]} \in \mathbb{R}^{1 \times w_s}$ is a filter weight vector, $b_F^{[k]}$ is the bias, and h_F is the activation function. Similar to the holistic filter, at each possible position, the dot product of $\mathbf{W}_F^{[k]}$ and the part of the sentence embedding the filter overlaps with is taken, the bias is added, and the result is passed through the activation function. Mathematically, a vector $\mathbf{out}_{\mathbf{F}}^{[\mathbf{k}]} \in \mathbb{R}^{|s|-w_s+1}$ is generated, in which each entry i is

$$\mathbf{out}_{\mathbf{F}^{[\mathbf{k}]}}[i] = h_F(\mathbf{W}_F^{[k]} \cdot \mathbf{S}_{i:i+w_s-1}^{[k]} + b_F^{[k]}) \tag{3.2}$$

where $\mathbf{S}_{i:i+w_s-1}^{[k]}$ denotes the k-th row (dimension) of the i-th to $i+w_s-1$-th columns of $\mathbf{S}$ inclusive and $1 \leq i \leq |s| - w_s + 1$.

The intuition behind per-dimensional convolution is that each filter, restricted to a specific dimension, may be able to capture and exploit distinctive information from each dimension that the holistic filters cannot capture.

3.2.2 Multiple Pooling Types

Typically, only max pooling is used for extracting the most important features out of the convolutional feature maps. However, MP-CNN experiments with two additional types of pooling, average (mean) pooling and min pooling, which takes the mean and minimum value of each convolutional feature map respectively, with the hope of retaining more information about the representation.

Here, MP-CNN defines some objects that will be used in the similarity measurement layer: groups and building blocks. A group is a tuple consisting of a convolutional layer with width w_s, a pooling function, and an input sentence embedding. It can be denoted by $group(w_s, pooling, sent)$. There are two types of groups: $group_A$ for groups that use holistic filters of width w_s and $group_B$ for groups that use per-dimensional filters of width w_s. A building block consists of multiple groups. There are two types of building blocks, building block A for groups that use holistic convolution and building block B for groups that use per-dimensional convolution. One important detail to note is that pooling functions of a particular building block use independent convolutional layers, in accordance with the choice of the original MP-CNN model, rather than shared convolutional layers.

$$block_A = \{group_A(w_{s_a}, p, sentence) : p \in \{max, min, mean\}\} \tag{3.3}$$

$$block_B = \{group_B(w_{s_b}, p, sentence) : p \in \{max, min\}\} \tag{3.4}$$

Suppose the convolution layer in group $group_A(w_{s_a}, pooling, sentence)$ has N_A filters, then the output oG_A of the group with the sentence as input is a vector of length N_A, where

$$\mathbf{oG_A}[i] = pooling(\mathbf{out_F}[i]) \tag{3.5}$$

Suppose the convolution layer in group $group_B(w_{s_b}, pooling, sentence)$ has N_B filters for each of the d dimensions of the word vector. Then the output oG_B of the group with the sentence as input is a matrix of dimension $d \times N_B$, where

$$\mathbf{oG_B}[k][i] = pooling(\mathbf{out_{F[k]}}[i]) \tag{3.6}$$

3.2.3 Multiple Window Sizes

MP-CNN uses multiple filter widths to match n-grams of different granularity. For example, filters of $w_s = 1$ match unigrams, filters of $w_s = 2$ match bigrams, filters of $w_s = 3$ match trigrams, and so on. This idea has also been used in previous works such as convolutional neural networks for sentence classification by Kim [21].

In addition to using multiple widths of convolution filters, multiple types of pooling are used on the original sentence embedding. The paper argues that this retains the original information in the sentence. This essentially corresponds to using a filter of $w_s = \infty$, except the filter doesn't have weights associated with it.

3.3 Similarity Measurement Layer

Both input sentences have building blocks constructed from convolution using filters with shared parameters. Since the building blocks have structure, which separates the information coming from multiple convolution types, multiple pooling types, and multiple filter widths, it naturally points to comparison methods that utilize this structure to perform structured comparisons. An alternative would be to flatten all the building blocks from each sentence into a vector and then directly comparing the vector representations of the two sentences, but this discards potentially important structural information.

MP-CNN proposes two comparison algorithms over local regions of the sentence representations: the **horizontal** and **vertical** comparison algorithms. The horizontal comparison algorithm performs structured comparison between groups of the two building blocks, one from each sentence, that share the same holistic filters. That is, it compares the outputs of the groups that use the same type of pooling function, using the same holistic filters, between the two sentences. In contrast, the vertical comparison algorithm performs comparisons over groups of both holistic and per-dimensional filters. For holistic filter groups, comparison is performed between group outputs of all pairs of filter widths w_{s_1} and w_{s_2} for the two sentences, under the same pooling type. For per-dimensional groups, comparison is performed over corresponding groups of the same pooling type and width between the two sentences. The pseudocode for the exact algorithms are in Figure 3.4 and Figure 3.5. An illustration of the algorithms using a pair of inputs is illustrated in Figure 3.6.

The comparison algorithms use comparison units consisting of multiple types of similarity metrics. These similarity metrics include cosine distance, measuring the angle between two vectors, the L_2 Euclidean distance, and element-wise absolute differences. The first type of comparison unit $comU_1$ uses all three types of comparison units, while the second type of comparison unit $comU_2$ only uses the cosine distance and Euclidean distance. These choices are empirically motivated when they were chosen for MP-CNN.

$$comU_1(\mathbf{x}, \mathbf{y}) = \{cos(\mathbf{x}, \mathbf{y}), L_2Euclid(\mathbf{x}, \mathbf{y}), |\mathbf{x} - \mathbf{y}|\} \tag{3.7}$$

$$comU_2(\mathbf{x}, \mathbf{y}) = \{cos(\mathbf{x}, \mathbf{y}), L_2Euclid(\mathbf{x}, \mathbf{y})\} \tag{3.8}$$

```
HORIZONTAL-COMPARISON($S_1, S_2$)
$fea_H = []$
for $pooling = max, min, mean$
    for $width\ w_s = 1...n, \infty$
        $regM_1[*][ws_1] = group_A(w_s, pooling, S_1)$
        $regM_2[*][ws_1] = group_A(w_s, pooling, S_2)$
    for $i = 1...numFilters_A$
        $fea_h = comU_2(regM_1[i], regM_2[i])$
        $fea_H.extend(fea_h)$
return $fea_H$
```

Figure 3.4: The horizontal comparison algorithm. $regM_*$ is a $numFilters_A \times (n+1)$ dimensional matrix.

```
Vertical-Comparison($S_1, S_2$)
$fea_V = []$
for $pooling = max, min, mean$
    for $width\ w_{s_1} = 1...n, \infty$
        $oG_{1A} = group_A(w_{s_1}, pooling, S_1)$
        for $width\ w_{s_2} = 1...n, \infty$
            $oG_{2A} = group_A(w_{s_2}, pooling, S_2)$
            $fea_a = comU_1(oG_{1A}, oG_{2A})$
            $fea_V.extend(fea_a)$
    for $width\ w_s = 1...n$
        $oG_{1B} = group_B(w_s, pooling, S_1)$
        $oG_{2B} = group_B(w_s, pooling, S_2)$
        for $i = 1...numFilters_B$
            $fea_b = comU_1(oG_{1B}[*][i], oG_{2B}[*][i])$
            $fea_V.extend(fea_b)$
return $fea_V$
```

Figure 3.5: The vertical comparison algorithm.

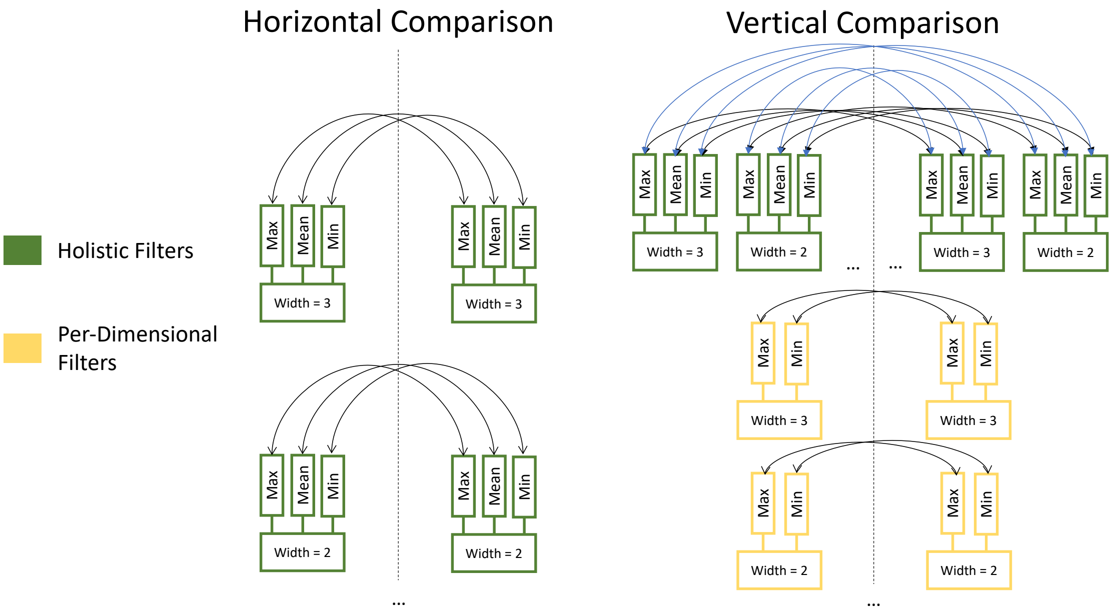


Figure 3.6: An illustration of the similarity comparison algorithms. The horizontal comparison algorithm only compares corresponding pooling outputs of the same filter. In the vertical comparsion algorithm, for per-dimensional filters, only corresponding pooling outputs of the same filter are compared. However, for holistic filters, for each pooling type, outputs from all pairs of filter widths are compared.

3.4 Fully-Connected Layer

The output vectors of the Horizontal-Comparison and Vertical-Comparison algorithms are concatenated together to form a single feature vector. This vector is used as input into a two layer fully-connected network for classification. `tanh` is used as the activation in all places. Since MP-CNN is not a stacked deep network, we do not need to be as concerned about the vanishing gradient problem when `tanh` becomes saturated.

3.5 Simple Modifications

We suggest several simple modifications to the base MP-CNN that may help to improve the model. We will conduct experiments to evaluate the effectiveness of these modifications.

3.5.1 Wide Convolution vs. Narrow Convolution

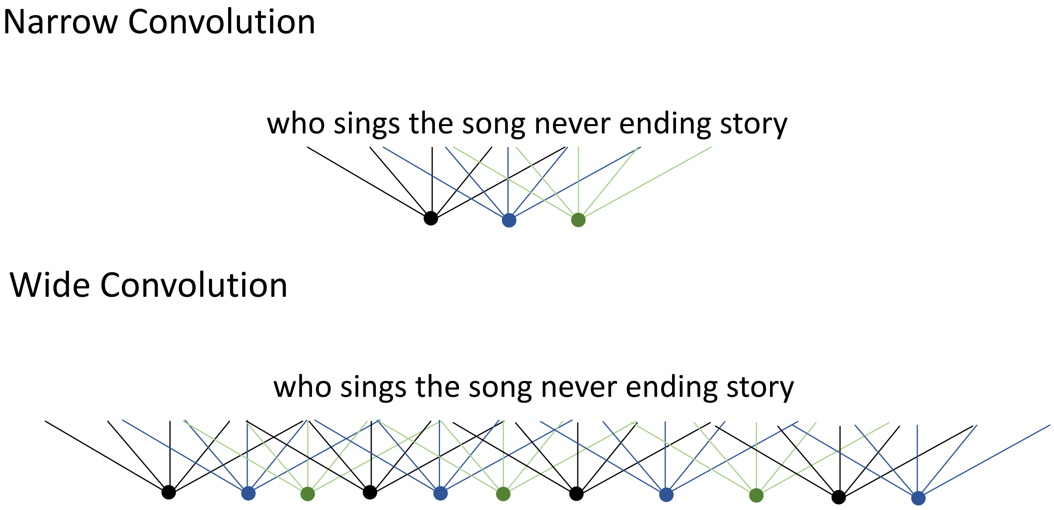

Narrow Convolution
who sings the song never ending story
Wide Convolution
who sings the song never ending story

Figure 3.7: Narrow vs. wide convolution. The filter is of width 5 and each dot indicates one element in the convolutional feature map that is computed. The lines connecting the words to the dot indicates the words in each window used to compute the feature map.

MP-CNN uses narrow convolution, which means that the filter will start at the first word in the sentence and it stops sliding when the right end of the filter reaches the last word (Figure 3.7). If the sentence has $|s|$ words and the filter is of width w, then narrow convolution creates a convolutional feature map of $|s| - w + 1$ elements in total. A requirement of narrow convolution is that $w \leq |s|$.

An alternative to narrow convolution is wide convolution. In wide convolution, zero padding is added on either side of the sentence so that we ensure that all words in the sentence get included in some sliding window position an equal number of times. Furthermore, in wide convolution there is no requirement for $w \leq |s|$, so regardless of the size of the filter relative to the sentence a valid feature map can always be produced [20]. Wide convolution can better handle words at the margin, which maybe especially important for short sentences.

3.5.2 Attention

MP-CNN computes the interaction between the two sentences indirectly through comparison algorithms in the similarity measurement layer, on the max, min, and mean elements of the convolutional feature maps. However, there is a more direct approach for computing the interaction between the original sentences, through the use of attention. In contrast to the approach of Yin et al. [55], Hua et al. [17] proposed an attention mechanism that does not require extra parameters to be added to the model with the exception of adapting the size of the convolution filters to match the new sentence embedding size.

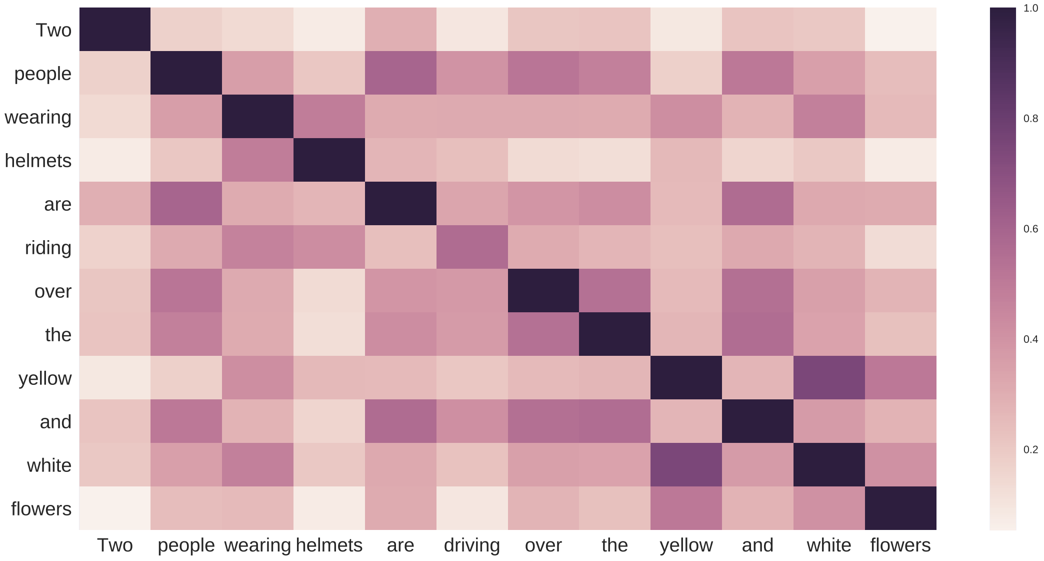


Figure 3.8: An attention matrix of two sentences.

Suppose we have input sentences s_1 and s_2 of lengths m and n respectively, and construct corresponding sentence embeddings $\mathbf{S_1}$ and $\mathbf{S_2}$ of dimensions $d \times m$ and $d \times n$ respectively. We can compute an attention matrix $\mathbf{D} \in \mathbb{R}^{m \times n}$, where each element is computed by

$$\mathbf{D}[i][j] = cosine(\mathbf{S_1}[:,i], \mathbf{S_2}[:,j]) = \frac{\mathbf{S_1}[:,i] \cdot \mathbf{S_2}[:,j]}{||\mathbf{S_1}[:,i]||||\mathbf{S_2}[:,j]||} \tag{3.9}$$

where $\mathbf{S_1}[:, i]$ refers to the word vector of the i-th word in s_1 and $\mathbf{S_2}[:, j]$ refers to the word vector of the j-th word in s_2. Each element represents the word similarity between the i-th word of s_1 and the j-th word of s_2.

We can then create attention weight vectors $\mathbf{a_1} \in \mathbb{R}^m$ and $\mathbf{a_2} \in \mathbb{R}^n$ that contains information about how relevant is one word in one sentence compared all words in the other sentence. We can do this by summing up the rows of $\mathbf{D}$ to create a vector that tells us how relevant is each term in the first sentence with respect to the second sentence and sum up the columns of $\mathbf{D}$ to create a vector that tells us how relevant is each term in the second sentence with respect to the first sentence. These vectors can then be normalized using softmax.

$$\mathbf{a_1}[i] = softmax(\sum_j \mathbf{D}[i][j]) \tag{3.10}$$

$$\mathbf{a_2}[j] = softmax(\sum_i \mathbf{D}[i][j]) \tag{3.11}$$

The attention weight vectors can be used to reweigh the sentence embeddings so that word vectors corresponding to words that are relevant to the other sentence can be weighted higher than words not as relevant to the other sentence. This is simply multiplying every element of the word vector corresponding word i in the first sentence by $\mathbf{a_1}[i]$ and likewise for the second sentence. The reweighted sentence embedding can then be concatenated with the existing sentence embedding, creating a new sentence embedding $\mathbf{S}' \in \mathbb{R}^{2d \times |s|}$

$$\mathbf{S}' = concat(\mathbf{S}, \mathbf{a} \odot \mathbf{S}) \tag{3.12}$$

Often, we may notice that there are many stop words in the two sentences that co-occur. This will result in elements in the attention matrix to match, and may contribute highly to the column and row sums of the attention matrix. We do not want stopwords to overly influence the attention weight vectors, hence we can make two simple modifications. In Equations 3.10 and 3.11, we can weigh each $\mathbf{D}[i][j]$ by the inverse document frequency, or *idf*, of word i in sentence one and word j in sentence two respectively. The inverse document frequency is the number of sentence pairs in which the word appears. Furthermore, after summation but before the softmax, we can further reweigh each element of the attention weight vector by the *idf* of the corresponding word in the sentence. We will evaluate the version of attention with and without *idf* reweighting later on.

Chapter 4

Data and Implementation

4.1 Datasets

We use three datasets across semantic textual similarity and answer selection to conduct our experiments: SICK, TrecQA, and WikiQA. Note that there exists a popular dataset for paraphrase identification, the Microsoft Research Paraphrase Corpus (MSRP), but we do not evaluate against it since the literature on MSRP is heavily dominated by non-standardized feature engineering across different papers. For example, Yin et al. [55] used 15 machine translation features but do not report the scores of their model without using the additional features. Socher et al. [41] used three additional features for their model. We will demonstrate that MP-CNN and its variants can perform well on other tasks without any need for manual feature engineering or external resources.

4.1.1 Semantic Textual Similarity

The Sentences Involving Compositional Knowledge (SICK) dataset is often used for the semantic textual similarity task. It was developed for the SemEval 2014 competition Task 1, consisting of 9927 pairs of sentences in total. Sentences in the SICK dataset aim to test the ability of the computer system to understand sentences that are challenging due to semantic compositionality-related reasons rather than encyclopedic knowledge. SICK has labels for both the semantic textual similarity (relatedness) task and the natural language inference (entailment) task. We focus on the relatedness tasks, where sentence pairs are given a score on 5-point continuous scale indicating different degrees of semantic relatedness in the range [1, 5]. Sentences in SICK are short, having an average length of 10 words, with the shortest sentence only having 3 words and the the longest has 32 words. More details about the dataset can be found in [25].

The original MP-CNN paper also uses the Microsoft Video Paraphrase Corpus (MSRvid) dataset. However, since MSRvid is one of the data sources of SICK, we will not include this dataset with this thesis.

Split	# of Sentence Pairs	Relatedness Score Distribution
Train	4500	[9.7%, 14.1%, 38.8%, 37.4%]
Dev	500	[7.4%, 13.8%, 38.4%, 40.4%]
Test	4927	[9.2%, 13.7%, 39.9%, 37.2%]
Total	9927	[9.2%, 13.9%, 39.3%, 37.5%]

Table 4.1: Statistics of the SICK dataset showing the number of sentence pairs and distribution of scores. The distribution describes the percentage of examples in each of the following range: $[1, 2], (2, 3], (3, 4], (4, 5]$.

Sentence A	Sentence B	Relatedness
A man is in a parking lot and is playing tennis against a friend	A man is pointing at a silver sedan	1.3
Two angels are making snow on the lying children	Two children are lying in the snow and are making snow angels	2.9
A woman is playing with two young boys at a park	There is no woman playing with two young boys at a park	4.3
A lone biker is jumping in the air	A biker is jumping in the air, alone	5

Table 4.2: Example sentences from the SICK dataset, showing various graded levels of semantic relatedness.

4.1.2 Answer Selection

The two datasets usually used for evaluating answer selection are TrecQA and WikiQA. The TrecQA dataset is prepared by Wang et al. [49] and comes from the TREC Question Answering tracks 8-13. Data from tracks 8-12 are used for training and data from track 13 is used for development and testing. Candidate answers in the development and testing set are selected from sentences in the question's document pool that contain at least one non-stop word that is also in the question and their relevance is judged by humans. For the training set, the candidate answers contains in addition sentences that have the correct answer pattern, although only the first 100 sentences' relevance are manually judged. All sentences have no more than 40 words. The average sentence length for TrecQA is 18 words. Statistics about the TrecQA dataset is summarized in Table 4.3 and examples can found in Table 4.4 [52].

Split	# of Questions	# of Sentence Pairs	% of Relevant Pairs
Train	1227	54317	13.6%
Dev (raw)	81	1148	19.3%
Dev (clean)	65	1117	22.5%
Test (raw)	95	1517	23.0%
Test (clean)	68	1442	18.7%
Total (raw)	1403	56982	14.1%
Total (clean)	1360	56876	14.0 %

Table 4.3: Statistics of the clean TrecQA dataset.

Question / Answer	Relevant
Q: Where is the highest point in Japan?	
A1: It was not alone, just one of the hundreds gathering on the summit of Mt Fuji, at 12,388ft the highest point in Japan.	Yes
A2: Climbing Mt Fuji is more than just getting to the top of a mountain.	No
A3: No camping is allowed on Mt Fuji.	No

Table 4.4: Example question and candidate answers from the TrecQA dataset.

Yang et al. created the WikiQA dataset to address some of the shortcomings of the TrecQA dataset [51]. They argue that WikiQA has less selection bias (e.g. question and answer do not necessarily need overlap non-stop word). The questions are selected from actual questions users ask on Bing and sentences from the summary section of the relevant Wikipedia article are candidate answers, which are judged on a crowdsourcing platform. Questions without correct answers are excluded and answers longer than 40 tokens are truncated to 40 tokens. Sentences on average have 15 words. Statistics about the WikiQA dataset can found found in Table 4.5 and example can be found in Table 4.6.

Split	# of Questions	# of Sentence Pairs	% of Relevant Pairs
Train	1040	8672	13.6%
Dev	140	1130	14.1%
Test	293	2351	14.2%
Total	1473	12153	13.8%

Table 4.5: Statistics of the clean WikiQA dataset.

Question / Answer	Relevant
Q: What does it mean to be a common wealth state?	
A1: Commonwealth is a traditional English term for a political community founded for the common good.	Yes
A2: Historically, it has sometimes been synonymous with "republic".	No
A3: More recently it has been used for fraternal associations of some sovereign nations.	No

Table 4.6: Example question and candidate answers from the WikiQA dataset.

4.2 Implementation

On the SICK dataset, we use the KL-divergence loss, following the setup of Tai et al. [43]. We define a target distribution p that we want to learn from the data. We convert the graded continuous score $y \in [1, 5]$ to this target distribution as follows, for $1 \leq i \leq 5$:

$$p_i = \begin{cases} y - \lfloor y \rfloor, & \text{if } i = \lfloor y \rfloor + 1 \\ \lfloor y \rfloor - y + 1, & \text{if } i = \lfloor y \rfloor \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

We then compute the L_2-regularized loss of the KL divergence between the predicted distribution and the target distribution.

On the TrecQA and WikiQA datasets, we use the cross-entropy loss, treating the task as a binary classification problem.

All code is written in PyTorch v0.4.0 with torchtext v0.2.3 for data preprocessing. Tokenization is performed using the Penn Treebank tokenizer implemented in Python's NLTK library. We use the GloVe 300 dimension, 840 billion token embedding pre-trained on Common Crawl. All tokens not found in the GloVe embedding (out-of-vocabulary) are numericalized using a special `<unk>` word vector, whose elements are drawn i.i.d. from a normal distribution with mean of zero and standard deviation of 0.01. The implementation of the MP-CNN model is available at https://github.com/castorini/Castor and the many variations of the model used for experiments are available at https://github.com/tuzhucheng/MP-CNN-Variants.

Chapter 5

Experiments

We conduct a series of experiments to evaluate the contribution of each of the model component choices towards the overall model accuracy. One of the main criteria for deciding which model choice is superior compared to others is the robustness of the model under different hyperparameter configurations.

5.1 Evaluation Metrics

For semantic textual similarity tasks, we often resort to the Pearson's correlation coefficient (Pearson's r) and Spearman's rank correlation coefficient (Spearman's ρ) for evaluation. Pearson's r measures the similarity between the model's predicted sentence pair similarity score and the golden set's sentence pair similarity score. Spearman's ρ measures the similarity in the ranks of the sentence pairs using their sentence similarity score between the model's predictions and the golden set. If S is the set of sentence pairs, y_i and t_i are the model's predicted and golden sentence pair similarity score for sentence pair i respectively, $\overline{y}$ and $\overline{t}$ are means of the predicted and golden scores respectively, and r_{y_i} and r_{t_i} are the ranks of sentence pair i based on their pair similarity scores respectively, then,

$$Pearson(y,t) = \frac{\sum_{i=1}^{|S|}(y_i - \overline{y})(t_i - \overline{t})}{\sqrt{\sum_{i=1}^{|S|}(y_i - \overline{y})^2(t_i - \overline{t})^2}} \tag{5.1}$$

$$Spearman(y,t) = 1 - \frac{6\sum_{i=1}^{|S|}(r_{y_i} - r_{t_i})^2}{|S|(|S|^2 - 1)} \tag{5.2}$$

Both correlation coefficients have a value between -1 and 1, with -1 indicating complete negative linear correlation/monotonicity and 1 indicating complete positive linear correlation/monotonicity. The higher the correlation coefficients, the better is the model. We use the implementation of Pearson's r and Spearman's ρ in the popular SciPy library.

For answer selection, we resort to the Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR) - metrics used in information retrieval for ranked lists. Both metrics use a single number to reflect the quality of the ranked list, with MAP considering the average precision over different levels of recall and MRR considering only the rank of the top result. Suppose Q is the set of questions, m_j is the number of relevant answers for the j-th question in the golden set, R_{jk} is a ranked list that contains the top k answers from most relevant to least relevant as determined by the model, r_j is the rank of the first actual relevant answer in the rank list for the j-th question, and $Precision(R)$ is the ratio of relevant answers to total number of answers in the ranked list R. The higher the MAP and MRR score, the better. We use the implementation of MAP and MRR in the `trec_eval` tool developed by the TREC community.[1]

Then,

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk}) \tag{5.3}$$

$$MRR(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{r_j} \tag{5.4}$$

5.2 Methodology

Training neural networks is sensitive to many conditions, including but not limited to network weight initialization, hyperparameters, the optimization method used, and more. In this section we describe the methodology we used for experiments in this work in detail, which is heavily influenced by the experimental procedure of Reimers et al. [35].

Reimers et al. [35] found the difference between the best and worst score on unseen data due to different random seeds is statistically significant. They argue the practice of only reporting the result of a single run is insufficient, since a good result could be a

[1] https://github.com/usnistgov/trec_eval

consequence of a favourable seed number creating favourable conditions rather than due to actual model improvement. Comparing two recent works for Named Entity Recognition, the result due to different random seeds could make the difference between a state-of-the-art system and mediocre system. Therefore, instead of reporting the score of a single execution, researchers should compare the score distribution across multiple executions.

As an illustration of the variability of model accuracy due to different random seeds, consider the distribution of MAP and MRR scores on the TrecQA dataset on the original MP-CNN model shown in Figure 5.1 using a fixed set of hyperparameters with 10 different random seeds. The 95% confidence interval of the estimation of the mean test set MAP is $[0.7711, 0.7876]$ and the 95% confidence interval of the estimation of the mean test set MRR is $[0.8208, 0.8361]$, assuming these scores follow a Student's t-distribution with unknown standard deviation.

Furthermore, when choosing a model architecture, it is desirable that it can achieve good, stable results for a wide landscape of hyperparameters. If a model is too sensitive to the range of hyperparameters and can only yield good accuracy for a narrow range of hyperparameters, it could be difficult to adapt the model to new tasks and datasets [35]. Hence, model robustness is a desirable property to consider in architecture selection.

Hence, in the following experiments, we compare model architectural variations with respect to robustness to hyperparameter selection. For each trial, we randomly select a seed and randomly select a learning rate and regularization factor from a range describe in Table 5.1, collectively referred to as a configuration. Note that we fix most hyperparameters (Table 5.2) instead of sampling them from a range or set, since otherwise the large combinatorial space of options will require a much larger set of trials which will be more computationally demanding to run. The learning rate and regularization strength are also the most important hyperparameters. The hyperparameters used in Table 5.2 are consistent with those in the original MP-CNN paper [15], except that we use Adam instead of SGD with a custom learning schedule that we found to work better than SGD.

Hyperparameter	Range	Comments
Learning rate (LR)	$[1 \times 10^{-4}, 1 \times 10^{-3}]$	Sampled uniformly over the exponent
Regularization (λ)	$[1 \times 10^{-5}, 1 \times 10^{-3}]$	Sampled uniformly over the exponent

Table 5.1: Ranges for different hyperparameters for drawing a random set of hyperparameters.

Fixing the configuration, we evaluate the statistical accuracy of the architectural variation on multiple datasets. We repeat this procedure in N trials, which means we will

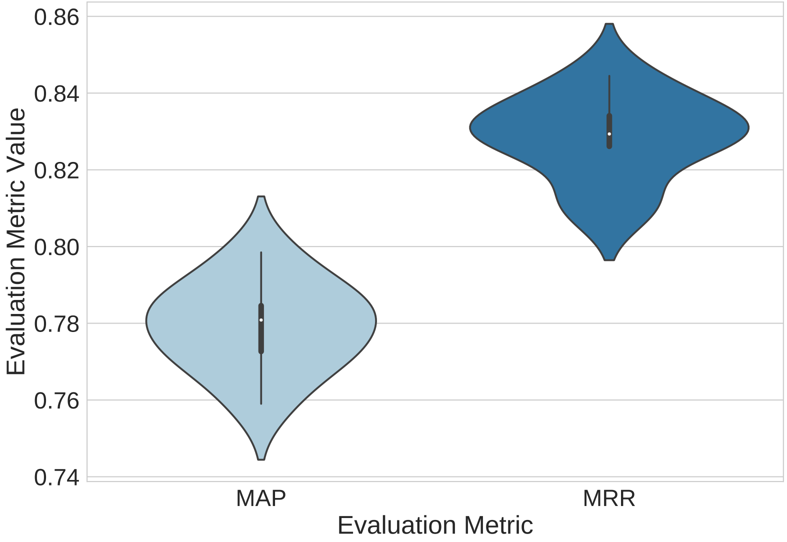


Figure 5.1: Fixing the hyperparameters and only varying the random seed can lead to highly different outcomes. This violin plot shows the MAP and MRR for the test set of TrecQA on a fixed set of hyperparameters on the original MP-CNN model on 10 runs using 10 different random seeds.

evaluate N configurations in total. Note that N could be different for different sets of experiments since some experiments are more computational demanding to run and we have limited computational budget. Following the work of Reimers et al. [35], we use one architectural variation as a baseline to compare to and report the median change in evaluation metric relative to the baseline. Suppose the baseline architecture is denoted by B, and a model architectural variation is denoted by C. Let B_i and C_i denote the test evaluation metric of baseline architecture B and architecture variation C under configuration i respectively. Then, the incremental score change Δ is defined by $median(C_1 - B_1, C_2 - B_2, ..., C_N - B_N)$. The median is less sensitive outliers which may skew the mean. We also report the 95% confidence interval (CI) of the estimation of the mean for each evaluation metric under each architecture variation and dataset. We often draw a violin plot of the distribution of the scores obtained in the N trials using the Python seaborn visualization library.[2] The violin plot shows a kernel density estimation of the underlying distribution with a miniature box plot as the middle line.

[2] https://seaborn.pydata.org/

Parameter	Value	Comments
Optimizer	Adam	Adam is a good default optimizer to use
Batch size	64	
Num. of holistic filters	300	
Num. of per-dimensional filters	20	
Num. of hidden units	150	
Max window size	3	
Activation unit	tanh	
Dropout	0	absent from MP-CNN, but we add later
Learning rate reduce factor	0.3	factor by which to reduce LR
Learning rate patience	2	epochs to wait before reducing LR

Table 5.2: Hyperparameters to fix when comparing model architectural choices. This learning schedule is experimentally determined to work well - the learning rate gets reduced by the reduce factor after patience number of epochs if the validation set score does not improve.

We report the evaluation metric on the test set. For each architecture variant with each configuration, we evaluate the model at the end of every epoch, saving a snapshot of the model if it yields the best score on the validation set. Hence, the model that has the best score on the validation set is used to make predictions on the test set. Architecture comparison experiments are run for 5 epochs for TrecQA, 10 epochs for WikiQA, and 19 epochs for SICK. These were observed to be sufficiently long enough for convergence on the respective datasets.

5.3 Results

Here are all the model choices we investigate:

- Effects of Multiple Pooling Types
- Effects of Multiple Window Sizes
- Effects of Per-Dimensional Convolution
- Effects of Similarity Measurement Algorithms

- Effects of Similarity Comparison Units
- Lightweight MP-CNN
- Effects of Wide Convolution
- Effects of Attention

5.3.1 Effects of Multiple Pooling Types

Finding: Using max pooling alone achieves competitive accuracy and yields the best model robustness. There is no other pooling variation that consistently outperforms max pooling on all datasets and evaluation metrics by appreciable margins.

Table 5.3 summarizes the experiments with different kinds of pooling architectures, visualized by Figure 5.2. We compare the architecture variations against MP-CNN, which uses max, min, and mean pooling for horizontal comparison and max and min for vertical comparison. "Max" refers to an architecture with everything the same as MP-CNN except it uses only max pooling. "Max/Min" uses only max and min pooling in both comparison algorithms, and "All" refers to using max, min, and mean pooling for both comparison algorithms. There are two metrics used for every dataset, and for each metric the 95% confidence interval (CI) of the estimation of the mean is shown as well as the median incremental difference relative to MP-CNN, as defined earlier.

Using max pooling alone gives us the most robust results, meaning it will perform reasonably well over a wide of hyperparameters, rather than mediocre over a large range of the hyperparameter space and only well over a small space. This is evident from the 95% confidence interval. We observe that in the lower limit of the confidence interval of max pooling is always better than that of MP-CNN by non-negligible margins, and it is the only pooling variant to do so. This means that no matter what we pick to be the hyperparameter, it is unlikely that max pooling would yield extremely poor accuracy on the testing set relative to other model choices. Furthermore, the confidence interval of max pooling is also the tightest out of any model architecture on all datasets, indicating that its score is the most stable and predictable. In fact, picking max pooling over the pooling choices of MP-CNN can actually boost the median incremental score difference sometimes, as seen for the TrecQA dataset.

However, we observe that other models often have higher upper limit on the confidence interval of the score compared to that of max pooling. For example, adding min pooling on top of max pooling improved the upper limit on all score metrics of all three datasets with the exception of Pearson's r for SICK. However, these gains are not guaranteed across all datasets (e.g. going from max pooling to MP-CNN pooling for the MRR metric on TrecQA). A higher upper limit relative to that of max pooling indicate that with careful hyperparameter tuning, we may be able to achieve superior score with respect to max pooling. A tradeoff is that architectures that have higher upper limit also have smaller lower limit relative to max pooling and it forms a relatively large interval with the upper limit. When the hyperparameters are not selected well, the model score may be quite poor.

Metric		**MP-CNN**	**Max**	**Max/Min**	**All**
Num. of Params		8.66×10^6	5.00×10^6	8.22×10^6	11.43×10^6
TrecQA					
MAP	CI	[0.7324, 0.7985]	[0.7702, 0.7841]	[0.7100, 0.8057]	[0.7622, 0.7885]
	Δ	–	+0.0062	+0.0165	+0.0028
MRR	CI	[0.7794, 0.8433]	[0.8380, 0.8544]	[0.7815, 0.8628]	[0.8136, 0.8339]
	Δ	–	+0.0232	+0.0248	0.0000
WikiQA					
MAP	CI	[0.6440, 0.7223]	[0.6855, 0.6947]	[0.6445, 0.7043]	[0.6199, 0.7173]
	Δ	–	-0.0167	-0.0181	-0.0101
MRR	CI	[0.6546, 0.7359]	[0.6992, 0.7096]	[0.6513, 0.7194]	[0.6264, 0.7306]
	Δ	–	-0.0167	-0.0148	-0.0086
SICK					
r	CI	[0.8049, 0.8899]	[0.8640, 0.8720]	[0.8646, 0.8704]	[0.8465, 0.8726]
	Δ	–	-0.0010	+0.0019	+0.0004
ρ	CI	[0.7599, 0.8251]	[0.7995, 0.8100]	[0.8003, 0.8120]	[0.8034, 0.8116]
	Δ	–	-0.0027	-0.0006	+0.0012

Table 5.3: Effects of pooling methods relative to MP-CNN on 10 different sets of hyperparameters and seed configurations. "Max" refers to using max pooling only for Building Blocks A & B, "Max/Min" refers to using max and min pooling for both building blocks, and "All" refers to using all three pooling methods for both building blocks. MP-CNN uses max, min, mean pooling for Building Block A and max, min pooling for Building Block B. Median incremental difference is relative to first column.

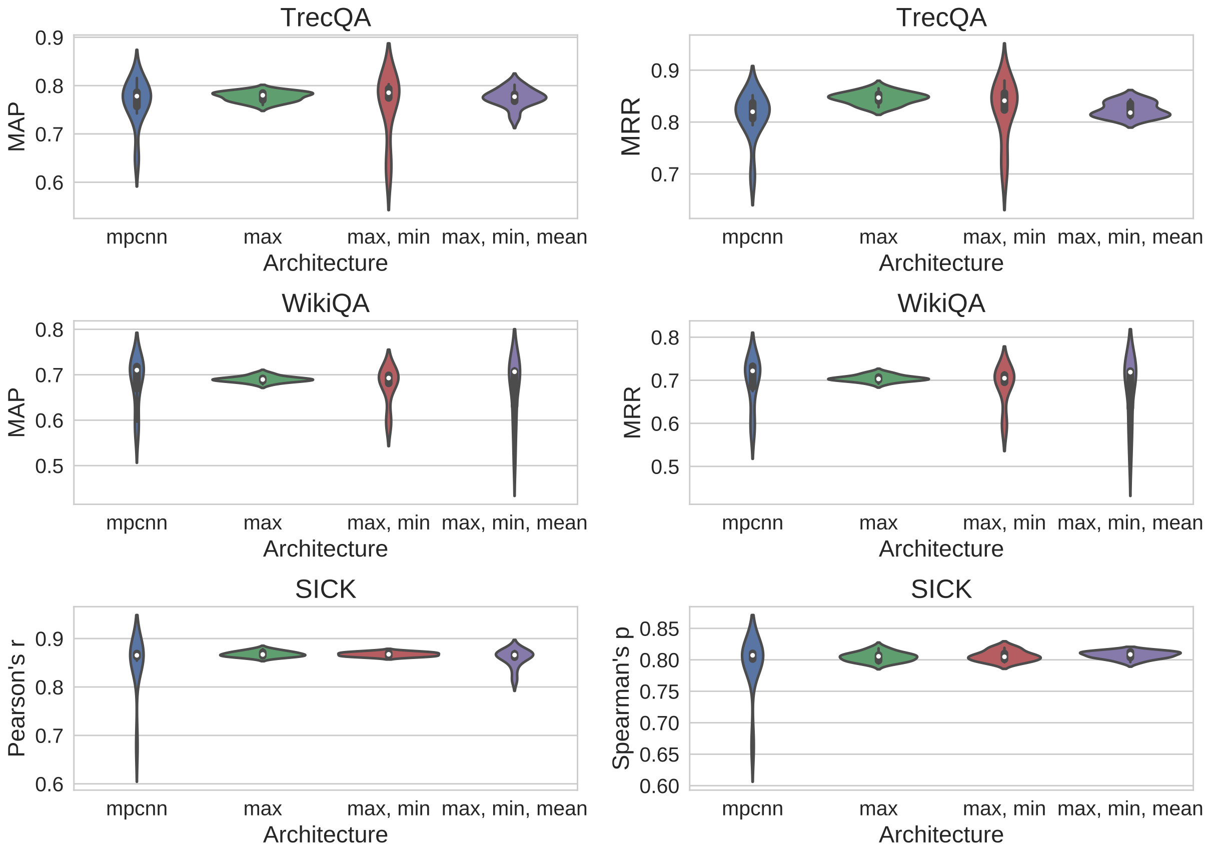


Figure 5.2: Visualization of score distributions with different pooling configurations in Table 5.3, showing using other pooling types often exhibit long tails, leading to poor results if hyperparameters are not tuned carefully.

We conclude that for maximum model robustness across a wide range of hyperparameters, max pooling is the best choice to use. However, to get the best possible score, adding minimum pooling and mean pooling may help, although careful hyperparameter tuning is needed since there may be a large range of hyperparameters that will result in worse score relative to max pooling. It should be noted that adding min and mean pooling to max pooling more than doubles the number of parameters, without bringing about consistent, appreciable gains in the scores.

5.3.2 Effects of Multiple Window Sizes

Finding: Performing convolution using multiple windows sizes yields considerable and consistent score gain compared to using a single window size across all datasets and evaluation metrics.

Next, we investigate the effect of using multiple sizes of convolution filters on the model score. The intuition behind using filters of different window sizes is that they may be able to capture patterns at different granularities, such as at the word, bigram, and trigram levels. Table 5.4 and Figure 5.3 compares the evaluation metrics of TrecQA, WikiQA, and SICK on three model variants. MP-CNN uses filters of widths 1, 2, 3, as well as performing pooling over the entire sentence matrix. The "Single Width" model only uses a single width of filter of size 3, while "Single Width/Inf" uses a width 3 filters in addition to performing pooling over the entire sentence matrix.

Unlike the effects of multiple pooling types, which have mixed results compared to max pooling, using multiple window sizes demonstrates clear superiority over using only a single window size across all datasets and all evaluation metrics. The improvements are also quite noticeable rather than marginal. Using multiple filter window sizes resulted in a median incremental difference of at least two points except in one case. This is larger than the difference between consecutive state-of-the-art models on these datasets (e.g. Table 5.11). These results also show that pooling over the entire sentence matrix, in effect using the infinitely wide filter, has mixed results, since it often yield worse results compared to using single width only.

Figure 5.4 shows that filters of different widths are able to detect n-gram patterns at different granularities. The top visualization corresponds to unigram filters, the middle visualization corresponds to bigram filters, and the bottom visualization corresponds to trigram filters. For each width, suppose we have N filters, and for each individual filter, we keep track of its highest activation value and the window that caused the highest activation. Then, we visualize how often each window causes the highest activation for all filters of a particular width, with each count weighted by the amount of activation. We observe that unigram filters activates highly when they see the word "Europe", but the activation of bigrams and trigrams that contain the word "Europe" is more subdued. Unigram filters cause "July" to activate highly, but not so much the adjacent words "late" and "1914", whereas bigram and trigram filters can find detect larger patterns: "July 1914" and "late July 1914". Unigram filters have high activation for words "conflicts" and "hostility" individually, and trigram filters may group them into one larger pattern "conflicts and hostility" that activates highly. Bigram filters miss these words due to the presence of "and" in between. Filters of different widths can act as an ensemble, picking out patterns that others miss.

Metric		**MP-CNN**	**Single Width**	**Single Width/Inf**
Num. of Params		8.66×10^6	2.81×10^6	3.08×10^6
TrecQA				
MAP	CI	[0.6956, 0.8115]	[0.7212, 0.7492]	[0.6878, 0.7254]
	Δ	–	-0.0383	-0.0605
MRR	CI	[0.6860, 0.8473]	[0.7596, 0.8199]	[0.6824, 0.8270]
	Δ	–	-0.0012	-0.0306
WikiQA				
MAP	CI	[0.5355, 0.7264]	[0.6646, 0.6787]	[0.6640, 0.6730]
	Δ	–	-0.0272	-0.0345
MRR	CI	[0.5410, 0.7387]	[0.5770, 0.7315]	[0.6797, 0.6900]
	Δ	–	-0.0215	-0.0283
SICK				
r	CI	[0.7843, 0.9005]	[0.8358, 0.8419]	[0.8366, 0.8413]
	Δ	–	-0.0281	-0.0284
ρ	CI	[0.7653, 0.8320]	[0.7740, 0.7788]	[0.7766, 0.7820]
	Δ	–	-0.0312	-0.0241

Table 5.4: Effects of using convolution over multiple window sizes on 10 different sets of hyperparameters and seed configurations. MP-CNN uses filters of widths 1-3 and performs pooling over the entire sentence matrix, "Single Width" uses only filters of width 3, and "Single Width/Inf" uses filters of width 3 and performs pooling over the entire sentence matrix. Median incremental difference is relative to first column.

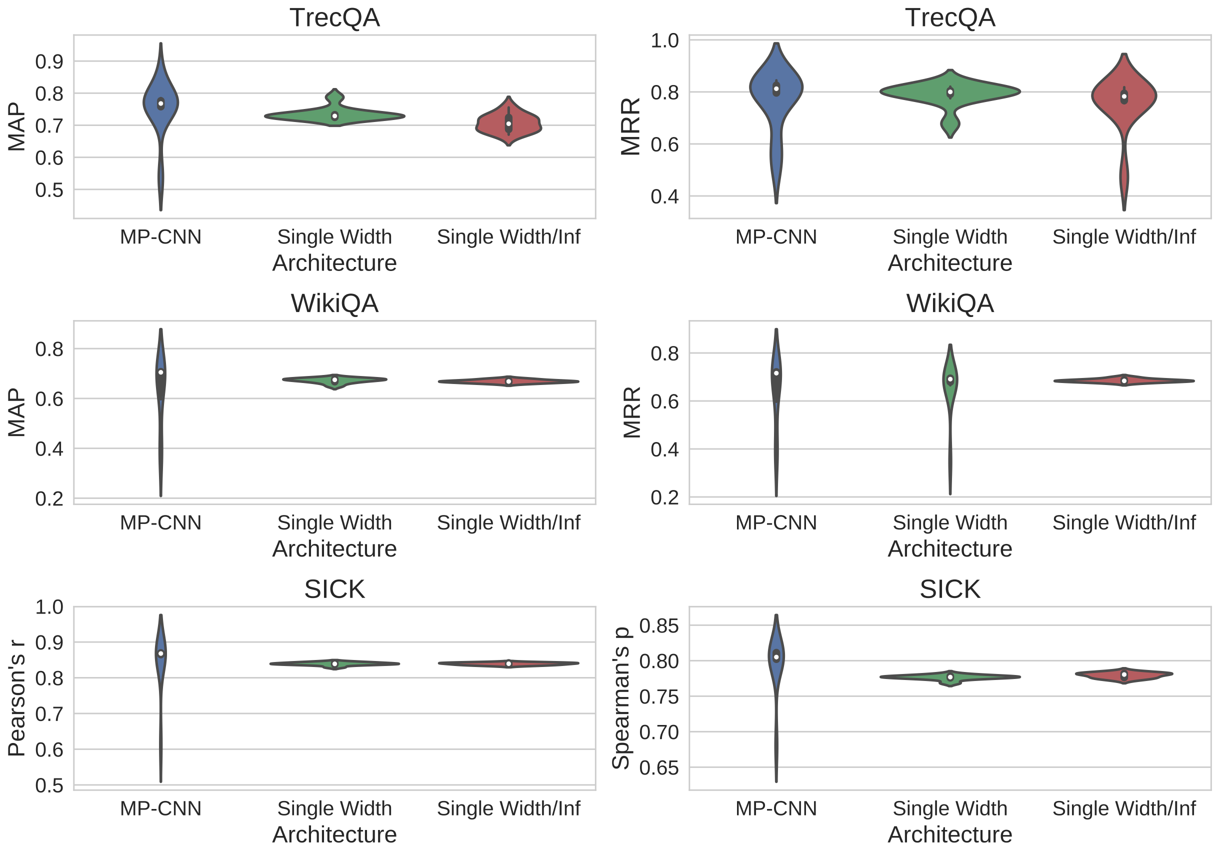


Figure 5.3: Visualization of score distributions with different window configurations in Table 5.4, showing the clear effectiveness of using filters of multiple sizes (MP-CNN).

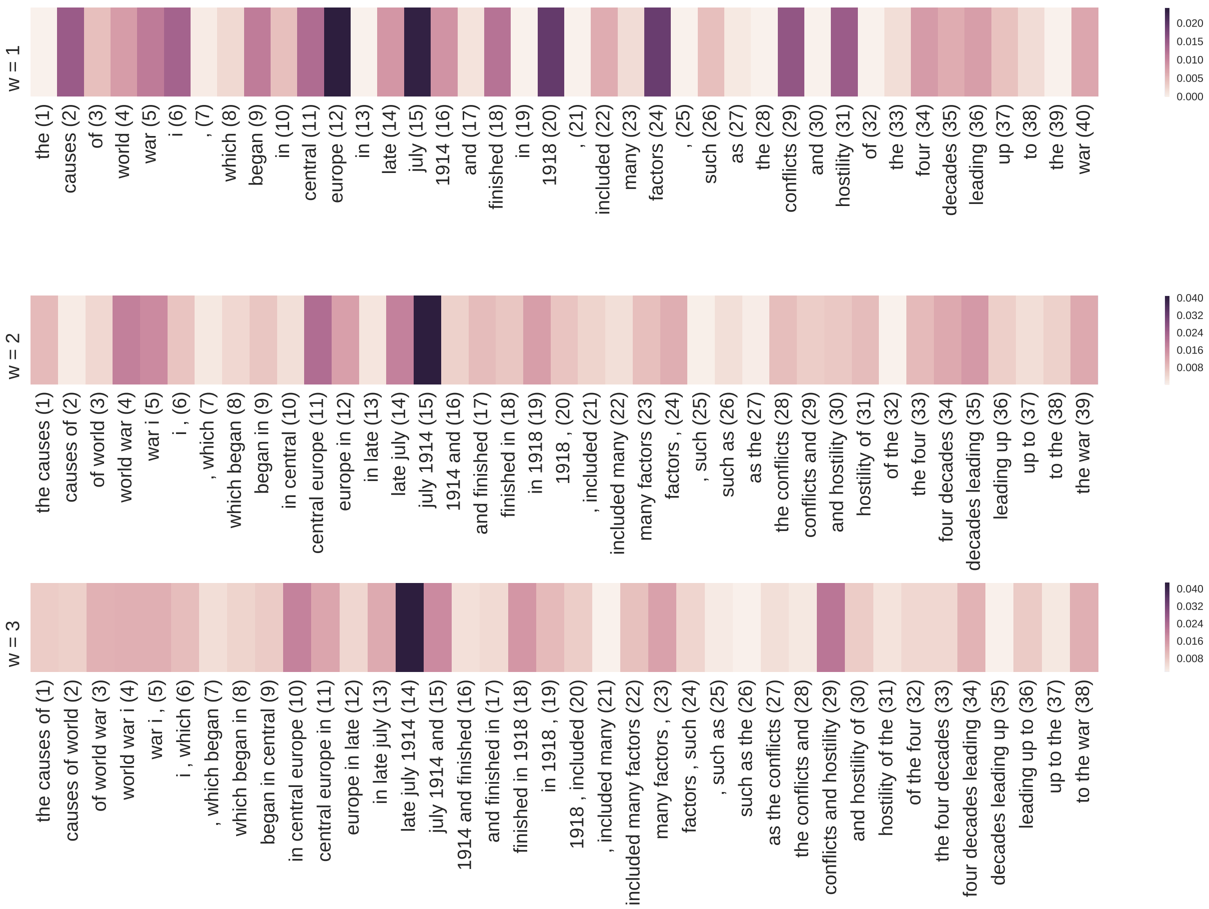


Figure 5.4: Filter activation visualizations for filters of different widths for "*the cause of World War I, which began in central Europe in late July 1914 and finished in 1918, included many factors, such as the conflicts and hostility of the four decades leading up to the war.*" The top, middle, and bottom figures correspond to filters of width 1, 2, and 3 respectively. The intensity of each sliding window indicates a weighted proportion of filters that activate the most highly in that window when they are slid across all windows of a sentence.

5.3.3 Effects of Per-Dimensional Convolution

Finding: Per-dimensional convolution does not provide appreciable boost to model score. It makes the model less robust and often degrades model score.

Metric		**MP-CNN**	**Holistic Only**
Num. of Params		8.66×10^6	3.12×10^6
TrecQA			
MAP	CI	[0.7343, 0.7879]	[0.7427, 0.7926]
	Δ	–	+0.0035
MRR	CI	[0.7967, 0.8381]	[0.8015, 0.8408]
	Δ	–	-0.0042
WikiQA			
MAP	CI	[0.5048, 0.6869]	[0.6227, 0.7302]
	Δ	–	+0.0093
MRR	CI	[0.5109, 0.6989]	[0.6320, 0.7441]
	Δ	–	+0.0107
SICK			
r	CI	[0.7950, 0.8998]	[0.8561, 0.8712]
	Δ	–	-0.0029
ρ	CI	[0.7849, 0.8172]	[0.8016, 0.8076]
	Δ	–	-0.0022

Table 5.5: Effects of using holistic filters only compared to using both holistic and per-dimensional filters in MP-CNN, on 10 different sets of hyperparameters and seed configurations. Median incremental difference is relative to first column.

Intuitively, per-dimensional convolution filters may be able to pick out patterns restricted to particular dimensions in the word vector space and help the model. However, experiment results in Table 5.5 show that they make only a negligible contribution to the model score on the SICK dataset, and do not improve the scores consistently on the TrecQA and WikiQA datasets.

In fact, on both the TrecQA and WikiQA datasets, using only holistic convolution increases both the lower and upper limits of the confidence interval on all evaluation metrics. The margin of the lower limit can increase by large margins when using just holistic convolution - the MAP and MRR lower limits of WikiQA increased by more than 10 points. Furthermore, on WikiQA, the median incremental difference actually improved by close to a single point for both MAP and MRR when using holistic convolution only. Only the median incremental difference of MRR on TrecQA worsened, but this drop is no more than half a point. On the SICK dataset, the median incremental difference worsened when using holistic convolution only, but the magnitude is small (< 0.3 points) and the lower limit of the confidence interval increased. These observations suggest that using holistic convolution only makes the model more stable to hyperparameters, since it is more likely a randomly selected set of hyperparameters from some range will yield lower score when per-dimensional convolution is used.

5.3.4 Effects of Similarity Measurement Algorithms

Finding: The vertical comparison algorithm yields consistent positive benefits while the horizontal comparison algorithm has a more nuanced impact on the model score relatively. Furthermore, in the vertical comparison algorithm itself, the pairwise holistic filter comparison is crucial.

Recall that the similarity comparison layer uses two comparison algorithms, the horizontal and the vertical, for comparing the features detected by shared convolution filters on the two input sentences. The horizontal comparison algorithm compares the features between the two sentences that are generated by the same holistic filter index using the same pooling type. The vertical comparison algorithm compares the features between the two sentences that are generated by the same per-dimensional filter using the same pooling type as well as the interactions between holistic filters of different widths using the same pooling type. In this experiment we analyze the effect of using only a subset of these comparisons, using the original comparison units contained in the subset.

We observe from Table 5.6 that if the vertical comparison algorithm is omitted, the median incremental difference decreases by one point or more consistently in all scenarios. Furthermore, the upper limit of the confidence interval also decreases in all scenarios, indicating the achievable peak performance is decreased. However, we do observe the lower limit of the confidence interval is often improved when omitting the vertical comparison algorithm, indicating some aspects of this algorithm might contribute in instability in accuracy. We can further dissect the vertical comparison algorithm by examining the model results when we just have the comparisons involving holistic filters in the vertical comparison algorithm, omitting the horizontal comparison algorithm and per-dimensional comparison in the vertical comparison algorithm, which is shown in the last column. Comparing the second column "Omit Vert." and the last column "Vert. Holistic", we see that holistic filter comparisons in the vertical comparison algorithm definitively contribute to the model's accuracy, since all confidence interval lower and upper limits as well as the median incremental difference improved when we add them.

Metric		**MP-CNN**	**Omit Vert.**	**Omit Horiz.**	**Vert. Holistic**
Num. of Params		8.66×10^6	2.00×10^6	8.39×10^6	2.85×10^6
TrecQA					
MAP	CI	[0.6811, 0.8095]	[0.7255, 0.7562]	[0.7773, 0.7942]	[0.7612, 0.7859]
	Δ	–	-0.0261	+0.0168	-0.0024
MRR	CI	[0.7408, 0.8707]	[0.8044, 0.8280]	[0.8276, 0.8576]	[0.8218, 0.8440]
	Δ	–	-0.0118	+0.0179	+0.0061
WikiQA					
MAP	CI	[0.5790, 0.7327]	[0.6865, 0.6962]	[0.6984, 0.7092]	[0.6917, 0.7022]
	Δ	–	-0.0168	-0.0040	-0.0123
MRR	CI	[0.5857, 0.7465]	[0.7000, 0.7094]	[0.7106, 0.7224]	[0.7047, 0.7173]
	Δ	–	-0.0182	-0.0038	-0.0142
SICK					
r	CI	[0.8660, 0.8714]	[0.8561, 0.8587]	[0.8643, 0.8707]	[0.8620, 0.8677]
	Δ	–	-0.0115	-0.0011	-0.0036
ρ	CI	[0.8037, 0.8098]	[0.7948, 0.7987]	[0.7990, 0.8065]	[0.7973, 0.8043]
	Δ	–	-0.0100	-0.0038	-0.0062

Table 5.6: Effects of similarity measurement algorithm on 10 different sets of hyperparameters and seed configurations. "Omit Vert." refers to MP-CNN without vertical comparison, "Omit Horiz." refers to MP-CNN without horizontal comparison, and "Vert. Holistic" means MP-CNN only using all widths pairwise holistic comparison in the vertical algorithm, without horizontal comparison and without per-dimensional comparison. Median incremental difference is relative to first column.

Removing the horizontal comparison algorithm has less impact on the model accuracy relative to that of the vertical comparison algorithm. The median incremental difference of the evaluation metrics did not decrease as much when the horizontal comparison algorithm is removed relative the the vertical comparison algorithm, and in some cases even obtained a boost. The upper confidence interval limits decreased, indicating that the horizontal comparison algorithm could improve the upper range of the accuracy. The lower confidence interval limits improved with the omission of the horizontal comparison algorithm however, indicating perhaps its inclusion can hurt stability.

5.3.5 Effects of Similarity Comparison Units

Finding: Using a multitude of similarity metrics generally aid model accuracy than using any single similarity metric alone.

Metric		**MP-CNN**	**Cosine**	**Euclidean**	**Abs. Diff**
Num. of Params		8.66×10^6	1.89×10^6	1.89×10^6	8.89×10^6
TrecQA					
MAP	CI	[0.7752, 0.8020]	[0.7486, 0.7926]	[0.7473, 0.7607]	[0.7686, 0.7875]
	Δ	–	-0.0277	-0.0386	-0.0116
MRR	CI	[0.8169, 0.8406]	[0.5808, 0.8735]	[0.8111, 0.8367]	[0.8159, 0.8403]
	Δ	–	-0.0099	-0.0020	-0.0033
WikiQA					
MAP	CI	[0.6538, 0.7239]	[0.6870, 0.6990]	[0.6865, 0.6939]	[0.6220, 0.7272]
	Δ	–	-0.0089	-0.0073	-0.0030
MRR	CI	[0.6607, 0.7391]	[0.6961, 0.7101]	[0.6993, 0.7080]	[0.6320, 0.7414]
	Δ	–	-0.0105	-0.0070	-0.0046
SICK					
r	CI	[0.8500, 0.8746]	[0.8351, 0.8406]	[0.8552, 0.8591]	[0.8686, 0.8730]
	Δ	–	-0.0302	-0.0095	+0.0034
ρ	CI	[0.8030, 0.8109]	[0.7614, 0.7689]	[0.7943, 0.7996]	[0.8038, 0.8113]
	Δ	–	-0.0407	-0.0105	+0.0007

Table 5.7: Effects of similarity comparison units on 10 different sets of hyperparameters and seed configurations. Median incremental difference is relative to first column.

In Table 5.7 we compare MP-CNN which uses cosine distance, Euclidean distance, and absolute element-wise difference for comparison unit 1 and cosine and Euclidean distance for comparison unit, with model variation that use cosine, Euclidean, and absolute element-wise difference alone respectively. We notice that the median incremental difference is always negative, when we change from the MP-CNN architecture, with the exception of using absolute element-wise difference on SICK.

5.3.6 Lightweight MP-CNN

Finding: We propose a lightweight variant of MP-CNN that does not use per-dimensional convolution and multiple types of pooling and find it obtains similar effectiveness as MP-CNN with improved model robustness.

Previous experiments show that multiple types of pooling and per-dimensional convolution made the model much less robust and yielded insignificant or negative accuracy gains. In addition, they significantly increased the number of model parameters. This motivates us to explore a lightweight variant of MP-CNN, which we call MP-CNN Lite, that only uses max pooling instead of multiple types of pooling and only uses holistic convolution and no per-dimensional convolution. We also compare to SM-CNN with the comparison layer to show that MP-CNN Lite significantly outperforms SM-CNN due to having multiple windows whereas MP-CNN has comparable accuracy to MP-CNN Lite. The only difference between SM-CNN and MP-CNN Lite is that MP-CNN has multiple filter widths. An illustration of MP-CNN Lite can be seen in Figure 5.5.

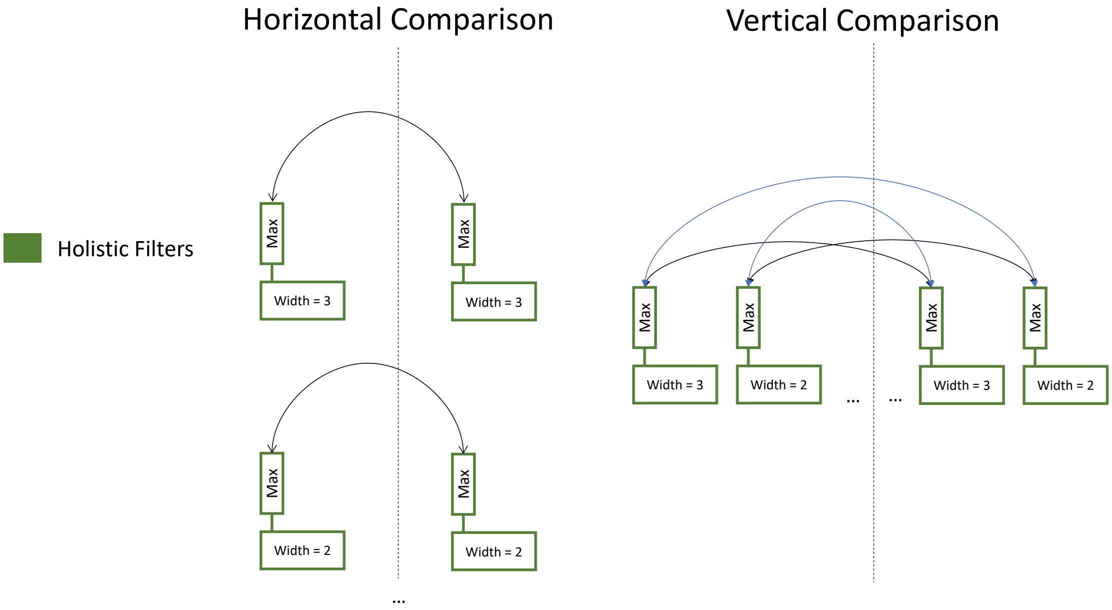


Figure 5.5: An illustration of the similarity measurement layer of the lightweight model. Per-dimensional convolution and min, mean pooling are removed. Dramatic simplification can be seen when compared to Figure 3.6.

Metric		**MP-CNN**	**MP-CNN Lite**	**SM-CNN with Comparison**
Num. of Params		8.66×10^6	1.04×10^6	0.41×10^6
TrecQA				
MAP	CI	[0.7252, 0.8184]	[0.7561, 0.7840]	[0.6644, 0.6808]
	Δ	–	-0.0245	-0.1256
MRR	CI	[0.7550, 0.8406]	[0.8299, 0.8550]	[0.7553, 0.7788]
	Δ	–	+0.0388	-0.0356
WikiQA				
MAP	CI	[0.5796, 0.6868]	[0.6815, 0.7009]	[0.6449, 0.6577]
	Δ	–	+0.0916	+0.0476
MRR	CI	[0.5803, 0.6957]	[0.6969, 0.7143]	[0.6609, 0.6730]
	Δ	–	+0.1050	+0.0634
SICK				
r	CI	[0.8327, 0.8829]	[0.8637, 0.8716]	[0.8211, 0.8290]
	Δ	–	-0.0022	-0.0432
ρ	CI	[0.8025, 0.8126]	[0.7995, 0.8102]	[0.7616, 0.7680]
	Δ	–	-0.0055	-0.0432

Table 5.8: Comparison of MP-CNN with MP-CNN Lite and SM-CNN with Comparison on 10 different sets of hyperparameters and seed configurations. Median incremental difference is relative to first column.

Table 5.8 compares MP-CNN Lite and SM-CNN with the similarity comparison layer to MP-CNN. Note that the similarity comparison layer consists of horizontal and vertical comparison algorithms but only with max pooling and no per-dimensional feature map comparison. The median incremental score change for MP-CNN Lite compared to MP-CNN is no more than 2.45 points worse for TrecQA and WikiQA and in fact often show improvement, whereas for SICK it is no more than 0.55 points worse. There are two main advantages gained.

MP-CNN Lite is much more **robust and less sensitive to hyperparameters**, with tigher confidence interval limits and more predictable accuracy. With MP-CNN Lite, when the lower confidence interval limit is higher than that of MP-CNN, it beats that of MP-CNN by a large margin (e.g. $\sim$11 points for WikiQA). When it is worse, the margin is in comparison rather small. This is visualized by Figure 5.6 - one can notice that the distribution of scores for MP-CNN Lite is concentrated in a small range of the score that is much better than SM-CNN with similarity comparison layer and competitive with MP-CNN. Meanwhile, the score distribution is stretches out on the y-axis for MP-CNN, meaning it is more sensitive to hyperparameters and less predictable.

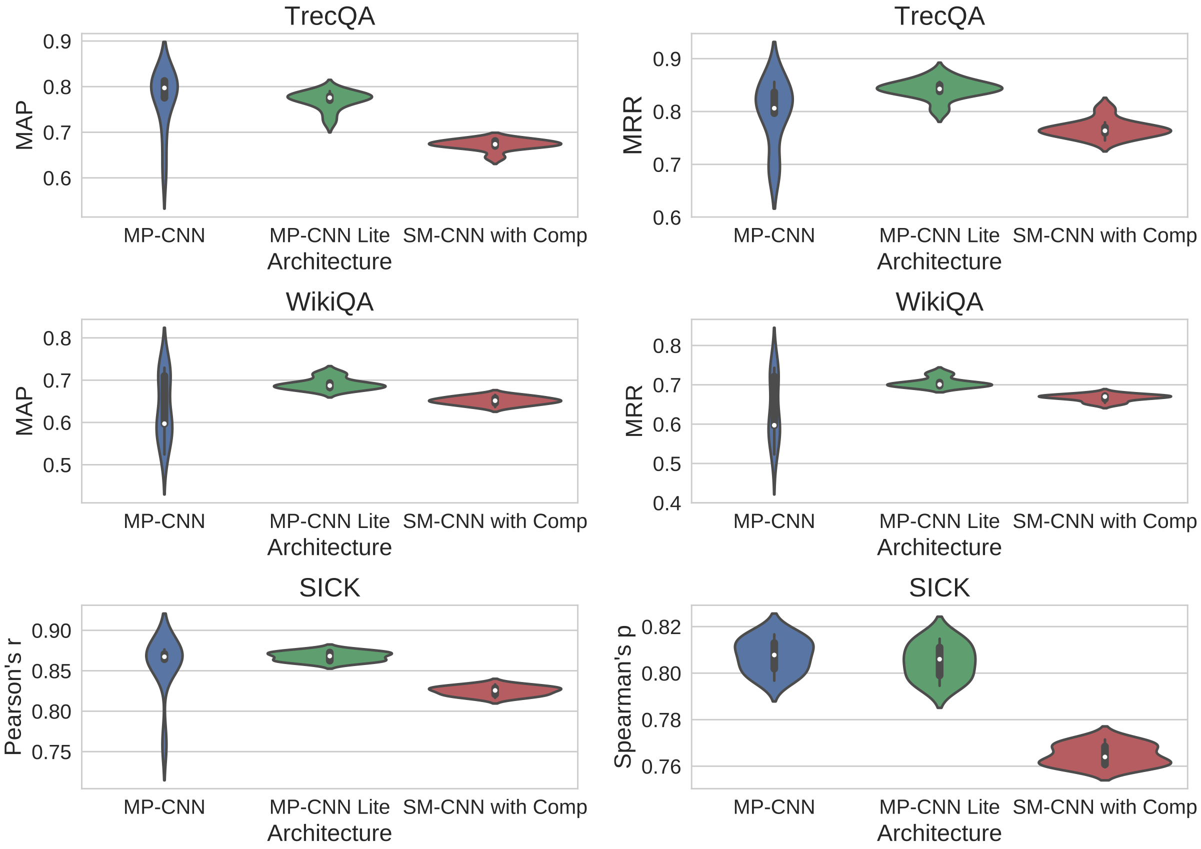


Figure 5.6: Visualization of score distributions of the results in Table 5.8. MP-CNN Lite has essentially the same accuracy as MP-CNN with 8.3x fewer parameters. Both outperform SM-CNN with Comparison.

MP-CNN Lite has much fewer parameters, a 8.3x reduction compared to MP-CNN. This means the model is less prone to overfitting and will use less power. We previously showed that the number of multiplies and parameters are good proxies of the power consumption of a model for convolutional neural networks, and removing per-dimensional filtering will result in less multiplies [44]. Furthermore, smaller model size and fewer operations can benefit the deployment of the model in new environments, such as in the web browser [24] or on serverless architectures [46].

5.3.7 Effects of Wide Convolution

Finding: Wide convolution generally shows slight improvement over narrow convolution, without increasing the number of parameters.

Metric		**MP-CNN Lite Narrow**	**MP-CNN Lite Wide**
Num. of Params		1.04×10^6	1.04×10^6
TrecQA			
MAP	CI	[0.7676, 0.7844]	[0.7649, 0.7846]
	Δ	–	-0.0035
MRR	CI	[0.8317, 0.8493]	[0.8347, 0.8553]
	Δ	–	+0.0070
WikiQA			
MAP	CI	[0.6831, 0.6999]	[0.6975, 0.7151]
	Δ	–	+0.0020
MRR	CI	[0.6975, 0.7151]	[0.7021, 0.7183]
	Δ	–	+0.0021
SICK			
r	CI	[0.8634, 0.8701]	[0.8709, 0.8763]
	Δ	–	+0.0062
ρ	CI	[0.7995, 0.8065]	[0.8097, 0.8175]
	Δ	–	+0.0088

Table 5.9: Effects of wide convolution vs. narrow convolution on lightweight variant of MP-CNN, on 10 different sets of hyperparameters and seed configurations.

We find that although using wide convolution instead of narrow convolution is a simple modification which does not increase the number of parameters, the results are generally better than those obtained using narrow convolution (Table 5.9). With the exception of

MAP for TrecQA, the confidence interval limits as well as median incremental change all showed slight improvement.

Furthermore, using wide convolution allows us to experiment with using bigger filter sizes that detect longer n-gram patterns, which was previously not possible since a restriction of narrow convolution is that the width of the filter cannot be bigger than the number of words. We find that depending on the dataset the optimal max window size can vary (Figure 5.7): for TrecQA it is 3, for WikiQA it is 4, and for SICK it is 2 on the validation set.

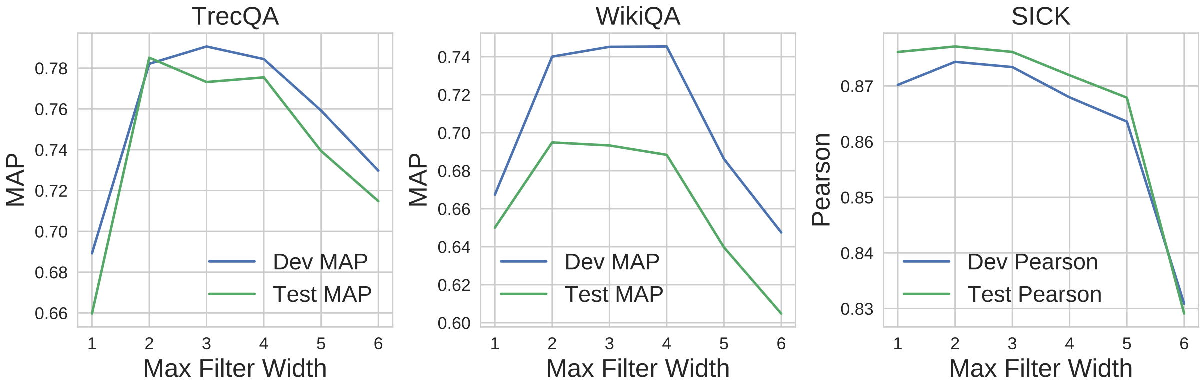


Figure 5.7: Optimal max window width when using multiple filters.

5.3.8 Effects of Attention

Finding: Attention can improve the accuracy of the model, but *idf*-weighted attention does not always work better than basic attention.

Finally, we experiment with adding attention to the MP-CNN lightweight model. Due to GPU memory constraints, it is not straightforward to apply attention to the original MP-CNN model on the TrecQA dataset. We find that attention consistently improved the median incremental score across all metrics and datasets (Table 5.10). However, the improvement is not always major, and it is not clear which of basic attention or *idf*-attention works better (Figure 5.8). It suggests the choice of attention could be an experimental choice depending on the dataset.

Metric		**MP-CNN Lite**	**Basic Attention**	***idf*-Attention**
Num. of Params		1.04×10^6	1.58×10^6	1.58×10^6
TrecQA				
MAP	CI	[0.7487, 0.7781]	[0.7641, 0.7867]	[0.7716, 0.7869]
	Δ	–	+0.0169	+0.0141
MRR	CI	[0.8165, 0.8475]	[0.8261, 0.8494]	[0.8242, 0.8545]
	Δ	–	+0.0032	+0.0024
WikiQA				
MAP	CI	[0.6730, 0.6879]	[0.6809, 0.6924]	[0.6833, 0.6977]
	Δ	–	+0.0048	+0.0109
MRR	CI	[0.6851, 0.7010]	[0.6963, 0.7070]	[0.6968, 0.7124]
	Δ	–	+0.0071	+0.0135
SICK				
r	CI	[0.8598, 0.8649]	[0.8617, 0.8663]	[0.8622, 0.8670]
	Δ	–	+0.0013	+0.0022
ρ	CI	[0.7953, 0.8015]	[0.7981, 0.8037]	[0.7999, 0.8047]
	Δ	–	+0.0023	+0.0032

Table 5.10: Effects of adding attention to MP-CNN lightweight model, on 10 different sets of hyperparameters and seed configurations. Median incremental difference is relative to first column.

5.4 Final Results

In this section we compare the lightweight MP-CNN model and MP-CNN with popular models in the literature across these datasets.

Unlike the methodology described in Section 5.2 and practiced in Section 5.3, in this section we do not fix the hyperparameters and perform a randomized grid search. We select the best model based on the validation set score. We first rank the models by the sum of the MAP/MRR for TrecQA/WikiQA and sum of Pearson's r and Spearman's ρ for SICK, and then select the model with the best validation MAP for TrecQA/WikiQA and Pearson's r for SICK. This enables us to make decisions based on multiple signals.

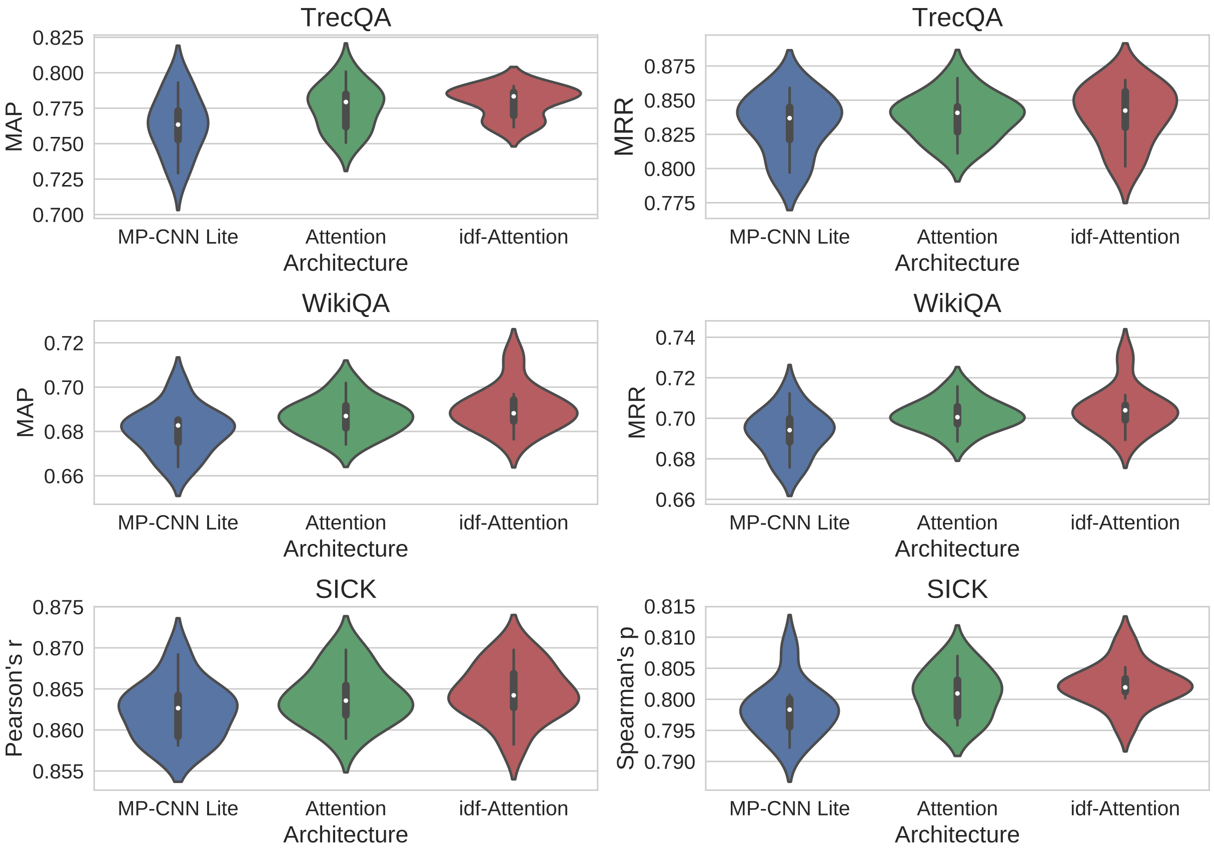


Figure 5.8: Visualization of score distributions of the results in Table 5.10. "Attention" refers MP-CNN Lite with basic attention and "*idf*-Attention" means MP-CNN Lite with *idf*-weighted attention.

MP-CNN Lite is strictly a subset of MP-CNN - it is simply MP-CNN without per-dimensional convolutional and multiple types of pooling. MP-CNN Lite + is MP-CNN Lite with optionally using wide convolution, basic or *idf*-weighted attention, dropout, and sparse features. Dropout is added after the hidden layer in the fully-connected portion of the network. Sparse features are common features used in literature for TrecQA/WikiQA, including proportion of word overlap, word overlap without stop words, and *idf*-weighted variants of these two features.

We see that MP-CNN Lite beats MP-CNN on the TrecQA and WikiQA datasets in Table 5.11. Here MP-CNN Lite + actually performs worse than MP-CNN Lite. In Table

5.12, we see that on SICK again MP-CNN Lite beats MP-CNN, but MP-CNN Lite + can provide additional gain on top of MP-CNN Lite.

Method	**TrecQA**		**WikiQA**	
	MAP	MRR	MAP	MRR
Baseline (2015) [51]	0.695	0.763	0.652	0.665
Neural Answer Selection Model (2016) [26]	0.734	0.812	0.689	0.707
Attentive Pooling CNN (2016) [36]	0.753	0.851	0.689	0.696
ABCNN (2016) [55]	–	–	0.692	0.711
Lexical Decomposition and Composition (2016) [50]	0.771	0.845	0.706	0.723
HyperQA (2017) [45]	0.784	0.865	0.712	0.727
Bilateral Multi-Perspective Matching (2017) [29]	0.802	0.875	0.718	0.731
Inter-Weighted Alignment Network (2017) [39]	0.822	0.889	0.730	0.744
Dynamic-Clip Attention (2017) [2]	0.821	0.899	0.753	0.764
MP-CNN	0.777	0.836	0.693	0.709
MP-CNN Lite	**0.777**	**0.847**	0.704	0.721
MP-CNN Lite +	0.774	0.838	**0.715**	**0.730**

Table 5.11: Test result comparison of MP-CNN Lite and MP-CNN with popular models in the literature for TrecQA (clean) and WikiQA. MP-CNN results are used from [33]. MP-CNN Lite uses just 1.04×10^6 parameters.

We can examine the predictions made by MP-CNN and its lightweight variant to identify some of their strengths and weaknesses.

We observe from Table 5.13 that when the structure of the two sentences closely mirror each other, but with only words replaced with their synonyms, MP-CNN tend to do very well. For example, in the first sentence, the word “little” in sentence one is replaced by “young” in sentence two, and both MP-CNN models, the original model and the lightweight model, give a close prediction. However, when the structure of the sentences changes dramatically, MP-CNN can fail. This is evident in “A person is performing tricks on a motorcycle” and “The performer is tricking a person on a motorbike”, which have very different meanings, but MP-CNN thinks their semantic meaning is close. Also, “A little girl is looking at a woman in costume” is quite different from “A little girl in costume looks like a woman”, but MP-CNN predicts a high relatedness score for the pair.

Method	**SICK Relatedness**		
	r	ρ	MSE
Smooth Inverse Frequency Baseline (2017) [1]	0.8603	-	-
Bi-LSTM (2015) [43]	0.8567	0.7966	0.2736
Skip-Thoughts (2015) [22]	0.8655	0.7995	0.2561
Tree-LSTM (2015) [43]	0.8676	0.8083	0.2532
Pairwise Word Interaction (2016) [16]	0.8784	0.8199	0.2329
Att Tree-LSTM (2016) [56]	0.8730	0.8117	0.2426
MaLSTM (2016) [30]	0.8822	0.8345	0.2286
Inter-Weighted Alignment Network (2017) [39]	0.8833	0.8263	0.2236
MP-CNN	0.8686	0.8047	0.2606
MP-CNN Lite	**0.8751**	**0.8148**	–
MP-CNN Lite +	0.8740	0.8124	–

Table 5.12: Test result comparison of MP-CNN Lite and MP-CNN with popular models in the literature for SICK.

Besides the inability of MP-CNN to do well on examples where the structure is strastically different between the two sentences, another common pitfall seems be the inability of MP-CNN to model hierarchical relationships, such as hierarchical relationships of entities in the sentences. For example, inTable 5.14, we see that "dancing" is a form, or hyponym, of "exercising", but MP-CNN predicts the two sentences have extremely low semantic relatedness. However, for different entities belonging to same parent category, we need to be able to discern them as different. "Tofu" and "butter" are different food entities, and so are "carrots" and "potatoes", but MP-CNN gives high scores to pairs of sentences containing these entitites when they should have a much lower score. We also observe from the table that MP-CNN can benefit from knowing the parts-of-speech information, since in first sentence of the last pair "milk" is an object but "milking" is a verb, and if MP-CNN can distinguish between them it is likely MP-CNN will not give such a high score to the pair.

Sentence A	Sentence B	Gold Label	Prediction
A little girl is looking at a woman in costume	A young girl is looking at a woman in costume	4.7	4.8/4.8
A little girl is looking at a woman in costume	The little girl is looking at a man in costume	3.8	4.4/4.4
A little girl is looking at a woman in costume	A little girl in costume looks like a woman	2.9	4.2/4.0
A sea turtle is hunting for fish	A sea turtle is hunting for food	4.5	4.3/4.1
A sea turtle is not hunting for fish	A sea turtle is hunting for fish	3.4	3.8/3.8
A person is performing acrobatics on a motorcycle	A person is performing tricks on a motorcycle	4.3	4.6/4.7
A person is performing tricks on a motorcycle	The performer is tricking a person on a motorcycle	2.6	4.3/4.3

Table 5.13: Example predictions from the SICK test dataset, using same examples as [22]. The prediction shows the prediction of the full MP-CNN model and the prediction of the lightweight MP-CNN model respectively.

Sentence A	Sentence B	Gold Label	Prediction
A person is exercising	A man is dancing	3.385	1.283/1.211
A woman is slicing butter	Tofu is being sliced by a woman	2.7	4.8/4.8
Carrots are being sliced by a woman	A woman is cutting potatoes	2.1	4.5/4.2
The milk is being drunk by a cat	A drunk man is milking a cat	2.4	4.8/4.4

Table 5.14: Predictions of the SICK test set that differs the most from the ground truth. The prediction shows the prediction of the full MP-CNN model and the prediction of the lightweight MP-CNN model respectively.

Chapter 6

Conclusion

Through careful experimental analysis of MP-CNN across multiple hyperparameters and random conditions, we devised a lightweight model based on MP-CNN that uses strictly a subset of the components of MP-CNN that outperforms MP-CNN on semantic textual similarity and answer selection tasks. Here, outperforming is defined not just in terms of superior accuracy after hyperparameter tuning, but also stability to a wide range of hyperparameters to optimize for model robustness. This is also achieved while using eight times as fewer parameters as MP-CNN. Specifically, we found that per-dimensional convolution and min/mean pooling did not contribute appreciably to the model accuracy or instead hurt model accuracy, and the vertical comparison algorithm and using multiple filter sizes were the most critical components of MP-CNN. Without these latter components, the model accuracy of MP-CNN would be significantly worse. Furthermore, we suggested several modifications to the model, such as using wide convolution and *idf*-weighted attention, that may improve the model accuracy. These results led to extremely competitive results of MP-CNN relative to popular 2016 models on the same tasks, without any need for manual feature engineering.

A promising future direction includes the integration of hierarchical information into MP-CNN. We found during the error analysis that MP-CNN did not perform well in terms of recognizing the hierarchical relationship between entities in the sentence. Recent work on order-embeddings by Vendrov et al. [47] and hyperbolic embeddings by Tay et al. [45] inspire us in this direction. The integration of more linguistic features into the model could also be interesting, as MP-CNN can benefit from being able to recognize parts-of-speech, etc. as discovered during the error analysis.

References

[1] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. 2017.

[2] Weijie Bian, Si Li, Zhao Yang, Guang Chen, and Zhiqing Lin. A compare-aggregate model with dynamic-clip attention for answer selection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1987–1990. ACM, 2017.

[3] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[4] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744, 1994.

[5] Chris Callison-Burch, Miles Osborne, and Philipp Koehn. Re-evaluation the role of bleu in machine translation research. In *11th Conference of the European Chapter of the Association for Computational Linguistics*, 2006.

[6] Ming-Wei Chang, Dan Goldwasser, Dan Roth, and Vivek Srikumar. Discriminative learning over constrained latent representations. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 429–437. Association for Computational Linguistics, 2010.

[7] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051*, 2017.

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

[9] William B Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

[10] Abdessamad Echihabi and Daniel Marcu. A noisy-channel approach to question answering. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 16–23. Association for Computational Linguistics, 2003.

[11] Samuel Fernando and Mark Stevenson. A semantic similarity approach to paraphrase detection.

[12] Yoav Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017.

[13] Weiwei Guo and Mona Diab. Modeling sentences in the latent space. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-volume 1*, pages 864–872. Association for Computational Linguistics, 2012.

[14] Sanda Harabagiu and Andrew Hickl. Methods for using textual entailment in open-domain question answering. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 905–912. Association for Computational Linguistics, 2006.

[15] Hua He, Kevin Gimpel, and Jimmy Lin. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1576–1586, 2015.

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

[17] Hua He, John Wieting, Kevin Gimpel, Jinfeng Rao, and Jimmy Lin. Umd-ttic-uw at semeval-2016 task 1: Attention-based multi-perspective convolutional neural networks for textual similarity measurement. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 1103–1108, 2016.

[18] Yangfeng Ji and Jacob Eisenstein. Discriminative improvements to distributional sentence similarity. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 891–896, 2013.

[19] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

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

[21] Yoon Kim. Convolutional neural networks for sentence classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1746–1751, 2014.

[22] Ryan Kiros, Yukun Zhu, Ruslan R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302, 2015.

[23] Finley Lacatusu, Andrew Hickl, Kirk Roberts, Ying Shi, Jeremy Bensley, Bryan Rink, Patrick Wang, and Lara Taylor. Lccs gistexter at duc 2006: Multi-strategy multi-document summarization.

[24] Yiyun Liang, Zhucheng Tu, Laetitia Huang, and Jimmy Lin. Cnns for nlp in the browser: Client-side deployment and visualization opportunities. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 2018.

[25] Marco Marelli, Luisa Bentivogli, Marco Baroni, Raffaella Bernardi, Stefano Menini, and Roberto Zamparelli. Semeval-2014 task 1: Evaluation of compositional distributional semantic models on full sentences through semantic relatedness and textual entailment. In *Proceedings of the 8th international workshop on semantic evaluation (SemEval 2014)*, pages 1–8, 2014.

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

[27] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[28] George Miller, Christiane Fellbaum, Judy Kegl, and Katherine Miller. Wordnet: An electronic lexical reference system based on theories of lexical memory. *Revue quebecoise de linguistique*, 17(2):181–212, 1988.

[29] Sewon Min, Minjoon Seo, and Hannaneh Hajishirzi. Question answering through transfer learning from large fine-grained supervision data. *arXiv preprint arXiv:1702.02171*, 2017.

[30] Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. 2016.

[31] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[32] Vasin Punyakanok, Dan Roth, and Wen-tau Yih. Mapping dependencies trees: An application to question answering.

[33] Jinfeng Rao, Hua He, and Jimmy Lin. Noise-contrastive estimation for answer selection with deep neural networks. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1913–1916. ACM, 2016.

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

[35] Nils Reimers and Iryna Gurevych. Reporting score distributions makes a difference: Performance study of lstm-networks for sequence tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 338–348, 2017.

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

[37] Royal Sequiera, Gaurav Baruah, Zhucheng Tu, Salman Mohammed, Jinfeng Rao, Haotian Zhang, and Jimmy Lin. Exploring the effectiveness of convolutional neural networks for answer selection in end-to-end question answering. *arXiv preprint arXiv:1707.07804*, 2017.

[38] Aliaksei Severyn and Alessandro Moschitti. Learning to rank short text pairs with convolutional deep neural networks. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 373–382. ACM, 2015.

[39] Gehui Shen, Yunlun Yang, and Zhi-Hong Deng. Inter-weighted alignment network for sentence pair modeling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1179–1189, 2017.

[40] David A Smith and Jason Eisner. Quasi-synchronous grammars: Alignment by soft projection of syntactic dependencies. In *Proceedings of the Workshop on Statistical Machine Translation*, pages 23–30. Association for Computational Linguistics, 2006.

[41] Richard Socher, Eric H Huang, Jeffrey Pennin, Christopher D Manning, and Andrew Y Ng. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *Advances in neural information processing systems*, pages 801–809, 2011.

[42] Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza. Learning to rank answers on large online qa collections. *Proceedings of ACL-08: HLT*, pages 719–727, 2008.

[43] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 1556–1566, 2015.

[44] Raphael Tang, Weijie Wang, Zhucheng Tu, and Jimmy Lin. An experimental analysis of the power consumption of convolutional neural networks for keyword spotting. *arXiv preprint arXiv:1711.00333*, 2017.

[45] Yi Tay, Luu Anh Tuan, and Siu Cheung Hui. Hyperqa: Hyperbolic embeddings for fast and efficient ranking of question answer pairs. *arXiv preprint arXiv:1707.07847*, 2017.

[46] Zhucheng Tu, Mengping Li, and Jimmy Lin. Pay-per-request deployment of neural network models using serverless architectures. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 2018.

[47] Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. Order-embeddings of images and language. *arXiv preprint arXiv:1511.06361*, 2015.

[48] Stephen Wan, Mark Dras, Robert Dale, and Cécile Paris. Using dependency-based features to take thepara-farceout of paraphrase. In *Proceedings of the Australasian Language Technology Workshop 2006*, pages 131–138, 2006.

[49] Mengqiu Wang, Noah A Smith, and Teruko Mitamura. What is the jeopardy model? a quasi-synchronous grammar for qa. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.

[50] Zhiguo Wang, Haitao Mi, and Abraham Ittycheriah. Sentence similarity learning by lexical decomposition and composition. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1340–1349, 2016.

[51] Yi Yang, Wen-tau Yih, and Christopher Meek. Wikiqa: A challenge dataset for open-domain question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2013–2018, 2015.

[52] Xuchen Yao, Benjamin Van Durme, Chris Callison-Burch, and Peter Clark. Answer extraction as sequence tagging with tree edit distance. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 858–867, 2013.

[53] Wen-tau Yih, Ming-Wei Chang, Christopher Meek, and Andrzej Pastusiak. Question answering using enhanced lexical semantic models. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1744–1753, 2013.

[54] Wenpeng Yin and Hinrich Schütze. Convolutional neural network for paraphrase identification. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 901–911, 2015.

[55] Wenpeng Yin, Hinrich Schütze, Bing Xiang, and Bowen Zhou. Abcnn: Attention-based convolutional neural network for modeling sentence pairs. *arXiv preprint arXiv:1512.05193*, 2015.

[56] Yao Zhou, Cong Liu, and Yan Pan. Modelling sentence pairs with tree-structured attentive encoder. *arXiv preprint arXiv:1610.02806*, 2016.