Convolutional Neural Networks for Automated Essay Assessment

Youmna Farag
Murray Edwards College

UNIVERSITY OF CAMBRIDGE

A dissertation submitted to the University of Cambridge in partial fulfilment of the requirements for the degree of Master of Philosophy in Advanced Computer Science

University of Cambridge
Computer Laboratory
William Gates Building
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom

Email: yf273@cam.ac.uk

June 10, 2016

Declaration

I Youmna Farag of Murray Edwards College, being a candidate for the M.Phil in Advanced Computer Science, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Total word count: 14,993

Signed:

Date:

Abstract

We employ convolutional neural networks to the task of automated essay assessment. Our findings show that the task is better casted as a regression problem rather than a classification one. We construct three models that leverage pre-trained word embeddings and operate by applying convolving filters over different textual units: characters, words and sentences. The best results are obtained by the sentence-level model and the performance is further enhanced by combining the predictions of this model with the ones generated by using word convolutions on the document level. We study the effects of different hyperparameters and network configurations such as filter sizes, number of feature maps, activation functions and pooling operations. The impact of fine-tuning the input vectors is examined and the effect of training on more data is investigated. We further conduct a series of experiments and an error analysis to explicate the networks' behaviour. Our results are promising and indicative of the ability of convolutional architectures to evaluate writing quality directly from the text with no recourse to handcrafted features.

Acknowledgements

I would like to thank my supervisors Dr. Marek Rei and Prof. Ted Briscoe for their guidance and support. I am also indebted to Marek for providing the Theano tutorial that was a good starting point for me and for his valuable feedback and constant help. I also acknowledge Dr. Helen Yannakoudakis and Dr. Ronan Cummins for their valuable suggestions throughout the course of this project and would like to thank Helen for letting me use her code for parsing the errors in the dataset.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The task of assessing students' essays has traditionally relied on human graders and their judgement of writing quality. A plethora of attempts have been made to develop systems that are able to automatically evaluate this task [1], [2], [3], [4]. The merits of employing such systems are numerous. It reduces the human labour involved in the task which makes the grading process more cost- and time-efficient. Additionally, essay evaluation is subjective, which leads to potential inconsistencies in the final scores when many human graders are involved. Adopting an automated system can remedy this since all the essays are graded according to the same criteria. Another advantage of automated scoring is enhancing the tutoring process by giving feedback to students about what lowered their grades, hence helping them improve their writing abilities.

Several automated essay assessment (hereafter AEA) systems rely on excessive feature engineering where different textual features are extracted from the training data and fed to a classification algorithm to be mapped to grade classes or scores [1], [5], [6]. Such features might include linguistic, grammatical and discourse properties for the essays. Generating these features is a daunting process that either involves manual annotation or relies on the outputs of other imperfect systems such as syntactic parsers or coreference resolution tools. Other systems leverage shallow statistical features such as

sentence or essay length and use naive classifiers for predictions, which is an oversimplification for the problem [3]. Furthermore, there are latent complex textual features that encode aspects of writing quality yet they are hard to handcraft and hence missed out even by the heavily feature-engineered systems.

To remedy the shortcomings of the aforementioned systems, it is key to develop models that rely on surface features, yet can utilize them to enrich the feature space with more high-level hard-to-encode features. Thereby, we propose employing *Convolutional Neural Networks (CNNs)* to address the task of essay assessment. The network operates on simple word vector representations and extracts more complex features with its different layers to use in the final score prediction. To our knowledge, this is the first implementation of CNNs for essay scoring. We construct three models that apply convolutional operations over different textual units (characters, words and sentences) in an attempt to detect writing quality features and grade the essays accordingly. A word convolution works as a feature detector to extract local contextual information from each window of words in the text and learn possible erroneous word combinations. It aggregates these local features later to extract a global feature vector for the whole essay to be assessed. Character convolutions can be incorporated with convolutions ones to leverage word morphological forms and intra-representations; hence, potentially, detect some writing mistakes such as spelling ones. Sentence convolutions employ a deeper network that convolves over words in each sentence then over the sentences in the essay as a way to represent the bottom-up structure of the essay and detect sentence-relationship features. This deep sentence-level architecture outperforms the document-level word and character models. Moreover, the system performance is further enhanced with a simple combination of the sentence and word models as a way to represent sentence-level and document-level features.

Yannakoudakis et al. [6] compiled the First Certificate in English (FCE) dataset from the Cambridge Learner Corpus (CLC) exam scripts and built the state-of-the-art system to rank them. They exploited numerous lexical

and grammatical features that were used to train rank preference Support Vector Machines (SVMs) [7]. They relied on other resources such as parsers and large corpora to handcraft their features. In the work described herein, we address the same task using convolutional models. We train the system on the public version of the FCE dataset as well as another bigger FCE dataset (full FCE) and test our model on the same test scripts used by the state-of-the-art to make our results comparable to theirs. Although our system underperformed Yannakoudakis et al.'s model, it has interesting results that show that CNNs are a promising avenue in AEA and indicate their ability to learn inherent features for essay scoring directly from the text with no recourse to handcrafted features.

Chapter 2

Background

2.1 Feedforward Neural Networks

2.1.1 Overview

Feedforward neural networks (or *multilayer perceptrons (MLP)*) are one of the most widely used models in supervised machine learning tasks. A network aims at learning a function f that maps a set of labelled input instances to their corresponding outputs:

$$y = f(x; \theta) \tag{2.1}$$

where y is the network's predicted output, x is its input and θ represents the parameters the network learns to make correct predictions. Neural networks do not simply approximate one function, but rather evaluate multiple nested functions:

$$y = f_i(...f_2(f_1(x; \theta_1); \theta_2)...; \theta_i) \tag{2.2}$$

These functions represent the *network layers* and cascading them gives the depth of the model, hence the notion of *deep learning* [8]. In the above equation, f_1 is the *first layer*, f_i is the *output layer* and all the layers in between are *hidden layers*. With this composition of functions, the network

is able to extract higher-level features that are hard to manually encode.

An inherent part of neural models is the non-linear activation functions interleaved with the network's layers. Applying non-linearity enables the network to detect even more complex non-linear features that cannot be extracted by simple linear operations. Additionally, non-linearity is a way to limit the parameters to a certain range of values[1]. The following is a description of integrating the non-linearity in the model. In a neural network, the i-th *neuron* (computational unit) in the j-th layer calculates its output $a_i^{(j)}$ by:

$$a_i^{(j)} = g(\sum_k^{n^{(j-1)}} \theta_{ki}^{(j-1)} a_k^{(j-1)} + b_i^{(j)}) \tag{2.3}$$

where $n^{(j-1)}$ is the size of the previous layer, $\theta^{(j-1)} \in \mathbb{R}^{n^{(j-1)} \times n^j}$ is the weight matrix controlling the mapping between the $(j-1)$-th layer and the j-th one, g is a non-linear activation function and $b_i^{(j)} \in \mathbb{R}$ is a bias term. For the first layer, $a^{(j-1)}$ represents the network's input. Accordingly, the output of the j-th layer is a composition of its neurons' outputs and can be represented as:

$$a^{(j)} = \begin{bmatrix} a_1^{(j)} \\ \vdots \\ a_{n^j}^{(j)} \end{bmatrix} \tag{2.4}$$

where n^j is the number of neurons in the layer. Hence, Equation 2.3 can be written in vector form as:

$$a^{(j)} = g((\theta^{(j-1)})^T \cdot a^{(j-1)} + b) \tag{2.5}$$

where T is the matrix transpose operation. The most commonly used non-linear functions are:

- Logistic function, which is the most prominent type of sigmoid func-

[1] *Sigmoid function* limits the range to [0,1], *tanh* to [-1,1] and *ReLU* eliminates the negative values.

tions:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

- Hyperbolic tangent (tanh), which is a type of sigmoid functions:

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.7}$$

- Rectified linear unit (ReLU), which is a half-wave rectifier [9]:

$$f(x) = max(0, x) \tag{2.8}$$

The final layer in a network performs the system's prediction. The functionality of this layer varies according to the task which includes:

- Linear Regression: It is suitable for problems that have a real-valued output (e.g. the prediction of house prices). In these problems, the size of the output layer is a single neuron representing the system's prediction and calculated by applying Equation 2.5 with $\theta^{(j-1)} \in \mathbb{R}^{n^{(j-1)}}$ and $b \in \mathbb{R}$.

- Softmax: In classification tasks, such as sentiment analysis or image detection, where the output belongs to a finite set of labels, softmax output layers are used. The size of the output layer $\in \mathbb{R}^c$ where c is the number of output labels and Equation 2.5 is applied with $\theta^{(j-1)} \in \mathbb{R}^{n^{(j-1)} \times c}$ and $b \in \mathbb{R}^c$. The following step is to apply a softmax operation to the output vector to "squash" its elements to real values between $[0, 1]$ that add up to one. The output of this operation is vector $z \in \mathbb{R}^c$ representing the probability distribution over the classes. To generate z, the probability of the i-th class in z is calculated by:

$$z_i = P_i(a^{(last)}) = \frac{e^{a_i^{(last)}}}{\sum_{k=1}^{c} e^{a_k^{(last)}}} \tag{2.9}$$

where $a^{(last)} \in \mathbb{R}^c$ is the last layer output. The predicted label is finally

generated by an $argmax$ operation over z to select the class with the highest probability:

$$prediction = argmax(z) \tag{2.10}$$

2.1.2 Training

The previous section describes the process of *forward propagation* wherein each layer's output is dependent on the parameters and outputs of the previous layers. In this section, we detail how MLPs are trained via the process of *backward propagation* (*backpropagation*) [10] and how the network's parameters are optimized.

Objective Function. The goal of neural networks is to learn the values of parameters (θ) that minimize the network's *loss* (error). The *loss function* (*objective function*) varies according to the task. Two of the most commonly used functions are:

1. *Least-squares cost*: This approach is used in linear regression problems, where the system's predictions are real values. The cost for the training instance (x_i, y_i) is calculated by:

$$J(\theta; x_i, y_i) = \frac{1}{2}(h_\theta(x_i) - y_i)^2 \tag{2.11}$$

   where $h_\theta(x_i)$ is the network's prediction for x_i using parameters θ and y_i is the gold label.

2. *Cross-entropy*: It is widely-used in classification tasks where the network's output is a probability distribution over the classes and the final prediction is the class with the highest probability. In order to compute the loss function, the network calculates the negative log-likelihood of

the gold labels given the network's predicted probability distribution:

$$J(\theta; x_i, y_i) = -\sum_{i=1}^{c} Q(y_i) log P_\theta(y_i) \tag{2.12}$$

where c is the number of classes in the system, $Q(y_i) \in \{0, 1\}$ is the ground-truth probability of the i-th class and $P(y_i)$ is the network's predicted probability of this class.

Optimization. Equations 2.11 and 2.12 calculate the cost at the last layer in the network. In order to calculate the errors at previous layers, a *back-propagation* algorithm is implemented:

$$\delta^j = ((\theta^{(j)})^T \cdot \delta^{(j+1)}) \cdot * g'^{(j)} \tag{2.13}$$

where δ^j is the error at layer j, $\cdot *$ is an element-wise multiplication, $g'^{(j)}$ is the derivative of function g from Equation 2.5 and δ of the last layer is calculated by $\frac{\partial}{\partial \theta} J(\theta; x, y)$ which is the partial derivative of $J(\theta)$ from Equations 2.11 or 2.12.

From Equation 2.13, it is obvious that the error at any layer is dependent on the error of the next one, hence the errors *backpropagate* through the network's layers. The next step is to minimize the overall network error. One of the commonly used algorithms to achieve this is *Stochastic Gradient Descent* (*SGD*). This algorithm iterates over the input examples and updates (fine-tunes) the network's parameters θ w.r.t the processed example as follows. First, θ is initialized randomly. Second, the network is fed the first training example (x_1, y_1) and it predicts $h_\theta(x_1)$ via a feedforward process. The error at each layer is calculated and backpropagated to the previous one. The role of SGD is to calculate the derivatives (gradients) of the different layers' errors and use the results to update θ according to a predefined step-size known as the *learning rate*. The network is then fed the second example and the whole process is repeated for each training instance. Processing all the instances indicates one iteration of training. The training proceeds for a

certain number of iterations (epochs); at each iteration the network's cost is further minimized. The gradients are estimated by:

$$\frac{\partial}{\partial \theta^{(j)}} J(\theta) = (a^{(j)})^T \cdot \delta^{(j+1)} \tag{2.14}$$

where $a^{(j)}$ is calculated by Equation 2.5. The parameters are then updated according to:

$$\theta^{(j)} := \theta^{(j)} - \alpha \frac{\partial}{\partial \theta^{(j)}} J(\theta) \tag{2.15}$$

where α is a hyperparameter representing the learning rate.

Preventing Overfitting. As mentioned before, the more the network is trained the more its loss is minimized. This might result in an *overfitting* (*high variance*) problem wherein the trainable parameters highly fit the training set yet fail to generalize over new unattested examples. To remedy this, various approaches could be implemented, we discuss a few here:

- Regularization: It is a way to penalize the network's parameters. Adding a regularization (weight decay) term to cost calculations lowers the performance on the training set, yet makes the model more generalizable. There are different methods for regularization including L_1, L_2 and *dropout* [11]. In this paper, we focus on the second type. L_2 regularization discourages large weights by applying a quadratic penalty to the network's parameters. The loss function is modified to:

  $$J(\theta) := J(\theta) + \lambda \sum_i \theta_i^2 \tag{2.16}$$

  where λ is a hyperparameter representing the *regularization rate* (a larger rate indicates a larger penalty).

- Early stopping [12]: A common way to avoid overfitting is to divide the training data into a training set and a development (dev) one. After each iteration (or a few iterations) of training on the training set, the network is tested on the dev set. The training and dev costs are

observed. Ideally, both the training and dev costs should decline until a certain number of iterations. After that, the training cost will continue decreasing yet the dev cost will start increasing. Training should stop at this optimal point.

- Training on more data: Networks trained on small datasets are more prone to overfitting. Conversely, training on more data makes it harder for the model to overfit. The rationale behind this is intuitive; with a few number of input instances, it is easier for the network to find the parameters that perfectly fit these examples. However, adding more instances makes the problem of searching for these parameters more difficult.

2.2 Convolutional Neural Networks

Convolutional neural networks (*CNNs*) [13], [14] are a special type of feed-forward neural networks. They have been successfully used in various computational tasks including image recognition [15], [16], [17], video detection [18], [19] and text categorization [20], [21], [22]. The input data of a CNN is a multi-dimensional matrix (tensor) that represents various data types including images, videos or texts. The core mathematical operation evaluated by a CNN is a linear *convolutional* function. The convolutional layers are followed by *pooling* layers in most of the network architectures. Figure 2.1 is a graphical depiction for a full network architecture. The following subsections provide a detailed description for CNNs.

2.2.1 Convolutions

Convolutional layers perform the key operation in CNNs. This operation includes 3 essential tensors: *input* $x \in \mathbb{R}^{m \times n}$, *filter* (*kernel*) $w \in \mathbb{R}^{h \times l}$, where $h \leq m$ and $l \leq n$ and *output* (*feature map*) $k \in \mathbb{R}^{m-h+1 \times n-l+1}$. This is a simple case of $2D$ tensors, for instance the input can be a $2D$ image where

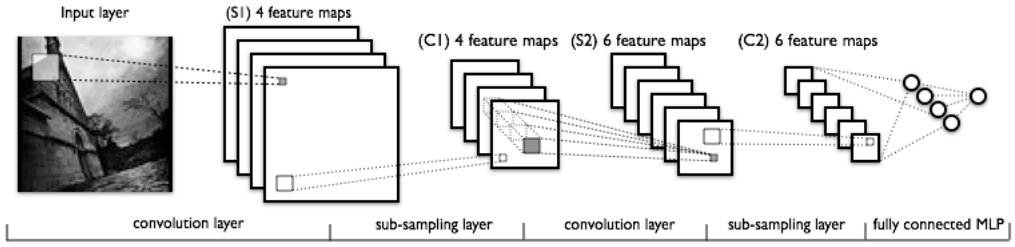


Figure 2.1: Convolutional Neural Network architecture. Figure is from http://deeplearning.net/tutorial/lenet.html.

each pixel has a singular value. Higher order of magnitude tensors can also be applied. For example, $3D$ tensors could be used with images to represent pixel RGB values. In the case of text inputs, the additional dimensions could indicate different representations for words. Traditionally, the higher input dimensions are known as *channels* (from RGB channels). Filters can also have multiple dimensions to create numerous feature maps.

A filter is simply a weight tensor. The idea behind how filters are applied is simple (see Figure 2.2). The filter $w \in \mathbb{R}^{h \times l}$ convolves around the input $x \in \mathbb{R}^{m \times n}$ and at each location, an element-wise multiplication between w and a window of size $h \times l$ in the input is applied. The elements of the resulting matrix are summed up to form a new feature $k_{i,j}$:

$$k_{i,j} = w \ \cdot * \ x_{i:i+h,j:j+l} \tag{2.17}$$

Applying the filter on various positions in the input creates a feature map k of the different detected $k_{i,j}$ features:

$$k = [k_{1,1}, \ldots, k_{1,n-l+1}; \ldots; k_{m,1}, \ldots, k_{m-h+1,n-l+1}] \tag{2.18}$$

In the above equations, the filter shifts vertically and horizontally in the input image. However, it is possible for it to move in one direction which is common in Natural Language Processing (NLP) tasks. Furthermore, the equations demonstrate the simple case of applying a $2D$ filter. Nevertheless, a $3D$ filter $w \in \mathbb{R}^{h \times l \times d}$ would produce d feature maps, hence the dimensions

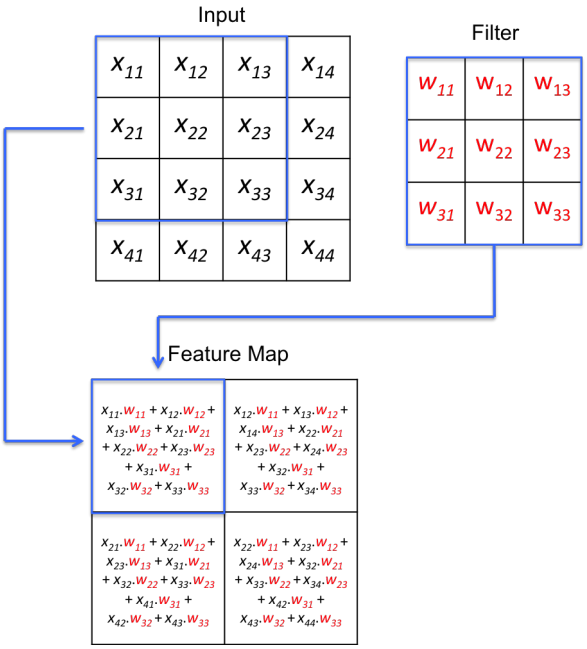


Figure 2.2: Convolutional operation.

of k become $\mathbb{R}^{m-h+1 \times n-l+1 \times d}$, where d is a hyperparameter.

The convolving filter works as a feature detector that extracts local features as it is applied to local windows in the input. For instance, in image recognition, the filters can detect shapes or objects at different locations in an image. In NLP, the filters can extract contextual information from sentence, word or character sequences in the input text. Such information varies according to the task. For instance, it could represent the sentiment of a window of words in sentiment analysis or a detected error in this window in an essay assessment task, as will be discussed in this paper.

So far, the network is a simple linear model that extracts local features from the input data. In order to extract higher-level non-linear features, the linear convolutional layer is interleaved with a non-linear activation function (Equations 2.5 through 2.8). This function is applied element-wise to the feature maps extracted earlier to produce local non-linear features.

2.2.2 Pooling

In CNNs, a pooling operation is traditionally applied to the feature maps extracted by the convolutional layer. Pooling is an aggregation function such as max, average, weighted average and $L2$ norm [8]; the most commonly used ones are max and average pooling [20]. Max pooling attempts to highlight the important features by selecting the highest-value feature(s) in each feature map, while average pooling averages these features. In general, pooling is a way to extract global features from the local ones detected previously by the convolutional operation. There are various merits of pooling. First, extracting the features of the highest values (max pooling) or averaging them helps keep the salient information from the input and reduce the possible noise. Second, pooling is a dimensionality reduction (subsampling) mechanism which leads to more efficient computations, especially in deep networks. It also creates a unified representation for all the variable-sized input instances; therefore, the network's prediction layer would receive inputs of the same dimensionality. Finally, pooling achieves *transitional invariance*. A feature map indicates the occurrence of a specific feature in the input. By aggregating the values in this map, this feature is detected regardless of where it appears in the input. This could be helpful in some tasks where spatial invariance is required, while it is less efficient in others that are more sensitive to feature order.

2.2.3 Fully Connected

As discussed before, the convolutional neurons are locally connected to certain regions in the input which creates a *sparsely connected* network [8]. This is different from the traditional feedforward neural architectures, where each input neuron is connected to each output neuron (Section 2.1). This full connection is achieved at the final layer in CNNs where the output of the last pooling operation is fully connected to the prediction layer. The output layer could perform linear regression or softmax operations based on the task (Section 2.1.1).

2.2.4 Convolutional Premises

In this section, we present some of the core concepts associated with CNNs.

Wide and narrow convolutions indicate the kernel's behaviour around the boundaries of the input instances. In Figure 2.2, the input value x_{11} gets only multiplied by w_{11} in the convolving filter. This is a case of *narrow* convolution (*valid* convolution [8]). One of the disadvantages of narrow convolutions is that the filter size has to be smaller than the input size. To remedy this, *wide* (*full*) convolutions are applied [22]. In this model, the input is padded with enough zero vectors to multiply each x_i value in the input tensor with all the values in the w filter. As a result, the input values near the boundaries are given equal attention as the rest of the values. Additionally, the kernel size becomes independent of the input size. Figure 2.3 illustrates the two convolution types. There is a third type of convolution that lies between the aforementioned two types, wherein zero-padding is added only when necessary [8]. For instance, padding vectors can only be added if the input tensor is smaller than the filter to make them of equal size. Another case is to unify the input instances' size by padding them to be of the same size as the largest instance amongst them.

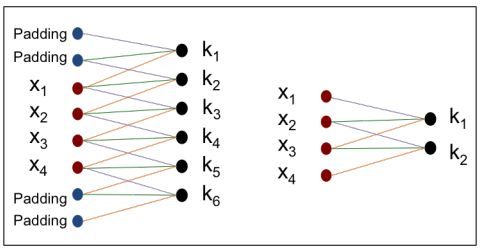


Figure 2.3: The diagram on the left represents a wide convolution while the one on the right represents a narrow one. Filter size = 3 is used in both.

Stride size represents how much the filter shifts at every step. For instance, in image processing, a stride of size 1 indicates a filter shift by one

pixel at a time[2]. Similarly, size 2 indicates that the filter skips one pixel in every shift. Figure 2.4 is an illustration for stride sizes.

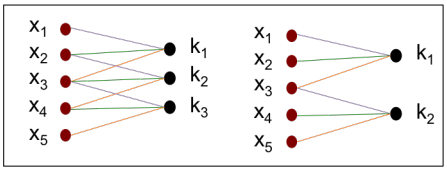


Figure 2.4: The diagram on the left represents a stride of size 1 and the one on the right represents stride of size 2.

Weight sharing. As mentioned earlier, full connectedness in neural networks means that all the input neurons are connected to all the output ones. This implies that each transition from an input to an output neuron requires a different weight parameter. Hence, high-dimensional input tensors and large hidden units would require maintaining high-dimensional weight parameters which is computationally expensive. CNNs remedy this by its local connectivity. In a CNN, the weight tensor is shared between all the input regions (by convolving over them) which dramatically reduces the number of trainable parameters in the network.

2.3 Word Embeddings

An inherent question that is always posed when addressing NLP tasks is how the input words should be represented. Vector representations of words are widely-used as the input format for many machine learning models, including neural networks. In the literature, these vectors are traditionally distributed in a semantic space (word embedding space) where the semantic regularities of words are captured. Such models are known as distributional semantic models (DSM) [23], [24], [25]. They represent each word by a vector that embeds its co-occurrence with neighbouring words. DSMs are spurred by the notion that "Words which are similar in meaning occur in similar

[2]All the previous equations assume the stride size is 1.

contexts."[26]. For instance, words like "*football*" and "*soccer*" are expected to have similar vectors in the embedding space as they appear in similar contexts. The similarity between two vectors a and b is traditionally measured by their cosine similarity [27], [28], [29]:

$$sim(a, b) = cos(\theta) = \frac{a \cdot b}{\|a\| \, . \, \|b\|} \tag{2.19}$$

where θ is the angle between the two vectors.

There are various ways to construct the continuous word space. Neural language models have attracted much attention as efficient methods for creating the embedding space. Various neural architectures have been proposed for this task such as Feedforward Neural Net Language Models (NNLM) [30], Recurrent Neural Net Language Models (RNNLM) [31], [32] and `word2vec` models: Continuous Bag-of-Words (CBOW) and Continuous Skip-gram [33]. Training these models on large corpora (such as Google News or Wikipedea) generates "pre-trained" word representations that can be leveraged in various NLP tasks [34], [35], [36].

2.4 Character Embeddings

Recently, several NLP problems have exploited character-based representations for words [37], [38], [39], [40]. The idea behind character embeddings is to create vector representations for characters and represent the words by combining their character vectors. Representing the words using this technique is a way to capture their morphological forms that word-embeddings fail to capture. For instance, with character representations, words like "*eventful*", "*eventfully*", "*uneventful*" and "*uneventfully*" should have embeddings structured in a related way in the semantic space [37]. Building character embeddings can be achieved by different approaches including CNNs [37], [38], [39] and Recursive Neural Networks [40].

Chapter 3

Related Work

The work described herein is related to two areas of NLP research. In this section we discuss some of the work accomplished in these areas.

3.1 Automated Essay Assessment

There are various attempts in the literature to build automated systems for essay evaluation by modelling a few of the essay grading criteria. Some of these systems aim at evaluating writing coherence. Miltsakaki and Kukich [5] applied the concept of *rough shifts* in the *Centering Theory*[1] [41] to measure the "incoherence" of student essays. Their method relied on manual annotations of coreferential expressions and transition types for centres. Another attempt to evaluate essays by modelling coherence was by Burstein et al. [42] using entity-grids[2] [43] integrated with other "quality writing features" such as spelling checks. Somasundaran et al. [44] leveraged *lexical chains* to assess essay coherence, by building chains of semantically-related words, from

[1] In the Centering Theory, entities are the centres of attention in text and the transition types between them are indicators for text coherence. A rough shift indicates an abrupt transition in topic which is a sign of text incoherence.

[2] Entity-grids is an approach that models the distribution of entity transitions throughout the text.

WordNet, throughout the text and using their properties to train a gradient boosting regressor to evaluate new ungraded essays.

Numerous AEA systems operate by exploiting grammatical and lexical features in the essays. The Project Essay Grade (PEG) [4] is one of the earliest AEA systems that relied on manually extracted surface textual features that are used as "proxies" for the quality of writing. Such features included essay length, counts of different Part-of-Speech (POS) tags and word length. Another system is e-Rater [1] that represented the essays as vectors of weighted features related to grammar, style, vocabulary usage, lexical complexity and discourse properties. New unmarked essays are evaluated based on the cosine similarity between their vectors and the ones of the training essays. Intelligent Essay Assessor (IEA) [2] applied Latent Semantic Analysis (LSA) [45] and Singular Value Decomposition (SVD) to represent essays as $2D$ matrices indicating the relationships between words and their contexts. The evaluation of test essays is accomplished by computing the cosine similarity between the test document matrix and the training ones and assigning the test essay the grade of the most similar training essay. In addition to matrix similarity, IEA leverage other text properties representing style and grammar. The Bayesian Essay Test Scoring sYstem (BETSY) [3] used shallow textual features such as sentence length, word unigrams and bigrams and the number of verbs and verb arguments. It applied Naive Bayes classifiers to categorize the essays into different classes (such as pass or fail).

Yannakoudakis et al. [6] addressed AEA as a rank preference problem and trained SVMs to accomplish the ranking task and discriminate between the essays based on writing quality. Their system relied on a wide-variety of lexical and grammatical features. For the lexical features, they used word unigrams and bigrams. For the grammatical ones, they used: (1) POS unigrams, bigrams and trigrams, (2) phrase structure rules generated from the sentence parse trees to encode the grammatical constructions in essays, which helps detect long-distance syntactic errors, and (3) grammatical relation (GR) features by calculating the distance between heads and dependents in GRs as a way to capture the "grammatical sophistication" in essays. Additional fea-

tures were used such as script length and error-rate. Error-rate was estimated by building a language model from the trigrams extracted from ukWaC corpus [46] plus trigrams from the highly-scoring scripts in the CLC. A trigram encountered in a test essay that is not found in this model is considered an error. Yannakoudakis et al. composed the FCE dataset of CLC exam scripts to evaluate their system (a detailed description of the dataset is provided in Section 5.1). They achieved the state-of-the-art results on this dataset with high correlations with the human graders. In this work, we use their dataset and implement CNN models to address the task.

3.2 Convolutional Neural Networks for NLP

Word convolutions. One of the earliest CNN implementations for text classification were the models by Collobert et al. [20]. They applied two convolutional approaches: a window approach for POS tagging, chunking and Named-Entity Recognition and a sentence one for semantic role labelling. The core difference between the two approaches is the absence of pooling in the window approach. Inspired by Collobert et al.'s sentence approach, Kim [21] implemented a CNN that was generalizable for numerous tasks including sentiment analysis, question type classification and opinion polarity detection. They applied three different convolutional filter sizes and aggregated their outputs with max-pooling. Kalchbrenner et al. [22] constructed a deeper network with two convolutional and two max-pooling layers and a folding layer (performs element-wise summation) after the first pooling layer. They applied their architecture to sentiment and question type classification problems.

Character convolutions. Character convolution is a new area of research in NLP. The following are a few attempts in that area. Santos and Zadrozny [38] built a deep CNN that exploits character-level and word-level representations to perform POS tagging. Inspired by their work, dos Santos and

Gatti [47] incorporated character embeddings with distributed word representations to analyse tweets' sentiments. Other research has attempted to leverage only character representations with no recourse to word embeddings. Kim et al. [37] proposed a network that only relies on character-level representations to build language models for different languages. They concluded that the character model is particularly efficient with morphologically rich languages. Zhang et al. [39] implemented a convolutional character model for text classification of various datasets. All the aforementioned character models build word representation by convolving over the characters in the word then apply a max-pooling operation to generate fixed-size word vectors.

Chapter 4

Design and Implementation

In this work, we implement three convolutional models. We describe the features exploited by these models in the first part of this chapter. In the second part, we detail the models' architectures.

4.1 Features

4.1.1 Word Embeddings

All the models described herein, leverage word vector representations in semantic spaces as their input. We initialize the networks with the pre-trained `word2vec` Skip-gram model trained on Google News articles [36]. This corpus contains around 100 billion words with 3 million unique words and phrases. In this pre-trained model, the words are represented as 300-dimensional vectors. The motivation behind using a model that was trained earlier on a large corpus is to initialize the network with contextually rich word representations, hence help mitigate the effects of data sparsity.

The words that occur in the training set compose the model's vocabulary V. In V, each word has a unique index i, where $1 \leq i \leq |V|$. The distributional vector for each word in V is retrieved from the pre-trained model. As a

result, an embedding matrix $E^{wrd} \in \mathbb{R}^{|V| \times n^{wrd}}$ is constructed, where n^{wrd} is the dimensionality of the pre-trained space (300 in our case). In matrix E^{wrd}, the i-th row corresponds to the semantic vector of the word with index i in V:

$$E^{wrd} = \begin{bmatrix} embed_1 \\ \vdots \\ embed_{|V|} \end{bmatrix} \tag{4.1}$$

The words that are not found in the pre-trained embeddings are initialized randomly with values drawn from a *Gaussian distribution* with mean = 0 and scale = 0.1.

4.1.2 Character Embeddings

The character embedding model is very similar to the word one. In this feature model, vectors are constructed for the characters in a predefined set C. We define C in our experiment as:

ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

0123456789‘ ~!#$%^&*()-_=+,./;’[]\<>?:”{}|

To build the character embeddings, each character in C is randomly initialized with an n^{chr}-dimensional vector, similar to word vector initialization. As a result, a matrix $E^{chr} \in \mathbb{R}^{|C| \times n^{chr}}$ is composed, where the i-th row corresponds to the vector representation of the i-th character in C. The subsequent step is to build the word representation from its character vectors as will be detailed in Section 4.2.2. Any characters not in C (e.g. foreign letters) are assigned a single randomly initialized vector that represents unknown characters.

The decision to exploit character representations as features is motivated by their ability to:

- **alleviate the effects of data sparsity and the out-of-vocabulary words.** In the word-embedding model, unknown words are initialized randomly giving them unmeaningful positions in the semantic space.

However, using character embeddings regularizes this random process. If words like "*eventful*", "*eventfully*", "*uneventful*" and "*uneventfully*" are encountered for the first time in the test set, the character embedding model should be able to represent them in a structured way and to identify that the relation between the first and the second words is similar to the relation between the third and the fourth.

- **detect spelling errors.** With sub-word information, the system can learn that there are "prohibited" sequences of characters and recognize them if they are encountered. For instance, words that end with "*sh*" are pluralized, in case of nouns, or converted to the present tense, in case of verbs, by adding the suffix "*es*". Adding just an "*s*" is a wrong suffix. By representing the words with their character vectors, the model should be able to identify that words like "*crash-es*" and "*slash-es*" are correctly spelled while "*crash-s*" and "*slash-s*" are not.

- **capture grammatical errors.** As mentioned earlier, the premise behind our models is not to rely on hardcoded features or features generated by other systems such as POS tags. In essay grading, identifying POS tags is beneficial as it helps detect grammatical errors. As character models can identify the word morphemes, they can work as a proxy for grammatical rules and hence, identify grammatical errors with no recourse to POS taggers. For instance, the character model can detect that the word "*I*" followed by a verb ending with the suffix "*es*", is an unusual sequence. While this model is expected to be less accurate than POS tags, it can still approximate some grammatical mistakes that word embedding models cannot capture.

4.2 Models

4.2.1 Document-level with Word Convolution (DWC)

The Document-level with Word Convolution (DWC) model processes the input document as a flow of words with no account for sentence boundaries. Figure 4.1 graphically illustrates the DWC architecture[1]. The only features leveraged by this model are word embeddings. The following is a detailed description of the model's architecture:

Input Layer. The first layer in the model works as a projection layer to retrieve the vector representations of the input words from the semantic space. As mentioned before in Section 4.1, each word in the vocabulary V has a unique index i and the word-embedding feature space is represented by a matrix $E^{wrd} \in \mathbb{R}^{|V| \times n^{wrd}}$, where n^{wrd} is the embedding dimensionality. The task of the input layer is twofold. First, it translates each input document into a sequence of its word indices. Second, using this sequence, it performs a lookup operation in matrix E^{wrd} to retrieve the semantic representations of the document's words. The output of the lookup operation is a document embedding matrix $M \in \mathbb{R}^{m \times n^{wrd}}$, where m is the document length.

For instance, if the index sequence representing the document is $\{2, 14, 5, 60\}$, then:

$$M = \begin{bmatrix} embed_2 \\ embed_{14} \\ embed_5 \\ embed_{60} \end{bmatrix} \tag{4.2}$$

Convolutional Layer. This layer is responsible for applying a linear convoluional operation over the input embeddings in order to extract local features. The same equations from Section 2.2.1 are applied. However, in the

[1]This model is inspired by the implementation of Farag [48] in sentiment analysis, which follows the model of Kim [21].

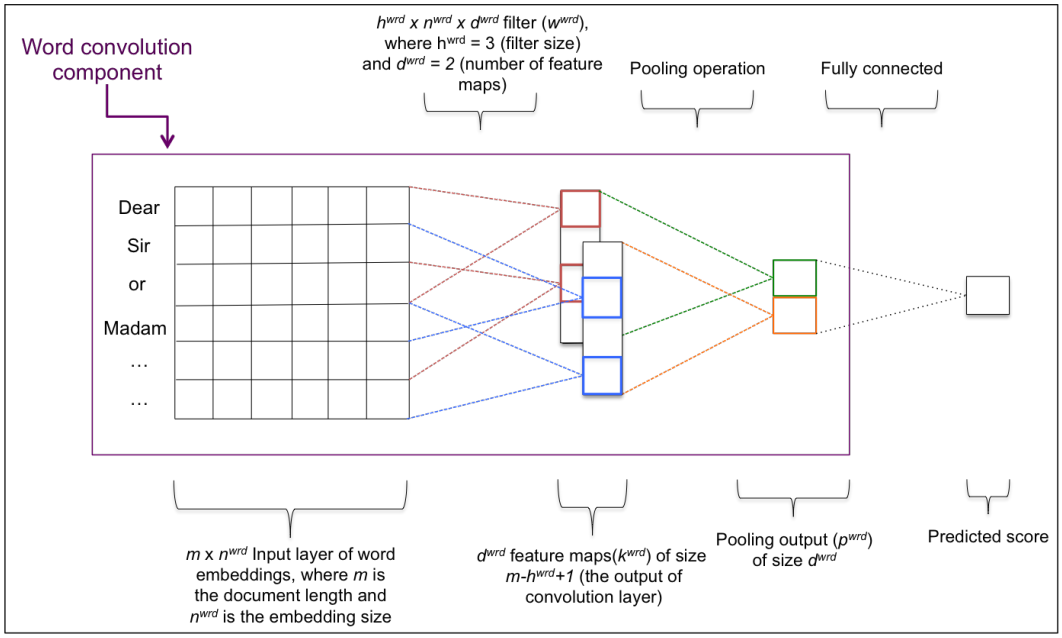


Figure 4.1: Document-level with Word Convolution (DWC) model architecture.

convolution discussed in 2.2.1, the filter moves in both directions. In this task, it is more reasonable to shift the filter only vertically since the document is expressed as a sequence of words (see Figure 4.1). Accordingly, filter $w^{wrd} \in \mathbb{R}^{h^{wrd} \times n^{wrd}}$ is applied sequentially to each position in the input document $[x_1, \ldots, x_m]$ producing a feature map $k^{wrd} \in \mathbb{R}^{m-h^{wrd}+1}$, where:

$$k_i^{wrd} = w^{wrd} \cdot * \, x_{i:i+h^{wrd}} \tag{4.3}$$

Subsequently, a non-linearity is applied element-wise to each value in the feature map:

$$k_i^{wrd} := g(k_i^{wrd} + b) \tag{4.4}$$

where $b \in \mathbb{R}$ is a bias term and g is a non-linear function. We test both the ReLU and tanh functions, as will be discussed later in the experiment.

All the above equations demonstrate the simple case of generating one feature map. In order to further enrich the extracted feature space and exploit the merits of deep learning, a $3D$ filter is applied; hence, the dimensionality of

w^{wrd} becomes $\mathbb{R}^{h^{wrd} \times n^{wrd} \times d^{wrd}}$ and b becomes $\in \mathbb{R}^{d^{wrd}}$, where d^{wrd} is a hyper-parameter representing the number of generated feature maps. As a result, the dimensionality of the convolutional output k^{wrd} becomes $\mathbb{R}^{m-h^{wrd}+1 \times d^{wrd}}$.

In all the models in our experiment, we apply a stride of size one, meaning that the convolutional operation only shifts by one word. With this stride size, overlapping filters are applied which helps extract richer contextual information. Furthermore, the DWC model uses narrow convolutions. The reason is that the filter tensor convolves around the whole document and its size is much smaller than the document length. Additionally, wide convolutions can be helpful in image detection in order to perform multiple convolving operations over the image boundaries. However, in essay scoring, document boundaries might not be as informative as image ones; thereby, including them in one convolution should be sufficient.

Pooling Layer. The local features detected by the convolutional operation pass through a pooling layer in order to extract more global high-level features as described in Section 2.2.2. Aggregating the values of each of the d^{wrd} feature maps results in a vector $p^{wrd} \in \mathbb{R}^{d^{wrd}}$. This is a way to highlight the useful features generated from the convolutional layer and capture a more high-level global context for the essay. In addition, as the input documents are of variable lengths, pooling overcomes such discrepancies and produces vectors of unified lengths for all the documents. In this model, both average and max pooling are examined.

Fully Connected Layer. The final layer in the model is the fully-connected prediction layer that outputs the estimated score of the essay. The input of this layer is the pooled vector $p^{wrd} \in \mathbb{R}^{d^{wrd}}$ and the output is a number representing the score. We experiment the two approaches mentioned in Section 2.1.1: linear regression and classification. On one hand, using linear regression is straightforward, where the model predicts a real-valued score. On the other hand, essay scoring can be casted as a classification task, where 40 labels representing the scores from 1 to 40 are defined. The output layer

applies a softmax then argmax operations to the 40-dimensional vector that represents the class probabilities (see Equations 2.9 and 2.10).

4.2.2 Document-level with Word and Character Convolutions (DWCC)

The Document-level with Word and Character Convolutions (DWCC) model extends the aforementioned DWC model by augmenting it with character convolutions. This section describes the character convolution component (Figure 4.2) and how it is integrated with the DWC model (Figure 4.3).

Character Convolutions are very similar to word convolutions. The core difference between the two methods is that character convolutions are applied over the characters of each word, whereas word convolutions are applied over the words of the whole document. The result of the character convolution operation is a character vector representation for each word (as depicted in the upper part of Figure 4.2). The steps for building these representations are:

- Input Layer: As discussed in Section 4.1.2, the character embedding space $E^{chr} \in \mathbb{R}^{|C| \times n^{chr}}$ is constructed by randomly initializing each character $\in C$ with an n^{chr}-dimensional vector. Similar to the input layer in DWC, the task of the input layer in the DWCC model is to retrieve the character embeddings for each word in the input essays. To achieve that, each input word is tokenized into its sequence of characters which is translated to a sequence of indices $[c_1, \ldots, c_{q_i}]$, where q_i is the length of this word. These indices are used to perform a lookup operation in matrix E^{chr} to retrieve the character embedding representation for each word $N_i \in \mathbb{R}^{q_i \times n^{chr}}$.
- Padding: In order to handle the discrepancies in word lengths, padding vectors are added to each word. Padding is simply adding n^{chr}-dimensional zero vectors to the top and bottom of the embedding N_i to make its

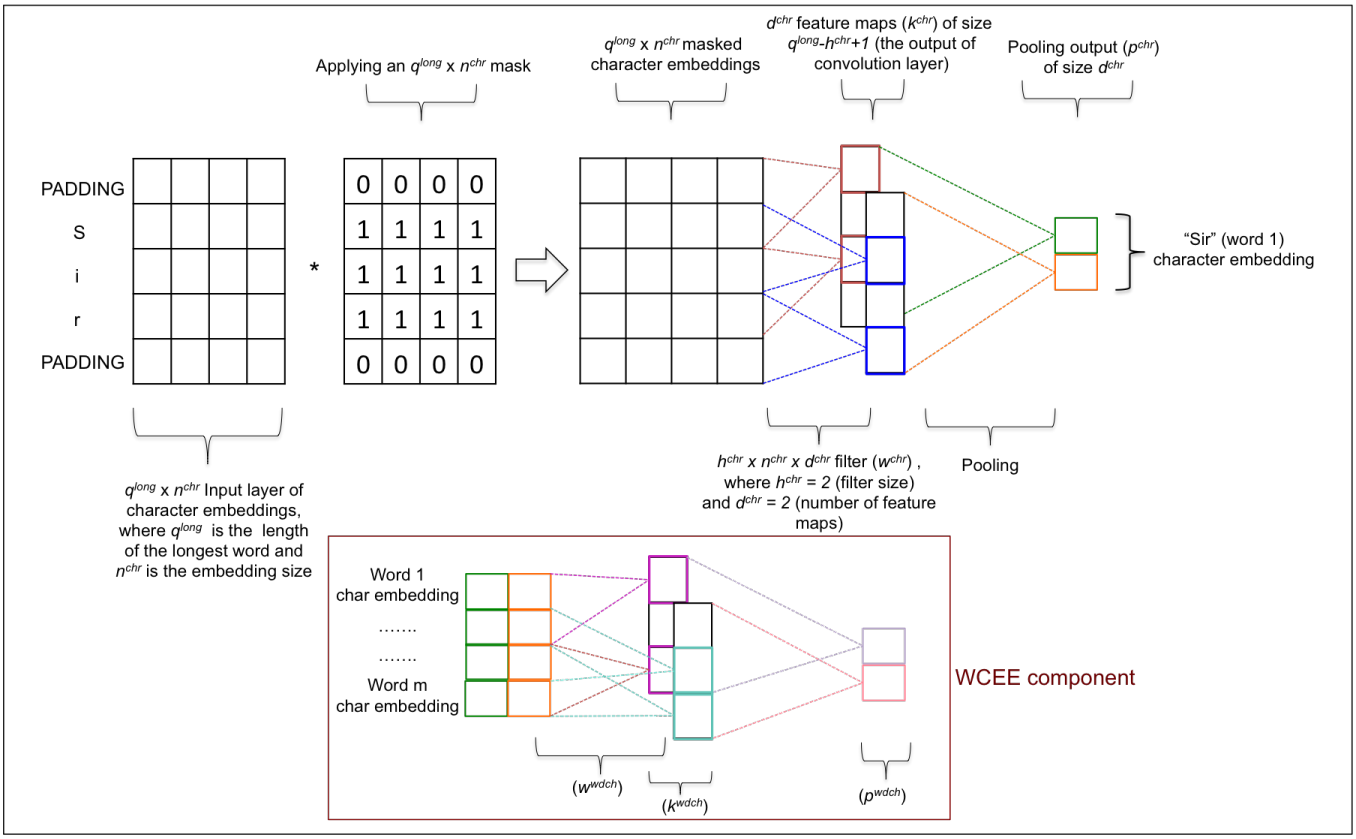


Figure 4.2: Character Convolutions for the Document-level with Word and Character Convolutions (DWCC) model.

dimensionality equal to the one of the longest word's embedding. Let the longest word in the document be of length q^{long}, and the processed word is of length q_i, padding is done as follows: embedding N_i for this word is top-padded with $P_t \in \mathbb{R}^{(q^{long}-q_i)/2 \times n^{chr}}$ and, likewise, bottom-padded with $P_b \in \mathbb{R}^{(q^{long}-q_i)/2 \times n^{chr}}$ (if $q^{long} - q_i$ is an odd number then one padding side will be a row larger than the other). As a result, each i word in the same document will have a character-based embedding $N_i \in \mathbb{R}^{q^{long} \times n^{chr}}$. Unifying the size of character representations for words is computationally more efficient than using variable-sized representations.

- Masking: As character embeddings are fine-tuned via backpropagation, padding sub-tensors will be fine-tuned as well, which produces noise in

the data. To remedy this, a masking operation is performed over the embeddings N to keep all the padding vectors zero-valued and only fine-tune the actual characters in words. Masking is simply an element-wise multiplication of $N \in \mathbb{R}^{q^{long} \times n^{chr}}$ and a mask $\in \mathbb{R}^{q^{long} \times n^{chr}}$ with zero values opposite to the padding vectors and ones elsewhere.

- Convolutional Layer: A convolutional operation is applied to the masked embeddings. Similar to the DWC model, a convolving filter $w^{chr} \in \mathbb{R}^{h^{chr} \times n^{chr} \times d^{chr}}$ is applied to every character position in every word to extract d^{chr} feature maps, where d^{chr} is a hyperparameter. A non-linearity is then applied to the generated local character feature maps $k^{chr} \in \mathbb{R}^{q^{long} - h^{chr} + 1 \times d^{chr}}$. One of the main differences between character convolutions and the word ones applied in the DWC is padding. In the DWC, there is no padding; therefore, narrow convolutions are applied. In the character model, padding is added, but only to unify the word lengths to the longest word. This means that in the long words narrow convolution is applied while in the short ones, the convolution is wide; hence, the convolution here is in between the two types (Section 2.2.4). This should not affect the results due to the masking operation.

- Pooling Layer: Pooling is applied to character feature maps with the same approach as word maps in DWC. The output of this operation for each word is a pooled vector $p^{chr} \in \mathbb{R}^{d^{chr}}$ representing the character-based representation of this word. Since padding is applied, the feature maps would contain zero vectors, which makes average pooling inaccurate, specially for short words. Therefore, max pooling is a more effective technique in this model since the max feature is guaranteed to be of a positive value after applying a ReLU activation function.

Word Convolutions with Character Embeddings (WCCE). This component of the model is similar to the word convolution component in DWC (see Figure 4.1), yet the embeddings for the input words are the estimated character-based representations ($p^{chr} \in \mathbb{R}^{d^{chr}}$), instead of the pre-trained word vectors. The previously described character convolutions can

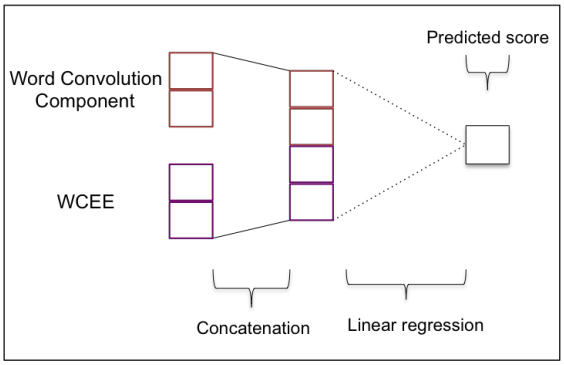


Figure 4.3: Document-level with Word and Character Convolutions (DWCC) model architecture.

be viewed as a component that initializes word representations based on word morphological forms. The input document is translated into a sequence of this new representation and the same operations in Section 4.2.1 are applied. The lower part of Figure 4.2 illustrates the WCCE component. We assign the parameters in this component a *wdch* superscript as it convolves over words represented based on their characters. This component uses a weight filter $w \in \mathbb{R}^{h^{wdch} \times n^{wdch} \times d^{wdch}}$ to produce feature maps $k^{wdch} \in \mathbb{R}^{m-h^{wdch}+1 \times d^{wdch}}$, where m is the document length. The pooling operation is then applied to produce the pooled vector $p^{wdch} \in \mathbb{R}^{d^{wdch}}$.

Integration with Word Convolutions and Fully Connected Layer. Figure 4.3 depicts the integration of both the word convolution component (from the DWC model, see Figure 4.1) and the WCCE one. The outputs of the two components are the pooled vectors $p^{wrd} \in \mathbb{R}^{d^{wrd}}$ and $p^{wdch} \in \mathbb{R}^{d^{wdch}}$ respectively. The integration part is simply concatenating the two vectors which results in $p^{wrd/wdch} \in \mathbb{R}^{d^{wrd}+d^{wdch}}$. Linear regression is then applied to $p^{wrd/wdch}$ in order to predict the score. With this implementation, both character and word representations are used for prediction.

4.2.3 Sentence-level with Word and Sentence Convolutions (SWSC)

Sentence-level with Word and Sentence Convolutions (SWSC) employs a deeper neural network than the aforementioned two models. This model is closer to image detection algorithms by leveraging the compositionality feature of deep convolutional networks. In image detection, the early filters in the network could detect edges from raw input pixels. Subsequent filters could use these edges to detect shapes and further filters could identify objects in the image from these shapes. In NLP, a document could be viewed the same way as an image. It consists of words that are grouped together to form the sentences that eventually compose the whole document. This could be helpful in the task of essay assessment as a human grader evaluates the composition of each sentence individually and the whole structure of the document as a group of sentences. The SWSC is inspired by this premise of compositionality. It consists of two convolutions each one is followed by a pooling layer. The first convolution applies feature detectors (filters) to the raw vector representations of the words in each sentence to extract intermediate-level features that are aggregated by a pooling operation. Accordingly, each pooled feature vector is a representation for each sentence in the essay. In the second convolution, the filters convolve over the sentence representations to detect high-level features followed by a second pooling operation. The last aggregated vector is a representation for the whole document and thereby, used for the final scoring. Figure 4.4 shows the detailed architecture of the model. The main system components are:

Input Layer. The model's input is the same word embeddings used in the DWC system. The main difference is that instead of feeding the system a sequence of all the words in the document, the input is a sequence of words in each sentence. In the SWSC model, each sentence is translated into a sequence of word indices used to retrieve their embeddings, hence creating the sentence embedding.

Padding. Each sentence embedding is padded, using the same technique in Section 4.2.2, to make the length of all the sentences in the same document equal to the length of the longest sentence in this document. As a result, each sentence is represented as an embedding matrix $\in \mathbb{R}^{l^{long} \times n^{wrd}}$ where l^{long} is the length of the longest sentence and n^{wrd} is the word embedding size. Accordingly, the whole document is represented as a tensor $\in \mathbb{R}^{l^{long} \times n^{wrd} \times s}$, where s is the number of sentences in the document. Padding is particularly helpful with short sentences to enable selecting convolutional filter sizes irrespective of the sentence length.

Masking. Similar to the masking technique applied in Section 4.2.2, each sentence embedding is masked to exclude the padding vectors from the fine-tuning process.

First Convolution and Pooling. The same convolutional methods applied in the other models are applied here to the words in each sentence, without crossing sentence boundaries. The model convolves a filter $w^{wrd} \in \mathbb{R}^{h^{wrd} \times n^{wrd} \times d^{wrd}}$ to generate s sets of feature maps $k^{wrd} \in \mathbb{R}^{m-h^{wdch}+1 \times d^{wdch}}$, or in another notation, generate feature maps $k^{wrd} \in \mathbb{R}^{m-h^{wdch}+1 \times d^{wdch} \times s}$. A max pooling operation is then applied to produce s pooled vectors $p^{wrd} \in \mathbb{R}^{d^{wrd}}$ (or a matrix $p^{wrd} \in \mathbb{R}^{d^{wrd} \times s}$). The choice of max pooling here is for the same reasons as mentioned in Section 4.2.2 for character convolutions.

Second Convolution and Pooling. The output of the first convolution and pooling is a vector for each sentence highlighting its most prominent features. In order to construct a higher representation for the whole document, a second convolution is applied to $p^{wrd} \in \mathbb{R}^{d^{wrd} \times s}$. This convolution uses a filter $w^{sent} \in \mathbb{R}^{h^{sent} \times d^{wrd} \times d^{sent}}$, where h^{sent} is a hyperparameter representing the sentence filter height, d^{wrd} is the number of feature maps, per sentence, from the first convolution (filter width), and d^{sent} is a hyperparameter representing the number of feature maps to be extracted by this convolution. The output is a second set of d^{sent} feature maps of size $\in \mathbb{R}^{s-h^{sent}+1}$. A second

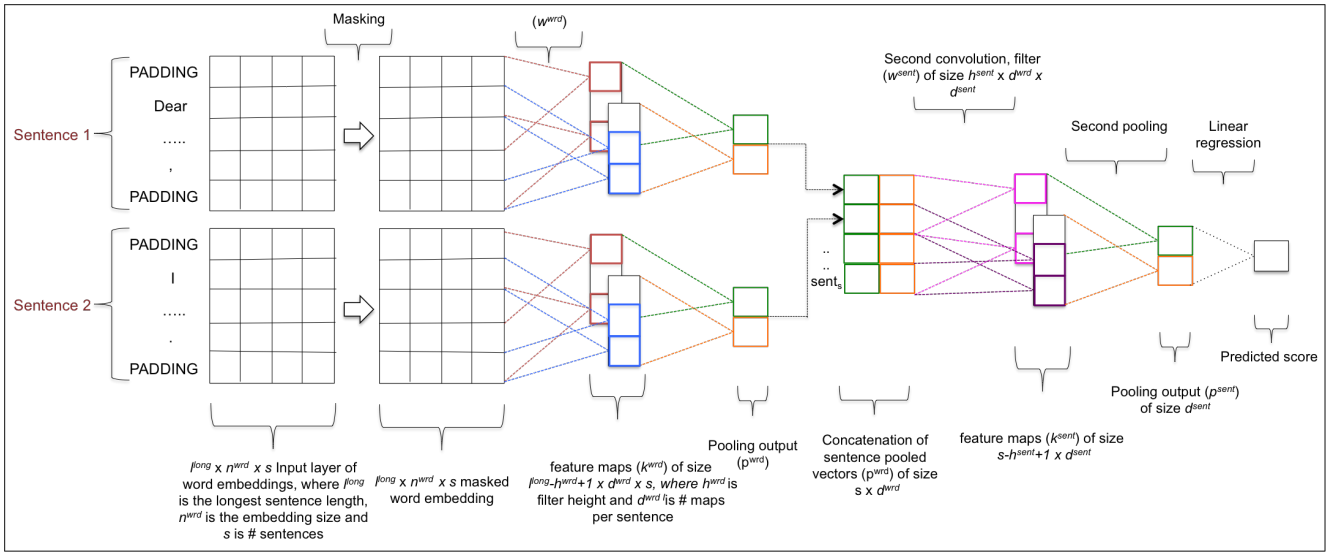


Figure 4.4: Sentence-level with Word and Sentence Convolutions (SWSC) model architecture.

pooling operation is then applied to extract global features for the whole document in a vector $p^{sent} \in \mathbb{R}^{d^{sent}}$.

Fully Connected Layer. The prediction layer is the same as the one in the previous two models. Linear regression is applied to the output of the second pooling operation to generate the predicted score.

Chapter 5

Evaluation

5.1 Datasets

We evaluate our essay scoring system using two datasets extracted from the Cambridge Learner Corpus (CLC). This corpus consists of exam scripts from Cambridge Assessment's English as a Second or Other Language (ESOL) examinations. The scripts were written by English language learners from all around the world. The CLC is a collaborative project between Cambridge University Press and Cambridge Assessment [6]. The extraction of the datasets from the CLC is achieved as follows:

Public First Certificate in English (FCE) Dataset. Yannakoudakis et al. [6] compiled the public First Certificate in English (FCE) dataset[1] from exam scripts written by learners of upper or intermediate levels. This dataset consists of a total of 1,238 scripts, 1,141 scripts from the year 2000 for training and 97 scripts from the year 2001 for testing. The scripts in each set are written by distinct learners and represented in XML format. Each script contains two answers to two different prompts asking the learner to write either an article, a letter, a report, a composition or a short story. A

[1] http://ilexir.co.uk/applications/clc-fce-dataset/

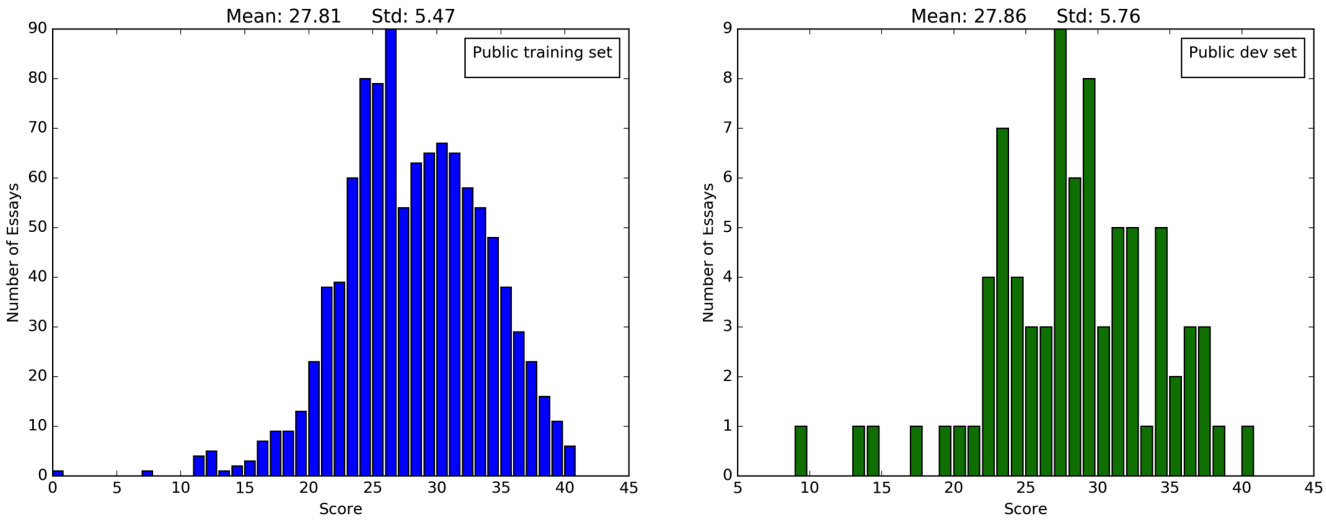


Figure 5.1: Distribution of the public FCE dataset. The x axes represent the scores and the y axes represent the number of essays graded with this score. Mean and standard deviation (std) values are illustrated.

script is annotated with an overall score for the two answers; the range of the scores are from 1 to 40. Additionally, scripts contain meta-data about the learners' native language and age. The script answers are annotated with 80 types of linguistic errors [49]. For example, a spelling error is encoded as:

> *In our* <e type="S">*Acadamy*<c>*Academy*</c></e>
> *we are not allowed to smoke.*

where is the detected error and <c> is its suggested correction. We do not use these annotations for training; they are only used in the error analysis as will be discussed later in this chapter.

We divide the training data randomly into 1,061 scripts for training and 80 scripts as a dev set. The distribution of the scores in the training and dev sets is illustrated in Figure 5.1, where the mean score is around 27, which is normal among upper-intermediate level exam takers.

Full First Certificate in English (FCE) Dataset. It is an extended version of the public FCE; it contains the same scripts from the public FCE

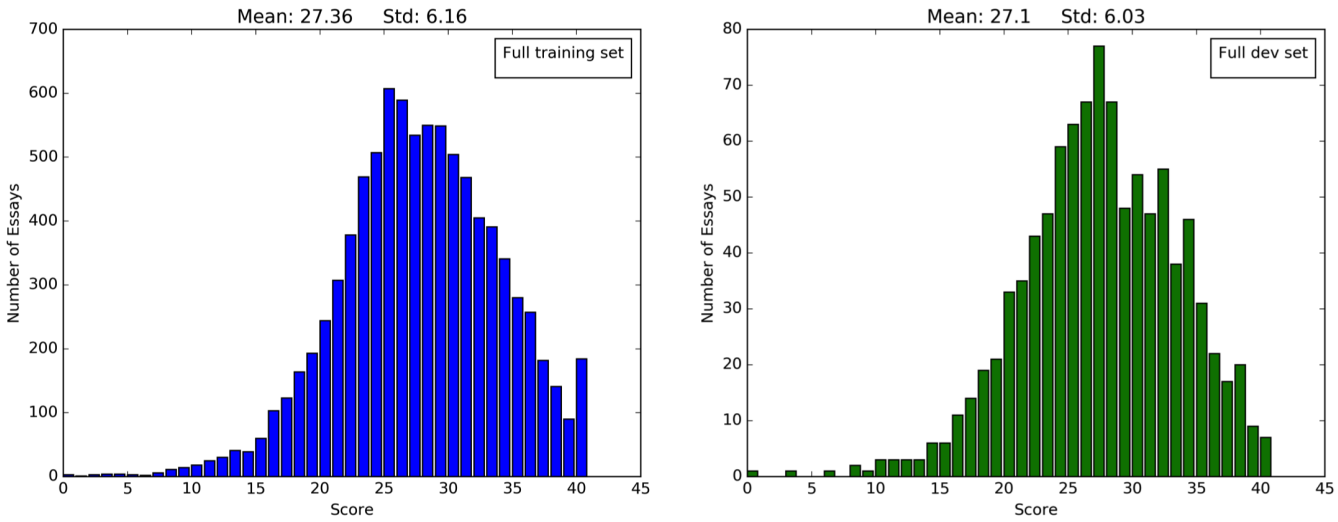


Figure 5.2: Distribution of the full FCE dataset. The x axes represent the scores and the y axes represent the number of essays graded with this score. Mean and standard deviation (std) values are illustrated.

plus additional ones, following the same format. This dataset is not published yet. It has a total of 11,900 scripts. We remove all the scripts with invalid scores which lowers the number of scripts to 9,804. The dataset is randomly divided into 8,824 scripts for training and 980 for the dev set. The full FCE shares the same test set with the public FCE. The distribution of the training and dev sets is depicted in Figure 5.2. The following table is a summarization of the datasets:

Dataset	Training	Dev	Test
Public FCE	1,061	80	97
Full FCE	9,804	980	

Table 5.1: Summarization of the datasets.

5.2 Data Preprocessing

Prior to retrieving the vector representations of the words in the scripts, the following data preprocessing is applied:

Merging answers. As the grade assigned to a certain script includes its two answers, these answers are concatenated as one training instance with an *end_of_answer* tag inserted at the end of each answer. In an initial experiment, we evaluated the task using the two answers as two different instances assigned the same score, but the merging technique yielded better results.

Tokenization. The documents are split on white spaces and punctuation marks (standard tokenization) using the Robust Accurate Statistical Parsing (RASP) system [50].

Unknown words. One of the main issues in NLP algorithms is handling the unknown words. The system's vocabulary V is naturally extracted from the training set, which is problematic when a word out of this vocabulary is encountered. For this reason, we replace all the words that occur only once in the training set with $<UNK>$ tag. Accordingly, there will be a single vector representation for these words. This solves the out-of-vocabulary words problem as these words are initialized with $<UNK>$ as well. The system views the $<UNK>$ words as *rare* words that can be grouped together. It is hard for the system to learn features from a word that occurs only once.

5.3 Evaluation Metrics

We use three evaluation metrics to measure the models' performance:

- *Root mean square error (RMSE):* It measures the standard deviation of the differences between the predicted scores and the ground-truth

ones. Therefore, it gives a quantitative measure for the performance. The lower the RMSE is, the more accurate the predictions are. RMSE is computed by:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(h(x_i) - y(x_i))^2}{n}} \tag{5.1}$$

where $h(x_i)$ is the system's prediction for the i-th document, $y(x_i)$ is the true score for this document and n is the number of test instances.

- *Pearson's product-moment correlation coefficient*: It is also referred to as Pearson's correlation and denoted by (r). It measures the strength of linear dependence between two variables. In our case, the variables are the predicted scores $h(x_i)$ and the gold ones $y(x_i)$, both with size n. Pearson's correlation is estimated by:

$$r = \frac{\sum_{i=1}^{n}(h(x_i) - \mu_h)(y(x_i) - \mu_y)}{\sqrt{\sum_{i=1}^{n}(h(x_i) - \mu_h)^2}\sqrt{\sum_{i=1}^{n}(y(x_i) - \mu_y)^2}} \tag{5.2}$$

$$\text{where: } \mu_h = \frac{1}{n}\sum_{i=1}^{n} h(x_i)$$

$$\text{and analogously: } \mu_y = \frac{1}{n}\sum_{i=1}^{n} y(x_i)$$

The value of $r \in [-1, 1]$. $r = 1$ means there is a total positive correlation, $r = 0$ means there is no correlation and $r = -1$ means there is a total negative correlation. Basically, Pearson's coefficient attempts to fit the data points (h_i, y_i) to a single line; thereby, it is sensitive to outlier points. The higher r is the more accurate the predictions are.

- *Spearman's rank correlation coefficient*: It is also referred to as Spearman's correlation and denoted by (ρ). It is the nonparametric version of Pearson's correlation; it measures the degree of association between two ranked variables. Spearman's correlation is only dependent on the ordinal arrangement of the variables. Hence, unlike Pearson's correlation, it is not sensitive to outlier scores, which makes it a more reliable

metric. In order to calculate Spearman's correlation between $h(x_i)$ and $y(x_i)$, a monotonic function is computed by first, ranking the two variable sets to generate $rank(h(x_i))$ and $rank(y(x_i))$ respectively, then ρ is estimated by:

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} (rank(h(x_i)) - rank(y(x_i)))^2}{n(n^2 - 1)} \tag{5.3}$$

Accordingly, $\rho = 1$ means a total positive correlation, $\rho = 0$ means no correlation and $\rho = -1$ means a total negative correlation. The higher ρ is the more accurate the predictions are. Ranking ties are handled by a *midrank* approach. For instance, if the values of a variable are $[0.1, 0.3, 0.3, 0.5]$ this would yield ranks $[1, 2.5, 2.5, 4]$.

5.4 Other Models

We compare our approaches with two models:

5.4.1 Random Baseline

The idea behind the random baseline is to generate the mean scores that would minimize the prediction error. We generate two sets of scores for the two datasets (public and full FCE). The generation process for each set works as follows. The scores for the test essays are generated randomly in the range $[\mu_{train} - \sigma_{train}, \mu_{train} + \sigma_{train}]$, where μ_{train} is the mean score and σ_{train} is the standard deviation in the training set. The process is repeated 100 times, with different seeds, and each time the three evaluation metrics are calculated using the generated scores and the gold ones. The final evaluation values are the average of the values estimated in all the trials. This baseline is to ensure that the CNN is actually learning useful features and not just generating random results. From the mean and standard deviation values in Figures 5.1 and 5.2, the baseline generates random scores in the range $[22, 33]$

in case of the public FCE and [21, 34] in the full FCE.

5.4.2 Rank Preference SVMs

As described in Section 3.1, the state-of-the-art model for this task was implemented by Yannakoudakis et al. [6] who used a rank preference SVM with various lexical and grammatical features (detailed in Section 3.1 and summarized in Table 5.13). We only compare our system to their Pearson's and Spearman's correlations, since they were the only provided evaluation metrics.

5.5 Experimental Setup

In this section, we describe the various experiments conducted using the three models described in the previous chapter. In all the models, the network's parameters are initialized with values drawn randomly from a Gaussian distribution with mean = 0 and scale = 0.1. The initialized parameters include: weights, biases, embeddings for the words not found in the pre-trained space, character embeddings and word embeddings in the random initialization scenario as will be discussed in the next subsection. The models are constructed using Theano, a library that efficiently evaluates and optimizes mathematical expressions involving multi-dimensional arrays [51]. All the models are trained using SGD with early stopping based on the dev sets. The calculated objective function is least-squares cost in regression models and cross-entropy in classification ones (see Section 2.1.2).

Params	Public FCE	Full FCE
learning rate	{0.0001, **0.001**, 0.01, 0.1}	{0.001, **0.0001**}
word-embed size	300	300
word filter size (h^{wrd})	{2, **3**, 5, 7}	3
# word feature maps (d^{wrd})	{100, 200, **300**, 400, 500}	{**100**, 200, 300}
word pooling	{**avg**, max}	avg
L2 rate	{0, 0.001, **0.0001**}	0.0001
lowercase characters	{yes, **no**}	no
word initialization	{**pre-trained**, random}	pre-trained
prediction	{**regression**, classification}	regression
non-linearity	{**ReLU**, tanh}	ReLU
Fine-tuning	{**yes**, no}	yes
Stride size	1	1

Table 5.2: Hyperparameters and configurations of the DWC model, for both the public and full FCE dev sets. The values in bold are the ones that performed the best on dev sets, in case multiple values were examined.

5.5.1 Document-level with Word Convolution (DWC)

We implement the DWC model as described in Section 4.2.1. Different hyperparameters and network configurations[2] are tested as revealed in Table 5.2; the best performing values are in bold in case multiple values were tried. These values are tuned on the dev sets of both the public and full FCE datasets. In one of the experiments, we cast the problem as a classification task and compare it to regression on the public FCE dataset (Table 5.8). On the same dataset, we explore the effect of lowercasing the words before retrieving their embeddings (this is part of the preprocessing stage) and the impact of using pre-trained embeddings vs. random vector initialization. Additionally, we test average and max pooling operations and ReLU and tanh activation functions. Moreover, we conduct an experiment to investigate the effect of changing the convolving filter size (h^{wrd}) on the performance.

The impact of fine-tuning the word embeddings and how this impact varies relative to the filter size are examined by implementing two experimental setups with each filter size (Table 5.5). The first setup fine-tunes the word embeddings by including them in the network's optimized parameters. In

[2]Network configurations are: pooling technique, lowercasing characters, word initialization, prediction strategy, non-linearity and fine-tuning. For simplification we refer to hyperparameters and configurations as "params" in tables.

the second one, these embeddings are excluded from the trainable parameters, hence functioning as *static* inputs. In the second setup, we perform a slight change in the data preprocessing step; we retrieve the vectors from the semantic space for all the words that appear in the training set, even if they occur once. In addition, words that are absent from this space are randomly initialized with unique vectors, regardless of their frequency in the training set; we do not use the $<UNK>$ technique. For the test set, words are initialized from the training embedding space and if not found, they are initialized from the the pre-trained space. If words are not found in both spaces, they are initialized with unique random vectors. The rationale behind this change is that since the words in this setting are not fine-tuned, they are only represented according to their positions in the pre-trained space. Fine-tuning shifts the vector space to be more tailored for the task; hence, in the fine-tuned setting the vector space, from which the test words should be retrieved, is different from the pre-trained one. Nevertheless, in the static setting the vector space is identical to the pre-trained one; thereby, test words can be retrieved directly from the pre-trained space even if they were not seen before in the training data[3].

We tuned the hyperparameters by trying a few of the values commonly used in the literature [52], [20], [21], [22], [47], [53]. For instance, for the number of feature maps (d^{wrd}), the commonly used range of values is $[100-600]$ [52]. Accordingly, we start with a baseline size of 100 as it was used in similar architectures [20], [21] and tune the other parameters with this size. We later investigate the change in performance when using larger numbers of feature maps (Table 5.7 and Figure 5.3(a)). The hyperparameter selection in this model influences the initial selection of the hyperparameters in the other two models. Table 5.11 depicts the results of the public and full FCE dev sets and the final results on the test data are illustrated in Table 5.12.

[3]We tried using the $<UNK>$ technique in the static space but, removing it yielded better results.

Params	Public FCE	Full FCE
learning rate	{0.0001, **0.001**, 0.01, 0.1}	{0.001, **0.0001**}
char-embed size	{15, 20, 30, **50**}	50
word-embed size	300	300
char filter size (h^{chr})	{**1**, 3, 5}	1
word with char-embed filter size (h^{wdch})	3	3
word filter size (h^{wrd})	3	3
# char feature maps (d^{chr})	{50, **100**, 300}	{**100**, 300}
# word with char embed feature maps (d^{wdch})	{50, 100, **300**}	100
# word feature maps (d^{wrd})	{100, **300**}	{**100**, 300}
char pooling (p^{chr})	max	max
word with-char embed pooling (p^{wdch})	avg	avg
word pooling (p^{wrd})	avg	avg
L2 rate	{0, **0.0001**}	0.0001
lower-case characters in char embeddings	{yes, **no**}	no
word initialization	pre-trained	pre-trained
prediction	regression	regression
non-linearity	ReLU	ReLU
Fine-tuning	yes	yes
Stride size	1	1

Table 5.3: Hyperparameters and configurations of the DWCC model, for both the public and full FCE dev sets. The values in bold are the ones that performed the best on dev sets, in case multiple values were examined.

5.5.2 Document-level with Word and Character Convolutions (DWCC)

We implement the DWCC model as described in Section 4.2.2. The examination of hyperparameters is depicted in Table 5.3. The hyperparameters' initialization is guided by the values used in the literature as well as the results from the DWC model. For instance, for the character convolutional filter size (h^{chr}), 3 and 5 were of the most commonly used values [38], [37], [54], [39]. Similar to the DWC experiment, two DWCC models are created for the two datasets (see Table 5.11 for the results on the dev sets) and tested on the test set (see Table 5.12).

Params	Public FCE	Full FCE
learning rate	{**0.0001**, 0.001}	0.0001
word-embed size	300	300
word filter size (h^{wrd})	3	3
sentence filter size (h^{sent})	{1, **2**, 3, 4}	2
# word feature maps (d^{wrd})	{**100**, 200}	100
# sentence feature maps (d^{sent})	{**100**, 200, 300}	{**100**, 200}
word pooling (p^{wrd})	max	max
sentence pooling (p^{sent})	avg	avg
L2 rate	0.0001	0.0001
lower-case characters	no	no
word initialization	pre-trained	pre-trained
prediction	regression	regression
non-linearity	ReLU	ReLU
Fine-tuning	yes	yes
Stride size	1	1

Table 5.4: Hyperparameters and configurations of the SWSC model, for both the public and full FCE dev sets. The values in bold are the ones that performed the best on dev sets, in case multiple values were examined.

5.5.3 Sentence-level with Word and Sentence Convolutions (SWSC)

We implement the SWSC model as described in Section 4.2.3. This model has a special addition in the data preprocessing stage. Since it relies on sentences, sentence boundary detection is required. For this purpose, we use the standard sentence boundary detection by the RASP system and mark all the boundaries with a special *sentence_end* tag. This way, the model can process each sentence separately.

The hyperparameter tuning on the dev sets is detailed in Table 5.4. Sentence filters (h^{sent}) of sizes $[1, 4]$ are tested and the results are revealed in Table 5.6. As the performance decreased when increasing the filter size, we did not try larger sizes, which was an intuitive decision as well. Similar to the previous experiments, two SWSC models are built for the two datasets (see Table 5.11 for the results on the dev sets) and tested on the test set (see Table 5.12).

5.6 Results and Discussion

A summarization of the final models' results is depicted in Tables 5.11 and 5.12. The following subsections provide a detailed discussion about the models' performance.

5.6.1 Hyperparameters

Word Filter Size. As depicted in Table 5.5, the best filter size in the DWC model is 3; hence, this size is applied to other models. Using filter size 3 puts each word in the context of its previous and next words. For the task of essay scoring, defining each word in this context was enough to make the most accurate predictions by the model. Another rationale behind the efficiency of this size is that many writing errors occur in a window of three words, including spelling errors, word ordering, verb agreement (in short-distance dependency case) and replacement errors. A filter of size 3 might fail to capture long-distance dependency errors; however, it is still more accurate than larger filters. Table 5.5 also shows that fine-tuning improves the performance for all filter sizes, rather than using static embeddings (more analysis about fine-tuning is discussed in the next section). Additionally, an interesting observation in this table is that the impact of fine-tuning becomes less noticeable when increasing the filter size. For instance, the difference in the RMSE between the fine-tuned and the static models with filter size 2 is 0.33, this difference decreases gradually with increasing the filter size until it reaches 0.12 with size 7. The intuitive explanation for the relationship between the filter size and the impact of fine-tuning is that with bigger filters, words are defined in larger contexts; therefore, the influence of fine-tuning becomes less obvious[4].

Sentence Filter Size. Table 5.6 illustrates the impact of changing the sentence filter size (h^{sent}) in the SWSC model when testing on the public

[4]This analysis is inspired by Farag [48].

Filter size	2		3		5		7	
Fine-tuning	yes	no	yes	no	yes	no	yes	no
RMSE	4.75	5.08	**4.74**	4.95	4.81	5.01	4.88	5.00
r	0.57	0.47	**0.58**	0.51	0.57	0.49	0.55	0.49
ρ	0.57	0.49	**0.59**	0.51	0.57	0.48	0.55	0.47

Table 5.5: Results of using different filter sizes and the effect of fine-tuning in the DWC model on the dev set of the public FCE dataset. Feature map size 100 is used and the other hyperparameters/configurations are the best values from Table 5.2.

Evaluation	sentence filter size			
	1	2	3	4
RMSE	5.14	**5.01**	5.11	5.29
r	0.47	**0.51**	0.49	0.42
ρ	0.42	**0.45**	0.43	0.40

Table 5.6: Results of changing sentence filter size (h^{sent}) in the SWSC model on the dev set of the public FCE data set. The hyperparameters/configurations used are the best values from Table 5.4.

FCE dev set. The best results in all evaluation metrics are obtained using $h^{sent} = 2$. There are various interpretations for why this size works the best. First, applying this size, resembles LSA models that are often used to measure text coherence by estimating the similarity between consecutive sentences in text [45], [55], [43]. Convolving over two sentences enables the network to extract features indicating the relatedness between sentences and thereby, model text coherence. On the other hand, decreasing the filter size to 1 fails to extract this relatedness and increasing it over 2 overfits the data and leads to performance decay as the table depicts.

Number of Feature Maps. The number of the feature maps generated by the convolutional operation indicates the number of features the system detects in the essay. Intuitively, the less feature maps the model produces, the more it underfits the data and the more it generates the more it overfits. We experiment with the numbers commonly used in the literature, on the dev sets of both datasets; the results for the DWC model are displayed in Table

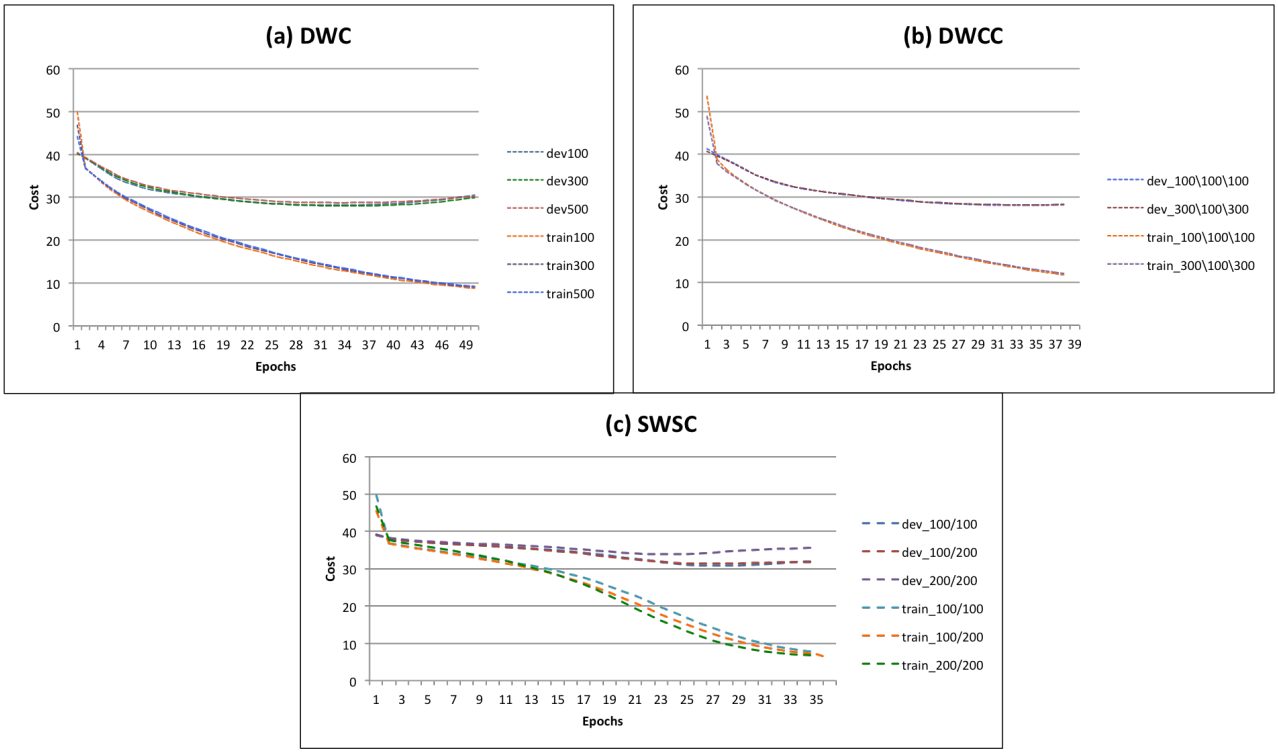


Figure 5.3: A depiction of the effect of changing the number of feature maps in the three models. In DWC, the number in the legend indicates d^{wrd}. In DWCC, the first number is d^{wrd}, the second is d^{chr} and the third is d^{wdch}. In SWSC, the first number is d^{wrd} and the second is d^{sent}.

5.7. Additionally, Figure 5.3 plots the change in the training and dev costs on the public FCE dataset with different number of maps in the three models. While in DWC there is one map to tune (k^{wrd}), in DWCC there are three (k^{chr}, k^{wdch} and k^{wrd}) and in SWSC there are two (k^{wrd} and k^{sent}). From Table 5.7 and Figure 5.3, it is obvious that the differences in the results, when changing the number of feature maps, are ignorable. This supports Collobert et al. [20]'s claim that the choice of this number, provided it is large enough, has a negligible effect on performance. Furthermore, from Figure 5.3, it is apparent that the training process for all the models is monotonic, which is a sign of their stability.

Feature map size	Public FCE			Full FCE		
	RMSE	r	ρ	RMSE	r	ρ
100	4.74	**0.58**	**0.59**	**5.22**	**0.57**	**0.55**
200	4.73	**0.58**	0.58	5.25	0.56	0.54
300	**4.70**	**0.58**	**0.59**	5.28	0.56	0.54
400	4.77	0.57	0.58	-	-	-
500	4.76	0.57	0.58	-	-	-

Table 5.7: Evaluation when using different numbers of feature maps in the DWC model on the dev sets of both public and full FCE datasets. The hyperparameters/configurations used are the best values from Table 5.2.

5.6.2 Classification vs. Regression

As illustrated in Table 5.8, using a regression model in the AEA outperforms classification in all evaluation metrics. Our initial explication for the poor performance of classification was that it predicts the most common scores in the training set. Nonetheless, analysing the classification output reveals that a variety of scores are predicted. In fact, the standard deviation of the classification predictions on the public FCE dev set is 4.1 while in the regression model, it is 3.6. However, classification makes more errors that are closer to the errors by the random baseline (running the random baseline on the public FCE dev set results in $RMSE = 6.7$ vs 6.6 by classification). Additionally, the learning rate for classification had to be higher than regression (0.1 vs. 0.001 respectively). This choice is motivated by the unstable non-monotonic learning curve obtained by the classification model when applying low learning rates. This further indicates that classification is not the right approach for the task as the training process is not stable and the model fails to learn the features associated with each grade class. Furthermore, as indicated in Table 5.8, tanh was the nonlinearity choice for classification as ReLU resulted in exponential growth of network parameters (*exploding gradient problem*).

Prediction	Non-linearity	learning rate	# feature maps (d^{wrd})	*RMSE*	r	ρ
classification	tanh	0.1	100	6.6	0.24	0.35
regression	ReLU	0.001	100	**4.74**	**0.58**	**0.59**

Table 5.8: Classification vs. Regression results using the DWC model on the public FCE dev set. The hyperparameters/configurations not mentioned here are the best values from Table 5.2.

5.6.3 Fine-tuning

As described earlier, fine-tuning enhances the performance by shifting the continuos embedding space to be more task-specific. As a further investigation into the effect of fine-tuning, we explore the embedding space by plotting a sample of it in a $2D$ graph. Reducing the dimensionality of word vectors into a $2D$ space is accomplished using t-Distributed Stochastic Neighbor Embedding (t-SNE) — a technique for dimensionality reduction [56]. We choose a few words from the public FCE training set that were redistributed in the space after fine-tuning and compare their distribution prior to fine-tuning (from the pre-trained space) and after. Figure 5.4 is a graphical depiction for our findings. In the figure, the red crosses represent the static vectors and the blue circles represent the fine-tuned ones. It can be noticed that the words "*on*", "*in*" and "*for*" get clustered together after fine-tuning. Similarly, "*there*", "*their*", "*it*" and "*they*" are grouped. One possible interpretation for that is that the exam takers frequently use the words that belong to the same cluster interchangeably, i.e. they use "*on*" in the context where "*in*" should be used and confuse "*there*" with "*their*" ...etc. We examine the public FCE training set to find how these words were misplaced for each other by investigating the error tags and their suggested corrections. The output confusion matrix is illustrated in Table 5.9. The numbers in the upper left part of the table that represent the first cluster are more indicative than the ones in the lower right representing the second cluster. This confusion matrix is a possible explanation for how some words get redistributed in the space; however, there could be other factors that lead to that, all related to the co-occurrence of words in the training set. To conclude, the exam essays

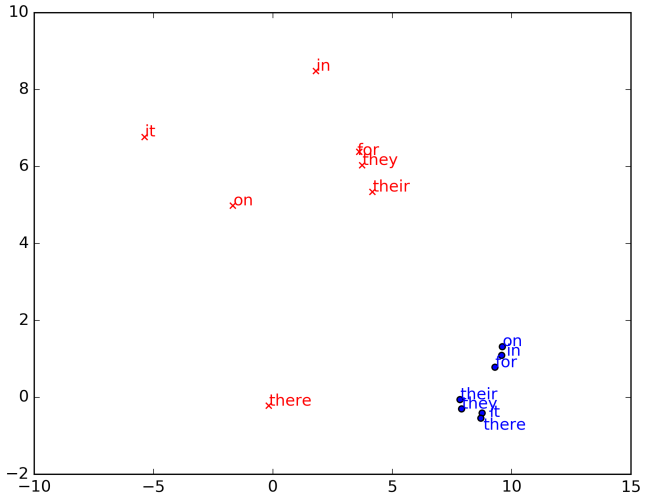


Figure 5.4: The effect of fine-tuning on a few words in the semantic space. The red crosses represent the static word vectors and the blue circles represent the ones fine-tuned on the training set of the public FCE dataset, using the DWC model.

provide new contexts for words and fine-tuning them accordingly makes the semantic space more tailored for the task.

5.6.4 Other Configurations

Nonlinearity. Using ReLU as an activation function outperforms tanh. The rationale behind this could be that ReLU is more efficient in handling the problem of *vanishing gradients*. As the errors backpropagate in the network and with the small gradient range of tanh, the gradients calculated at earlier layers become very small; hence, the parameters of these layers do not get updated or are updated with negligible values. Using ReLU can remedy this problem. This was verified by observing the parameters and the gradients when applying each function. Nevertheless, ReLU might cause the opposite problem of "*exploding gradients*", where the parameters grow exponentially. However, our models do not face this problem except for the

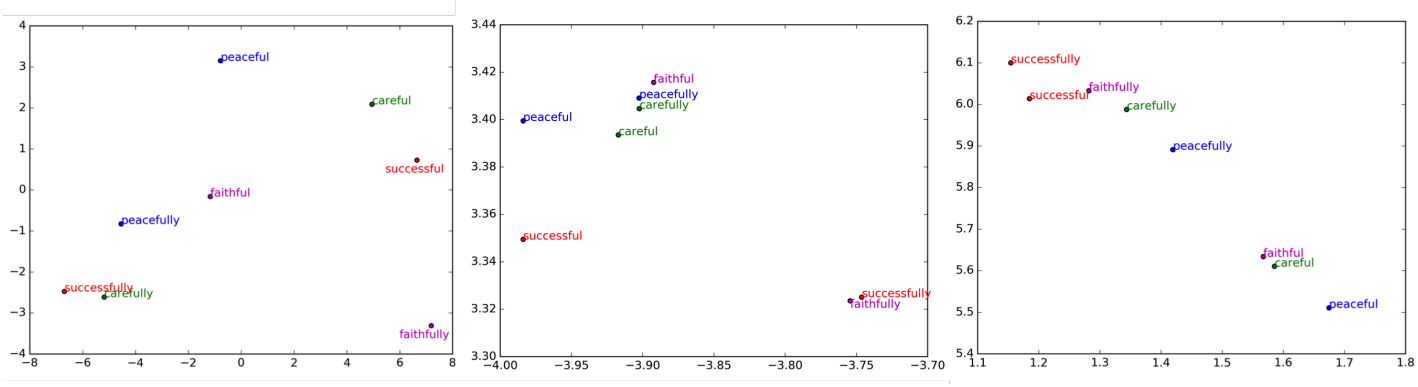
Used Word	Correction						
	in	on	for	their	there	it	they
in	-	153	28	-	-	-	-
on	292	-	11	-	-	-	-
for	21	40	-	-	-	-	-
their	-	-	-	-	8	0	1
there	-	-	-	5	-	13	4
it	-	-	-	0	27	-	37
they	-	-	-	2	11	6	-

Table 5.9: Confusion Matrix for a few words in the public FCE training set.

DWC classification model, where tanh was the better choice.

Lowercasing When the input words are lowercased, the system's performance declines. We conjecture that this is attributed to the fact that while lowercasing increases the frequencies of words and hence, motivates learning more contexts for words, word casing is one of the criteria used in essay grading and using wrong cases is usually penalized. Therefore, better results are achieved with keeping the words in their original form. For instance, if the DWC network encounters a lowercased word after a full stop, it should be able to detect that this is a wrong sequence.

Pooling Unlike the results obtained by similar architectures on other tasks [20], [21], average pooling outperformed max pooling in our models. This can be explicated by the fact that max pooling selects the highest-value feature which might discard other useful information. On the other hand, estimating the average of these features might give a more informative representation for every feature in the essay, which could be more efficient in the task of essay assessment. The only case where max pooling is more efficient is with padded textual units in the DWCC and SWSC models.



(a) Word embeddings

(b) Char embeddings Filter 1

(c) Char embeddings Filter 3

Figure 5.5: Word representations in a $2D$ semantic space created: (a) from the fine-tuned word embeddings in the DWC model, (b) from character embeddings of the words, in the DWCC model, using a character filter (h^{chr}) of size 1 and (c) the same as (b) but with filter size 3. The representations are generated from the public FCE training set.

5.6.5 The Effect of Integrating Character Convolutions

We expected that integrating character convolutions with the DWC model would enhance the performance for the reasons discussed earlier in Section 4.1.2. However, the results of the DWCC model are very close to the ones obtained by the DWC model as depicted in Tables 5.11 and 5.12, which shows that the DWCC model does not help, if not harm, the performance. We further investigate into the effect of applying character convolutions by visualizing a few word vectors with specific morphological regularities in a $2D$ space, using t-SNE, as demonstrated in Figure 5.5. In the figure, the word representations are created: (a) from the fine-tuned vectors in the DWC model, (b) from the character embedding representations of words[5] using a character filter size $h^{chr} = 1$ and (c) the same as (b) but with $h^{chr} = 3$. All the representations are generated from the public FCE training set. In (c), there is a noticeable pattern of how the adverbs and their adjectives are structured in the space. In order to get from an adjective (ending with "*ful*") to its adverb (ending with "*fully*"), the shift should be towards the upper left

[5]This embedding is the output of the upper part of Figure 4.2.

side of the space. It is also interesting how the adverbs are grouped together and the same for the adjectives, except for "*successful*". This structure gets disarranged in (a) and (b). This shows that the character convolutions of size 3 are able to detect the morphological structure of words. In the case of using a filter of size 1, there is actually no convolution operation applied; hence, max pooling operates over the features of individual characters not a sequence where morphological properties can be detected. Surprisingly, size 1 outperforms size 3 and thereby was used in the final DWCC model; although, as graphically revealed, size 3 is more informative. From these observations, we come up with two main conclusions:

1. Despite being successful in a variety of tasks [54], [38], [37], integrating character convolutions with word ones has an ignorable effect in the task of essay assessment.

2. It is possible that the way character convolutions are integrated with word convolutions is not efficient and the model perceives them as noise in the data rather than informative representations. Further investigation is needed to figure out better methods for integration.

5.6.6 Training on More Data

The motivation behind using two datasets, while one is the subset of the other, is to examine the effect of increasing the training data size on the performance. Table 5.11 shows that in the three models, the public FCE (small dataset) outperforms the full FCE when evaluating on their dev sets in all evaluation metrics. However, this is not an even comparison since they are evaluated on different dev sets. When both datasets are evaluated on the same test set, the full set performs better for the three models in all evaluation metrics (Table 5.12).

Interestingly, the SWSC model is the one that gets enhanced the most when training on more data. Its RMSE decreases by 0.77 and its Pearson's and Spearman's correlations increase by 0.19 and 0.17 respectively. This could

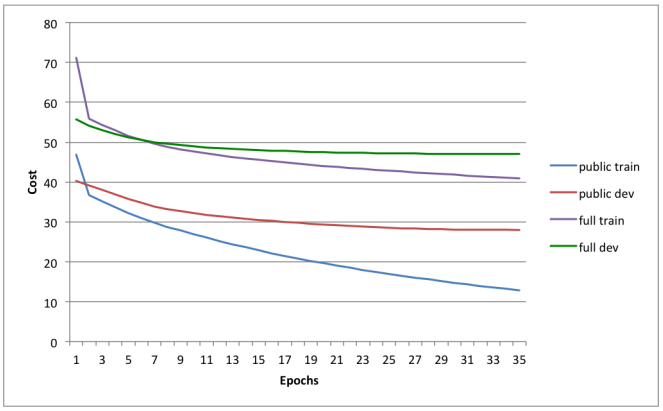


Figure 5.6: Comparison of learning curves of the DWC model on the public and full FCE train and dev sets.

indicate that deeper models need more data to improve their performance. It also makes intuitive sense for the SWSC model to be more sensitive to the size of the training data. Learning from combinations of sentences is sparser than leveraging words; hence, adding more training data helps remedy this sparsity.

In general, with more data, the models process more instances and thereby, learn more features. Furthermore, additional data makes the models more robust against overfitting (Section 2.1.2). This can be clear from Figure 5.6 that compares the learning curves of the public and full FCE DWC models, when using the same number of feature maps (300).

5.6.7 Sensitivity to Order

As described earlier in Section 2.2.2, CNNs are transitionally invariant due to the pooling operation that detects features wherever they occur in text. This property might make our convolutional models less sensitive to word order. To verify this, we conduct two experiments. The first is to shuffle the words randomly in the test set and run the DWC public FCE model. The second is to shuffle the sentences randomly in the test set and run the SWSC public FCE model. Shuffling for both experiments is done 5 times

Model	Evaluation		
	RMSE	r	ρ
DWC with original test set	**5.29**	**0.50**	**0.45**
DWC with shuffled words	5.49	0.45	0.40
SWSC with original test set	**5.59**	**0.38**	**0.39**
SWSC with shuffled sentences	5.86	0.24	0.25

Table 5.10: Evaluation of DWC and SWSC trained on public FCE and tested on: the original test set, shuffled-word test set (with the DWC model) and shuffled-sentence test set (with the SWSC model).

with different seeds and the resulted evaluation metrics are averaged among the trials to eliminate the chance factor. The results are depicted in Table 5.10. The gold scores for these shuffled essays are not provided. However, the target of these experiments is to verify whether the models are sensitive to order or they just mimic Bag-Of-Word models. The results in the table reveal that the performance decreases in all evaluation metrics in the two shuffled models. This shows that word order (in DWC) and sentence order (in SWSC) play an inherent role in the prediction. In the DWC model, the order is maintained between the words inside each window of the size of the convolving filter. Shuffling the words would then result in different detected features; hence, different results. In the SWSC model, consecutive sentences are expected to be related by repetition of words for instance. When the sentence order changes, unrelated sentences would be grouped which would lead to different predictions. This is a further evidence that the SWSC system can model essay coherence.

5.6.8 Comparison between Models

Table 5.12 reveals that out of the three models (DWC, DWCC and SWSC), DWC performs the best when training on the public dataset in the three evaluation metrics. However, on the full dataset, the SWSC model outperforms the other two in RMSE, performs equally in Spearman's correlation and worse in Pearson's correlations. Since Pearson's correlation is sensitive

Model	Public FCE			Full FCE		
	RMSE	r	ρ	RMSE	r	ρ
DWC	**4.70**	**0.58**	**0.59**	5.22	0.57	0.55
DWCC	4.71	**0.58**	**0.59**	5.25	0.57	0.55
SWSC	5.01	0.51	0.45	**5.16**	**0.58**	**0.56**

Table 5.11: Results of the three models on the public and full FCE dev datasets.

to outliers and hence, is not a very robust measurement, we conclude that SWSC trained on the full set is the best model.

We measure the correlations between the three models to verify whether they complement each other, hence combining them is beneficial, or this combination is redundant. We calculate the models' inter-Spearman's correlations when trained on the full set. The correlations show that the DWC and DWCC models are almost identical, since they correlate with 0.99. This further proves that integrating character convolutions was redundant and the model did not learn additional features from it. The DWC and DWCC models highly correlate with the SWSC model with 0.84 and 0.83 respectively. While this is a very high correlation, it still indicates that the SWSC model is able to detect some features that the other two fail to detect and vice-versa. We conduct an experiment to combine the SWSC and the DWC models[6]. We simply average the scores, of the test set, predicted by both models. The output scores have a lower RMSE, higher Spearman's and the same Pearson's correlations in comparison to all the other models. As depicted in Table 5.12, this model (DWC + SWSC (Full)) achieves the best results: RMSE = 4.69 , $r = 0.62$ and $\rho = 0.58$. This is accomplished by a simple method of averaging the scores. Nevertheless, further investigation is needed for other ways to combine the models and leverage the ability to extract document-level features (DWC) and sentence-level ones (SWSC).

[6]We choose the DWC rather than the DWCC because it performs better and is simpler in implementation and faster in running.

Model	Evaluation		
	RMSE	r	ρ
DWC (Public)	5.29	0.50	0.45
DWCC (Public)	5.34	0.48	0.44
SWSC (Public)	5.59	0.38	0.39
DWC (Full)	4.93	**0.62**	0.56
DWCC (Full)	4.95	**0.62**	0.56
SWSC (Full)	4.82	0.57	0.56
DWC + SWSC (Full)	**4.69**	**0.62**	**0.58**

Table 5.12: Results of the three models trained on the public and full FCE datasets when tested on the final test set.

5.6.9 Comparison with Other Models

Table 5.13 compares our best model to the random baselines, generated from the public and full datasets, and the SVM system. The comparison of our system and the random baseline reveals that all our models on all the datasets outperform this baseline. Although this random benchmark tries to minimize the error by predicting the "safe" scores that do not deviate much from the mean score, it still produces higher errors than our system. It also has zero correlations which is normal since it does not differentiate between good and bad essays. The comparison with this baseline shows that the models implemented herein can learn writing quality features that are useful to predict essay grades.

Despite the promising results of our system, it still underperforms the state-of-the-art SVM model [6]. This comparison is using only correlation metrics since no RMSE evaluation was provided in the SVM system. As shown in Table 5.13, the SVM model involves heavy feature engineering whereas our model only leverages word representations. Our system has the advantage of fully automating the prediction process with minimal reliance on outer systems (machines or humans) for feature handcrafting. Since CNNs could extract high-level features that are hard to be manually defined, it would be interesting to combine the CNN and SVM systems in order to learn comple-

Model	Features	Evaluation		
		RMSE	r	ρ
DWC + SWSC (Full)	Word embeddings	**4.69**	0.62	0.58
Random Baseline (Public)	-	6.83	0	0
Random Baseline (Full)	-	7.20	-0.01	-0.01
SVM State-of-the-art	Word unigrams Word bigrams POS unigrams POS bigrams POS trigrams Phrase structure rules GR distance measures Script length Error-rate	-	**0.74**	**0.77**

Table 5.13: Comparison with Other Systems.

mentary features of different levels.

5.6.10 Error Analysis

We further analyse our models' predictions and the properties of the tested essays to investigate any correlations between the two. The essay properties are divided into two sets: general properties and error types.

General Properties. We examine the following properties in the test essays and measure their Spearman's correlations with the 6 models' predictions (three models on two datasets) as well as the gold scores of the essays:

1. Total number of errors: counted as the number of erroneous words detected in the error tags (<e></e>).
2. Number of unknown words (unk): counted as the number of unknown words encountered in the test essays yet do not occur in the public FCE

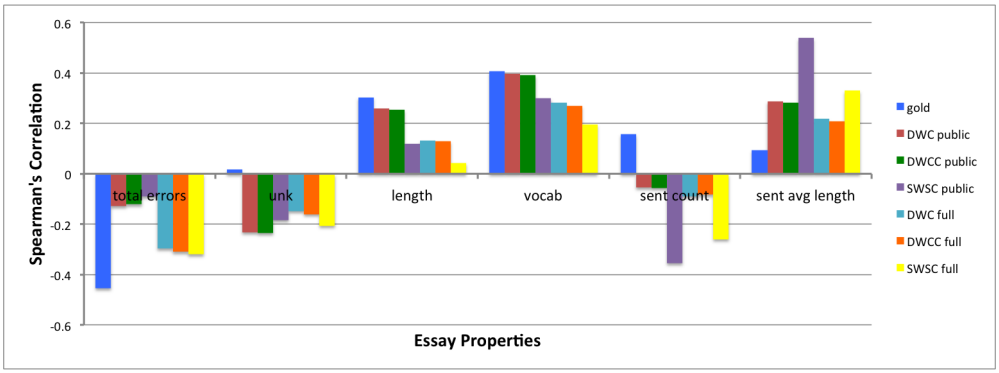


Figure 5.7: Spearman's correlations between the predictions of the different systems and the test essays' properties. The properties from left to right are: number of total errors, number of unknown words (unk), essay length, size of vocabulary, number of sentences and average sentence length.

training set (or occur once). We measure the correlation between this property and the test scores generated by the public and full models. The unknown words that are absent from the full FCE training set are not used in order to unify the comparison criteria, since it is inaccurate to compare the public and full models on different unk counts. This is especially valid since the the average number of unknown words per essay when training on the public FCE is 24 with std = 14 while on the full FCE the average is 8 with std = 9. The last two numbers indicate that using unknown word counts from the full FCE is not very informative. Therefore, we compare the public and full models with the unk counts from the public FCE training data.

3. Essay length: counted as the number of words in the essay.

4. Vocabulary size: counted as the number of unique words used in the essay.

5. Number of sentences.

6. Average sentence length.

The results are plotted in Figure 5.7. Analysing the figure, we come up with a few observations:

- It is obvious that all the models correlate with the total errors, essay length, vocabulary and average sentence length in the same direction as the gold scores, which is a good sign.

- The models' sensitivity to the total number of errors gets higher when training on more data (full FCE), which shows that adding more data improves their ability to detect writing errors.

- On one hand, the gold scores do not correlate with the number of unknown words, which is natural. On the other hand, the public models are more sensitive to this number, since they are trained on less data (less words), than the full models. The exception here is the SWSC model; its correlation slightly increases in the negative direction when adding more data. There is no clear reason for that, yet this can be attributed to the fact that the sentence models are less sensitive to word features than the word models. Therefore, the correlation with the unk words is less indicative in this model. In general, the negative correlation with the number of unknown words, while they do not have an effect on the ground-truth scores, indicates that this number is one of the factors that leads to prediction errors.

- The public DWC and DWCC models' correlations with the essay length are very close to the gold scores. This positive correlation might show that better essays are usually longer. Nonetheless, it is unclear why the correlations get lower with more data. Furthermore, the correlations are sightly lower in the SWSC model. As mentioned before, this could stem from the assumption that this model is more sensitive to sentence features rather than word ones.

- The public DWC and DWCC models' correlations with the vocabulary size is almost identical to the gold scores. This positive correlation is intuitive since good writers tend to use a wide variety of vocabulary. The correlation gets slightly lower, yet still positive, with more data, which needs further investigation. The SWSC model behaves differently for the same reasons mentioned in the previous two points.

Error code	Description	Frequency	Error code	Description	Frequency
R	Replace Error	752	RN	Replace Noun	165
RP	Replace Punctuation	513	UP	Unnecessary Punctuation	140
W	Word Order Error	413	UD	Unnecessary Determiner	133
S	Spelling Error	363	AS	Argument Structure Error	131
MD	Missing Determiner	350	MT	Missing Preposition	106
TV	Incorrect Tense of Verb	324	UT	Unnecessary Preposition	97
MP	Missing Punctuation	289	AGV	Verb Agreement Error	90
RT	Replace Preposition	275	ID	Idiom Wrong	87
RV	Replace Verb	227	RJ	Replace Adjective	86
FV	Wrong Verb Form	193	MA	Missing Anaphor	77

Table 5.14: The most common 20 errors in the test essays with their frequencies.

- The gold scores slightly correlate in the positive direction with the number of sentences. However, all the DWC and DWCC models negatively correlate with this number but with correlations very close to zero so, they can be ignored. This is an expected result since these models are agnostic to sentences. On the other hand, the SWSC models are more sensitive to the number of sentences as a result of how they operate.
- The average sentence length has the highest impact on the SWSC models, as expected, specially the public model (0.5 correlation). This shows that this model is biased towards essays with longer sentences whereas, from the previous point, it is biased against models with many sentences.
- It would be interesting to find a method to moderate the SWSC model's positive bias towards sentence length and negative bias towards sentence count.

Error Types. We further investigate whether the systems have any biases towards any types of errors by measuring Spearman's correlation between their predictions and the frequency of each error type in the test essays. We measure the same correlation with the ground-truth scores. The selected error types for comparison are the most frequent 20 types; their description is depicted in Table 5.14[7]. The errors are detected as the number of erroneous

[7]For more details about error types, we refer the reader to Nicholls [49].

words for each error type. Figure 5.8 is a visualization for the correlations. There are a few observations to draw from this figure:

- There are no high correlations between any error type and the models' predictions or the gold scores, except for -0.45 between the gold scores and spelling errors. This shows that none of the systems or the human graders have strong biases towards certain error types.
- For most of the errors, the models' correlations are in the negative direction, similar to the gold scores, which indicates the system's ability to identify writing errors. The same conclusion can be drawn from the correlations with the total number of errors.
- The ID error type, which means using wrong idioms, positively correlates with all the models with the highest correlation with the gold scores. A possible explanation is that good essay writers who use a variety of vocabulary might tend to write overcomplicated expressions that are linguistically incorrect. For instance, one of these detected errors is:

  <e type="*ID*">*as far as I am concerned*<c>*in my opinion*</c></e>

- The three models, specially their full FCE versions, have close correlations to the gold ones with AGV error, which could indicate their ability to identify grammatical errors.
- In general, adding more data makes the models more sensitive to most of the error types. For instance:
  - The three models become more sensitive to spelling errors after training on more data.
  - The gold scores are slightly sensitive to R, RV and UT error types; whereas, all the systems correlate poorly with these errors. However, adding more data improved these correlations. For instance, with R and RV errors, the correlations transformed from positive to negative with more data.

- It was expected that the DWCC model would perform better on spelling and AGV errors (see the motivations in 4.1.2); however, it performs almost identical to the DWC model.

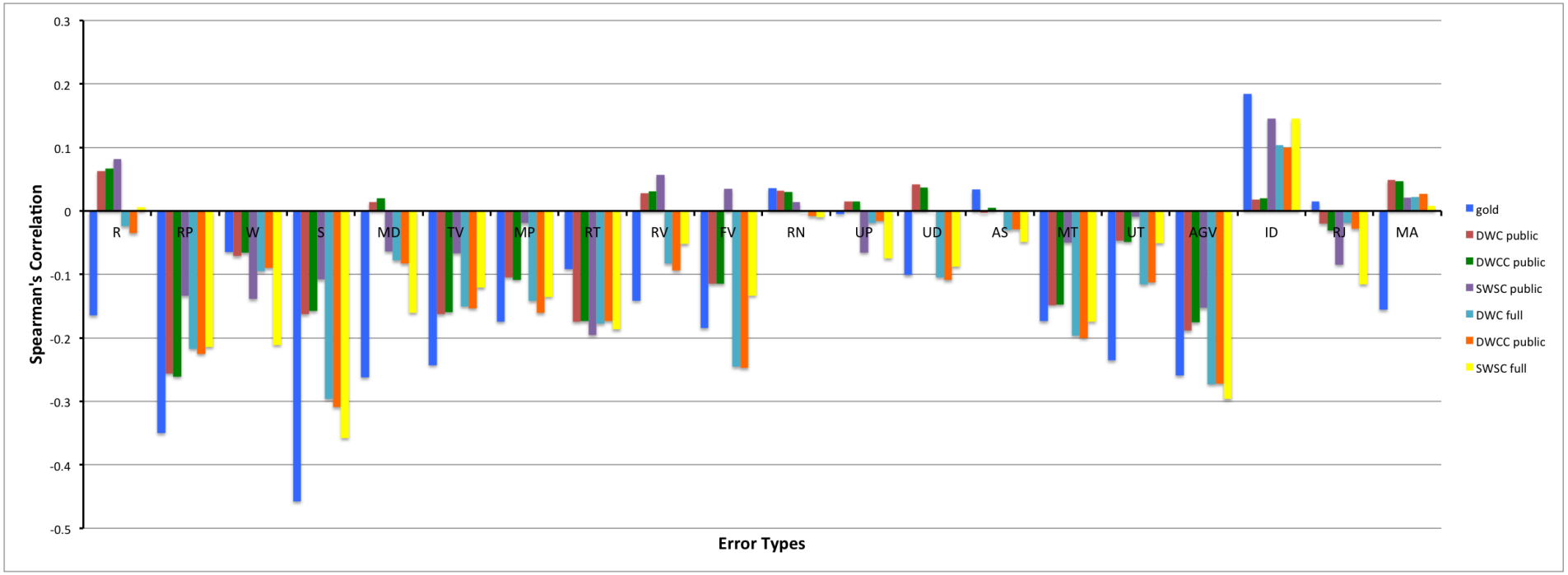


Figure 5.8: Spearman's correlations between different systems and the most 20 common error types in the test essays.

Chapter 6

Summary and Conclusions

We proposed employing convolutional networks to address the task of essay scoring. With initial experimentation, we conclude that this task is better casted as a regression problem rather than classification. This stems from the unstability of the training process and producing prediction errors close to those by the random baseline in the classification model. We built three convolutional architectures. The first two are document-level models with word convolutions in the first and word plus character convolutions in the second. The third is a sentence-level model with word and sentence convolutions. Our results show that the sentence-level model outperforms the document-level ones when trained on more data. The performance was further enhanced by averaging the predicted scores of the first and third models, which motivates investigating more efficient ways to combine them. This combination could provide two perspectives about the essay: an overall perspective as a sequence of words and a more detailed one as a sequence of sentences. On the other hand, leveraging character embeddings did not have any noticeable effect on the performance. We attribute this to two possible reasons: either character embeddings are not useful for essay assessment or other methods for integrating them with word convolutions need to be examined.

Training the neural models revealed the importance of tuning their parameters. Various experiments with filter sizes showed that the best values are

three for word filters, one for characters and two for sentences. Empirical evaluation also exhibited that average-pooling, ReLU activation function and using pre-trained embeddings outperform max-pooling, tanh activation and random vector initialization respectively. Fine-tuning the word embeddings was essential for better results while changing the number of feature maps had minimum effect on the output. Furthermore, training on more data enhanced the performance, especially with the sentence model.

Further investigation into the results led to interesting conclusions. An experiment that shuffles the test essays' words and another that shuffles the sentences revealed that the models are not agnostic to word and sentence order despite the transitionally-invariant nature of CNNs. This could also be an indication that the sentence model can capture writing coherence aspects. A further error analysis showed that the models' predictions are sensitive to essay length, vocabulary size and the number of unknown words. The sentence model is particularly sensitive to the number of sentences and their average length. Moreover, the correlations between the systems' predictions and the number of errors as well as error types in test essays indicate the models' ability to detect writing errors, particularly after adding more training data. In addition, these correlations showed that the models do not have strong biases towards any error types.

When comparing with other systems, all our models outperformed the random baseline yet underperformed the state-of-the-art SVM model. The SVM system relied on heavy feature engineering to handcraft various linguistic and grammatical features. Encoding these features was achieved by other resources such as parsers or large corpora to estimate error rates. On the other hand, our models relied only on word vector representations yet produced promising results that indicate their ability to learn features associated with writing quality. Furthermore, our system has the advantage of being more generalizable and can be applied to other datasets since it only depends on word embeddings as features. Nonetheless, the SVM model lack this generalization ability due to its heavy reliance on other resources and handcrafted features. For instance, the SVM system needs a lot of modifications (rein-

venting the wheel) to be applied to other datasets of other languages, while this should not be a problem in our convolutional models. In general, it would be interesting to combine the SVM and convolutional models in the essay assessment task in order to learn complementary features of different levels.

Bibliography

[1] Yigal Attali and Jill Burstein. Automated essay scoring with e-rater® v. 2. *The Journal of Technology, Learning and Assessment*, 4(3), 2006.

[2] Thomas K Landauer, Darrell Laham, and Peter W Foltz. Automated scoring and annotation of essays with the intelligent essay assessor. *Automated essay scoring: A cross-disciplinary perspective*, pages 87–112, 2003.

[3] Lawrence M Rudner and Tahung Liang. Automated essay scoring using bayes' theorem. *The Journal of Technology, Learning and Assessment*, 1(2), 2002.

[4] Ellis Batten Page. Project essay grade: Peg. *Automated essay scoring: A cross-disciplinary perspective*, pages 43–54, 2003.

[5] Eleni Miltsakaki and Karen Kukich. Automated evaluation of coherence in student essays. In *Proceedings of LREC 2000*, 2000.

[6] Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. A new dataset and method for automatically grading esol texts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 180–189. Association for Computational Linguistics, 2011.

[7] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[8] Yoshua Bengio Ian Goodfellow and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016. URL `http://www.deeplearningbook.org`.

[9] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[10] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learn-

ing representations by back-propagating errors. *Cognitive modeling*, 5 (3):1, 1988.

[11] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15 (1):1929–1958, 2014.

[12] Nelson Morgan and Hervé Bourlard. Generalization and parameter estimation in feedforward nets: Some experiments. In *NIPS*, pages 630–637, 1989.

[13] Yann LeCun et al. Generalization and network design strategies. *Connections in Perspective. North-Holland, Amsterdam*, pages 143–55, 1989.

[14] B Boser Le Cun, John S Denker, D Henderson, Richard E Howard, W Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. Citeseer, 1990.

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

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[18] Bo Chen. *Deep learning of invariant spatio-temporal features from video*. PhD thesis, The University of British Columbia (Vancouver, 2010.

[19] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, pages 568–576, 2014.

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

[21] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[22] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[23] Geoffrey E Hinton. Learning distributed representations of concepts. In *Proceedings of the eighth annual conference of the cognitive science society*, volume 1, page 12. Amherst, MA, 1986.

[24] Jordan B Pollack. Recursive distributed representations. *Artificial Intelligence*, 46(1):77–105, 1990.

[25] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6): 391, 1990.

[26] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[27] Peter D Turney, Patrick Pantel, et al. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37(1):141–188, 2010.

[28] Stephen Clark. Vector space models of lexical meaning. *Handbook of Contemporary Semantics, Wiley-Blackwell, à paraître*, 2012.

[29] Katrin Erk. Vector space models of word meaning and phrase meaning: A survey. *Language and Linguistics Compass*, 6(10):635–653, 2012.

[30] Yoshua Bengio, Holger Schwenk, Jean-Sébastien Senécal, Fréderic Morin, and Jean-Luc Gauvain. Neural probabilistic language models. In *Innovations in Machine Learning*, pages 137–186. Springer, 2006.

[31] Yoshua Bengio, Yann LeCun, et al. Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5), 2007.

[32] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Honza Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE, 2011.

[33] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

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

[35] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

[36] 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.

[37] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. Character-aware neural language models. *arXiv preprint arXiv:1508.06615*, 2015.

[38] Cicero D Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1818–1826, 2014.

[39] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, pages 649–657, 2015.

[40] Thang Luong, Richard Socher, and Christopher D Manning. Better word representations with recursive neural networks for morphology. In *CoNLL*, pages 104–113. Citeseer, 2013.

[41] Barbara J Grosz, Scott Weinstein, and Aravind K Joshi. Centering: A framework for modeling the local coherence of discourse. *Computational linguistics*, 21(2):203–225, 1995.

[42] Jill Burstein, Joel Tetreault, and Slava Andreyev. Using entity-based features to model coherence in student essays. In *Human language technologies: The 2010 annual conference of the North American chapter of the Association for Computational Linguistics*, pages 681–684. Association for Computational Linguistics, 2010.

[43] Regina Barzilay and Mirella Lapata. Modeling local coherence: An entity-based approach. *Computational Linguistics*, 34(1):1–34, 2008.

[44] Swapna Somasundaran, Jill Burstein, and Martin Chodorow. Lexical chaining for measuring discourse coherence quality in test-taker essays. In *COLING*, pages 950–961, 2014.

[45] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[46] Adriano Ferraresi, Eros Zanchetta, Marco Baroni, and Silvia Bernardini. Introducing and evaluating ukwac, a very large web-derived corpus of english. In *Proceedings of the 4th Web as Corpus Workshop (WAC-4) Can we beat Google*, pages 47–54, 2008.

[47] Cícero Nogueira dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *COLING*, pages 69–78, 2014.

[48] Youmna Farag. Convolutional neural network for sentiment analysis. *A project report for: Advanced Topics in Natural Language Processing (R222)*, 2016.

[49] Diane Nicholls. The cambridge learner corpus: Error coding and analysis for lexicography and elt. In *Proceedings of the Corpus Linguistics 2003 conference*, volume 16, pages 572–581, 2003.

[50] Ted Briscoe, John Carroll, and Rebecca Watson. The second release of the rasp system. In *Proceedings of the COLING/ACL on Interactive presentation sessions*, pages 77–80. Association for Computational Linguistics, 2006.

[51] The Theano Development Team, Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

[52] Ye Zhang and Byron Wallace. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*, 2015.

[53] Rie Johnson and Tong Zhang. Effective use of word order for text categorization with convolutional neural networks. *arXiv preprint arXiv:1412.1058*, 2014.

[54] Xiang Zhang and Yann LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

[55] Mirella Lapata and Regina Barzilay. Automatic evaluation of text coherence: Models and representations. In *IJCAI*, volume 5, pages 1085–1090, 2005.

[56] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.