Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Deep Learning Combined With Topic Modeling for Automated Essay Scoring

Author:
Georgios Vasilakis

Supervisor:
Dr. Benny Lo

Submitted in partial fulfillment of the requirements for the MSc degree in Computing Science / Machine Learning of Imperial College London

September 2018

Abstract

Automated Essay Scoring (AES) is the utilization of specialized software to assess essays written in an academic context. With the recent growing interest in Massive Open Online Courses and Graduate Admission Tests, systems that are able to deliver this kind of services are highly motivated since they could potentially reduce rising costs of education, cover the need for grading standards, and provide students with instant feedback, while, at the same time, maintain robustness and objectivity in the grading scheme. Although existing systems have proved to perform well, there are still challenges to be faced in order to fully incorporate AES systems into educational institutions.

The aim of this dissertation is to address these challenges and develop a robust, meaningful and transparent system that could be used in an academic environment on a consistent basis. Motivated by the recent success of Convolutional Neural Networks in Language Modeling applications, this work utilizes a combination of a hierarchical CNN structure with a Topic Modeling algorithm in order to achieve the best possible results. In order to get a better insight of how the system actually assigns scores to essays, a visualization technique was adopted and extended.

Overall, the results of the experiments were satisfying and the system was also able to capture the content of an essay and assign low scores to essays that did not make sense, or were out of topic. Although, in terms of accuracy, the system did not reach the state of the art results, it was the first system to be implemented on the existing high level dataset, laying the foundations for potentially rewarding avenue for future AES systems.

Acknowledgments

I would sincerely like to thank Dr. Benny Lo for his help and guidance throughout my MSc Research Project. His ideas were always insightful and practical, and he was always willing to provide me with all kinds of suggestions. Without his contribution, the successful completion of this report would not be possible. I would also like to thank Frank Lo and Yingnan Sun for their technical suggestions and discussions on my research topic. Finally, I would like to thank my family for their support and encouragement throughout this period.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Purpose of Investigation . . . 2
- 1.3 Contributions . . . 3
- 1.4 Ethics Consideration . . . 3
  - 1.4.1 Ethics Statement . . . 5
- 1.5 Report Structure . . . 5

2 Background and Related Work **7**
- 2.1 Deep Learning Preliminaries . . . 7
  - 2.1.1 Neural Word Embeddings . . . 7
  - 2.1.2 Multi-Layer Perceptrons (MLPs) . . . 9
  - 2.1.3 Convolutional Neural Networks (CNNs) . . . 10
  - 2.1.4 Gated Linear Units (GLUs) . . . 12
- 2.2 Latent Dirichlet Allocation (LDA) . . . 13
- 2.3 Literature Review . . . 14
  - 2.3.1 Automated Scoring in Massive Open Online Courses (MOOC) . . . 14
  - 2.3.2 Automated Essay Scoring Problem . . . 15
    - 2.3.2.1 Pre-Deep Learning Approaches . . . 15
    - 2.3.2.2 Deep Learning Approaches . . . 18

3 Dataset **20**
- 3.1 GMAT Analytical Writing Assessment (AWA) Dataset Overview . . . 20
- 3.2 GMAT AWA Scoring Guide . . . 21
- 3.3 Dataset Augmentation . . . 21

4 Approach **22**
- 4.1 Data Preprocessing . . . 22
  - 4.1.1 Tokenization . . . 22
- 4.2 Network Architecture . . . 23
  - 4.2.1 Neural Word Embeddings . . . 23
  - 4.2.2 Gated Convolutional Neural Network (GCNN) Model . . . 23
    - 4.2.2.1 Lookup Table Layer . . . 23
    - 4.2.2.2 Convolutional and Gated Linear Unit Layer . . . 24
    - 4.2.2.3 Max Pooling Layer . . . 25
    - 4.2.2.4 Dropout Layer . . . 26
    - 4.2.2.5 Fully Connected Layer with Softmax Activation . . . 26
  - 4.2.3 Topic Modeling using Latent Dirichlet Allocation . . . 27

4.3 Fusion of the two Machine Learning Models 28

5 Technical Implementation **31**
5.1 TensorFlow 31
5.2 AES System Implementation 31
5.2.1 Data Preprocessing 31
5.2.2 Building and Reading TFRecords 32
5.2.3 Topic Modeling 33
5.2.4 Training and Evaluation 33

6 Training Procedure **35**
6.1 Trainable Parameters 35
6.2 Stochastic Gradient Descent (SGD) 36
6.3 Model Hyper-parameters 37
6.3.1 Learning rate 37
6.3.2 Batch size 37
6.3.3 Momentum 38
6.3.4 Dropout 38
6.3.5 Gradient Clipping 38
6.3.6 Number of Training Steps 38
6.3.7 Additional Hyper-parameters 38
6.4 Automated Essay Scoring System Training 39

7 Evaluation **43**
7.1 Evaluation Results 43
7.2 Setting a Threshold 49
7.3 Discussion 50

8 Black-Box Visualization **52**
8.1 Visualization Approach 52
8.2 Visualization Examples 54

9 Conclusions and Future Work **56**

List of Figures

2.1 Image taken from https://algorithmia.com/algorithms/nlp/Word2Vec. Example of relation between words after the training of the model. 8
2.2 Image taken from Efficient Estimation of Word Representations in Vector Space, 2013. Presentation of the two models that word2vec is comprised of. 8
2.3 Image taken from https://github.com/cazala/synaptic. Vanilla MLP, with one hidden layer. The nodes of each layer correspond to values while the connections represent the weights of the network. 10
2.4 Image taken from hhttps://medium.com/@rohanthomas.me/convolutional-networks-for-everyone-1d0699de1a9d. Example of a CNN architecture. 11
2.5 Image taken from http://cs231n.github.io/convolutional-networks/. Example of a Max Pooling operation 12
2.6 Source: https://en.wikipedia.org/wiki/Latent_Dirichlet_Allocation . Plate notation representing the LDA model. 14

4.1 Concatenation of the neural word embedding vectors into a Lookup Table. . . 23
4.2 GCNN model architecture.After the Lookup Table Layer, there are a fixed number of stacked Convolutional and Gating Layers. Source: "Language Modeling with Gated Convolutional Networks", 2017. 24
4.3 Neural Network architecture of the model presented with five gated convolutional layers, a max pooling layer and a fully connected one. 26
4.4 Block diagram of the Topic Modeling procedure. A topics latent vector is first extracted from the essay, which is later forwarded to a fully connected layer that outputs a vector that will be used to predict the essay's score. 27
4.5 Architecture of the fused model. The first latent vector is produced by the Neural Networks GCNN model, while the other vector is produced by performing Topic Modeling on the essay. The two latent vectors are then combined to produce the final probabilities distribution over the possible classes. 30

5.1 Pipeline of the training and evaluation of the model. 34

6.1 Training loss graph for each of the four datasets. 41
6.2 Global step/sec for each of the four datasets. 42

7.1 Accuracy for each of the four datasets. 45
7.2 Cohen's Kappa for each of the four datasets. 46
7.3 Evaluation Loss for each of the four datasets. 47

List of Tables

6.1 Word Embeddings Layer parameters along with their description. 35
6.2 Convolutional and Gating Layer parameters along with their description. . . . 35
6.3 Fully Connected Layer in Topic Modeling parameters along with their description. 36
6.4 Fully Connected Layer in GCNN parameters along with their description. . . . 36
6.5 Table with all the additional hyper-parameters and their values 39
6.6 The four different datasets that the network was trained on, along with the number of essays in each one. 40
6.7 Distribution of the number of scores of the essays in the four separate datasets. 40

7.1 The sizes of the four datasets that the network was evaluated on, alongside with the number of essays in each one. 43
7.2 The four different datasets that the network was trained on, along with the number of essays in each one. 44
7.3 Confusion Matrix for Dataset A. 48
7.4 Confusion Matrix for Dataset B. 48
7.5 Confusion Matrix for Dataset C. 48
7.6 Confusion Matrix for Dataset D. 49
7.7 ***Accuracy*** for different values of threshold on the four datasets 49
7.8 ***Cohen's Kappa*** metric for different values of threshold on the four datasets. . 50
7.9 Proportion of the initial size of the dataset that is left after a threshold is applied. 50
7.10 Confusion Matrix for augmented Dataset C. 51

Chapter 1

Introduction

1.1 Motivation

”Automated Essay Scoring (AES) is the use of specialized software to assign grades to essays written in an educational setting” [wik, 2018]. With the recent growing interest in Massive Open Online Courses and Graduate Admission Tests, systems that are able to deliver this kind of services are highly motivated since they could potentially reduce rising costs of education, cover the need for grading standards, and provide students with instant feedback, while, at the same time, maintain robustness and objectivity in the grading scheme.

Grading essays in an automated manner has been a research topic of interest for many years. Currently, in order to maintain a consistent and robust grading scheme across standardized examinations, multiple graders are employed, that proves to be inefficient in terms of cost and time. With the recent advancements in Machine Learning, and, especially, in Deep Learning, there is a strong belief that those hindrances could be overcome. AES proposes a system that, although is not yet able to totally eliminate the human factor, it can certainly reduce the need for multiple markers and assist the remaining ones to a great extent. Besides economic gains, traits of such automated systems could also benefit less developed regions.

There are a lot of AES approaches cited in literature, from manually hard-coded efforts to derive essay scores to Deep Learning techniques. The main difference between Deep Learning and the rest approaches can be detected in the way feature engineering takes place. Feature engineering is the most crucial part of AES systems, and is the exact reason why Deep Learning approaches have proved to perform more efficiently. Manually trying to extract features from essays is considered to be a very complicated task and it is almost impossible to locate all features necessary for grading an essay, whereas, a Deep Learning approach requires no manual feature engineering at all. The trade-off between those two approaches lies in the transparency of each system, since Neural Networks operate as black boxes.

Existing AES systems have proved to perform in a very satisfying way, based on the measurement of the inter-rater agreement between human graders and the system’s predictions. However, the meager employment of those systems in education and industry is, mostly, owed to three basic features that most AES systems lack: *meaningfulness*, *transparency* and *robustness*.

By **meaningfulness** it is implied that, not only a system should achieve high inter-rater agreement, but it should also assess essays based on criteria that make sense. **Transparency** consists a great challenge mainly in Deep Learning systems, where feature engineering takes place through a black-box operation. Being able to visualize the way Neural Networks operate is of paramount importance. Finally, **robustness** refers to the ability of the system to effectively detect and correct errors in an essay. These errors not only include spelling and grammatical mistakes, but also measure how precise the author is in the content and the ideas of the essay, according to the specific question of the essay. Although most of the existing systems are capable of locating and correcting grammar and spelling mistakes, none of them is able to correctly asses the content, ideas and organization of the essays, while they can be easily fooled by essays consisting of rarely used words that do not make any sense.

Inspired by the recent advancements in Deep Learning and Language Modeling, this investigation aims to address those challenges above, by utilizing a novel system architecture in the Automated Essay Scoring problem field, consisting of a fusion of a Deep Learning model - made up of exclusively Convolutional Neural Networks without the use of any Recurrent Neural Networks - and a Latent Dirichlet Allocation algorithm that performs Topic Modeling. CNNs automatically detect and extract features that are the most informative of the defined problem, from low-level features such as spelling and grammar mistakes, to higher level ones such as style and organization of the essay, owed to the gradual convolutional architecture. Additionally, Topic Modeling contributes to the capture of higher level features such as the content, which constitute the most difficult ones to detect. Additionally, the system cannot be fooled by randomly generated essays, since it is trained to score this kind of essays with the lowest possible score. In this way, both meaningfulness and robustness can be ensured. In order to ensure transparency, a visualization technique that was first presented by [Yannakoudakis et al.] has been employed and improved. This technique is able to identify words or sentences that are discriminative of writing quality. Additionally, it can be used in effective error analysis of the deep neural network.

1.2 Purpose of Investigation

The purpose of this work is to conduct a systematic investigation of a novel fused architecture of deep learning and machine learning techniques on assessing high level texts. More specifically, the aims of this study are to:

1. Carry out comprehensive literature review on existing techniques of AES and their limitations
2. Design and develop and AES system using a combination of CNNs and Topic Modeling models
   (a) Perform analysis on the model in determining writing quality and score
   (b) Produce a set of optimal hyper-parameters that maximize the performance of the AES system
3. Employ visualization techniques to evaluate and provide an insight into the error analysis of the system
4. Lay foundations for future research in the area of AES

1.3 Contributions

The main contributions the investigation makes to the field of Automated Essay Scoring as as follows:

1. A proposal of a novel system architecture using a fusion of CNNs and Topic Modeling that is able to effectively capture higher level features and avoid being fooled by nonsense essays, while it is also fully parallelizable in a GPU context. It also provides confidence levels with each score.
2. A systematic investigation of the above system.
3. Training and evaluation on a high-level dataset that has never been used before.
4. Extension of an error analysis visualization technique to improve transparency of the black-box model.

1.4 Ethics Consideration

	Yes	**No**
Section 1: HUMAN EMBRYOS/FOETUSES		
Does your project involve Human Embryonic Stem Cells?		✓
Does your project involve the use of human embryos?		✓
Does your project involve the use of human foetal tissues / cells?		✓
Section 2: HUMANS		
Does your project involve human participants?		✓
Section 3: HUMAN CELLS / TISSUES		
Does your project involve human cells or tissues? (Other than from Human Embryos/Foetuses i.e. Section 1)?		✓
Section 4: PROTECTION OF PERSONAL DATA		
Does your project involve personal data collection and/or processing?	✓	
Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)?		✓
Does it involve processing of genetic information?		✓
Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc.		✓
Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets?		✓
Section 5: ANIMALS		
Does your project involve animals?		✓

Section 6: DEVELOPING COUNTRIES		
Does your project involve developing countries?		✓
If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned?		✓
Could the situation in the country put the individuals taking part in the project at risk?		✓
Section 7: ENVIRONMENTAL PROTECTION AND SAFETY		
Does your project involve the use of elements that may cause harm to the environment, animals or plants?		✓
Does your project deal with endangered fauna and/or flora /protected areas?		✓
Does your project involve the use of elements that may cause harm to humans, including project staff?		✓
Does your project involve other harmful materials or equipment, e.g. high-powered laser systems?		✓
Section 8: DUAL USE		
Does your project have the potential for military applications?		✓
Does your project have an exclusive civilian application focus?		✓
Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items?		✓
Does your project affect current standards in military ethics e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons?		✓
Section 9: MISUSE		
Does your project have the potential for malevolent/criminal/terrorist abuse?		✓
Does your project involve information on/or the use of biological-, chemical-, nuclear/radiological-security sensitive materials and explosives, and means of their delivery?		✓
Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied?		✓
Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cyber-security related project?		✓
Section 10: LEGAL ISSUES		
Will your project use or produce software for which there are copyright licensing implications?		✓
Will your project use or produce goods or information for which there are data protection, or other legal implications?	✓	
Section 11: OTHER ETHICS ISSUES		
Are there any other ethics issues that should be taken into consideration?		✓

1.4.1 Ethics Statement

This research project and the subsequent data raise ethical issues. So to eliminate these issues, certain ethical considerations were observed and dealt with.

The dataset was collected by the GMAC Research team from graduate students who take the GMAT test. The essays that form the dataset come from the Analytical Writing Assessment section of this test and are written by students that none of them are:

- Under the age of 18
- Unable to consent

Those essays are written by students though, and thus the project involves collection and processing of personal data. However, all of the essays are anonymous and none of them contains personal data or information that could help someone infer the identity of the person authoring the essay.

Regarding the legal issues, the dataset was delivered to me by the GMAC Research Team specifically for the needs of this research project and it is not an open-source dataset. Thus, I do not own the rights, title and interest, including ownership of this dataset and this is the reason why this dataset or part of it cannot be included in the submission file. Additionally, there are restrictions on disclosure and presentation of the dataset, as well as on possible publications of the findings that could result from this research project. All those restrictions and further information of the agreement between me, my supervisor and GMAC can be found in the Dataset and Data License Agreement that has been signed by all three parts.

1.5 Report Structure

The dissertation takes on the following structure:

1. **Chapter 2** provides all the necessary background information of the building blocks of the proposed architecture of the system. It also provides a detailed literature review of existing AES systems and their limitations.
2. **Chapter 3** provides detailed information and statistics of the dataset that was used for the training and evaluation of the system, which is not publicly available.
3. **Chapter 4** presents an overview of the design and implementation of the novel system architecture and its building blocks, alongside with justifications of the design decisions that were made.
4. **Chapter 5** includes a presentation of the software developed for the system; Libraries and API's that were used are presented in this chapter, as well as software engineering techniques.
5. **Chapter 6** provides an insight into the training procedure of the system, that is which optimization algorithms were chosen and how hyper-parameters were tuned.
6. **Chapter 7** provides a quick overview of the experimental setup, results on various essay questions with different number of essays and a detailed analysis of the model in general.

7. **Chapter 8** describes the visualization technique developed to get an insight into the system and provide a valuable error analysis.

8. **Chapter 9** draws general conclusions regarding the investigation and measures the contributions made against the key aims set out at the beginning of the study. In this chapter, proposals for future research are also provided.

Chapter 2

Background and Related Work

This chapter will present the relevant background material required to thoroughly understand the design and implementation phase of this investigation. Additionally, it will aim to provide some intuition regarding the design decisions made throughout this study.

2.1 Deep Learning Preliminaries

Automated Essay Scoring, due to the extensive feature engineering that requires, was revolutionized by Deep Learning. Therefore, it would be useful to summarize fundamental Deep Learning concepts, which constitute the building blocks of various existing AES systems.

2.1.1 Neural Word Embeddings

One of the major breakthroughs in the field of Statistical Natural Language Processing was the introduction of *word embeddings* [Bengio and Senecal, 2008] - a representation of words that allows words with similar meaning to have similar representation, capturing, in this way, the semantics of each word into a low-dimensional fixed sized vector, contrary to the one-hot vector representations where each word vector has the size of the vocabulary. Word embeddings constitute part of the learning process of a system and their efficiency depends on both the quality and the volume of the dataset. Neural distributional representation is a popular technique of training word embeddings and has been extensively cited throughout literature [Mnih and Hinton, 2007] [sem]. The distributed form of word embeddings is perhaps one of the key elements that led projects of Natural Language Processing based on Deep Learning techniques to perform in an impressive way and this is the main reason why the AES system needs to adopt such a method of representing words.

One of the most popular and widely used models that trains word embeddings is *Word2Vec*, developed by [Mikolov et al., 2013b] at Google as a response to make the neural-network-based training of the embedding more efficient and since then it has become the standard choice for developing pre-trained word embeddings. In this model, words are viewed as discrete states for which the transitional probabilities have to be calculated. The work involves analysis of the learned vectors and the exploration of the vector math on the representation of words. For example, as seen in Figure 2.1, vectors that have a relation of country-capital appear to be closer in the new dimensional space.

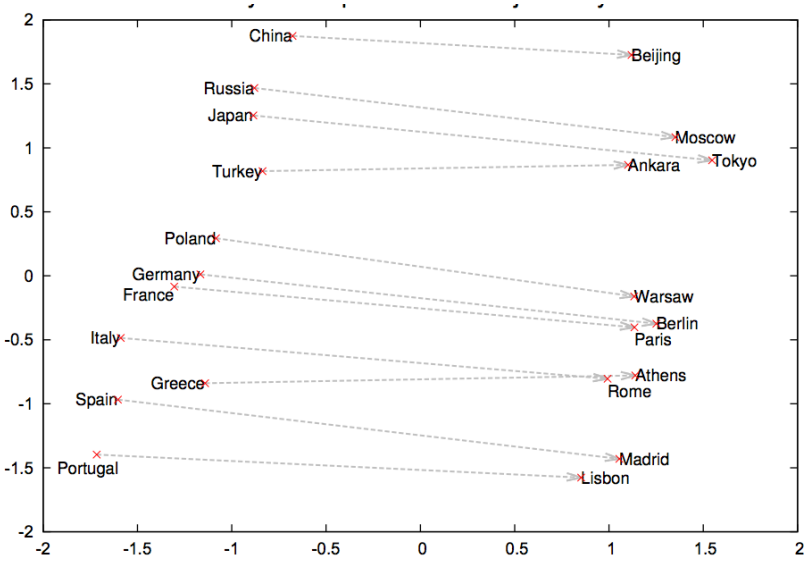


Figure 2.1: Image taken from https://algorithmia.com/algorithms/nlp/Word2Vec. Example of relation between words after the training of the model.

Word2Vec is comprised of two models:

- Continuous Bag-of-Words, or CBOW model [Mikolov et al., 2013a]
- Continuous Skip-Gram Model [Mikolov et al., 2013b]

The CBOW model learns the embedding by predicting the current word based on its context, while the skip-gram model learns by predicting the surrounding words given a current word as seen in Figure 2.2

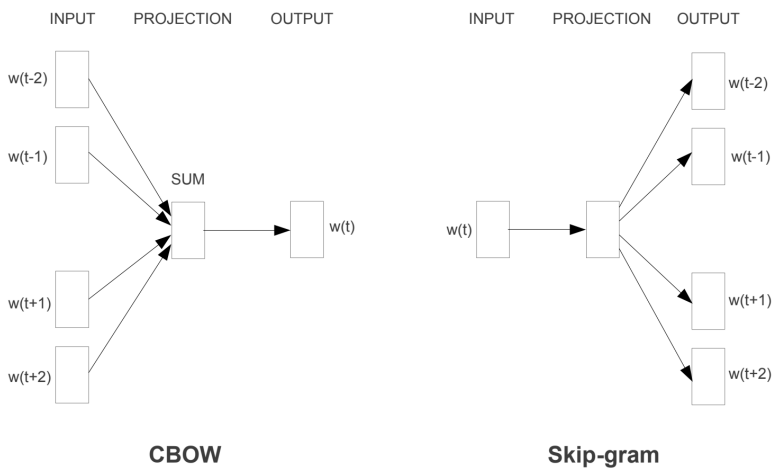


Figure 2.2: Image taken from Efficient Estimation of Word Representations in Vector Space, 2013. Presentation of the two models that word2vec is comprised of.

2.1.2 Multi-Layer Perceptrons (MLPs)

In order to get a better understanding of CNNs, a revisit on the basics of the feed-forward networks, and specifically on Multi-Layer Perceptrons (MLPs) would be very useful.

Multilayer perceptrons, also known as feed-forward neural networks are the quintessential deep learning models. They usually consist of three layers:

- an input layer x_i
- a hidden layer h_i
- and an output layer o_i

In the base case as seen in Figure 2.3, an MLP with only one hidden layer is a function: $f : R^I \rightarrow R^O$ where I, O are the sizes of the input and the output vector respectively. The hidden layer can then be defined as follows:

$$h_i = \sigma(x_i * W_x + b_x) \tag{2.1}$$

, where W and b are the weight matrices and bias respectively that have to be optimized throughout the training process and their size should be determined by the size of each of the three above-mentioned layers (input, hidden, output), while σ is a non-linear activation function such as *sigmoid* or *tanh*. The hidden layer then feeds into the output layer o_i. The output layer can then be defined as follows:

$$o_i = G(h_i * W_o + b_o) \tag{2.2}$$

, where G is also a non-linear function that can take the form of a softmax function and will be described later in this chapter. h_i is the output of the hidden layer which is multiplied by W_o which is another weight parameter matrix.

Multilayer Perceptrons, like most of feed-forward neural networks, are trained by employing the Stochastic Gradient Descent (SGD) algorithm. For each of the parameters of the model $\theta = (W_x, W_o, b_x, b_o)$, SGD calculates its gradients and propagates the errors back to the model using the **backpropagation** algorithm [Rumelhart et al., 1986].

Softmax Function

In classification problems, such as the one that this research project presents, a softmax activation function is usually applied on the output of the MLPs. What it, essentially, does is to transform the output vector $\mathbf{y} \in \mathbb{R}^m$ of a neural network based classifier into a new vector $\mathbf{s} \in \mathbb{R}^m$ that represents a probability distribution over m possible output classes. Softmax function can be defined as follows:

$$s_j = s_j(\mathbf{y}) = \frac{exp(y_j)}{\sum_{r=1}^{m} exp(y_r)}, \quad j = 1, \ldots, m \tag{2.3}$$

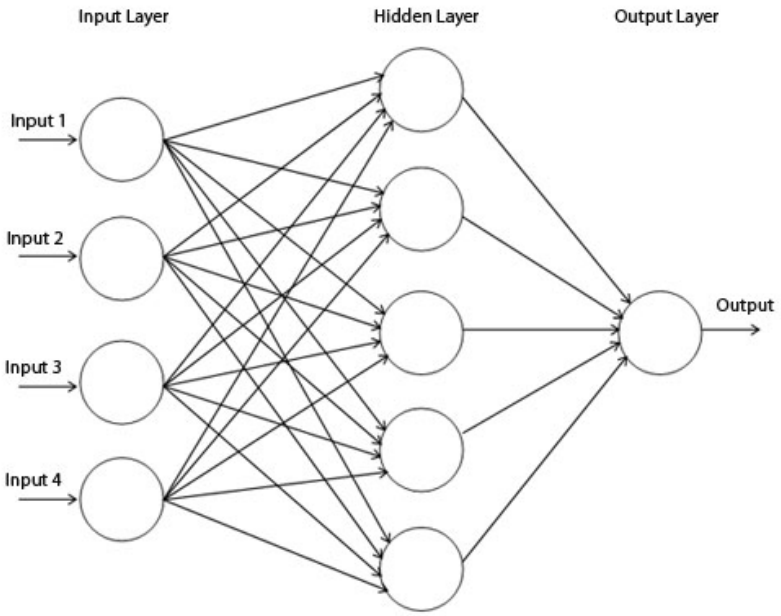


Figure 2.3: Image taken from https://github.com/cazala/synaptic. Vanilla MLP, with one hidden layer. The nodes of each layer correspond to values while the connections represent the weights of the network.

2.1.3 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are a special class of artificial neural networks which are mostly applied on problems where data is distributed in a grid topology, such as images. Similar to MLPs, CNNs are made up of neurons that have learnable weights and biases. CNNs have been applied with great success not only on computer vision problems, but on Language Sequence Modeling problems as it has been cited in the respective literature [Dauphin et al., 2016]

A simple CNN is a sequence of layers, and every layer transforms one volume of activations to another through a differentiable function. A CNN usually consists of the following layers:

- An input layer which holds the raw values of the input, which has one or more channels. For example, an image has three input channels, RGB.
- A sequence of hidden layers that compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.
- A Rectified Linear Unit (ReLU) that applies an element-wise activation function, such as the $max(0, x)$ thresholding at zero. This operation leaves the size of the input unchanged.
- A pooling layer that performs a down-sampling operation along the spatial dimensions.

- A fully-connected layer that computes the class scores followed by a softmax function.

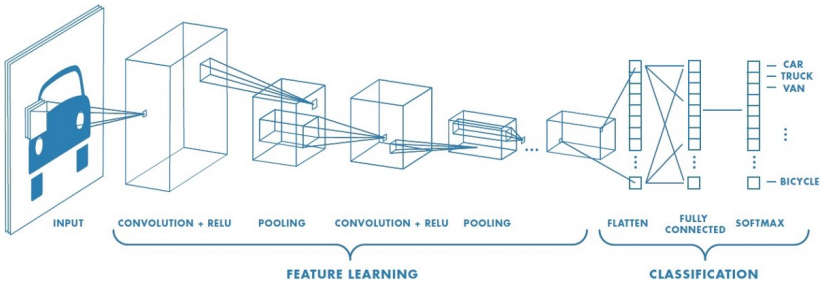


Figure 2.4: Image taken from hhttps://medium.com/@rohanthomas.me/convolutional-networks-for-everyone-1d0699de1a9d. Example of a CNN architecture.

An example of a CNN architecture can be seen in Figure 2.4.

Convolutional Layer

The convolutional layer is the core building block of a Convolutional Neural Network. This layer derives its name from the 'convolution' operation and its essential role is to extract features from the input vector. The trainable parameters of this layer consist of a set of learnable filters or kernels. Each filter has a set of weights and a bias. If we suppose that a convolutional layer has D_2 filters and D_1 input channels, then the weights should have the form: $\boldsymbol{W} \in \mathbb{R}^{D_1 \times F \times F \times D_2}$, where F is the filter size. The way Convolutional layers work is by sliding those filters across the outputs of the previous layers - be it the the input or the output of a previous hidden layer. Thus, if the input of a convolutional layer has dimensions $W_1 \times H_1 \times D_1$, then the current hidden layer has dimensions $W_2 \times H_2 \times D_2$ where:

$$W_2 = W_1 - F + 2P_1 + 1, \quad H_2 = H_1 - F + 2P_1 + 1$$

In the case where the sliding of the filters takes place with a step bigger than one, then a new parameter S is introduced, that is defined as *stride*. Then, the dimensions of the current hidden layer are respectively $W_2 \times H_2 \times D_2$, where:

$$W_2 = \frac{W_1 - F + 2P_1}{S} + 1, \quad H_2 = \frac{H_1 - F + 2P_1}{S} + 1$$

Pooling Layer

It is a common technique when building CNNs to periodically apply pooling layers on the outputs of the convolutional layers. The essential function of this layer lies on the reduction of the spatial dimensions of the representation, that results in fewer parameters,lighter training of the model and control of the over-fitting, while it also contributes to the maintenance of the translation invariance of the model. One of the most commonly used pooling operations is the *Max Pooling* in which the pooling layer would independently operate on every depth slice of the input vector and take the maximum element in each sliding window. More generally, the pooling layers operates as follows:

- Takes an input vector of size $W_1 \times H_1 \times D_1$
- Uses two parameters:
  - the spatial extent ***F***
  - the stride ***S***
- Outputs a vector with dimensions $W_2 \times H_2 \times D_2$, where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$

An example of a Max Pooling operation can be seen in Figure 2.5

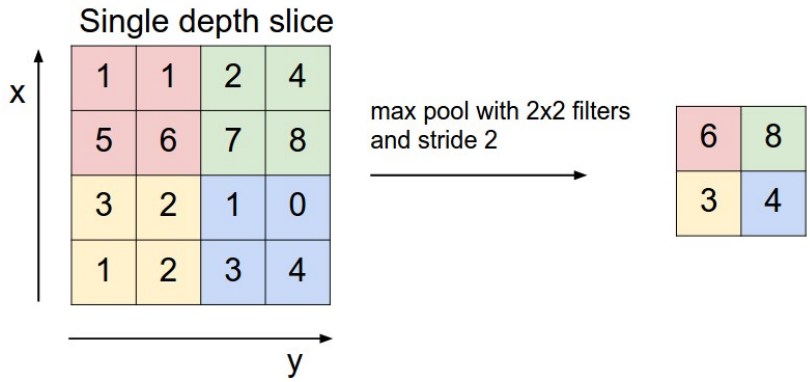


Figure 2.5: Image taken from http://cs231n.github.io/convolutional-networks/. Example of a Max Pooling operation

Fully-Connected Layer

Fully connected layers are usually placed after the convolutional and pooling layers in a CNN, and are, essentially, responsible for generating predictions. They have full connections to all activations in the previous layer, as seen in MLPs, since they are essentially feed-forward networks. Thus, their activations are usually computed by matrix multiplications among their neurons.

2.1.4 Gated Linear Units (GLUs)

Gating mechanisms were first proposed as an effective technique to deal with the vanishing gradients problem that mostly appears in Recurrent Neural Networks (RNNs), by letting information flow unaffected through potentially many timesteps. However, they have been proven to be effective not only for Recurrent Neural Networks, but for the convolutional modeling of images as well [van den Oord et al., 2016].

[Dauphin et al., 2016] first introduced Gated Linear Units (GLUs) as a simplified gating mechanism that is capable of reducing the vanishing gradient problem by having linear units coupled to the gates, while also retaining the non-linear capabilities of the layer and has proven to be very effective when combined with CNNs.

The gradient of the GLU

$$\nabla[\boldsymbol{X} \otimes \sigma(\boldsymbol{X})] = \nabla\boldsymbol{X} \otimes \sigma(\boldsymbol{X}) + \boldsymbol{X} \otimes \sigma'(\boldsymbol{X})\nabla\boldsymbol{X} \tag{2.4}$$

has a path $\nabla\boldsymbol{X} \otimes \sigma(\boldsymbol{X})$ without downscaling for the activated gating units in $\sigma(\boldsymbol{X})$.

2.2 Latent Dirichlet Allocation (LDA)

Topic Modeling is one of the most commonly used techniques for Natural Language Processing applications. In Topic Modeling, a topic is a set of words or n-grams that appear together in statistically significant methods. The topics inferred by a Topic Modeling algorithm can provide a useful view of the document as a whole at a higher level.

One of the most popular and effective methods for performing Topic Modeling is *Latent Dirichlet Allocation (LDA)*. LDA is an unsupervised generative probabilistic algorithm for modeling of a set of documents and was first introduced by [Blei et al., 2003]. The main idea of LDA lies in the fact that each document could be represented with a random mixture over latent topics, where each topic is considered to be a set of words and that all document topics distributions share a common *Dirichlet prior*. Usually, words with the highest probabilities can give a good idea of what the topic is about.

If a corpus *D* is assumed that consists of *M* documents and each document *d* having N d words ($d \in \{1, \ldots, M\}$), LDA applies the following steps in order to perform Topic Modeling:

1. Choose a multinomial distribution ϕ_t for topic t: $t \in \{1, \ldots, T\}$ from a Dirichlet distribution with parameter β.

2. Choose a multinomial distribution θ_d for document d: $d \in \{1, \ldots, M\}$ from a Dirichlet distribution with parameter α.

3. For a word $w_n, \quad n \in \{1, \ldots, N_d\}$ in document d,

   (a) Select a topic z_n from θ_d

   (b) Select a word w_n from ϕ_{zn}

ϕ and θ are considered to be latent variables, while α and β are considered to be hyper-parameters of the model. The maximization of the following function on the dataset D provides the optimal variables and hyper-parameters of the model:

$$p(D|\alpha, \beta) = \prod_{d=1}^{M} \int p(\theta_d|\alpha)(\sum_{n=1}^{N_d} p(z_{dn}|\theta_d)p(w_{d_n}|z_{d_n}, \phi)P(\phi|\beta))d\theta_d d_p \tag{2.5}$$

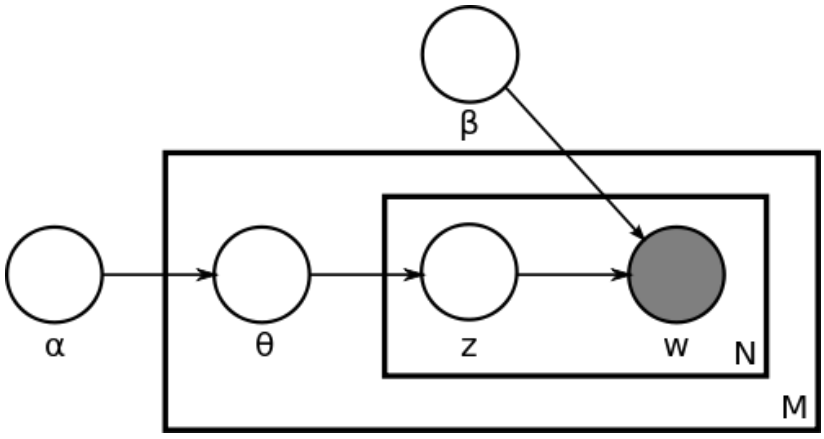


Figure 2.6: Source: https://en.wikipedia.org/wiki/Latent_Dirichlet_Allocation . Plate notation representing the LDA model.

2.3 Literature Review

2.3.1 Automated Scoring in Massive Open Online Courses (MOOC)

In the era of Massive Open Online Courses (MOOC) where the number of users exceeds 10 million, the problem of coursework grading constitutes a significant hindrance to the progress of the courses. The huge amount of courses makes it almost infeasible for humans to assess each coursework separately and the way grading takes place nowadays in MOOC is either manually, or in a peer-to-peer way, where users of a course assess other users' coursework. Developing automated effective and impartial grading systems for MOOC is a challenging problem in educational measurement and assessment, due to the diversity of answers and the subjectivity of the graders. A lot of efforts have been attempted until today to face these problems, including advances in short answers, mathematical equations and computer programs' grading.

[Jing] designed an automatic grading approach for short answers, based on the non-negative semi-supervised document clustering method. More specifically, they treated each short answer as a short document, and then applied a clustering algorithm on them, using similarity measures, and in this way determining the final score of each short answer.

A different approach to automated grading in MOOC was developed by [Lan et al., 2015], who focused on how Automated Grading could be used to assess and provide feedback to Open Response Mathematical Questions that figure prominently in STEM courses. What they essentially did was to employ Mathematical Language Processing techniques, inspired by the Natural Language Processing advances, and their method comprised three main steps. First they converted each solution to an open source mathematical question into a series of numerical features. Then they clustered the features from several solutions to uncover the structure of correct, partially correct, and incorrect solutions. Finally, they automatically graded the remaining solutions based on their assigned cluster and one instructor-provided grade

per cluster.

In assessing computer programs in MOOC things are undoubtedly simpler, since this can be easily done with the use of compilers and test cases. A step further into assessing computer programs was attempted by [Srikant and Aggarwal, 2013] who presented a machine learning framework to automatically grade computer programs by proposing a set of highly-informative features, derived from the abstract representations of a given program, that capture the program's functionality. These features were then used to learn a model to grade the programs, which are built against evaluations done by experts on the basis of a rubric.

However, the problem of Automated Essay Scoring, has not been, yet, faced in an efficient way that it could successfully be incorporated in MOOC, and this is what we are going to try to solve in this dissertation.

2.3.2 Automated Essay Scoring Problem

Automated Essay Scoring (AES) is a problem directly related to the discovery of features responsible for producing the final score. Throughout the years the process of detecting these features has changed dramatically. Starting from implementations where the feature engineering was taking place in a totally manual way, to reaching Deep Learning techniques where the human interference in the discovery of those useful features is considered to be zero. In between, a lot of other techniques were utilized to face the above-mentioned problem, with classical Machine Learning techniques, Statistical approaches and Natural Language Understanding implementations to considered as the most effective ones. Almost all of the systems that we are going to describe achieved very satisfying results, almost identical to those of the human graders - experts. However, the core difference between those systems is the effort needed to implement them - the way the feature engineering is taking place - as well as the robustness of those systems.

2.3.2.1 Pre-Deep Learning Approaches

Project Essay Grade (PEG) [Page, 1967] [Page, 1968] is one of the earliest systems developed for automated essay scoring. It's functionality primarily relies on *proxes*, i.e intrinsic features within a block of text that a human grader would look for, but constitutes a difficult problem for a computer. These *proxes* mainly consist of the surface linguistic features of an essay, including essay length, number of prepositions, number of relative pronouns, and variation in word length. Page utilized a multiple regression approach to this problem, by first extracting *proxes* from a set of training essays, and then training the system along with the given human grades to finally calculate the regression coefficients. These coefficients constitute the best approximation to human grades.

Latest experiments on PEG revealed results that reached a multiple regression corre-

lation as high as 0.87 with human graders. However, despite its undisputed success on predicting teachers' essay ratings, PEG did not manage to earn the trust of the education community, and it was mainly because of the fact that Page relied entirely on the surface features of an essay, without assigning any importance to the lexical context. He assumed that the *proxes* located by the experts would be enough to properly assess a block of text and this is the reason why he did not utilize any Natural Language Processing method.

Later on, [Landauer and Dumais, 1997] built the *Intelligent Essay Assessor (IEA)* and went a step further and deeper into solving the problem of automated essay grading, by employing a Latent Semantic Analysis (LSA) technique. LSA is a machine learning technology for simulating the meaning of words and passages and its main goal lies on going beneath the essay's surface vocabulary to quantify its deeper semantic content. It uses the Singular Value Decomposition technique in order to analyze a corpus of ordinary text of the same size and content as that from which students learn the vocabulary concepts, and knowledge needed to write an expository essay. Under the employment of LSA, every word and passage is represented as a point in a high dimensional semantic space. That means that relative distance between two words or passages implies similarity between those words or passages, and as a result, it deals effectively with the fact that there is an unlimited number of ways to express nearly the same meaning in different words. Moreover, LSA has proven to be significantly more accurate than traditional key word approaches that rely on the occurrence of the same words or word stems in two passages.

With LSA being the core element of IEA, the system is then trained on a large corpus of domain-representative text. LSA's basic role is to characterize essays by representing their meaning and compares them with highly similar texts of known quality. The three major elements that IEA computes and combines are content, style and mechanics used in a multiple regression setting on human scores.

A test conducted on GMAT essays using the IEA system, resulted in percentages for adjacent agreement with human graders between $85\% - 91\%$. However, despite the great performance of IEA, there are some disadvantages in using this system. First of all, LSA makes no use of word order since the authors claim that this is not the most important factor for grabbing the sense of a passage. Moreover, it requires large amounts of data to construct a suitable matrix representation of word use/occurrence, and due to the size of the matrices involved computations are very cumbersome.

It was then that Natural Language Processing techniques were first employed to confront the problem of automated essay grading. Natural Language Processing (NLP) is the application of computational methods to analyze characteristics of electronic files and they utilize tools such as syntactic parsers, discourse parsers, and lexical similarity measures. [Lonsdale and Strong-Krause, 2003] used the Link Grammar Parser [Lafferty et al., 1992] to analyze and score texts based on the average sentence-level scores calculated from the parser's cost vector.

Electronic Essay Rater (E-Rater) developed by [Burstein et al.] is another example of an NLP approach alongside with some statistical techniques, used for extracting linguistic features from the essays to be graded. Essays are evaluated against a benchmark set of human graded essays. The grading scheme of E-Rater relies on how relevant an essay is with the topic of the question, how strong, coherent and well-organized an argument structure is, and if it contains a variety of word use and and syntactic structure. The features that E-Rater is trying to detect and assess include the analysis of the discourse structure, of the syntactic structure and of the vocabulary usage. The application is designed to identify features in the text that reflect writing qualities specified in human reader scoring criteria and is composed by five main modules. Three of the modules detect features that may be used as scoring guide criteria for the syntactic variety, the organization of ideas and the vocabulary usage of an essay. A fourth independent module is used to select and weigh predictive features for essay scoring. Finally, the last module is used to compute the final score.

Although E-Rater is an example of a more sophisticated system that is able to extract features of an essay using Natural Language Processing Techniques, is also a far more complex system and requires a lot of training. Additionally, no on-line demonstration of E-Rater has been made available.

A more probabilistic solution to the problem was attempted by [Rudner and Liang, 2002], by developing the *Bayesian Essay Test Scoring sYstem (BETSY)*. BETSY is a program that classifies text based on trained material.

The goal of this system is to classify each essay into one of the four categories - extensive, essential, partial, unsatisfactory. The fundamental probabilistic models behind BETSY are the Multivariate Bernoulli Model (MBM) and the Bernoulli Model (BM). In the MBM model, each essay is considered to be a combination of features, and the probability of each score for a given essay is computed as the product of the probabilities of the features contained in the essay. Respectively, with the BM the conditional probability of each feature being captured in an essay is estimated by the proportion of essays within each category that contain the feature. Both models are assumed to be naive Bayes models, since they assume conditional independence. According to Rudner and Liang, BETSY contains all the best features of PEG, LSA, and E-Rate, plus several advantages of its own.

[Mitchell et al., 2002] aimed at developing a more robust system for evaluating free-text answers to open-ended questions and built *Automark*. Automark utilizes NLP techniques to mark open-ended responses. The system performs by employing a number of templates, each of which represents a form of a valid or a specifically invalid answer. It then applies the templates on the text, and makes inferences regarding the presence of spelling, typing, syntax and semantics errors.

[Chen et al., 2010] utilized a weakly supervised bag-of-words framework alongside with a voting algorithm to infer text scoring. [Yannakoudakis et al.] extracted deep linguistic features and employed a discriminative learning-to-rank model that

outperforms regression.

Reaching to more recent approaches, [Mcnamara et al., 2015] used a hierarchical classification approach to scoring, employing linguistic, semantic and rhetorical features, among others. [Farra et al., 2015] utilized variants of logistic and linear regression and developed models that score persuasive essays based on features extracted from opinion expressions and topical elements. Finally, [Crossley et al., 2015] identified student attributes, such as standardized test scores, as predictive of writing success and used them in conjunction with textual features to develop essay scoring models.

2.3.2.2 Deep Learning Approaches

All approaches discussed so far, despite their indisputable success in scoring text automatically, require a lot of manual feature engineering, which constitutes the most painful part of Automatic Essay Scoring systems. Moreover, the linguistic features that are manually extracted and are being used to assess an essay are tuned for specific domains. That means that there is not any general rule for extracting features and in order to achieve a good performance on different data, those features have to be adjusted to match this specific data.

In 2012, Kaggle [1], sponsored by the Hewlett Foundation, hosted the Automated Assessment Prize (ASAP) contest, to challenge engineers to solve the problem of Automated Text Scoring. Thus, they released a dataset consisting of around twenty thousand essays - 60% of which was human graded -, produced by middle-school English speaking students. Since this dataset became publicly available, researchers began focusing on the solution of this problem, employing Deep Learning techniques. We are going to focus our interest on two implementations.

Inspired by, at the time, recent advances in machine translation, [Alikaniotis et al., 2016] utilized Neural Networks to face the problem of Automated Essay Scoring. They achieved their best results by using a combination of a bi-directional LSTM, alongside with *Score Specific Word Embeddings (SSWEs)*. Taking the main idea from [Collobert and Weston, 2008], Alikaniotis et al. employed a neural network architecture for representing each word in a corpus based on its local context. A word w_t found in a n-sized sequence of words $S = (w_1, \ldots, w_t, \ldots, w_n)$, $\forall w_i \in S \mid w_i \neq w_t$ should be represented based on the other words that exist in the same sequence. That means that the model should be able to see the difference between S and some 'noisy' counterpart S' in which they had substituted the target word w_t with a randomly chosen word from the vocabulary, so as that: $S' = (w_1, \ldots, w_c, \ldots, w_n | w_c \sim V)$. Then, after mapping each word found in the vocabulary V to a real valued vector, they formed sets S and S' by concatenating the word embedding vectors of each sequence, which, later, they fed as inputs to a hard $tanh$ layer, feeding finally a single linear unit in the output layer. The way that the models learns those word embeddings is by ranking the activation of the true sequence S higher than the activation

[1] http://www.kaggle.com/c/asap-aes/

of its 'noisy' counterpart S'. The essential goal of this model is to minimize a hinge loss that ensures that the activations of the original and noisy sequences will differ by at least 1.

The contribution of Alikaniotis et al. to this word embedding model is the extension of this model to also identifying the contribution of each word to the final score, and not only capturing the its local linguistic environment, by introducing the Score Specific Word Embeddings (SSWEs). The architecture of the neural network is modified to contain a further linear unit in the output layer that performs linear regression by predicting the final score. The loss function of the added linear unit is the $mean\ squared\ error$ function, while the total loss of the word embedding model is computed as a weighted linear combination of the two respective individual losses.

After having obtained the SSWEs, each essay can be now represented as a continuous real valued vector. Each essay is then forwarded to the main core of the system, which is a bi-directional LSTM. The word vectors enter the input layer one at a time, zero-padding shorter sequences. By taking the activation of the LSTM layer of the last time-step at which a word is fed to the network, we get a prediction of the score of the essay. For the needs of the bi-directional LSTM, two independent passes of each essay have to be completed, and then are concatenated to get a prediction of the final score. The loss function being used in this model is the $mean\ squared\ error$, back-propagating the errors to the word embeddings. The model was trained on the Kaggle dataset and the Pearson's r correlation that they achieved was 92%.

A similar neural approach to solving the problem of AES was also adopted by [Taghipour and Ng, 2016]. The architecture of their system consisted of five core layers: the Lookup Table Layer, the Convolution Layer, the Recurrent Layer, the Mean Over Time Layer and the Linear Layer with Sigmoid Activation.

In the Lookup Table Layer each word is transformed into a word embedding vector with the use of an embedding matrix. Once each word has been projected to the real-valued vector space, they are being fed as input to a convolution layer responsible for extracting local features from the sequences. This layer could potentially capture local contextual dependencies in the essay and thus improve the performance of the system. The embeddings generated from the convolution layer are then forwarded into a recurrent layer, that is made up of a Long-Short Term Memory unit. The use of the recurrent layer is, essentially, to generate a representation for a given essay. The outputs of the recurrent layer are fed into a mean-over-time layer, responsible for aggregating the variable number of inputs into a fixed length vector. By using this layer, all of the recurrent states are taken into consideration and not just the last state for generating the final score, as it is proven to be more effective. Finally, the Linear Layer with Sigmoid Activation maps the outputs of the mean-over-time layer to a scalar value, representing the final score. The system is trained using the mean squared error function as a loss function, with the use of the RMSProp optimization algorithm. The dataset used to train the model was the Kaggle dataset.

Chapter 3

Dataset

3.1 GMAT Analytical Writing Assessment (AWA) Dataset Overview

The novel system presented in this research project was trained and evaluated on the *Graduate Management Admission Test (GMAT) Analytical Writing Assessment (AWA) Dataset*. GMAT is a computer-based adaptive test that aims to assess certain analytical, writing, quantitative, verbal and reading skills in written English for use in admission to a graduate management program.

The first section on the GMAT is the Analytical Writing Assessment (AWA), or essay section, where each student is asked to write two types of essays - an Analysis of an Argument and an Analysis of an Issue. In the Analysis of an Argument section, a short piece of text is presented which makes an assertion or states a point of view, and then gives evidence to support it. The student is then asked to critique the structure of the argument and explain how persuasive or not it is. In the Analysis of an Issue section, an issue is presented on which students are asked to elaborate on and explain their views , together with examples from their own experience, observations or readings.

Essays are graded from 0-6, rounded off if necessary to the nearest half-point. The essays are graded by two graders - one human and one computerized grader, the E-Rater. If there is a disagreement between those two raters, a third - human - grader is asked to make the final decision.

The dataset that was used to train and evaluate the current system consists of around *730,000* of both Analysis of an Issue and Analysis of an Argument essays graded from 1-6 collected by *Graduate Management Admission Council (GMAC)*. The number of questions answered by those essays were *113*, with an average of *6,500* essays per question and an average of *750* words per question. The respective scores and ids of the essays were provided alongside with the essays.

3.2 GMAT AWA Scoring Guide

The main elements that GMAT graders consider in the Analytical Writing Assessment in order to evaluate an essay, according to GMAC [1] are the following:

- Overall quality of ideas about the issue and argument presented
- Overall ability to organize, develop, and express those ideas
- The relevant supporting reasons and examples used
- Ability to control the elements of standard written English

It should be noted that in considering the elements of standard written English, readers are trained to be sensitive and fair in evaluating the responses of examinees whose first language is not English.

3.3 Dataset Augmentation

The GMAT AWA dataset was augmented with randomly created essays that do not make sense and were scored with the lowest possible grade 1, in order for the system to be able to efficiently deal with random essays, that have proved to 'fool' existing AES systems. With this data augmentation, the presented system became capable of dealing with essays that do not make any sense.

[1] https://www.gmac.com/gmat-other-assessments/accessing-gmat-exam-scores-and-reports/how-to-use-the-analytical-writing-assessment-score

Chapter 4

Approach

Automated Essay Scoring, as it can be seen from the relevant literature, requires a lot of feature engineering. Inspired by this and the recent breakthroughs in Language Sequence Modeling using Convolutional Networks, in this work a novel approach to AES is presented with a fusion of a supervised and an unsupervised Machine Learning algorithm. This chapter aims t to provide an overview of the approach used to develop this AES system using Deep Learning and Topic Modeling techniques. AES problem is treated as a classification problem in this work, where scores from 1-6 are considered the classes to be predicted, while a confidence level is provided with each prediction in order to assist the human grader. The chapter begins with providing information on the preprocessing of the dataset and continues with providing details on the system's architecture alongside with design decision justifications.

4.1 Data Preprocessing

This section provides information about the methodology followed for preprocessing the data from the GMAT AWA dataset. For the data to be able to be fed as input to the system presented, a couple of transformations had to take place.

4.1.1 Tokenization

The delivery of the dataset in .txt files required a lot of preprocessing. After separating the files into essay ids, essay answers and essay scores, the process of tokenization had to take place. *Tokenization* is the process of breaking up streams of text into individual words, called *tokens*. Additionally, each token was lowercased in order for the system to be case-insensitive. The next step was the creation of a *vocabulary*, that consisted of the *6,000* most frequent words observed in the dataset each of which was assigned an integer - starting from 1 - based on the number of times that appeared in the dataset. For example, the word '*the*' was the most frequent word of the dataset and thus it was assigned the integer *1*. Additionally, a special character '*' was inserted in the vocabulary for all the unknown words. In this way, each essay was represented as a list of integers that rendered it feasible for the system to understand and extract information from the essays.

4.2 Network Architecture

4.2.1 Neural Word Embeddings

As it has already been explained in this dissertation, in order to make each word of the essay ”machine-readable”, each word is mapped on to a neural word embedding of fixed size. In this way, the meaning of each word is captured in this fixed sized vector that is independent of the vocabulary size. It was decided not to initialize the word embeddings with some pre-trained representations such as *Word2Vec*, since there was observed no significant difference in the performance of the system when those embeddings were initialized. A word embedding matrix is, then created, of size $V \times W$, where V is the vocabulary size and W is the size of the embeddings vectors, that contains references (indexes) to every word in the corpus and its vector representation.

4.2.2 Gated Convolutional Neural Network (GCNN) Model

Inspired by the recent breakthroughs in Language Sequence Modeling using CNNs, [Dauphin et al., 2016], this section provides details on how Gated Convolutional Neural Networks were used as part of the system in this work. Contrary to the established belief of using Recurrent Neural Networks to confront the AES problem, this chapter presents a fully parallelizable Deep Learning model, that its gradual architecture is ideal for extracting features, starting from low level to the more abstract ones, while maintaining the sequential properties of RNNs with the use of the Gating Linear Units.

4.2.2.1 Lookup Table Layer

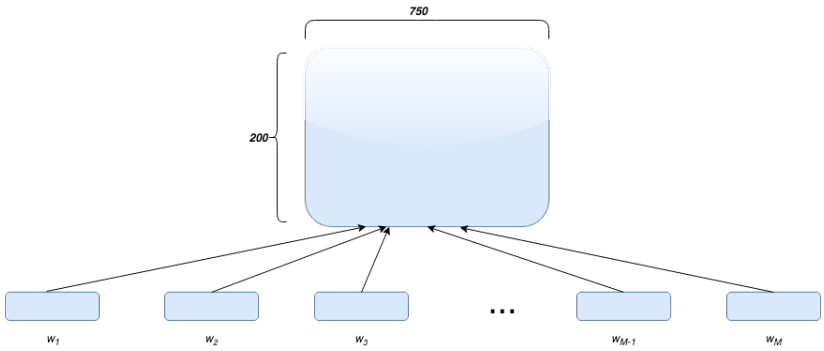


Figure 4.1: Concatenation of the neural word embedding vectors into a Lookup Table.

The first layer of the novel architecture presented in this work is a Lookup Table, which, essentially, is a concatenation of the 1-dimensional neural word embeddings of all the words in an essay into a 2-dimensional matrix E of size equal to $W \times L$ - where W is the size of the embeddings vectors and L is the length of the essay as it can be seen in Figure 4.1. At this step, the model takes the tokenized essay as input and for each token finds the corresponding word embedding vector from the Lookup Table described in 4.2.1. The concatenation of all those vectors make up the Lookup

Table.

4.2.2.2 Convolutional and Gated Linear Unit Layer

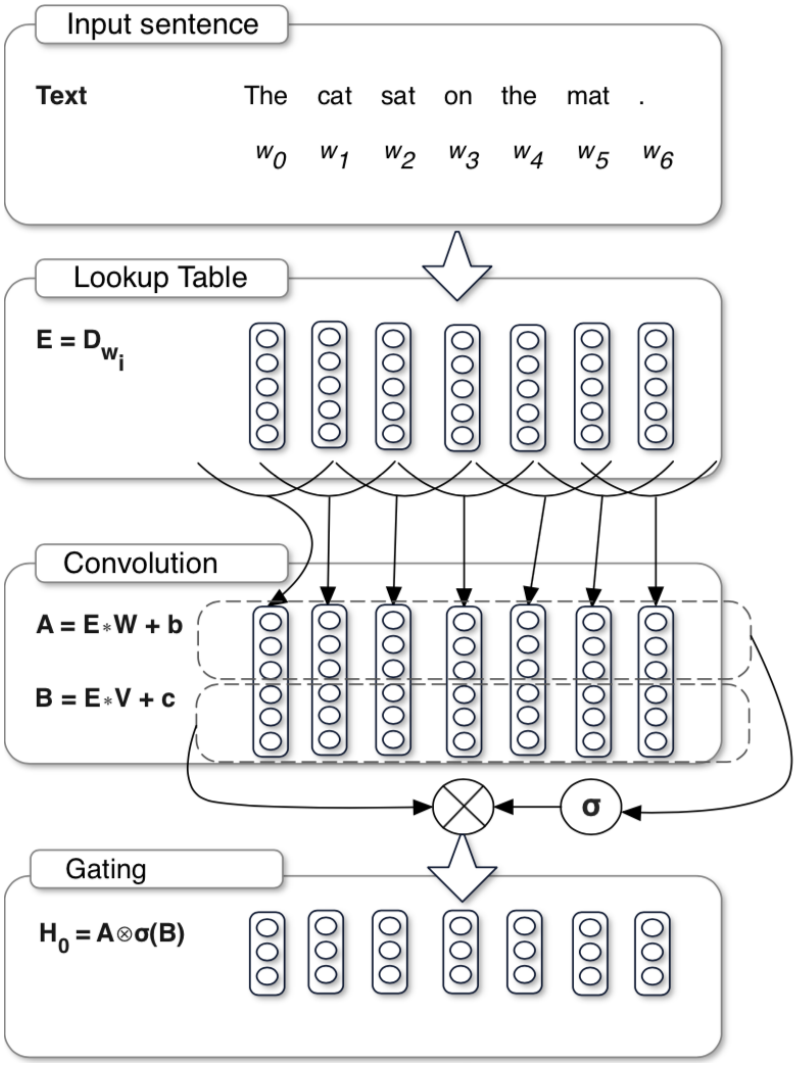


Figure 4.2: GCNN model architecture.After the Lookup Table Layer, there are a fixed number of stacked Convolutional and Gating Layers. Source: "Language Modeling with Gated Convolutional Networks", 2017.

The core building block of the Neural Network Architecture of the system presented is the Convolutional and Gated Linear Unit Layer. Input to this layer is the Lookup Table presented in section 4.2.2.1. This layer consists of a user defined number of stacked convolutions alongside with their gated linear units, named *gated temporal convolutions*. In this way, as it will be shown, all recurrent connections that are necessary for sequence modeling are replaced having, also, the advantage of parallelizability.

Inputs w to this layer are convolved with a function f to produce the output $\boldsymbol{H} = f * w$ for the next layer in a window fashion, the size of which is defined by the user. In this way, temporal dependencies are eliminated and thus convolutions can be effectively parallelized across the input.

In Figure 4.2 the GCNN model architecture for just one layer of gated temporal convolutions is illustrated. Words of an essay $w_0, w_1, \ldots, w_N$, represented by their embedding vectors matrix $\boldsymbol{E} = \boldsymbol{D}_{w_0}, \boldsymbol{D}_{w_1} \ldots, \boldsymbol{D}_{w_N}$ through the Lookup Table $\boldsymbol{D}^{|V| \times e}$, where $|V|$ is the vocabulary size and e is the embedding size, are given as input to the first stacked layer of the GCNN. Then each gated temporal convolution layer computes the input for the next layer. The hidden layers $h_0, \ldots, h_L$ of GCNN are computed as follows:

$$h_l(\boldsymbol{X}) = (\boldsymbol{X} * \boldsymbol{W} + \boldsymbol{b}) \otimes \sigma(\boldsymbol{X} * \boldsymbol{V} + \boldsymbol{c}) \tag{4.1}$$

, where $\boldsymbol{X} \in \mathbb{R}^{N \times m}$, $\boldsymbol{W} \in \mathbb{R}^{k \times m \times n}$, $\boldsymbol{b} \in \mathbb{R}^n$, $\boldsymbol{V} \in \mathbb{R}^{k \times m \times n}$, $\boldsymbol{c} \in \mathbb{R}^n$ are trainable parameters, whereas k is the window size, and m, n are respectively the number of input and output feature maps, σ is the sigmoid function and $\otimes$ is the element-wise product between matrices.

By observing equation 4.1 it is implied that the output of each layer is a linear projection $\boldsymbol{X} * \boldsymbol{W} + \boldsymbol{b}$ adjusted by the gates $\sigma(\boldsymbol{X} * \boldsymbol{V} + \boldsymbol{c})$, which actually decide what information should be passed on to the next levels of the hierarchy, mimicking, essentially, the way that RNNs work.

The hierarchical stack of the layers on top of the input $\boldsymbol{E}$ gives a representation of the context for each word $\boldsymbol{H} = h_L \circ \ldots \circ h_0(\boldsymbol{E})$, while convolutional and gated linear units are wrapped in a pre-activation residual block that adds the input of the block to the output.

Apart from the parallelizability and the RNN properties that GCNN models maintain, one of the main reasons for particularly choosing GCNN is the hierarchical structure of the model. Hierarchically analyzing essays contributes to the feature engineering that AES systems require. The goal of this model is to capture features in an essay according to the GMAT AWA scoring guide in section 3.2. Lower levels of the GCNN model are able to detect features that are related to low distance between the words of an essay such as spelling mistakes and grammar of sentences, while higher levels capture more abstract features that are related to high distances between the words, such as the overall quality of the ideas presented as well as the overall ability of the author to develop and express those ideas.

4.2.2.3 Max Pooling Layer

The output o_{GCNN} of the GCNN model is then forwarded into a Max Pooling layer for dimensionality reduction and translation invariance reasons. The function applied is as follows:

$$o_{pool} = \text{MaxPooling}(o_{GCNN})$$

4.2.2.4 Dropout Layer

One of the most commonly used and effective techniques for dealing with overfitting is by adding a Dropout Layer to the Deep Learning model. Adding a Dropout Layer to the model presented proved to be very effective in controlling overfitting since it randomly deactivates some neurons based on a user-defined probability:

$$o_{dropout} = \text{Dropout}(o_{pool})$$

4.2.2.5 Fully Connected Layer with Softmax Activation

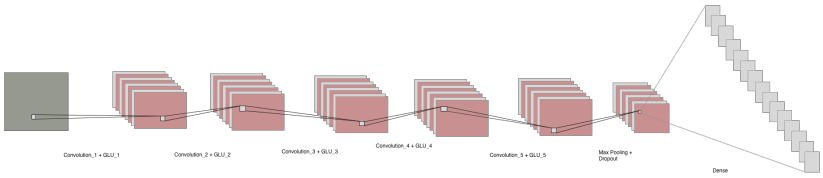


Figure 4.3: Neural Network architecture of the model presented with five gated convolutional layers, a max pooling layer and a fully connected one.

At this point, the output of the Dropout Layer is a 3-dimensional tensor of size $w_p \times l_p \times f$, where w_p is the pooled embeddings size, l_p is the pooled average length of an essay and f is the number of feature maps. This 3-dimensional tensor is essentially a representation of an essay. The goal of the system is to use this tensor in order to predict a class among 6 possible essay scores. Therefore, the $o_{dropout}$ vector is transformed into a new vector of size 6 using a Fully-Connected Layer as follows:

$$\boldsymbol{o}_{FC}^{GCNN} = \boldsymbol{W}_{FC}\boldsymbol{o}_{dropout}^{GCNN} + \boldsymbol{b}_{FC}^{GCNN}, \quad where \quad o_{FC}^{GCNN} \in \mathbb{R}^6$$

In order to get the desired probability distribution, the Softmax function is applied on the output of the fully connected layer:

$$\boldsymbol{p}_{FC}^{\prime GCNN} = \text{Softmax}(o_{FC}^{GCNN}) \tag{4.2}$$

The goal of the GCNN model, similar to other Machine Learning approaches is to minimize the cost function. The cost function being calculated in the supervised part of this model is ***cross-entropy loss***, a standard approach to classification problems. Cross entropy loss, in the context of the approach presented, can be defined as follows:

$$loss_{GCNN} = -\frac{1}{N}\sum_{i=0}^{N} \boldsymbol{p}_{FC}^{GCNN} \cdot log(\boldsymbol{p}_{FC}^{\prime GCNN}) \tag{4.3}$$

, where $\boldsymbol{p}_{FC}^{GCNN}$ is the ground truth probability distribution, N is the batch size and $\cdot$ is the vector product operation.

The output vector generated by the Fully Connected Layer is the first source of predictions of the system presented. The following sections will describe how this vector is going to be used in combination with the topics latent vector to produce the final

prediction.

The neural network architecture part of the model can be seen in Figure 4.3

4.2.3 Topic Modeling using Latent Dirichlet Allocation

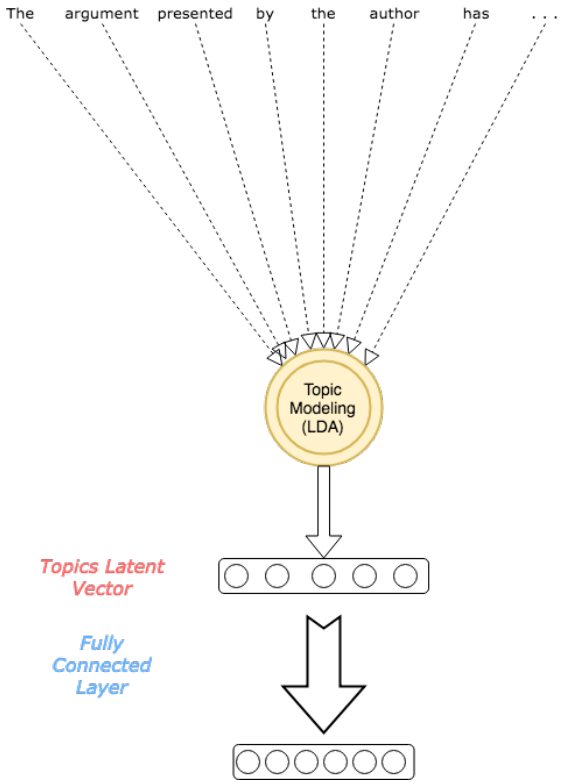


Figure 4.4: Block diagram of the Topic Modeling procedure. A topics latent vector is first extracted from the essay, which is later forwarded to a fully connected layer that outputs a vector that will be used to predict the essay's score.

As it has already been noted, *Latent Dirichlet Allocation* is the algorithm used in this work in order to perform Topic Modeling on the training dataset. LDA is an unsupervised Machine Learning algorithm which assumes there are k topics $T = \{t_1, t_2, \ldots, t_k\}$ in document corpus D. Each document $d \in D$ has a topic distribution θ_d over T and each topic has a word distribution ϕ over a fixed vocabulary. After LDA has been trained on corpus D, it is able then to generate a topics latent vector for each document it sees, which ,essentially, is a distribution over the - user-defined - number of topics.

Aim of this part of the system's architecture is to obtain a second source of predictions. The first step of this procedure, as it has already been mentioned is the extraction of the topics latent vector of the essay d that is given as input to the model:

$$\boldsymbol{o}^d_{LDA} = \text{TopicModeling(d)}$$

, where function $TopicModeling$ is analytically explained in section 2.2 and

$$\sum_{i=1}^{T}(o_{LDA}^{d})_i = 1$$

, where T is the number of topics.

Since Topic Modeling constitutes the second source of predictions, there is a need to generate a predictions vector with size equal to the possible number of classes. This is the reason why o_{LDA} is forwarded to a Fully Connected Layer, the output of which is defined as follows:

$$o_{FC}^{LDA} = \boldsymbol{W}_{FC}^{LDA}\boldsymbol{o}_{LDA}^{d} + \boldsymbol{b}_{FC}^{LDA}, \quad where \quad o_{FC}^{LDA} \in \mathbb{R}^6$$

Similarly to section $4.2.2.5$, the loss function calculated in this part of the model is the ***cross-entropy loss***, that can be defined as follows:

$$loss_{LDA} = -\frac{1}{N}\sum_{i=0}^{N}\boldsymbol{p}_{FC}^{LDA} \cdot log(\boldsymbol{p}_{FC}^{\prime LDA}) \tag{4.4}$$

, where $\boldsymbol{p}_{FC}^{LDA}$ is the ground truth probability distribution, N is the batch size and $\cdot$ is the vector product operation.

The goal of this layer is to provide the model with some additional information on the higher level features of the essay such as the ideas and content of the essays, features that are most difficult to detect. The block diagram of the Topic Modeling part can be seen in Figure 4.4.

4.3 Fusion of the two Machine Learning Models

At this point of the model, the two vectors o_{FC}^{neural} and o_{FC}^{LDA} produced by the Fully-Connected layers which are described in sections $4.2.2.5$ and $4.2.3$ respectively are going to be combined in order to produce the final probability distribution vector over the possible classes - scores.

The way of combining the two vectors is by a *weighted sum*. A new parameter α is introduced which represents the weight with which each of the two models - neural network and topic modeling - contributes to the final prediction. More specifically, the vector being generated from the fusion of the two models is as follows:

$$o_{fused} = \alpha \times o_{FC}^{GCNN} + (1-\alpha) \times o_{FC}^{LDA}$$

, where $\alpha \in [0, 1]$

The desired distribution vector on all possible classes $p_{softmax}$ is then generated by applying the Softmax Activation function on the fused vector o_{fused} as follows:

$$\boldsymbol{p}_{\text{softmax}} = \text{Softmax}(o_{fused})$$

, where:

$$\sum_{i=1}^{6} \boldsymbol{p}_i = 1$$

and Softmax function is defined as follows:

$$\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \quad for \quad j = 1, \ldots, K$$

Respectively, the loss function calculated and minimized can be defined as follows:

$$loss_{total} = \alpha \times loss_{GCNN} + (1 - \alpha) \times loss_{LDA} \tag{4.5}$$

, where $loss_{GCNN}$ and $loss_{LDA}$ are defined in sections $4.2.2.5$ and $4.2.3$ respectively.

The general fused architecture of the whole system can be seen in Figure 4.5.

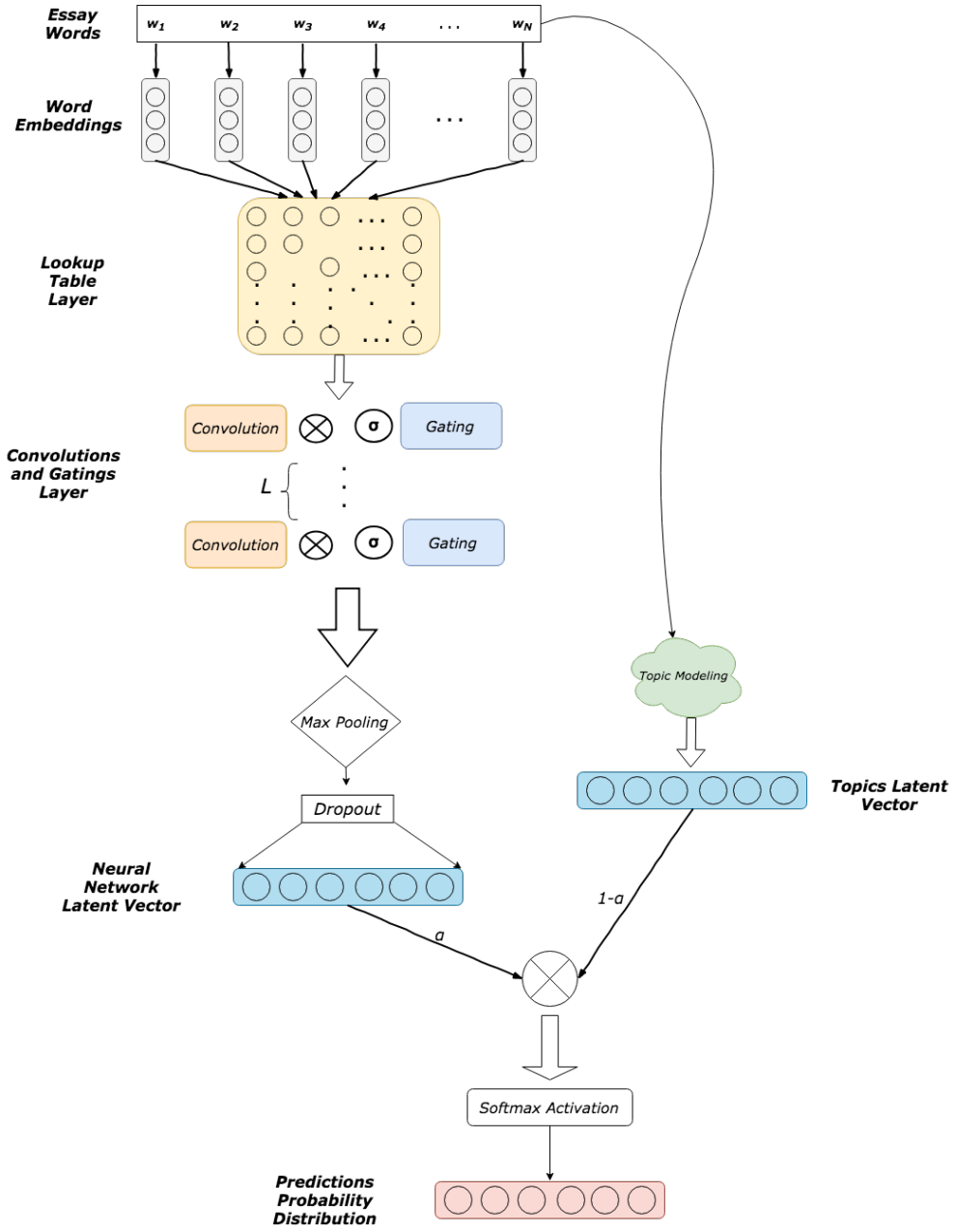


Figure 4.5: Architecture of the fused model. The first latent vector is produced by the Neural Networks GCNN model, while the other vector is produced by performing Topic Modeling on the essay. The two latent vectors are then combined to produce the final probabilities distribution over the possible classes.

Chapter 5

Technical Implementation

5.1 TensorFlow

Most of the Automated Essay Scoring System presented in this work was implemented in TensorFlow. TensorFlow [Abadi et al., 2016] is an open-source software library, employed widely in Machine Learning and Deep Learning applications, developed, originally, by the Google Brain team. It derives its name by '*tensors*' which are, essentially, multidimensional data arrays and it operates by using dataflow graphs that represent computations. TensorFlow is supported by many programming languages such as Python and C++. The whole project was implemented in Python and the system was trained and evaluated on an **Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz**.

5.2 AES System Implementation

5.2.1 Data Preprocessing

One of the first tasks in most of the Deep Learning projects is the data preprocessing task. In this work, essays were handed in raw .txt files together with their ids and scores, but with some differences in the file pattern that made it infeasible to load the data as is. By creating and employing regular expressions that can be found in the respective source code file, essays were brought into a form that made it able for the Python libraries to load the data into three separate columns:

- *Essay ID*
- *Essay*
- *Score*

After efficiently processing raw text files, it was possible to load the data set using the **Pandas** library [McKinney], which is, essentially, a Python library that provides tools for efficiently processing and working with structured data sets.

The next step to the data preprocessing task was the creation of a vocabulary, consisting of a user-defined number of words. Prior to this, tokenization and lower-capitalization of the words that make up the essays should be implemented.

The tokenization of the essays was implemented by employing the **NLTK** library [Loper and Bird, 2002]; an open-source Python library full of tools useful for Natural Language Processing tasks.

The *Vocabulary Class*, then, creates a vocabulary from the tokens of the whole dataset.

5.2.2 Building and Reading TFRecords

On of the main decision choices one has to make when building Deep Learning applications using TensorFlow is about the format of the input data when feeding the model. TensorFlow offers a various number of methods for importing data to the model. The format that TensorFlow recommends is ***TFRecords***, a supported format that makes it easier to mix and match data sets and network architectures. The *build_tfrecords* converts the data - essay, score and topic of each essay - to a TFRecord format. It actually gets the data, stuffs it in a protocol buffer, serializes the protocol buffer to a string, and then writes the string to a TFRecords file as it can be seen in the example code below:

```
def build_tfrecords(filename):
.
.
.
  with tf.python_io.TFRecordWriter(filename) as writer:
          for index in range(num_examples):
              #Create the protocol buffer
              essay = essays[index].tostring()
              score = scores[index].tostring()
              topic = topics[index][:,1].tostring()
              example = tf.train.Example(
                  features=tf.train.Features(
                  #Convert features to bytes
                  feature={
                  'essay':_bytes_feature(essay),
                  'score':_bytes_feature(score),
                  'topic':_bytes_feature(topic)
                  }))
              #Write the string to a TFRecords file
              writer.write(example.SerializeToString())
```

Listing 5.1: Building TFRecords

Then, using the TensorFlow *Dataset Class* TFRecords are decoded, batched, shuffled and are, then, fed as input to the model for training and evaluation as it can be seen in the following example code:

```
def read_tfrecords(tfrecords_filename):
.
.
.
    keys_to_features = {
        "essay": tf.FixedLenFeature((), tf.string),
        "score": tf.FixedLenFeature((), tf.string),
        "topic": tf.FixedLenFeature((), tf.string)
    }
    parsed = tf.parse_single_example(record, keys_to_features)
```

```
    # Decode features that are in bytes format.
    essay = tf.decode_raw(parsed["essay"], tf.int64)
    score = tf.decode_raw(parsed["score"], tf.int64)
    topic = tf.decode_raw(parsed["topic"], tf.float64)
    return {'essay':essay, 'topic':topic} ,score

# Use `Dataset.map()` to build a pair of a feature dictionary and a label
# tensor for each example.
dataset = dataset.map(parser)
dataset = dataset.shuffle(buffer_size=10000)
dataset = dataset.batch(params.batch_size)
dataset = dataset.repeat(10000)
iterator = dataset.make_one_shot_iterator()
```

Listing 5.2: Reading TFRecords

5.2.3 Topic Modeling

Topic Modeling was implemented using the **Gensim**[1] library, an open-source Python library for performing Space and Topic Modeling. Each essay had to be converted into a *Bag-of-Words* (**BoW**) format, while each word had been matched with its respective integer based on the vocabulary of the dataset.

5.2.4 Training and Evaluation

After building the TensorFlow graph for the model, the next step is to feed it with data in order to train and evaluate the model. Training and evaluation of the model were performed using the *TensorFlow Estimators* Class. TF Estimators are a high-level API of TensorFlow that simplifies major Deep Learning tasks such as training and evaluation of the models. Some of their advantages are the building of state of the art models with highly intuitive code and the automatization of many tasks when building a TensorFlow graph, such as variables initialization, exception handling, loading and saving checkpoints.

The whole pipeline of the training and evaluation of the model can be seen in Figure 5.1

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

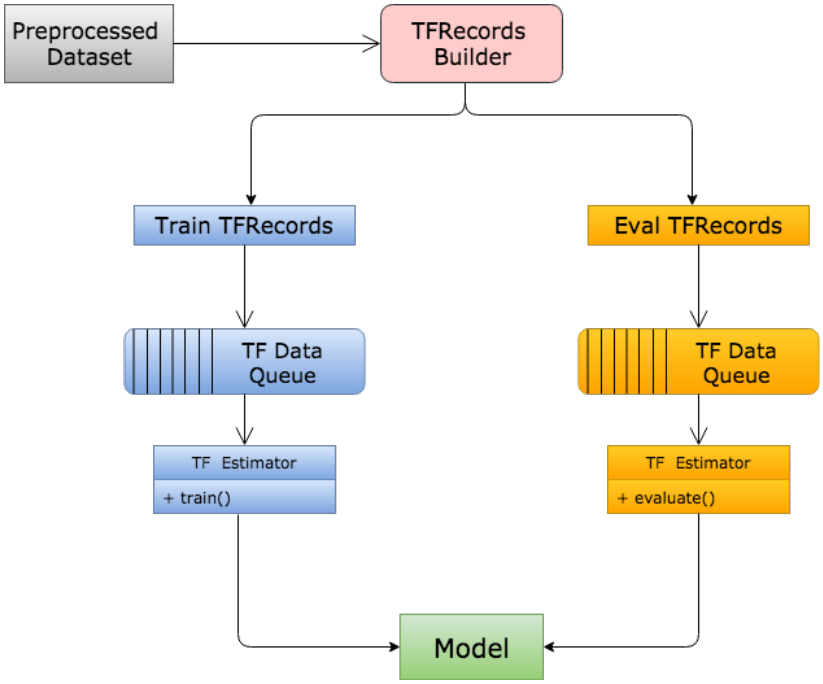


Figure 5.1: Pipeline of the training and evaluation of the model.

Chapter 6

Training Procedure

6.1 Trainable Parameters

During the optimization of the cost function that was defined in Equation (4.5) the values of the trainable variables are adjusted in order for the model to reach a local optimum. However, not all parameters are being trained. The trainable parameters are detected in the following layers and are provided with their respective dimensions in the following tables:

P	Dimensions	Description
WE	vocab_size $\times$ embedding_dimensions	weight matrix for the word embedding vectors

Table 6.1: Word Embeddings Layer parameters along with their description.

P	Dimensions	Description
C1W	filter_size $\times$ filter_size $\times$ number_of_feature_maps	weights matrix of the first convolution
C1B	number_of_feature_maps	bias vector of the first convolution
G1W	filter_size $\times$ filter_size $\times$ number_of_feature_maps	weights matrix of the first gating unit
G1B	number_of_feature_maps	bias vector of the first gating unit
C2W	filter_size $\times$ filter_size $\times$ number_of_feature_maps $\times$ number_of_feature_maps	weights matrix of the second convolution
C2B	number_of_feature_maps $\times$ number_of_feature_maps	bias vector of the second convolution
G2W	filter_size $\times$ filter_size $\times$ number_of_feature_maps $\times$ number_of_feature_maps	weights matrix of the second gating unit
G2B	number_of_feature_maps$\times$ number_of_feature_maps	bias vector of the second gating unit
...	...	...
C5W	filter_size $\times$ filter_size $\times$ number_of_feature_maps $\times$ number_of_feature_maps	weights matrix of the fifth convolution
C5B	number_of_feature_maps $\times$ number_of_feature_maps	bias vector of the fifth convolution
G5W	filter_size $\times$ filter_size $\times$ number_of_feature_maps $\times$ number_of_feature_maps	weights matrix of the fifth gating unit
G5B	number_of_feature_maps$\times$ number_of_feature_maps	bias vector of the fifth gating unit

Table 6.2: Convolutional and Gating Layer parameters along with their description.

P	Dimensions	Description
FCW	number_of_topics $\times$ number_of_outputs	weight matrix for the fully connected layer
FCW	number_of_outputs	bias vector for the fully connected layer

Table 6.3: Fully Connected Layer in Topic Modeling parameters along with their description.

P	Dimensions	Description
FCW	GCNN_output $\times$ number_of_outputs	weight matrix for the fully connected layer
FCW	number_of_outputs	bias vector for the fully connected layer

Table 6.4: Fully Connected Layer in GCNN parameters along with their description.

The total number of trainable parameters of the network was calculated to be *3,354,929* for the hyper-parameters provided in the next sections.

6.2 Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent Algorithm (SGD), originally introduced by [Kiefer and Wolfowitz, 1985], is an optimization algorithm used for iteratively optimizing any differentiable objective function. In the context of Neural Networks, the aim of SGD is to, usually, minimize a loss function. In contrast to the plain SGD, Stochastic SGD uses only a subset of the dataset when updating the model's trainable parameters in each iteration, known as ***batch***. The loss being minimized was defined in equation (4.5). If we assume a batch with samples $\{(\boldsymbol{x}_i, y_i)\}_{i=1,\dots,batch\_size}$, where $\boldsymbol{x}_i$ is the input essay and y_i the respective score of the essay, then the total loss function can be rewritten with respect to the trainable parameters $\boldsymbol{\theta}$ of the network as follows:

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} H(\boldsymbol{\theta}, \boldsymbol{x}_i, y_i) \tag{6.1}$$

, where N is the batch size and $H(\boldsymbol{\theta}, \boldsymbol{x}_i, y_i)$ is the sum of the cross entropy losses as defined in the previous sections. Equation (6.1) defines the total loss of the system which needs to be minimized in order for the model to reach a global minimum. If we assume that the set of trainable parameters $\theta = \{w_1, w_2, \dots, w_n\}$, where n is the number of the trainable parameters, then for $w \in \theta$ the derivative of the total loss can be computed as follows:

$$\nabla_w J(\boldsymbol{\theta}) = \nabla_w \Big(\frac{1}{N}\sum_{i=1}^{N} H(\boldsymbol{\theta}, \boldsymbol{x}_i, y_i) \Big) \tag{6.2}$$

Then, at each optimization step k, each trainable parameter w should be updated according to the Stochastic SGD algorithm as follows:

$$w^{(k)} := w^{(k-1)} - \eta \nabla_w J(\boldsymbol{\theta}^{(k-1)}) \tag{6.3}$$

or

$$w^{(k)} := w^{(k-1)} - \eta \nabla_w \Big(\frac{1}{N} \sum_{i=1}^{N} H(\boldsymbol{\theta}, \boldsymbol{x}_i, y_i) \Big) \tag{6.4}$$

,where η is the learning rate.

A small variation of the Stochastic SGD that is being employed in this work is the Stochastic SGD with Momentum. By the term 'Momentum' what is essentially implied is a moving average of the gradients when updating the parameters. Then, the updating rule having been defined in Equation (6.4) becomes as follows:

$$w^{(k)} := w^{(k-1)} - \eta \nabla_w \Big(\frac{1}{N} \sum_{i=1}^{N} H(\boldsymbol{\theta}, \boldsymbol{x}_i, y_i) \Big) + \upsilon (w^{(k-1)} - w^{(k-2)}) \tag{6.5}$$

, where υ is the momentum parameter.

6.3 Model Hyper-parameters

There are a lot of techniques that can be used in order to determine the hyper-parameters of a model, among of which are *Bayesian Optimization, Grid Search* and *Random Search*. Due to the restricted availability of hardware resources, the basic method followed throughout this work is Random Search, in which most of the hyper-parameters were set with a random value, while many of them were initialized based on [Dauphin et al., 2016], and then adjusted to the needs of the network.

6.3.1 Learning rate

Learning rate η is a hyper-parameter that controls how much the weights of the network are adjusted with respect to the loss gradient, and is considered to be one of the most important hyper-parameters that needs to be set, since it can make significant difference on the performance of the model. High values of this hyper-parameter can lead to divergence of the model, while too low values can potentially hinder the model from converging fast to a local minimum resulting in expensive computations. The best performance of the model was observed with the learning rate η set at *0.00001*. Higher order values for the learning rate caused the model to diverge

6.3.2 Batch size

Batch size defines the number of samples that the model has to see before any updates on the trainable weights of the network are made. Choosing a small batch size

leads to more frequent updates of the weights' values, that means less memory, but it also means that the model has a less accurate estimation of the gradient, compared to a bigger batch size, which is,however, computationally more expensive. The batch size was set to *16* in this work, since higher values could not fit in the memory of the CPU.

6.3.3 Momentum

Ideally, the goal of training a model is to make it reach a global optimum. In reality, however, this is very difficult, since the network can easily get stuck in local optima, and the training algorithm may think that it has reached a global one leading to sub-optimal results. *Momentum* is a term added in the cost function as defined in Equation 6.5, with values ranging from 0-1, that increases the size of the steps taken towards the global optimum in an effort to avoid local optima. The general rule when selecting values for momentum, is that when momentum term is large, then learning rate should be maintained in low levels. Thus, the value that was set to momentum was *0.9*.

6.3.4 Dropout

One of the most effective regularization techniques for dealing with over-fitting in neural networks is Dropout. At each training step, neurons of the network are "de-activated" by dropping them out of the network with a probability $1 - p$(neurons are kept with a probability p), that is the hyper-parameter that needs to be defined at this layer. Ingoing and outgoing edges to those nodes are also dropped out. A dropout layer with $p = 0.95$ was employed in the model presented, since with this value the network proved to have the best performance.

6.3.5 Gradient Clipping

Gradient Clipping [Pascanu et al., 2012] is a technique used mostly in Recurrent Neural Networks that deals effectively with the *Gradient Explosion Problem*. It does this by, essentially, clipping the gradients or capping them to a threshold value to prevent the gradients from getting too large. Although, in this work, there is no use of recurrent architecture, a gradient clipping value of *0.1* proved to help the network converge significantly faster.

6.3.6 Number of Training Steps

The training steps hyper-parameter refers to the number of batches seen by the model, and it is, essentially, the number of weight updates the model does towards optimization. The number of training steps for which the system was trained varied from *12,000-50,000* depending on the size of each dataset.

6.3.7 Additional Hyper-parameters

Hyper-parameters described in the previous subsections constitute the most important parameters that need to be set in a Deep Learning context. However, there are

a lot more hyper-parameters that affect the performance of the model they have to be carefully chosen. In table 6.5 can be seen all the additional hyper-parameters and the values that were set to them.

Hyper-parameter	Value
Embedding Dimensions	200
Convolution Window Size	5
Convolutional Filter Height	3
Convolutional Filter Width	3
Convolutional Filter Height	3
Number of Convolutional Filters	16
Number of Convolutional Layers	5
Number of Topics	5
Pooling Size	3
α	0.7
Vocabulary Size	7300

Table 6.5: Table with all the additional hyper-parameters and their values

6.4 Automated Essay Scoring System Training

The Automated Essay Scoring System, essentially, consists of two smaller networks, the GCNN part and the Topic Modeling part that employs the LDA algorithm. The whole network is being trained using **BackPropagation Through Time (BPTT)**. BPTT calculates the gradient of the cost function with respect to the trainable parameters through time and then employs the SGD algorithm that was described in chapter 6.2.

Regarding the weights initializations, it was decided not to initialize the word embeddings with any pre-trained weights, while the weights of the GCNN part were randomly initialized from a Gaussian Distribution with *zero mean* and *0.1* variance.

The whole network was trained on four different datasets consisting of four different questions. The decision of not training the network on a mixing dataset of different questions was made because of the fact that the system would not, then, be able to correctly assess the content and ideas of the essay, since it would not have any information regarding question of the essay. In case the question of the essays was available , along with the essay, a possible solution to this problem would be to feed the question of the essay along with the essay to the model, that would render the system capable of understanding to which question the essay is answering. Each dataset was augmented with random essays that received the lowest score possible. The number of random essays added to each dataset was 20% of the size of the dataset prior to the augmentation. Additionally, all essays were adjusted to consist

of 750 words, that was found to be the average length of an essay. Essays with fewer words were padded with zeros, while longer essays were clipped to 750 words.

The four different datasets were chosen so that they have different number of essays each. The number of training essays of each dataset can be seen in Table 6.6.

Dataset	Number of Training Essays	Number of Training Steps
Dataset A	14,080	50,000
Dataset B	8,336	42,000
Dataset C	4,752	15,000
Dataset D	816	10,000

Table 6.6: The four different datasets that the network was trained on, along with the number of essays in each one.

The distributions of the scores in each of the dataset can be seen in Table 6.7
The training process of the whole networks took place in two phases:

- ***Training Phase 1***

  During the first training phase, the training of the Latent Dirichlet Allocation (LDA) algorithm takes place. Since LDA is an unsupervised method of learning, the network is trained on each whole dataset - training and evaluation. The only parameters trained are those of the LDA algorithm, explained in chapter 2.2, and no parameters of the GCNN model are modified. LDA is trained to predict a topics distribution once it sees the whole essay.

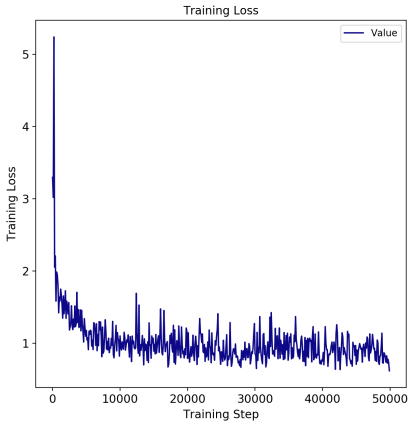
- ***Training Phase 2***

  The second and most important phase of the training involves the training of the whole network in an end-to-end fashion which is done in a supervised manner. The trained LDA is used to provide a topics distribution of the input essay. The total loss is, then, calculated as defined in Equation (4.5) and the error gradients are propagated through the GCNN model, in which the trainable parameters defined in section 6.1 are updated. No parameters of the LDA model are updated during BPTT.

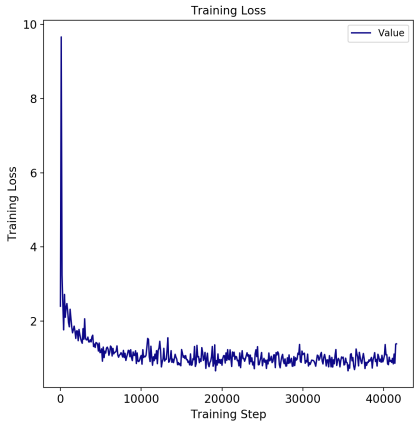
	1	2	3	4	5	6
Dataset A	3,048	122	959	3813	4172	1966
Dataset B	1,437	210	926	2,514	2,266	983
Dataset C	820	52	289	1,460	1,560	571
Dataset D	137	8	49	246	267	109

Table 6.7: Distribution of the number of scores of the essays in the four separate datasets.

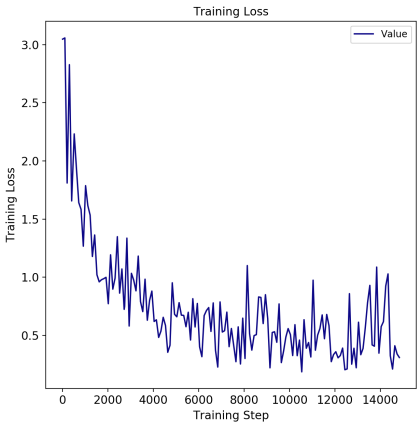
In the tables provided below, one can see the training losses graphs for the four different datasets, as well as the time each training step required.



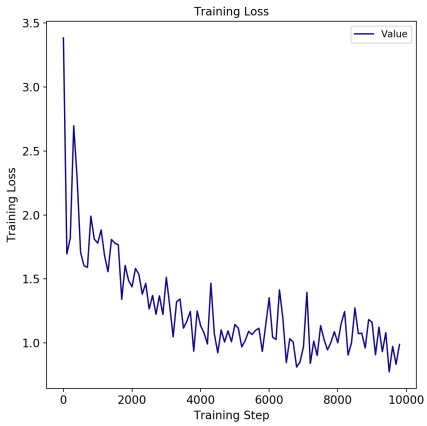
(a) Dataset A



(b) Dataset B

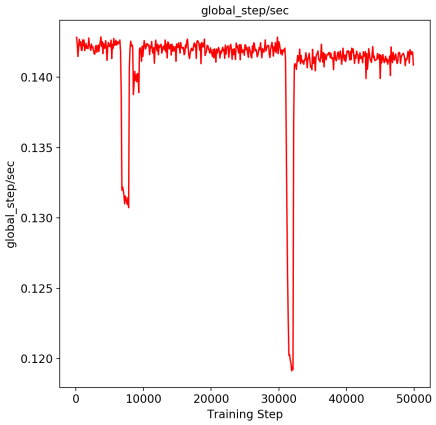


(c) Dataset C

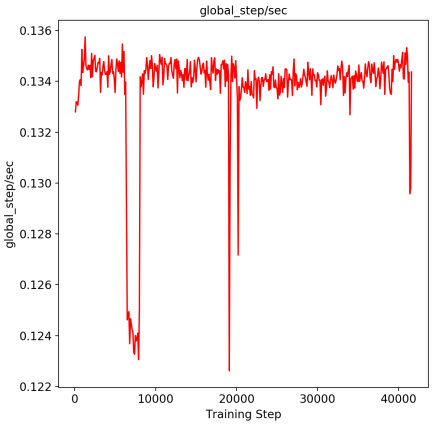


(d) Dataset D

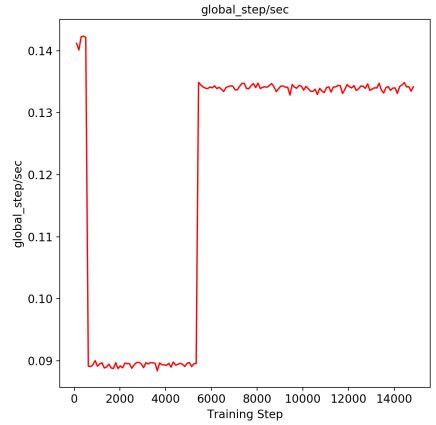
Figure 6.1: Training loss graph for each of the four datasets.



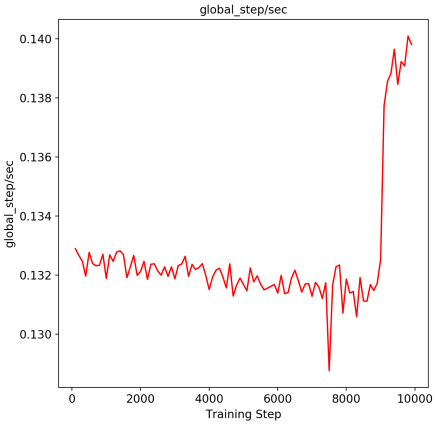
(a) Dataset A



(b) Dataset B



(c) Dataset C



(d) Dataset D

Figure 6.2: Global step/sec for each of the four datasets.

Chapter 7

Evaluation

7.1 Evaluation Results

The model was evaluated on the four separate datasets that was also trained on. The size of each evaluation dataset can be seen in Table 7.1.

Dataset	Number of Evaluating Essays
Dataset A	5,744
Dataset B	3,568
Dataset C	2,032
Dataset D	352

Table 7.1: The sizes of the four datasets that the network was evaluated on, alongside with the number of essays in each one.

The metrics that were employed to evaluate the model are the following:

Accuracy

Accuracy is the fraction of predictions the model got right to the total number of predictions. It can be defined as follows:

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

Cohen's Kappa

Cohen's Kappa or *Kappa Statistic* is a metric that compares and observed accuracy with an expected accuracy. In contrast to accuracy, Cohen's Kappa takes into account random chance as well, which makes it a more robust metric than accuracy.

Confusion Matrix

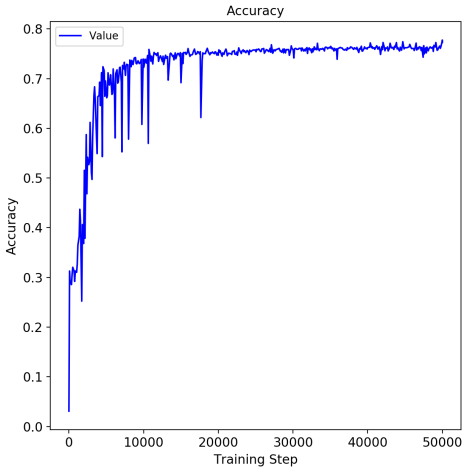
Confusion matrix, also known as *error matrix*, is a visualization table that summarizes up the performance of a, usually, supervised algorithm. Each row of the matrix

represents the assignments to classes that the prediction algorithm did, while the columns represent the actual classes those predictions belong to. By observing the confusion table one can notice between which classes the algorithm gets confused and, thus, make useful assumptions and corrections to the algorithm.

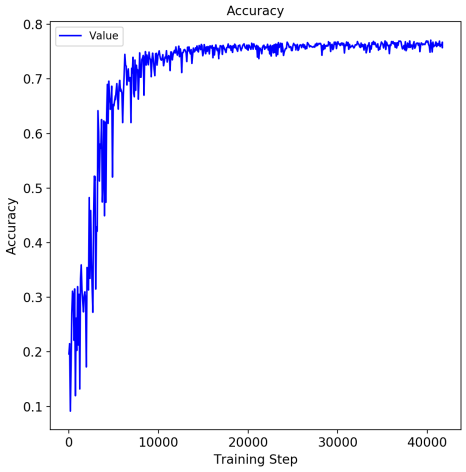
The final values for the for the accuracy and Cohen's Kappa metrics can be seen in Figure 7.2, while the graphs for the same metrics and the evaluation loss during the training can be seen in Figures 7.1 and 7.2 and 7.3. The confusion matrices for each dataset can be seen in tables 7.3, 7.4, 7.5 and 7.6 respectively for each one of the datasets.

Dataset	**Accuracy**	**Cohen's Kappa**
Dataset A	0.802	0.761
Dataset B	0.781	0.723
Dataset C	0.822	0.758
Dataset D	0.753	0.701

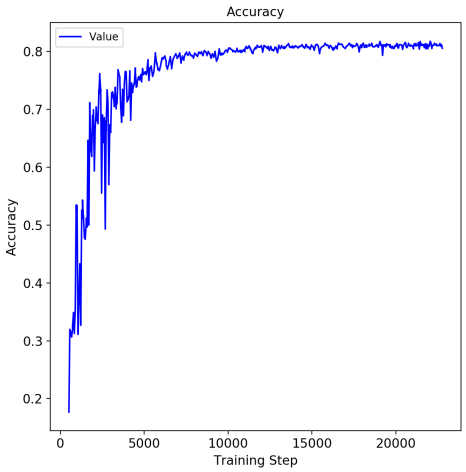
Table 7.2: The four different datasets that the network was trained on, along with the number of essays in each one.



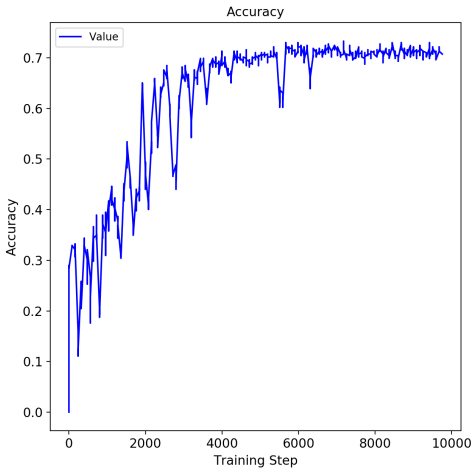
(a) Dataset A



(b) Dataset B

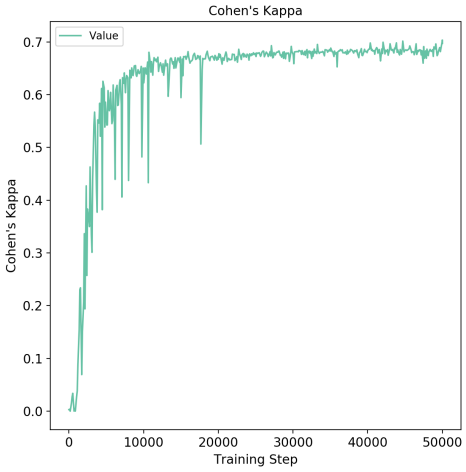


(c) Dataset C

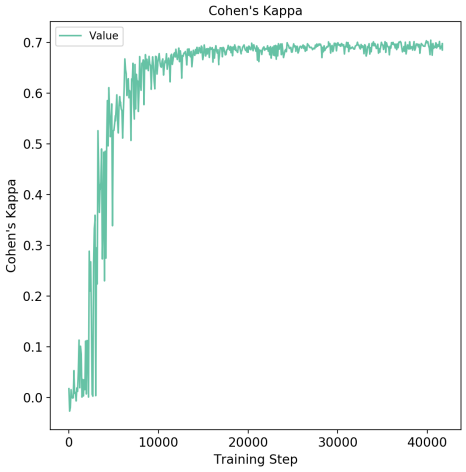


(d) Dataset D

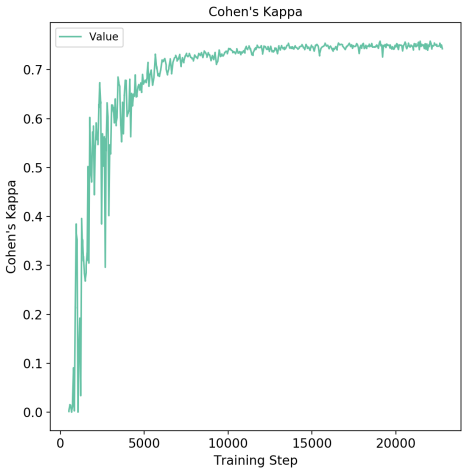
Figure 7.1: Accuracy for each of the four datasets.



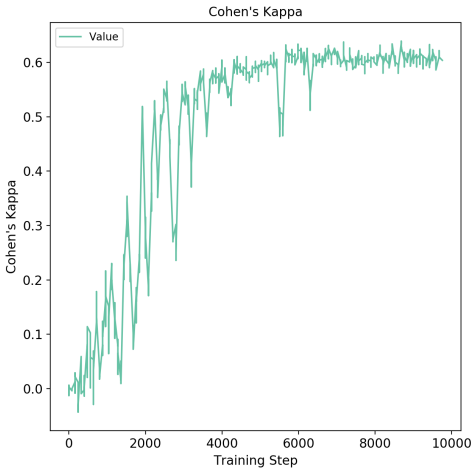
(a) Dataset A



(b) Dataset B

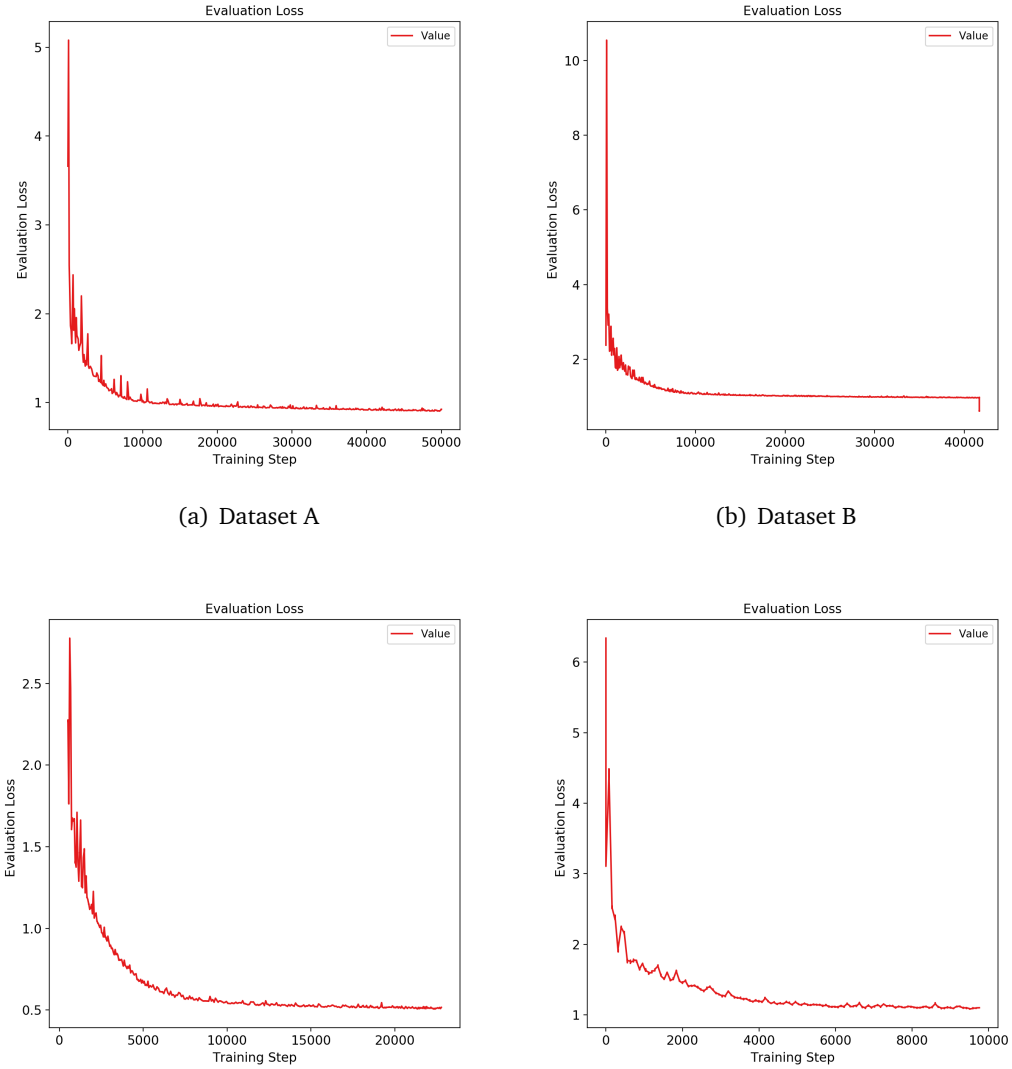


(c) Dataset C



(d) Dataset D

Figure 7.2: Cohen's Kappa for each of the four datasets.



(a) Dataset A

(b) Dataset B

(c) Dataset C

(d) Dataset D

Figure 7.3: Evaluation Loss for each of the four datasets.

Actual \ Predicted	Score 1	Score 2	Score 3	Score 4	Score 5	Score 6
Score 1	1009	7	1	0	0	0
Score 2	1	9	42	3	0	0
Score 3	0	8	159	225	14	2
Score 4	0	4	70	1231	354	4
Score 5	0	0	5	235	1473	85
Score 6	0	0	1	3	239	560

Table 7.3: Confusion Matrix for Dataset A.

Actual \ Predicted	Score 1	Score 2	Score 3	Score 4	Score 5	Score 6
Score 1	587	13	0	0	0	0
Score 2	0	62	18	1	0	0
Score 3	1	47	178	166	3	1
Score 4	0	15	54	903	183	8
Score 5	0	1	4	176	707	79
Score 6	0	0	0	1	83	277

Table 7.4: Confusion Matrix for Dataset B.

Actual \ Predicted	Score 1	Score 2	Score 3	Score 4	Score 5	Score 6
Score 1	359	7	1	0	0	0
Score 2	3	3	25	2	0	0
Score 3	0	1	103	43	1	0
Score 4	2	0	39	488	94	0
Score 5	0	0	3	63	571	25
Score 6	3	0	0	0	48	156

Table 7.5: Confusion Matrix for Dataset C.

		Predicted					
		Score 1	Score 2	Score 3	Score 4	Score 5	Score 6
Actual	Score 1	56	0	0	0	0	0
	Score 2	0	1	3	1	0	0
	Score 3	0	0	2	22	2	0
	Score 4	0	0	1	92	22	1
	Score 5	0	0	0	10	85	10
	Score 6	1	0	0	0	12	22

Table 7.6: Confusion Matrix for Dataset D.

7.2 Setting a Threshold

As seen in the previous section, the highest accuracy that the system achieved was 0.815. An AES system would be more useful, though, if it could assess fewer essays with a higher accuracy. Thus, a proposed approach to this problem would be to set a threshold on the probabilities distribution that the system generates. Generally, the system predicts a score class by giving as output the argmax of the probability distribution. Setting a threshold of confidence on the highest probability of this distribution, below which the essay with this distribution would not be graded, would result in assessing fewer essays but with a higher accuracy. In this work several threshold were tested. The results for threshold values ρ=0.5, 0.6, 0.7, 0.8, 0.9 for the different datasets are presented in tables 7.7 and 7.8, while in table 7.9 there are the percentages of the size of the data sets after the threshold had been applied.

Threshold	*Dataset A*	*Dataset B*	*Dataset C*	*Dataset D*
0.6	0.836	0.824	0.860	0.787
0.7	0.846	0.863	0.913	0.811
0.8	0.87	0.964	0.930	0.845
0.9	0.95	1	0.0	0.88

Table 7.7: ***Accuracy*** for different values of threshold on the four datasets

Threshold	*Dataset A*	*Dataset B*	*Dataset C*	*Dataset D*
0.6	0.776	0.762	0.811	0.715
0.7	0.753	0.812	0.878	0.747
0.8	0.75	0.898	0.883	0.792
0.9	0.0	0.0	0.0	0.83

Table 7.8: ***Cohen's Kappa*** metric for different values of threshold on the four datasets.

Threshold	*Dataset A*	*Dataset B*	*Dataset C*	*Dataset D*
0.6	64.35%	73.5%	82.5%	90%
0.7	42.75%	65.81%	71.88%	81.25%
0.8	13.28%	50.23%	59.45%	60.79%
0.9	0.0%	12.44%	25.52%	33.52%

Table 7.9: Proportion of the initial size of the dataset that is left after a threshold is applied.

7.3 Discussion

By observing the tables and the figures above, one can tell that the best performance is achieved by dataset C, whose training data set consists of 4,752 essays. By observing the confusion matrices it is clear that the cases that the system is mostly confused at is between classes of score 4 and 5, since it is possible that essays of those two scores are similar enough. It is also worth noting the high accuracy in class of score 1. In order for an essay to get a score of 1 it should be very short or out of topic. The number of essays graded 1 in the dataset were very few, but as it has already mentioned the dataset was augmented with random essays that were scored with 1. Thus, it can be inferred that the system reacts effectively when a random essay is given as input.

Setting a threshold value resulted in better metrics on a reduced dataset. Since a threshold value is set at the maximum probability, it means that many examples are dropped. However, having very high accuracy for some essays is some times preferable than having lower accuracy for all the essays.

Capturing The Content of Essays

An effort to capture the content of an essay was also made in this work. More specifically, we wanted to test how the system reacts when it receives essays that were graded with the highest score - 6 - but were answers for a different question than the one that the system was trained on. As expected, the system predicted high scores for those essays, although they should be scored with the lowest score possible.

The approach that was followed to deal with this problem was to augment the training the dataset - similar to the technique followed with the random essays - with highly graded essays which, however were taken from different datasets and thus were labeled with the lowest possible score - 1.

Evaluation of this system was carried out on an augmented dataset too. The original dataset was **Dataset C**, while out-of-topic essays were added from **Dataset B**. The out-of-topic essays that were added made up 20% of of the training and evaluations datasets. The highest accuracy that was recorded was **74.4%** in contrast to the 82.2% that Dataset C achieved without the dataset augmentation, accuracy that is satisfying for such a difficult problem as the capture of the content. The exact confusion matrix can be seen in Table 7.10. It is clear that the system gets confused between classes 1 and 6, although a satisfying percentage of the added essays were correctly identified as out of topic.

We strongly believe that appropriate proportions of the mixed datasets and hyperparameter optimization will lead to an effective way of the system capturing the content of an essay and being able to discriminate essays which essays are in or out of topic with a higher accuracy.

		Predicted					
		Score 1	Score 2	Score 3	Score 4	Score 5	Score 6
Actual	Score 1	383	0	16	1	61	98
	Score 2	0	0	31	2	0	0
	Score 3	0	0	75	72	1	0
	Score 4	0	0	26	528	69	0
	Score 5	1	0	1	85	508	37
	Score 6	45	0	0	1	42	189

Table 7.10: Confusion Matrix for augmented Dataset C.

Chapter 8

Black-Box Visualization

One of the main disadvantages in employing Deep Learning techniques is the obscure way in which the operations throughout the Neural Networks are taking place. Understanding how AES systems work and clarifying their criteria in essay scoring is of paramount importance, if such systems were to be employed in educational systems since they would, potentially, be able to provide students with feedback explaining the scores that their essays achieved.

[Alikaniotis et al., 2016] developed a technique for providing an insight into their system through the use of visualizations. They achieved this by measuring the quality of the individual trained word embedding vectors of the network. Inspired by their work, this chapter employs this technique to also assess the quality of single words, while it is also extended to assess the quality of sentences as well.

8.1 Visualization Approach

Assessing Words

The aim of this approach is to assess the quality of the word embedding vectors. A vector of high quality would contribute positively to an essay and would be related to highly graded essays. On the opposite, a low quality vector would contribute negatively to an essay and, thus, would be related to low graded essays. The quality of each vector can be measured by performing a single training pass on the network. To make an inference of how ”good” a vector in an essay is, at this training pass, the essay is given as input to the system alongside with the highest score an essay can achieve as label - in this case this score is six - instead of providing the essay with its real label. The error gradients of word embedding vectors are then computed based on this differentiated cost function. Vectors of high quality are going to have low values in the norm of their error gradients, that means that they are going to need little adjustment in their weights in order for the essay to achieve the highest grade possible. Respectively, to measure how ’bad’ a vector is, an essay is provided with the lowest possible grade to the network. Then, vectors of low quality are going to have minimal values in the norm of their error gradients in order for an essay to achieve the lowest grade possible. During this process, no weights are updated.

The cross entropy loss functions for both of the cases described above are as follows:

$$loss_{max} = -\sum p_{max} \log p\prime \tag{8.1}$$

$$loss_{min} = -\sum p_{min} \log p\prime \tag{8.2}$$

, where $p\prime$ is the distribution predicted by the model, while p_{min} and p_{max} are the lowest[1] and highest[2] score probability distributions respectively. For better accuracy, the two losses are combined into a total one as follows:

$$loss_{total} = loss_{max} - loss_{min} \tag{8.3}$$

Intuitively, the highest the total loss, the worse the word embedding vector is. The gradients of the loss with respect to the word embeddings are then calculated, the norm of which is used as a measure of quality of the vectors as defined in Equation 8.4.

$$||g_{word}||_2^i = ||\frac{\partial loss_{total}}{\partial w_i}||_2 \tag{8.4}$$

, where g_{word}^i are the error gradients and w_i the word vectors, for $i \in [1, \quad essay\_length]$.

Extending Approach to Assess Sentences

By producing visualizations of word embedding vectors one can be provided with a good insight into how words and short length relationships between them are assessed by the system. In this way, it is ensured that the system correctly assesses spelling and grammar.

Going a level higher on to the hierarchy, this section aims to provide an insight into the way assessment of sentences and connections between them is being done. In practice, it is an extension of the process described above.

Each essay is divided into sentences - dot to dot. After all the norms of the error gradients of the word embeddings are calculated, a mean error gradient is calculated for each sentence as follows:

$$g_{sentence}^i = \frac{1}{N_i} \sum_{j=1}^{N_i} ||g_{word}^j||, \tag{8.5}$$

, where N_i is the number of words in sentence i. Similarly, the higher the sentence gradient, the worse the sentence is.

[1] 1 in the score 1 position and 0 everywhere else
[2] 1 in the score 6 position and 0 everywhere else

8.2 Visualization Examples

This section presents some representative examples from words and sentences following the approach described in section 8.1. Words and sentences have been characterized based on their color as follows: *Low Quality vectors*, *Medium-Low Quality vectors*, *Medium-High Quality vectors*, *High Quality vectors*.

Example 1

. . . the executive officer is proposing to board of directors a way how to increase profits of the company. . .

Comments

In the above example, the system correctly identifies and rewards the phrase *'is rewarding'* and its correct tense. It assigns a low score to both words *'to'* and *'boards'*, since it detects the missing word *'the'* between those two words. Finally, it assigns low score to the word *'how'* since there is a missing *'of'* prior to this word, while the word *'profits'* that matches with *'increase'* gets a high score.

Example 2

. . . if we want to suggest a word that is exactly means a successful business . . .

Comments

The correct use of grammar is rewarded in the first words of the sentence presented, while the phrase *'is exactly means'* gets a low score since it a both syntactically and grammatically incorrect phrase. The adjective *'successful'* that characterizes the word *'business'* gets rewarded.

Example 3

. . . the standard motorcylce is cheaper to build. . .

Comments

In the example above, although the system praises the use of the adjective *'standard'*, it also identifies the spelling mistake of the word *'motorcycle'* and thus assigns it a low score.

Example 4

. . . the first assumption underlying the claim is that producing . . .

Comments

At this example one can see some flaws of the system since it assigns a low score to most of the words in the sentence, although the sentence seems to be a perfectly correct.

Example 5

While the memo comes to a plausible conclusion , the argument as it is currently presented makes a number of unstated assumptions . First , it fails to discuss how the plan affects profitability. In addition , it does not discuss market potential of the standard motorcycles line . Finally , it never discusses availability of robot used in the standard motorcycles assembly line .

Comments

In this example where sentences are assessed, one can see that the system rewards the connection between sentences by using connection words such as *'while', 'in addition', 'finally'* and, thus, assigns high scores to all sentences in this paragraph.

Example 6

Finally , the argument assumes that the robot used for standard motorcycles line is available . However , there it is possible that a robot is useless to the company . A company which used to produce the robot may have gone bankrupt , or the robot is too expensive to buy considering the cash availability of the company .

Comments

The system rates the sentences above with a low score since the grammar and syntax of those sentences is poor and also the connecting words like *'finally'* and *'however'* do not fit in this specific occasion.

Chapter 9

Conclusions and Future Work

The purpose of this dissertation was to develop an AES system that could perform efficiently in an educational system. By employing a combination of a supervised Machine Learning algorithm - Neural Networks - and an unsupervised one - Latent Dirichlet Allocation - this goal was achieved to a great extent. The architecture used in this project covers all the significant parts of an Automated Essay Scoring system, such as the satisfying performance against standard metrics (Accuracy, Cohen's Kappa) and the extraction of all levels of features based on the rubric score guide of the data set. Additionally, the transparency of the system became a feasible feature of the system, by employing a black-box visualization technique. In this technique, the error gradients of the word embedding vectors are calculated that made it possible for the system to assess both words and sentences and relations between them. As shown, the system was made capable of assessing the writing quality of the essays by detecting complex grammar, syntactical, spelling, and many other features.

The most important contribution of this work was the employment of Convolutional Neural Networks in AES systems, in contrast to the general belief that Recurrent Neural Networks perform better on Natural Language Processing applications. One of the main advantages of CNNs in AES is that their training and evaluation can fully be parallelized in a GPU context, whose importance is even higher due to huge load of datasets that educational institutions own. Additionally, the system presented was rendered capable of dealing in satisfying way with an issue that existing systems have not successfully confronted yet - content of the essay. Capturing the ideas and the content of an essay is a very difficult task. However, the hierarchical structure of the Neural Networks, the contribution of the Topic Modeling algorithm in a high level, and the dataset augmentation with essays that were grammatically and syntactically perfect but were totally out of topic and manually received the lowest score. In a similar way, the system was capable of dealing effectively with essays that were created randomly and also grade them with the lowest score.

Revisiting the initial aims of this work this work managed to succeed in most of the challenges of AES systems. However, there is plenty of area for additional improvement of the system. Some potential future improvements are proposed below.

Hyperparameter Tuning

The network presented has a lot of hyperparameters that need tuning. Due to limitations in time and resources it was not possible to optimize the values of the hyperparameters and achieve a better performance of the model. In this work, most of the hyperparameters were set intuitively, and based on existing experiments. Employing a more complex method of hyperparameter tuning such as Bayesian Optimization or Grid Search is likely to result in better performance of the system. Additionally, the lack of resources prevented the system from training on a bigger batch size than 16. Testing bigger batch sizes may also improve the system's performance.

Test on Different Combinations of the Dataset

The GMAT AWA Dataset consists of $\sim$780,000 essays. In the work presented just $\sim$10% of the whole dataset was used. Future work would include testing the model architecture on different parts of the dataset as well as mixing them. In addition, although the system dealt with random and out of topic essays, it did not do it on a consistent basis. The main reason of this is considered to be the mixing of the dataset. The fact that the time available for processing the dataset was not enough and in combination with the limited availability of resources for faster processing, there was not the chance for training the system on various combinations of the dataset. Better selection of proportions of the datasets mixed is believed to result in better results.

Bibliography

Natural-language processing, Apr 2018. URL https://en.wikipedia.org/wiki/Natural-language_processing.

Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016. URL https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

Dimitrios Alikaniotis, Helen Yannakoudakis, and Marek Rei. Automatic text scoring using neural networks. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2016. doi: 10.18653/v1/p16-1068.

Y. Bengio and J.-S. Senecal. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Transactions on Neural Networks*, 19 (4):713722, 2008. doi: 10.1109/tnn.2007.912312.

David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=944919.944937.

Jill Burstein, Joel Tetreault, and Nitin Madnani. The e-rater automated essay scoring system. *Handbook of Automated Essay Evaluation*. doi: 10.4324/9780203122761.ch4.

Yen-Yu Chen, Chien-Liang Liu, Tao-Hsing Chang, and Chia-Hoang Lee. An unsupervised automated essay scoring system. *IEEE Intelligent Systems*, 2010. doi: 10.1109/mis.2010.3.

Ronan Collobert and Jason Weston. A unified architecture for natural language processing. *Proceedings of the 25th international conference on Machine learning - ICML 08*, 2008. doi: 10.1145/1390156.1390177.

Scott Crossley, Laura K. Allen, Erica L. Snow, and Danielle S. Mcnamara. Pssst... textual features... there is more to automatic essay scoring than just you! *Proceedings*

of the Fifth International Conference on Learning Analytics And Knowledge - LAK 15, 2015. doi: 10.1145/2723576.2723595.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. *CoRR*, abs/1612.08083, 2016. URL `http://arxiv.org/abs/1612.08083`.

Noura Farra, Swapna Somasundaran, and Jill Burstein. Scoring persuasive essays using opinions and their targets. *Proceedings of the Tenth Workshop on Innovative Use of NLP for Building Educational Applications*, 2015. doi: 10.3115/v1/w15-0608.

Shumin Jing. Automatic grading of short answers for mooc via semi-supervised document clustering.

J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Collected Papers*, page 6064, 1985. doi: 10.1007/978-1-4613-8505-9_4.

John Lafferty, Saniel Sleator, and Davy Temperley. Grammatical trigrams: A probabilistic model of link grammar. Jan 1992. doi: 10.21236/ada256365.

Andrew S. Lan, Divyanshu Vats, Andrew E. Waters, and Richard G. Baraniuk. Mathematical language processing. *Proceedings of the Second (2015) ACM Conference on Learning @ Scale - L@S 15*, 2015. doi: 10.1145/2724660.2724664.

Thomas K. Landauer and Susan T. Dumais. A solution to platos problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104(2):211240, 1997. doi: 10.1037//0033-295x.104.2.211.

Deryle Lonsdale and Diane Strong-Krause. Automated rating of esl essays. *Proceedings of the HLT-NAACL 03 workshop on Building educational applications using natural language processing -*, 2003. doi: 10.3115/1118894.1118903.

Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

Wes McKinney. pandas: a foundational python library for data analysis and statistics.

Danielle S. Mcnamara, Scott A. Crossley, Rod D. Roscoe, Laura K. Allen, and Jianmin Dai. A hierarchical classification approach to automated essay scoring. *Assessing Writing*, 23:3559, 2015. doi: 10.1016/j.asw.2014.09.002.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013a. URL `http://arxiv.org/abs/1301.3781`.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger,

editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013b. URL http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

T. Mitchell, T. Russel, P. Broomhead, and N. Aldridge. Towards robust computerized marking of free-text responses. *Proceedings of the Sixth International Computer Assisted Assessment Conference*, 2002.

Andriy Mnih and Geoffrey Hinton. Three new graphical models for statistical language modelling. *Proceedings of the 24th international conference on Machine learning - ICML 07*, 2007. doi: 10.1145/1273496.1273577.

Ellis B. Page. Statistical and linguistic strategies in the computer grading of essays. *Proceedings of the 1967 conference on Computational linguistics -*, 1967. doi: 10.3115/991566.991598.

Ellis B. Page. The use of the computer in analyzing student essays. *International Review of Education*, 14(2), 1968. doi: 10.1007/bf01419938.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012. URL http://arxiv.org/abs/1211.5063.

L. M. Rudner and T. Liang. Automated essay scoring using bayes theorem. *Journal of Technology, Learning, and Assessment*, 1(2), Jun 2002.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–, October 1986. URL http://dx.doi.org/10.1038/323533a0.

Shashank Srikant and Varun Aggarwal. Automatic grading of computer programs: A machine learning approach. *2013 12th International Conference on Machine Learning and Applications*, 2013. doi: 10.1109/icmla.2013.22.

Kaveh Taghipour and Hwee Tou Ng. A neural approach to automated essay scoring. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016. doi: 10.18653/v1/d16-1193.

Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328, 2016. URL http://arxiv.org/abs/1606.05328.

Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. Implementing the new approach to esol. *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference*. doi: 10.1037/e577992011-001.