Context Dependent Natural Language Understanding with Deep Neural Networks

Stephen Fitz

University of Chicago
OVERVIEW

This manuscript has been produced in order to organize my current research endeavors, and will be submitted for the purpose of satisfying the requirements of the MA degree within the PhD program at the University of Chicago. My journey from pure mathematics into the field of computer science took me through vast and diverse fields of discourse. After exploring multiple topics I settled in the area of computational linguistics. Through conversations with my advisor, Prof. John Goldsmith, I was inspired to pursue computational approaches to natural language. Simultaneously, following publications in the field of artificial intelligence, and through my interactions with colleagues at the Toyota Technological Institute, I became fascinated with the recent advances in deep neural networks. It became clear to me that I want to pursue a research path for my PhD, that combines those two interest. Since deep learning is a young and fast developing field, I decided to first pursue more applied projects involving NLP and deep learning. Through these projects I was able to learn the state of the art in neural networks research, and develop an array of practical skills necessary for my subsequent research goals in deep learning for morphology. This thesis presents two of the three research projects that I’m involved with. Both projects are works in progress, and I hope to pursue them in parallel, in addition to my research on deep learning for morphology. I plan to develop them into a series of papers in the future, as advances are made. Hence, I think of them as research themes. The common feature of my research is the design of neural architectures for natural language understanding with specific goals in mind. The natural language input is interpreted in the context of the task, and sometimes
- as in the case of my research in robotics (Chapter 2) - also external inputs such as environment observations.

The first theme, presented in Chapter 1, developed out of my internships at an education corporation in Tokyo. Japanese ministry of education has been investing into applications of artificial intelligence to education, and I was tasked to explore the possibility of advancing the state of theart in automated essay scoring using deep neural architectures. In the United States, Educational Testing Service (ETS) has lead the efforts to automate essay grading using a variety of techniques from NLP and AI, which resulted in the e-rater engine that automatically identifies features related to writing proficiency in student essays so they can be used for scoring and feedback of SAT and TOEFL examinations. However, the e-rator methodology is based on a mixture of heuristics, traditional AI, and statistical machine learning, which limits its applicability to settings outside of the exams it was designed for. Since, the development of the e-rator engine, a shockwave of discoveries - the deep learning revolution of recent years - has swept over the AI community. Initially deep learning systems based on convolutional neural networks conquered the field of vision, achieving state-of-the-art results on all major datasets and tasks. The past 3 years saw and increasing success of recurrent neural networks such as LSTM in natural language processing. In this project, I develop a deep neural network based automated essay scoring framework using an array of methods inspired by the latest advances in deep learning, and recurrent neural networks in particular. Chapter 1 contains necessary background, literature review, theory and implementation details. I introduce the reader to the relevant history of the problem and tools used to formulate a solution. The system I have designed, called the Deep Text Scoring system (DTS), will be trained on a new data set being collected at thousands of schools in Japan this summer, and deployed for the purpose of assisting human graders in scoring of Japanese college entrance examinations. I am returning to Tokyo for 3 months this summer for another internship, to complete the experiments, data collection, and testing of DTS in a real world setting.
Chapter 2 presents the second theme of my research focusing on robot intelligence through perception. In this particular project, I study human-machine interaction using natural language. Prior approaches to this problem required humans to express themselves in artificially constrained domain specific languages. Here we approach human-machine communication in analogy to communication between humans using different natural languages - i.e. we phrase it as an instance of translation. We develop a deep recurrent network, Long-Short Term Memory (LSTM) based architecture. This system interprets the natural language commands in the context of another sequence of perceptual inputs coming from environment observations. In this particular project we focus on direction following for an autonomous vehicle (e.g. a self-driving car) navigating the streets of downtown Chicago. We develop an agent-based path generator using the Google Street View API, which is used to produce a large dataset of routes containing geolocation data together with images and environment metadata, aligned with natural language instructions describing navigation directions given to the vehicle. This dataset will be used to train a Husky Unmanned Ground Vehicle to follow directions expressed in natural language while navigating the streets of downtown Chicago.

Although the projects presented here are focused on the applied side of NLP, I used them as an opportunity to learn theory and skills that will form the foundation of subsequent research projects in computational linguistics. I believe that there is unexplored potential for these and other newly emerged deep models to explore more fundamental problems at the intersection of linguistics and computer science. The third theme of my research will focus on deep learning methods for natural language problems. Most early efforts in this field are directed towards more practical NLP, and engineering problems (e.g. named entity recognition, vector embeddings of text, sentiment analysis, compression, text mining, question answering, discourse classification). I am especially interested in more fundamental questions in computational linguistics, such as morphological analysis for languages with interesting inflectional paradigms. I would like to bring state-of-the art deep neural network
models to linguistics research, and develop new architectures tailored to answering more fundamental questions about language. Simultaneously, I am very interested in the nature of deep neural networks themselves. One of the avenues I plan to explore is visualization and mathematical analysis of advanced recurrent architectures for language problems. I would like to use visualization and statistics to shed light on what those models are learning when trained on language data. For instance, in a recurrent neural network character-level language model, we can visualize groups of correlated neural firing patterns corresponding to reading certain morphological features of input text. I aim to develop deep neural architectures to extract various types of structure from raw corpora of text, and then analyze what these models are learning through statistics, mathematics and visualization, in order to answer fundamental questions about the structure of language. Can neural networks provide new insights into the nature of linguistic phenomena, above what can be extracted through classical machine learning approaches? This is the question I plan to explore together with my advisor in my PhD dissertation.
Chapter 1

Automated Essay Scoring

Abstract

Automated Essay Assessment is an important practical problem in the intersection of Linguistics and Computer Science. Apart from being one of the open problems in NLP research, it draws great interest from industry, especially in countries like Japan, where aging society has caused human graders to be increasingly expensive. There have been some attempts at developing automated scoring systems for natural language responses. However, they all rely on traditional methods, domain-specific heuristics, and hand-crafted features. It seems that scoring text (which can be posed as an instance of classification) is a natural application area for deep neural network models, which have been extremely successful in recent years, outperforming traditional state of the art models in all major NLP tasks. However, there are obstacles to successfully training such models. Scarcity of quality data necessitates development of methods that can use existing data wisely. In this project, we deal with this problem by developing a deep recurrent network model which can learn to focus attention on the words and phrases within a paragraph, that matter most for assessment. The model uses Gated Recurrent Units (a variant of LSTM-RNN), as part of a larger modular neural network, which can learn to emphasize some of the learned information and forget parts that seem irrelevant to grade assessment. This is a fully general end-to-end approach without any heuristics, or hand-crafted
CHAPTER 1. AUTOMATED ESSAY SCORING

features. The necessary abstractions are discovered by the model automatically. The only external information is a training set of human-graded essays on a given topic. Arguably, the most innovative part of our approach is the incorporation of Dynamic Memory Network approach. We pose essay grading as an instance of the more general question answering problem, and implement a deep architecture based on the idea of episodic memory from neuroscience. Using this approach we go beyond what has been done in the area of automated essay scoring. Our model is able to perform transitive reasoning over facts retrieved from the essay, unlike the previous approaches, which were only able to assess grammar, style, spelling, and other lower level measures. We believe that more abstract concepts such as whether the writer answers the questions posed in the thesis, and develops coherent arguments, are a big part of essay quality assessment. Furthermore, we augment the DMN architecture with a new type of RNN cell designed using deep reinforcement learning methods. This type of cell exhibits improved perplexity results on language modeling tasks, over prior state-of-the-art for LSTM variants. It brings improved performance to our model in terms of essay scoring accuracy. We compare our model to previous state-of-the-art systems, and interpret empirical results. We also discuss future research directions inspired by this project.

1.1 Motivation

I have been interested in machine learning applications for natural language processing tasks for a while now. Last year I’ve started studying neural networks and deep learning as part of a smaller self-organized reading group with my advisor John Goldsmith. Deep neural models were first used with outstanding results in computer vision problems. However, recently they have been applied to a variety of NLP tasks, where they outperformed former state-of-the-art methods. Deep learning in NLP is a quickly expanding new field, rich in opportunities for original research.

Last summer I worked on NLP applications at a Japanese company in Tokyo. During that time, I became aware of a challenging NLP problem, solution to which
would have important practical implications for countries like Japan. The task is to build a machine learning system which could accept English essays written by students, and predict their corresponding expected grades (on a predefined scale), that human graders would likely assign to those essays. Additionally, for pedagogical reasons, it would be useful if the system could provide some kind of feedback on what factors contributed to its decision. There is great interest in developing such systems in Japan, where society is aging, and it is becoming harder to hire graders. Japanese government is currently investing in developing e-learning tools, so students can study topics such as English by themselves through the use of online courses. However, there is a need to assess students’ progress, and hiring human graders for essays is becoming increasingly expensive due to shortage of teachers. Millions of Japanese students take a college entrance exam, which contains English module. One of the components of the exam is writing short written responses on a given topic. It is advantageous if AI techniques could predict a distribution over likely grades in a population of student essays, and flag only a portion of them to be graded by humans. The savings gained through using such systems would be significant, and hence it motivates academic research in this area. I started reviewing the relevant literature, and quickly discovered that adequate methods to solving this problem haven’t been developed. However, I realized that deep learning techniques have great potential to make progress on this task. This project is aimed to explore deep learning techniques with the aim of advancing automated grading of short written responses in English.

I motivate the choice of this research topic for several reasons.

• It is a challenging problem, which ties together many NLP tasks, and neural network techniques together. Therefore, working on it has pedagogical benefits, since I have to learn a wide range of topics.

• I plan to continue academic research in deep learning, natural language processing, and computational linguistics in the future. Focusing on a tangible,
well defined application area like scoring essays, is a great starting point to developing approaches and theory, which can be abstracted away and generalized for future research endeavors.

- Due to my experience in Japan, and industry need, it will be possible to obtain grants and RA support for my future research in this field. I have already discussed this possibility with contacts in Tokyo. Since I want to pursue deep learning approaches to NLP, it would be helpful if I could purchase equipment such as GPUs and establish a "neural computational linguistics lab", so other interested students can join the effort. I also plan to start a reading group on deep learning for NLP, and I hope it can grow into an active lab, as other students become interested in those topics.

- I have access to school system (with thousands of schools) in Japan, and can accumulate datasets of human graded essays on a given topic. Collecting data is extremely important for training deep neural architectures, hence this data gathering capability is a great advantage to pursuing research on this particular topic.

- Finally, deep learning for computational linguistics and natural language processing is a new fast-growing field with many opportunities for discovery. While it could be said that some other application domains, such as computer vision, started saturating and most of the low-hanging fruit has been picked, NLP seems to be the next area to be conquered by deep learning methods. It is therefore an opportune time to pursue this direction, as many famous deep learning researchers remark.

- Based on my research over the past months, and experiments with deep language models, I am convinced that the model I have been developing has the potential to greatly outperform the state-of-the-art in Automated Essay Assessment.
1.2 Introduction

In this project we design, implement, and test a neural automated essay scoring model. The innovative approach here goes beyond just using deep learning neural networks to perform the scoring. The architecture described here, is able to focus attention on parts of the essay that are most informative to assessing the grade. This allows the system to give useful feedback to students, by highlighting problematic parts of the text. We will implement all modules in Python using Google’s TensorFlow framework. Subsequently, we evaluate this model on a dataset containing thousands of human-graded written responses in English on several topics. We visualize and discuss results of those experiments, and compare them against our implementation of selected prior approaches. We interpret those results, and suggest intuition from cognitive science and neuroscience.

1.3 Literature Review

Automatic Text Scoring (also Automated Essay Assessment) refers to a body of techniques from natural language processing, computational linguistics, machine learning, and statistics, aimed at automatically inducing mappings from strings of words to subsets of $\mathbb{R}$. The most common application is predicting a grade that a student essay would receive, if it was graded by a human (i.e. finding a map from essays on some topic to a set of possible grades). In addition to this general goal of obtaining a numerical grade without human involvement, in settings such as foreign language education we would also like to obtain pedagogically useful feedback. The main industrial motivation is reducing the cost and speeding up student evaluation process. In addition to this economical incentive, automated grading systems promote assessment equity, since they ensure consistent application of scoring criteria and reduce the variance inherent in human grading.

Attempts to develop AES systems began in mid 1960’s [Pag66]. One of the earliest implemented essay scoring systems motivated by the prospect of reducing
labor-intensive grading activities was Project Essay Grade (PEG) by Ellis B. Page [PP68] [Pag68] [Pag67]. We are especially interested in the setting of foreign language acquisition, and in particular assessment of essays by Japanese learners of English. There has been significant work since the PEG system focusing on scoring text produced by non-native English learners [AB04] [RL02] [Ell03] [LLF03] [BMA10] [YBM11] [SHM15]. Several overviews summarize main ideas [WXB12] [Dik06] [HS].

The general approach taken so far in the field has been a combination of techniques ranging from statistical machine learning to clustering based on an array of hand-crafted, linguistically informed features, and a large collection of auxiliary meta-data, such as hand annotation, syntactic parse trees, sentiment analysis, etc. The high level definition of the problem of automated essay scoring was supervised classification based on those features.

This work draws on ideas from deep neural networks, computational linguistics and neuroscience to pursue a differing approach to AES. Instead of relying on heuristics, and human designed features, we use a deep learning approach to extract useful features automatically from data. Additionally we implement an attention focus mechanism, that can learn to emphasize inputs relevant to the task of text scoring. This attention mechanism can then be used to provide useful feedback, by highlighting parts of the essay that contributed the most to the grading decisions. For instance, if the essay received an unusually low grade, we will be able to highlight words or sentences that are most likely problematic, and caused the decreased assessment. Students can then focus their attention on those highlighted parts, and learn from their mistakes. In the following sections we review necessary elements of background knowledge, and introduce concepts upon which our system is built.
1.4 Computational Linguistics and Natural Language Processing

Computational Linguistics is the an interdisciplinary field, usually considered to be more on the side of linguistics, concerning the use of computational methods to study linguistic phenomena. Natural Language Processing is similar, but located more on the side of computer science and engineering. A typical task in CL might be, for instance, developing a computational model in order to probe properties of morphological paradigms in languages with interesting inflectional classes, or developing an unsupervised syntactic category induction algorithm for an unknown language. Typical task in NLP, on the other hand, would be implementing an n-gram language model for English, a named entity recognition algorithm, or sentiment analysis classifier. We use many core ideas and techniques from both fields in order to develop our essay scoring system. These are broad subjects, going beyond the scope of this document. However, the architecture we have invented presents a completely novel approach to the AES problem based on neural networks. Therefore, we will focus on the relevant deep learning techniques and ideas from neuroscience research, rather than classical NLP and CL approaches. However, many of the deep methods used in our model design are inspired by classical NLP algorithms. For instance deep vector space embeddings used during pre-processing in the input module to our system draw inspiration from n-gram language models. However, the bulk of our system is based on latest state-of-the art results in deep neural networks (some published just several months ago), and goes far beyond what classical CL and statistical machine learning methods provide. Therefore we first introduce relevant deep learning background necessary to understand the design choices made in our system, and the resulting essay scoring architecture.
1.5 Neural Networks and Deep Learning

1.5.1 Historical Perspective

Artificial neural networks are computation architectures inspired by models of human brain. They consist of a graph of interconnected units, called neurons. Each neuron takes inputs from a collection of other neurons in the graph, and performs some operation, then propagates output (called activation), which becomes input to other neurons, or even back to itself (in case of recurrent networks).

Research into artificial neural networks began in the early 1940’s. The first model, called threshold logic, was developed by computational neuroscientists Warren McCulloch and Walter Pitts in 1943 [MP43]. The first ideas for neural learning mechanisms were published by a psychologist Donald Hebb in 1949 [Heb49]. Early ideas here go back to 18th century, since primitive supervised neural nets were essentially variants on the linear regression theory of Gauss and Legendre [Gau09] [Gau21] [Leg05]. Many design choices in artificial neural networks were inspired by neuroscience research. Early neural networks were based on two types of cells found in cat’s visual cortex [HW59] [HW62]. The first type of neuron, called simple cell, inspired sigmoid neurons. The second type, called complex cell, can be found in the primary visual cortex (V1), the secondary visual cortex (V2), and the Brodmann area 19 (V3). Complex cells were an early inspiration for deep convolutional neural networks, which led to deep learning revolution due to their enormous success in the field of computer vision. First single hidden layer perceptron networks were developed in 1960’s [Jos61] [Vig70]. Deep multilayer networks of perceptrons were developed in the same decade [Iva68] [Iva71] [IL65] [IL67]. First architectures that truly deserve the attribute deep were developed in late 1970’s and early 1980’s. Initial approaches focused on image recognition and introduced convnets [Fuk79] [Fuk80]. Those first approaches didn’t use supervised gradient descent training, but instead relied on had-engineering and some unsupervised techniques. Optimization of neural networks through gradient descent [Had08] has been discussed since 1960’s
Those first implementations of gradient descent were inefficient, and didn’t handle sparsity, recurrent connections, or links between nonsuccessive layers. Somewhat surprisingly, a 1969 book [Min69] on limitations of simple linear perceptrons with a single layer had a profound influence on the academic community, and discouraged researchers from further studying neural networks. Modern gradient descent optimization ideas for neural networks reemerged in 1980’s [LeC85] [Wer82]. In 1986 Hinton published an influential paper [RHW85], which popularized an efficient back-propagation algorithm for training of deep neural networks. It also showed empirical results suggesting the emergence of useful internal representations in hidden layers of deep architectures. This was the beginning of a wave of discoveries that formed a prelude to the modern deep learning revolution of recent years. Recent breakthroughs include long short-term memory recurrent neural networks, attention mechanisms, dynamic memory networks, neural Turing machines and deep reinforcement learning.

### 1.5.2 Perceptrons

Advances in neural networks started with the perceptron algorithm. It is a method of learning a binary classifier from data. The perceptron model takes a real-valued input vector $x$ and applies a linear activation neuron $a(x) = \sum_i w_i x_i + b$ where the weight vector $w$ and bias $b$ are to be learned from the training set. The result is then thresholded to obtain a decision boundary:

$$f(x) = \begin{cases} 1 & \text{if } \sum_i w_i x_i + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

The basic perceptron algorithm is guaranteed to converge if the training set is linearly separable. However, there are infinitely many possible solutions (fig. 1.1). A more advanced version of perceptrons called support vector machines [SB99] [Vap13] learn solutions with the widest margin.
Figure 1.1: perceptron solutions aren’t unique
1.5.3 Nonlinear Activation Neurons

Linear perceptrons are not general enough to learn most useful functions. The introduction of nonlinear activation functions (also called transfer functions) lead to renewal of interest in neural networks due to a result known as Universal Approximation Theorem, first proved by George Cybenko for sigmoid activation neurons [Cyb89]. The result implies that deep neural networks of sigmoid (and other types) neurons can learn to approximate continuous functions on compact subsets of \( \mathbb{R}^n \) with arbitrary precision.

A sigmoid neuron differs from the perceptron described above in that it applies a nonlinear activation function to the linear combination of its inputs:

\[
f(x) = \sigma(\sum_i w_i x_i + b)
\]

The range of the neuron’s output is the open interval \((0, 1)\) (fig. 1.2). This neuron type has biological inspiration. The output of the neuron can be thought of as "firing rate". Biological neurons fire electrical signals at varying rates per unit of time. Sigmoid model is supposed to imitate this behavior. Another important property is that the output is now a differentiable function of the inputs, and the derivative of a sigmoid function has nice algebraic form \( \frac{\partial}{\partial x} \sigma(x) = \sigma(x)(1 - \sigma(x)) \).

Figure 1.3 shows a diagram of a biological neuron, and figure 1.4 shows a diagram of a sigmoid neuron used in artificial neural networks. Dendrites correspond to inputs, cell body corresponds to the activation function, and terminal bulb propagates the output to the other neurons in the network.

Sigmoid neurons are one of top three most commonly used neuron types in modern neural architectures. The other two common activations are rectifier (ReLU) and hyperbolic tangent (Tanh). All these neurons satisfy the requirements of Universal Approximation, which means they can be used interchangeably.
Figure 1.2: logistic activation function $\sigma(x) = \frac{1}{1+e^{-x}}$

Figure 1.3: diagram of a biological neuron
1.5. NEURAL NETWORKS AND DEEP LEARNING

Figure 1.4: diagram of an artificial neuron

\[
\text{ReLU}(x) = \begin{cases} 
0 & \text{for } x < 0 \\
 x & \text{for } x \geq 0
\end{cases}
\]

\[
\text{Tanh}(x) = \frac{2}{1 - e^{-2x}} - 1
\]

1.5.4 Learning

Learning in neural networks generally refers to episodic adjustments in weights associated with the network’s edges, leading to incremental improvements in various measures of performance, called loss functions. Early efforts to develop learning algorithms for neural networks were inspired by neuroscience research pioneered by Donald Hebb. The Hebbian Theory introduced in his 1949 book titled "The Organization of Behavior - A Neuro-psychological Theory" [Heb49] tried to explain the basic mechanisms for synaptic plasticity (the adaptation of neurons in the brain during the learning process). It was later used as basis for learning rules in Hopfield networks [Hop82]. Key advance in learning algorithms for neural networks was the introduction of back-propagation algorithm [WH86]. This approach is used in
modern neural network research, and in this project. We will review the basics of back-propagation learning below.

Suppose we have a neural network \( \mathcal{N} \), which computes some function \( f : X \to Y \). Given a set (called training set) \( S \subset X \times Y \) of correct input-output pairs, we would like to adjust the weights of \( \mathcal{N} \) in order to better match the training set. In order to define the meaning of "better" we first introduce a measure of network’s performance, called a loss function (sometimes the term error function is also used) \( L^f : \mathcal{P}(X \times Y) \to \mathbb{R} \). In order for back-propagation to work, two basic assumptions about \( L \) have to be noted. First, it should decompose as \( L^f(S) = \frac{1}{n} \sum_x L^f_x \), where each \( L^f_x \) represents loss on a single training example \( (x, y) \in S \), and \( n = |S| \). Second, it should be a function of the outputs from \( \mathcal{N} \). The loss function is used to compute how far the network’s predictions, for inputs in the training set, differ from the correct answers. A standard choice is quadratic loss

\[
L^f(S) = \frac{1}{2n} \sum_x (y - \hat{y})^2
\]

where \( \hat{y} \) denotes the network’s prediction (i.e. \( f(x) \)) for the input example \( (x, y) \).

Other types of loss can be motivated by statistical or information theoretical considerations (e.g. we could compute KL-divergence between the distribution of the network’s outputs and the training examples). The back-propagation algorithm optimizes loss on the training set via stochastic gradient descent.

Here, we will derive the back-propagation update rules using a simple neural network model shown in figure 1.5. In the diagram of figure 1.5, squares denote input neurons (hence no activation function is applied there), and circles denote sigmoid neurons. Neurons are arranged in layers, which are denoted by named boxes I, J, K. We will use the following notation

- \( x^l_j \): input to node \( j \) of layer \( l \)

- \( W^l_{ij} \): weight from layer \( l - 1 \) node \( i \) to layer \( l \) node \( j \)
Figure 1.5: A simple neural network

- $\sigma(x) = \frac{1}{1+e^{-x}}$: sigmoid transfer function

- $\theta^l_j$: bias of node $j$ in layer $l$

- $O^l_j$: output of node $j$ in layer $l$

- $\tau_j$: target value of node $j$ in layer $l$

- $E = \frac{1}{2} \sum_{k \in K} (O_k - \tau_k)^2$: loss

In order to derive the back-propagation algorithm, we need to compute the derivative of our loss with respect to weights and biases of each node in the network. We split our computation into two cases.

First let us consider the output layer (K).
CHAPTER 1. AUTOMATED ESSAY SCORING

\[ \frac{\partial E}{\partial W_{jk}} = \frac{\partial}{\partial W_{jk}} \frac{1}{2} \sum_{k \in K} (O_k - \tau_k)^2 \]

\[ = (O_k - \tau_k) \frac{\partial}{\partial W_{jk}} O_k \]

\[ = (O_k - \tau_k) \sigma(x_k)(1 - \sigma(x_k)) \frac{\partial}{\partial W_{jk}} x_k \]

\[ = (O_k - \tau_k) O_k (1 - O_k) O_j \]

\[ = O_j \delta_k \]

where we defined \( \delta_k = (O_k - \tau_k) O_k (1 - O_k) \) which is basically the difference between the example and the prediction, scaled by the derivative of the transfer function.

Similarly, we compute the gradient of the loss function for the hidden layer (J).

\[ \frac{\partial E}{\partial W_{ij}} = \frac{\partial}{\partial W_{ij}} \frac{1}{2} \sum_{k \in K} (O_k - \tau_k)^2 \]

\[ = \sum_{k \in K} (O_k - \tau_k) \frac{\partial}{\partial W_{ij}} O_k \]

\[ = \sum_{k \in K} (O_k - \tau_k) \sigma(x_k)(1 - \sigma(x_k)) \frac{\partial}{\partial W_{ij}} x_k \]

\[ = \sum_{k \in K} (O_k - \tau_k) O_k (1 - O_k) \frac{\partial x_k}{\partial O_j} \frac{\partial O_j}{\partial W_{ij}} \]

\[ = \frac{\partial O_j}{\partial W_{ij}} \sum_{k \in K} (O_k - \tau_k) O_k (1 - O_k) W_{jk} \]

\[ = O_j (1 - O_j) \frac{\partial x_j}{\partial W_{ij}} \sum_{k \in K} (O_k - \tau_k) O_k (1 - O_k) W_{jk} \]

\[ = O_j (1 - O_j) \sum_{k \in K} (O_k - \tau_k) O_k (1 - O_k) W_{jk} \]

\[ = O_j \sum_{k \in K} \delta_k W_{jk} \]

\[ = O_j \phi_j \]

where we defined \( \phi_j = O_j (1 - O_j) \sum_{k \in K} \delta_k W_{jk} \), which can be thought of as a
weighted error at the output scaled by the derivative of the transfer function.

Because we can think of bias terms as connecting inputs always equal to 1, it is easy to verify that:

$$\frac{\partial E}{\partial \theta_l} = \begin{cases} \delta_l & \text{for } l \in K \\ \phi_l & \text{for } l \in J \end{cases}$$

The above derivation leads to the following algorithm for training the network from data:

- run the network forward on the inputs from the training set to compute its predictions
- for each output node compute $\delta_k = (O_k - \tau_k)O_k(1 - O_k)$
- for each hidden node compute $\phi_j = O_j(1 - O_j) \sum_{k \in K} \delta_k W_{jk}$
- compute $\Delta_l = \begin{cases} \delta_l & \text{for } l \in K \\ \phi_l & \text{for } l \in J \end{cases}$
- update:
  $$W \leftarrow W_{ij} - \eta O_i \Delta_j$$
  $$\theta_l \leftarrow \theta_l - \eta \Delta_l$$

We normally run the algorithm on smaller batches of examples instead of the entire training set, in what is called episodes. After repeating the training for a number of episodes we save the resulting weights, and then we can use the network to make predictions outside of the training set. There are various criteria we can use to determine when to stop. A popular choice is to stop when the change in loss between episodes slows down to some value. The learning rate, batch size, stopping criteria, are all examples of hyper-parameters. These aren’t learned but are rather
usually set by experts using some industry heuristics. There is a significant research effort aimed at automating hyper-parameter selection, but in general it is a hard problem, and good hyperparameter choices are part of the art of deep learning.

We have derived a basic version of stochastic gradient descent for optimization of loss function in neural networks known as back-propagation learning algorithm. There are many modifications to this basic approach used in modern deep learning research. Some of them are inspired by physics to simulate momentum of a rolling ball on the energy landscape defined by the loss function, and require computing higher order derivatives of the error. We will not go into detail on those extensions here, but a good overview is given in [Rud16].

1.5.5 Deep Neural Networks

Deep learning refers to a field in neural network research concerning the properties of complex, multi-layered, modular neural network architectures. Below, we will discuss several types of deep learning approaches that are incorporated into our AES system, such as ConvNet, LSTM-RNN, GRU, and DQN. A major contribution of deep neural architectures has been their ability to analyze data by inducing hierarchical representations, based on automatically extracted features. Higher level features are derived from lower level ones by the neural network when the information flows through consecutive layers (often entire modules composed of other deep networks). This leads to multiple levels of abstraction which are used to build up high-level concepts. Deep learning algorithms transform input data into distributed representations, where consecutive factors correspond to progressively higher levels of abstraction. Techniques used in deep learning are often inspired by neuroscience research. One example is neural coding - a study of the relationship between neuronal responses in the brain to various stimuli [O+96]. For instance, an image can be represented as a vector of pixel intensity values at the input layer, then gradually processed into a set of edges, regions of various shapes, and even topological properties. In classical machine learning approaches to such problems, e.g. a Viola-Jones
type object detection framework [VJ01], features are hand-selected (Haar feature set in this case), and the algorithm uses statistical theory to derive the best classifier. In contrast, a deep convolutional neural network would extract a hierarchical set of progressively more abstract representations of images automatically. Neural networks can be used just for this purpose alone.

A type of neural network known as auto-encoder does exactly that [HZ94] [Ben09]. It is trained to reproduce its own inputs as closely as possible. After it has been trained to a desired level, we can use the information stored in the hidden units as a compressed representation of the training data set. Deep learning is a fast growing field with new discoveries made daily. Because it is a recent area of discourse that has not been properly organized yet, and often lacks theoretical foundations, it is hard to give a single comprehensive reference for state-of-the-art knowledge. However, for the same reasons, it is a challenging and exciting topic to investigate.

There are many problems in deep learning, on which constant progress has been made over the past decade, and many new open problems that are yet to be conquered. Some of the main research topics currently pursued include:

- **Unstable gradients:** Due to the multiplicative nature of the chain rule for differentiation, which back-propagation algorithm implements, gradients often blow up or vanish for deep multi-layered networks. Much research is devoted to countering this problem. For instance, GRUs and LSTMs were developed to control the magnitude of the gradients during learning.

- **Generalization:** Is there a self-regularization effect induced by back-propagation in deep networks? Why do they generalize so well even though the number of parameters is often larger than number of training examples?

- **Regularization:** This refers to a range of methods in machine learning aiming at penalizing model complexity, in order to reduce overfitting. There are various approaches to regularization in deep learning, but most are based on heuristics and lack rigorous mathematical explanation. Why does regularization work in
neural networks? What are the best approaches to regularization and why?

- **Neuro-evolution:** Recently efforts have been directed at combining deep learning with genetic programming and other evolutionary approaches. These are optimization methods inspired by the mechanisms of biological evolution such as reproduction, mutation, recombination, and selection. Two main themes are evolving neural network topologies, and hyper-parameter optimization.

- **Big Data:** What are the types of asymptotic behavior of neural networks in the limit of large data sets?

- **Neurons:** What types of neurons are best for learning in different applications? For example, why do tanh neurons sometimes outperform sigmoid neurons?

- **Topological Regularization:** What is the statistical meaning of dropout and other approaches to regularization by selectively altering the topology of neural networks during learning? Dropout is a successful technique in deep learning that improves model robustness while reducing complexity. This is done by randomly disabling a fraction (usually half) of the neurons in the network during learning episodes. Because neurons cannot rely on particular neurons being present at all times, the network is forced to learn in a way that is less affected by any particular neuron. Hence, it decreases the probability that some weights become very large, because that would magnify output from those neurons, and likely make them more important.

- **Memory:** How to allow external memory storage in a way that is learnable using back-propagation? Neural Turing Machines and Memory Networks are some of the recent successful approaches to this problem. This research direction goes beyond what we’ve covered here, and generalizes neural networks to Turing-complete systems, which are able to learn algorithms from data.
• Attention: How to automatically distribute and focus attention on inputs or features most relevant for efficient learning? How to learn such attention distribution using back-propagation?

• Generative models: Deep Belief Networks are an example of a generative approach to neural networks. This topic was pursued initially, but then academic community shifted towards ConvNets instead. Recently it is gaining attention again as a promising approach to deep learning problems in some areas. Using RNNs in generative models is one avenue explored here.

• Recurrence: Various extensions, and novel approaches to recurrent neural networks have been proposed recently. LSTM-RNN, Grid-LSTM, GRUs, and many other are successful instances of recurrent architectures.

• Foundations: Efforts in foundations of deep learning include mathematical/statistical basis for their performance, and category theoretical/topological frameworks.

• Adverse behaviors: Many intriguing issues have been noted with neural networks, that aren’t fully understood. For instance, the existence of "adversarial images" in convolutional neural network research suggests that functions learned aren’t smooth or nearly noncontinuous in some cases. What is causing such discontinuities? Is it the cost functions? Interaction between layers? Activation functions? Regularization?

• Deep learning frameworks and architectures: New developments such as Generative Adversarial Nets, question answering frameworks, and architectural innovations such as DenseNet are achieving state of the art results while bringing insight into data. Much research is devoted into merging what we know, and developing broad frameworks and architectures that can solve a wide variety of machine learning problems.

I am currently interested in deep learning approaches to language problems,
therefore I will introduce some of the deep learning concepts which I believe have potential for my future research in this area, and are particularly relevant to this project.

1.5.6 Neural Vector Space Embeddings of Words

Our input layer pre-processes the essay by using a deep feed forward neural network, to embed words into a high dimensional vector space, where cosine similarity provides a measure of semantic and syntactic closeness. Furthermore, the distributional vector representation is compositional in nature, so the group operation of vector addition produces linguistically meaningful results. "Word embeddings" are a family of natural language processing techniques aiming at mapping semantic meaning into a geometric space. In the broadest sense it is a dimensionality reduction technique aimed at extracting linguistically meaningful features from raw text. Furthermore we require the embedding to have at least two properties:

- Some measure of distance (e.g. L2 or cosine) should correspond to linguistic similarity (semantic, syntactic, etc).

- Compositionality should have linguistic meaning. For instance, the path from "lecture" to "classroom" (i.e. the vector difference "classroom" - "lecture") should correspond to a vector in a neighborhood of words such as "location", and express the meaning of "where something occurs". Hence, if we call that difference vector v, we should have that "reading" + v is close to "library".

Multiple methods of computing distributed vector representations of words have been studied. The two most prominent examples are GloVe (Global Vectors for Word Representation) [PSM14] and Word2Vec [MSC+13]. GloVe is based on factorizations of the matrix of word cooccurrence statistics, and Word2Vec is based on a deep neural network encoding. We use Word2Vec embedding for input representation in our text scoring architecture.
In the simplest instantiation of the method, we have a single hidden layer feed-forward neural network with $V$ input neurons and the same number of output neurons. The number $V$ is set to the size of the lexicon. The hidden layer is meant to provide a lower dimensional explanation of the data, with $N \ll V$ neurons. There is a complete bipartite connection between successive layers (Figure 1.6).

![Figure 1.6: neural network for computing vector embeddings (bigram model version)](image)

The input is a sparse $V$-dimensional vector $x \in \mathbb{Z}_2^V$ with $\sum_i x_i = 1$ (i.e. there is a single 1 and all 0s in the input). The input vector corresponds to the words in the lexicon in some ordering, and the single 1 marks the word that is the current input to the network. The network in Figure 1.6 represents a bigram model. That is, the network reads the first word (marked by the 1 in the input) and tries to predict the next word in the bigram (marked by the maximum $y_i$ value). It is straightforward to extend this to a general k-gram model by adding extra sets of $V$ neurons to the input layer. We then train the network using back-propagation gradient descent algorithm on a raw corpus (e.g. Brown corpus). For instance, in a bigram model, we would generate training examples from a raw textual corpus by first generating all the bigrams (by sliding a window of size 2 through every position in the corpus). Subsequently, we feed (0,1)-vector formatted representations of the bigrams to the
network in form of training pairs \((x, y)\), and back-propagate the error adjusting the weights until some number of epochs.

After training is complete, we obtain a vector embedding of the \(i\)th word by reading all the paths from that word’s corresponding input neuron \(x_i\) to its output neuron \(y_i\). This gives us \(2N\) real numbers (the weights on the edges along the path through the hidden layer). We can either combine them in some way to get a \(\mathbb{R}^N\) embedding, or keep them separate and simply concatenate the weights for a full \(\mathbb{R}^{2N}\) embedding.

Such embeddings have many interesting properties. First, distance between vectors in the embedding space corresponds to syntactic and semantic similarity between the words they represent. This is because words with similar properties will appear in similar contexts, which means the words that can follow them are similar. Furthermore, the lower-dimensional representation described by the \(N\) weights allows for linguistically meaningful linear algebra operations. For example in our experiments with a neural network trained using Google News corpus, the word corresponding to the vector that is nearest to the vector sum \(\alpha(’bigger’) - \alpha(’big’) + \alpha(’cold’)\) was found to be the word ’colder’ (here \(\alpha\) denotes the word embedding computed by the neural network). Many other examples such as king - male + female = queen are famous in literature. We also noticed interesting morphological phenomena. For example, taking several -ed forms of verbs, adding them together, and subtracting their null forms creates a vector that can generate new ed forms from null forms of verbs. Adding this vector to new verbs in null form finds their corresponding past forms. In future research, we are interested in exploring linear subspaces of the embedding space that would allow decomposing words using linguistically meaningful basis. For example, finding a subspace corresponding to -ed inflection of verbs would allow projecting verb vectors onto that subspace to locate their -ed forms in the corpus.

This preprocessing step in our system is based on [MSC+13], [MCCD13], and [PSM14]. The reason we chose this embedding method, is because there is evidence
in literature that it provides a universal feature extractor for NLP tasks [SRASC14]. We only use it to represent the input as feature vectors in the first pre-processing step. Subsequent steps extract more task specific features, and take sequence ordering into account. Those higher order steps are based on two important ideas from deep learning: convolution and recurrence. These are the subjects of the following sections.

1.5.7 Convolutional Architectures

One of the most important deep architectures is known as Convolutional Neural Networks (ConvNet). It gets its name from a mathematical operation called convolution. I like to think of convolutions in terms of probability density functions of independent events. Under this interpretation, given two PDSs $f_A$ and $f_B$, describing distributions of independent random variables A and B, the PDF of their sum $C = A + B$, is given by the convolution of their $f_A$ and $g$.

$$f_C(x) = \int_{-\infty}^{\infty} f_A(y) f_B(x - y) dy = (f_A * f_B)(x)$$

Another good grounding for this concept comes from image processing. A common way to blur an image is to slide a localized function, called a kernel, through the image, and use it to compute the mean of the image under that kernel. We sum the values of the image pixels (first function) multiplied by values of the kernel at various positions, which is equivalent to convolving the image with the kernel. This method is common in statistics, e.g. non-parametric regression. In general a convolution of two functions $f$ and $g$ over domain $(D)$ is given by:

$$(f * g)(x) = \int_{a+b=x|a \in D} f(a)g(b) da$$

or in discrete case:
\[(f \ast g)(x) = \sum_{a+b=x|z \in \mathcal{D}} f(a)g(b)\]

The discrete case has also a nice interpretation of computing a probability of a sum of two independent random variables taking the value \(x\).

As we will see soon, we can think of convolutional neural networks as applying the discrete type of convolution to the input layer. First we need to introduce the CNN architecture. Normally consecutive pairs of layers in a feed-forward neural network would form a complete bipartite graph. In applications such as image processing, networks are quite deep and consist of large numbers of neurons per layer. For instance, we would often have a separate input neuron for each pixel in the image. This leads to a very large number of distinct weight parameters we need to train from data. Convolutional architectures counter that by applying local filters at each layer via convolution. This is implemented by tying of weights in groups. So we only need to learn the tied groups of weights (features). For example, we could have a window of 9 input neurons (3x3 pixel square in the input image) connected to a single neuron in the first hidden layer (one per a group of four input neurons), with no other cross connections. So every neuron in the second layer would only communicate with (i.e. receive input from) four distinct neurons in the input layer. We end up in linear number of distinct weights, instead of polynomial in a standard feed-forward network. It also reduces learning complexity, because we only need to learn a smaller number of distinct connections (features), which are applied to different sets of inputs (e.g. different parts of an image). We can repeat the convolution multiple times. Hence, we again feed fixed groups of neurons from a hidden layer into the next layer, and repeat the process. This leads to hierarchical feature extraction, and has been shown to model increasing levels of abstraction [Ben09]. We often include several special types of convolutional layers, performing various statistical summaries of the features extracted so far. For example a max pooling layer is very popular in computer vision. It simply fires if a certain feature is present in some region of a previous layer. This can be thought of as zooming
out, because we get a coarser view of the inputs, which is more robust under local noise. Figure 1.7 shows a diagram of a simple 1D convolutional neural network with a two neurons per window. Figure 1.8 shows a more realistic example of a 2D layer diagram with 4 neurons per window. Convolutional layers can get more complicated, and gates (like the pooling gate in our example) can even be entire neural networks with their own architectures [LCY13].

Geoff Hinton et al. [KSH12] introduced a wide array of new neural network techniques (including convolutional architecture, regularizaton via dropout, ReLU neurons, and the use of GPUs to train networks on large data sets) in their seminal paper on image classification. Their architecture (pictured in figure 1.9) is quite complex.

Using this architecture, Hinton et al. were able to automatically extract a number of features for image recognition. Some of them are visualized in figure 1.10.

The features learned include edges in various orientations (top of figure 1.10), as well as textures and color combinations (bottom of figure 1.10). Nobody designed those features by hand (as natural and useful as they appear for image processing). They emerged automatically as a result of back-propagation learning, from initially
CHAPTER 1. AUTOMATED ESSAY SCORING

Figure 1.8: CNN with max pooling for image processing

Figure 1.9: Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton - ImageNet classification
random weights. Representing images using such features instead of individual pixels reduces dimensionality of the problem, and makes the classifier robust to local noise.

Convolutional neural networks have been applied successfully to problems such as search query retrieval [SHG+14], semantic parsing [YHM14], sentence modeling [KGB14] as well as a variety of traditional NLP tasks [CWB+11]. Inspired by this, we will use convolutions to extract features from raw text. We will then use representations of essays in terms of such abstractions to process it through regularized recurrent neural network, which is the topic of the next section.

1.5.8 Recurrent Architectures

The deep neural network architectures we have studied so far, all share one major shortcoming. They have no lasting memory of past inputs, other than the general knowledge they’ve extracted from input-output pairs through back-propagation gradient descent. Recurrent neural networks address this problem. They allow information about previous inputs to persist, by feeding it back to themselves via recurrent connections. Because of this, they no longer form a simple acyclic graph like in previous examples, and their analysis is more challenging. We can conceptualize what’s happening by unwinding the recurrent connections into an infinite graph formed of copies of our network (as in figure 1.11).
Remembering previous inputs is especially useful if they form coherent sequences such as strings of words in a sentence. Therefore, RNNs are particularly useful in natural language processing. In 1991 Jeffrey L. Elman from the departments of Cognitive Science and Linguistics at UCSD published a seminal paper on distributed representations of text obtained with a simple recurrent neural network architecture. His paper showed that recurrent neural networks can learn long term dependencies in sequences of words (e.g. essays for our purpose), and extract deep linguistic features of their input [Elm91]. Elman’s work was a first step in a sequence of important discoveries, most prominent of which include long short-term memory networks and gated feedback recurrent units [SH97] [CGCB15]. We are going to use a modern version of recurrent neural network architecture to process words in input essays.

In the context of text processing, a recurrent neural network takes as input a sequence of words in some vector representation $w_t$, where $t$ is the index of a word in a sentence. Because words are fed into the network successively, $t$ is often called the time factor. Let $f$ denote the RNN cell, which can be a complicated function (such as an LSTM unit). In a single step at time $t$, an RNN cell takes the current input, and the previous hidden state $h_{t-1}$, and calculates the new hidden state $h_t$, which will be combined with the next input (consecutive word embedding in the text). Hence, for a single layer we have:

$$ h_t = f(W \cdot h_{t-1} + V \cdot w_t) $$

for some learnable matrices $W$ and $V$. In order to obtain a vector embedding
of the entire input sequence, we just take the final hidden vector $h_T$. In case of multiple layers, in order to compute the hidden state in layer $l$ at time $t$, we use the hidden state from the previous layer $l-1$ at time $t$ as input, and combine it with the current layer’s hidden state from the previous time step:

$$h_{t,l} = f(W \cdot h_{t-1,l} + V \cdot h_{t,l-1})$$

For the initial input we use the word embeddings ($h_{t,0} = w_t$) as in the single layer case.

There is also a case of bi-directional RNN. In this model we read the input words in both directions, and obtain two sets of hidden states:

$$\vec{h}_t = f(\vec{W} \cdot \vec{h}_{t-1} + \vec{V} \cdot w_t)$$
$$\overrightarrow{h}_t = f(\overrightarrow{W} \cdot \overrightarrow{h}_{t-1} + \overrightarrow{V} \cdot w_t)$$

We then use the concatenation of the two final directional state vectors, $[\vec{h}_1, \overrightarrow{h}_T]$, as the embedding.

Theoretically a plain version of RNN can solve problems with arbitrary temporal dependencies within inputs. However, in practice, especially if the gap between relevant information is large, it is very hard to train them to sufficient accuracy [BSF94]. This is related to the unstable gradient problem, which is magnified in case of recurrent networks. Long short term memory recurrent neural networks (LSTM) [SH97] and gated recurrent units (GRU) [CGCB15] (which is a special case of an LSTM) solve the unstable gradient problem in recurrent neural networks. LSTMs are composed of recurring cells that are able to retain important information and selectively forget some of the previous inputs. This is done through an internal circuit, with several types of gates. An input gate will add new relevant information into the state of the network, and a forget gate will filter out some information. The hidden cell state is passed on via recurrent connection and combined with input.
This can be described formally by a set of equations:

\[ f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \]
\[ i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \]
\[ \tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \]
\[ C_t = f_t \ast C_{t-1} + i_t \ast \tilde{C}_t \]

The cell state at previous time \( C_{t-1} \) is combined with current input \( x_t \) and previous output \( h_{t-1} \), filtered via the input and forget gates, to form the new cell state \( C_t \). The cell body together with recurrent connection can be unrolled to visualize the update as an infinite circuit, as in figure 1.5.8.

Since the original paper introducing the idea of long short term memory [SH97], several modifications have been proposed. We will make use of one of them here. The gated recurrent unit (GRU) [CGCB14] combines input and output gates into a single "update gate", as well as merges the cell and hidden state, among other augmentations:
1.5. NEURAL NETWORKS AND DEEP LEARNING

Figure 1.13: Gated Feedback Recurrent Unit

\[
\begin{align*}
    z_t &= \sigma(W_z \cdot [h_{t-1}, x_t]) \\
    r_t &= \sigma(W_r \cdot [h_{t-1}, x_t]) \\
    \tilde{h}_t &= \tanh(W \cdot [r_t \ast h_{t-1}, x_t]) \\
    h_t &= (1 - z_t) \ast h_{t-1} + z_t \ast \tilde{h}_t
\end{align*}
\]

Extensive empirical comparisons of these and many more recurrent neural network architectures (see [GSK+15] and [Zar15]) reveal that they are all comparable in performance, and GRU is one of the best while maintaining relative simplicity and intuitiveness. Because of that, it has become increasingly popular since its publication. We will make use of GRUs to process words of the input essays in our scoring engine.
1.5.9 Deep Reinforcement Learning

Reinforcement learning is a subfield of machine learning focusing on techniques inspired by behaviorist psychology, concerning agents acting in environments with a goal of maximizing a notion of cumulative reward. It is particularly applicable in robotics, game theory, control theory, evolutionary computation, and economics. Reinforcement learning is usually studied outside and in parallel to the main branches in machine learning, because it doesn’t exactly fit in the supervised/unsupervised classification of machine learning algorithms. Sub-optimal actions are not explicitly corrected, and correct input-output pairs are not available to the algorithm. However, there is a notion of supervision through the reward function. The agent might create its own training examples by trying things while exploring the environment, but the reward might come after a long sequence of actions. There are some unique issues in theory of reinforcement learning, such as the exploration vs. exploitation problem, which has been studied via Markov Decision Processes and Multi-Armed Bandit formulations in statistics. Two of the most studied algorithms for rein-

```python
def rnn(cell, input_list, initial_state):
    state = initial_state
    outputs = []
    for i, input in enumerate(input_list):
        output, state = cell(input, state)
        outputs.append(output)
    return (outputs, state)
```

Figure 1.14: Simple RNN in Python. Cell in our case is a GRU or NAS-cell discussed later.
forcement learning are Q-learning and policy search (in particular policy gradients). Deep reinforcement learning refers to reinforcement learning in deep neural networks. Many of the deep reinforcement learning techniques are adaptations of those classical algorithms to take advantage of optimization through back-propagation. For instance, the now famous paper from Google Deep Mind on learning human-level control for ATARI video games used a neural network adaptation of Q-learning. Similarly, the seminal result from Deep Mind on playing the game of Go (beating human champions, including Lee Sedol from South Korea, who held the world championship at the time) uses neural network version of policy gradients (PG) with Monte Carlo Tree Search (MCTS). We briefly discuss policy gradients below, because our essay scoring architecture uses a modification to the LSTM obtained by deep reinforcement learning. The main difference between reinforcement learning with policy gradients vs. regular supervised learning, is that we don’t have access to labels for each examples. Instead we assume temporary labels, which are drawn from a probabilistic policy. The goal is then to maximize the probability of examples that lead to success and decrease the probability of those that do not. Success can be understood as something that happens after many examples have been generated using the current policy. For example, in a game like chess, we might not know if the decisions that we make now lead to a success until much later in the game. We then penalize all decisions that led to a loss, and reward all decisions that lead to a win.

Describing reinforcement learning algorithms in detail is beyond the scope of this project document, as it plays a rather minor role in our design. We use it only to improve essay scoring performance of our system, by introducing a new type of RNN cell designed by means of deep reinforcement learning methods. This cell will allow our model to develop a better understanding of the English language, because experimental results show that it is able to facilitate higher perplexity language models than commonly used LSTM cells.
CHAPTER 1. AUTOMATED ESSAY SCORING

NAS-cell

Recently a new type of RNN cell was developed automatically through a technique known as Neural Architecture Search [ZL16] (figure 1.15). NAS uses a recurrent neural network to generate descriptions of neural network architectures, and that RNN is then trained using deep reinforcement learning techniques to improve performance of generated networks. This result is part of a recently growing research direction in deep learning focused on developing methods to automatically generate solutions to engineering problems. Another prominent example in this theme is the Neural Programmer [NLS15] [RDF15].

The best performing cell generated using NAS beats the LSTM type cells by a significant margin in multiple natural language processing tasks on benchmark datasets, and achieves state-of-the-art perplexity scores for language modeling on Penn Treebank. The PTB perplexity on test set for the NAS cell is 62.4, which is 3.6 perplexity better than previous state-of-the-art. The NAS-cell also achieved new record results in character-level language modeling on PTB (perplexity score of 1.214).

We will use the NAS-cell to obtain the final memory vector representation of each essay within a Dynamic Memory Network architecture, before passing it to the scoring module. DMN consists of RNN cells with an attention distribution mechanism, which will be discussed in later sections. The final encoding layer of RNN cells would normally be composed of Gated Feedback Recurrent Units. We use NAS-cells for the DMN encoding task.

1.5.10 Memory and Attention

Motivation

Humans are able to perform complex reasoning facts over large sets of facts with relative ease. This is possible partially due to the fact that human brain has the ability to filter through the data, and weigh relevant facts depending on the context
of the problem at hand. The brain is able to focus attention on the relevant information, instead of processing everything simultaneously. Moreover, our brain stores information together with meta-data describing its context (e.g. spatial or temporal). This allows it to find relevant information more efficiently, and focus attention on relevant facts in a given situation. Moreover, the meta-data stored together with memories in the brain contains a distribution of pointers to other relevant memories. This way the brain can build up a network of relationships between facts. This has been known in neuroscience as "episodic memory". In fact, research in neuroscience suggests that we have specific modules in the brain to allow for such functionality. For instance, the hippocampus is responsible for spatial, temporal, and sensory relations between memories [EC04]. The attention and memory module in our neural network architecture is inspired this research. While grading the essay, the dynamic memory network retrieves specific temporal states that are related to or triggered by the prompt and information from the first pass over the essay. A GRU module with such dynamic memory attention mechanism was shown to exhibit transitive inference capabilities on the Facebook bAbI question answering benchmark data set [KIS+15]. Interestingly, experiments show that disruption of the hippocampus impairs transitive inference capabilities in the brain [HZW+04], and that hippocampus neurons fire very actively during reasoning requiring connecting multiple facts.
This makes sense since making chains of logical reasoning steps on a collection of facts often requires focusing attention on the relevant data, and being able to access memories in relationship to the context (e.g. relevant domain knowledge).

Distribution of pointers to other memories stored in the brain has inspired research in memory modules for recurrent neural network architectures [WCB14]. This idea also allowed for new models of neural computation, such as neural Turing machines, because it makes storing and reading from memory into a differentiable operation, and hence allows for learning memory management automatically by back-propagation gradient descent on a training data set [GWD14]. Recent advances in recurrent neural networks research brought a number of powerful extensions to the RNN model. The most important directions in this area can be grouped into four main themes:

- **Attentional Interfaces**: allow RNNs to focus on relevant parts of the input.
- **Neural Turing Machines**: incorporate an interface to external memory, which is trainable using backpropagation.
- **Neural Programmers**: can generate programs by composing operations from a predefined set.
- **Adaptive Computation Time**: can perform varying amount of computation depending on the input.

We will focus on the first extension in this work, because it fits naturally with essay data. The idea is that not every word in an essay is equally relevant to the grade. We will train an attention mechanism to focus more on inputs (or as we’ll see later, features of words) that are relevant to score assessment.

**Attention Mechanisms for LSTM Recurrent Neural Networks**

The idea of attention was originally used in neural translation as an interface between two LSTM networks - an encoder and a decoder [BCB14]. The encoder module
1.5. NEURAL NETWORKS AND DEEP LEARNING

Figure 1.16: attention distribution

embeds sentences in the source language into a feature vector representation, and the decoder translates it into a sequence of words in the target language (figure 1.17). The attention module chooses which features to focus on. In order for the attention mechanism to be learnable by gradient descent, we need it to be differentiable. The technique used to allow for that is usually replacing categorical choices with a smooth attention distribution, which focuses on every input simultaneously to a different degree. One way to do this, is to generate a content-based attention distribution. We take a categorical attention query, apply it to the inputs, and feed it through a soft-max layer to generate a distribution (figure 1.16).

1.5.11 Dynamic Memory Networks

Most tasks in AI can be posed as question answering (QA) problems over adequate input pairs. For instance, machine translation into can be formulated in this framework as learning to generate output sequences in the target language based on pairs of input sequences representing the original text together with the question "What
is the translation into French?”. Sequence modeling tasks such as named entity recognition [PKM14] can be thought of as answering the question "What are the named entity tags in this sentence?". Sentiment analysis (an instance of classification problem in NLP), can be formed as a QA problem for the question "What is the sentiment of this sentence/paragraph?". Multi-sentence joint classification such as co-reference resolution answers questions of the form "Who does 'their' refer to?". We pose our text scoring problem in similar fashion, as a map from pairs $(T, Q) \rightarrow A$, where $T$ is the text of the essay, $Q$ is the question "What is the most likely grade the essay $T$ would receive, if graded by a human?", and $A$ is the predicted grade.

Question answering has attracted academic and industry interest in recent months. Facebook has recently released the bAbI dataset for training deep QA models. Socher et al. [KIS+15] [XMS16] introduced Dynamic Memory Networks - a neural framework for general question answering tasks, that can be trained using raw input-question-answer triplets. This approach is effective at solving sequence tagging tasks, classification problems, sequence-to-sequence tasks and question answering tasks that require transitive reasoning over inputs. We argue that assessing quality of essays often requires exactly such capabilities. Prior attempts at essay scoring treated input as a bag of words, without the ability to reason in a transitive manner over the word sequence. Good quality essays, represent much more than just diverse vocabulary usage, correct grammar and spelling, length, or style. They also need to answer the questions posed in the thesis. They need to build up arguments, and relate facts across multiple sentences in a coherent fashion. Dynamic Memory
is a step in that direction, and we believe that it captures some of the more abstract subtleties of essay writing. Below we review the general DMN framework.

A Dynamic Memory Network is composed of several modules (Fig. 1.18):

- **Input Module**: computes distributed vector representations of raw inputs. In our case it is converting an essay into a sequence of real valued vectors.

- **Question Module**: computes a distributed vector representation of the question asked. In our case it will be the encoding of the essay prompt.

- **Episodic Memory Module**: chooses which parts of the input representations to focus on during question answering. This is done via attention mechanism, and results in a memory vector representation. It is computed in an iterative fashion by going over the inputs several times. Each successive memory vector depends on the previous memory together with the inputs and the current question. Each iteration extracts newly relevant information from the inputs, allowing the model to focus on relevant facts while performing transitive inference.

- **Answer Module**: generates the output. In our case, this will produce the grade prediction for the essay.

In order to encode the input text, first a feed forward deep neural vector space embedding of the word2vec [MSC+13] type is computed. Then inside the Input Module, a recurrent neural network encodes the entire sequence of words into a single vector by utilizing the hidden states of the RNN as suggested by Elman in [Elm91]. More precisely, at time $t$, the hidden state depends on the embedding of the current word $w_t$, given by the embedding matrix $L$, and the previous memory state $h_{t-1}$:

$$h_t = RNN(L[w_t], h_{t-1})$$
CHAPTER 1. AUTOMATED ESSAY SCORING

Figure 1.18: DMN modules

Figure 1.19: Dynamic Memory Network architecture

Figure 1.20: Episodic Memory in a DMN
In our case, we will encode each sentence separately, together with a special word representing the end-of-sentence marker. Then use the final hidden state of the RNN (corresponding to the end of sentence marker) as the vector representation of each sentence. The entire essay is then a sequence of such vectors, which will be fed into the Episodic Memory Module.

The Episodic Memory Module will iterate over sequence of vectors representation of the essay, updating its internal memory in tandem. It is comprised of an attention mechanism and a Gated Recurrent Unit recurrent neural network (a form of an LSTM discussed above). The GRU and the attention mechanism work together in the following way. Each time, the internal memory \( m^i \) of the GRU is updated based on the previous memory \( m^{i-1} \), and episode \( e^i \):

\[
m^i = GRU(e^i, m^{i-1})
\]

Where the episode \( e^i \) is produced by the attention mechanism in the following way. In each cycle, the attention mechanism attends over the sequence of vectors produced by the Input Module, taking the RNN encodings of the input essay \( c \) and prompt \( q \), together with the previous memory \( m^{i-1} \) from the GRU to produce the episode \( e^i \):

\[
e^i = AM(c, m^{i-1}, q)
\]

We initialize memory \( m^0 \) to just the RNN encoding of the essay prompt. After a finite number \( T \) of episodes, the resulting memory \( m^T \) is returned to the Answer Module.

This iterative process allows the network to focus on different sentences in the essay during each episode, and use that information to perform a form of transitive inference.

The attention mechanism also uses a gating function. During episode \( i \), we distribute attention over sentences of the essay indexed by \( t \), to compute the scores:
\[ g^i_t = G(c_t, m^{i-1}, q) \]

Where \( G \) is a real valued function. \( G \) first produces a feature vector \( z(c, m, q) = \langle c, m, q, c \odot q, c \odot m, |c - q|, |c - m|, c^T W^{(b)} q, c^T W^{(b)} m \rangle \) where \( \odot \) denotes element-wise product on vectors. \( G \) is then a function of that feature vector, computed in our architecture by a simple two-layer feed forward neural network:

\[
G(c, m, q) = \sigma(W^{(2)} \tanh(W^{(1)} z(c, m, q) + b^{(1)}) + b^{(2)})
\]

In order to compute the memory update, we use the attention distribution to compute a weighted combination of vector representations over the sequence of input sentences from the essay. During episode \( i \) each RNN sentence embedding \( c_t \) is emphasized by the corresponding attention weight \( g^i_t \). The final state of the GRU will then be passed onto the Answer Module. The final equation for the \( i \)-th episode is:

\[
h^i_t = g^i_t \text{GRU}(c_t, h^i_{t-1}) + (1 - g^i_t)h^i_{t-1}
\]

\[
e^i = h^i_{T_C}
\]

Original application of dynamic memory was in the context of natural language generation for question answering. Because the answer in that context was another sequence of words, an additional GRU layer was used. However, we are concerned with predicting a scalar value representing a numeric evaluation of the essay, therefore our answer module differs from prior DMN architectures.

We experiment with two approaches to gluing the Episodic Memory Module to the Answer Module:

- **Average Memory:** We compute the mean \( \sum_{t=1}^{T} m^t \) of GRU hidden states from Episodic Memory Module
• Final Memory: We propagate just the last memory $m_T$ as input to the Answer Module

Let $x$ be the input vector to the Answer Module (either the final episodic memory or the mean). The Answer Module applies a final linear layer with sigmoid activation:

$$A(x) = \sigma(w^T x + b) \in (0, 1)$$

During training, we normalize all gold standard scores to be in the unit interval of the reals. At test time, we quantize the output of our model to predict the categorical grades on the scale 1-5.

1.6 Base Line Model

We compare our model against a base line implementation. Before beginning this project, I’ve done research on the state-of-the-art in Automated Essay Scoring. My first task was to investigate available methods for automatically predicting scores for short written responses written by Japanese learners of the English language. I realized that although active interest from industry and academia existed for decades, not much have been done in terms of innovation. Existing AES systems were results of year of incremental improvements based on tedious hard-coding of various special cases, based on linguistically informed features of the data. Of course, such systems weren’t robust, and needed constant adaptation. Most surprisingly, in spite of the recent success of deep learning models, none of those systems made use of neural NLP methods. I realized that LSTM-RNN neural networks, and recent advancements such as attention mechanisms and Dynamic Memory Networks were a natural candidate for tackling this problem. This is how I came up with the idea of this project. In future investigations, I want to build on what I have learned while working on this application. I’m especially interested in using visualization and statistical analysis of the neural firing patterns and weight evolution, to shed
light on what do such deep models learn while working with natural language data. I am convinced that there is great potential for deep learning approaches to this and other problems involving natural language data.

Our base line model was a state-of-the-art AES system at the time we began work on this project. It is based on statistical machine learning techniques. It was among the top performers during the 2012 Automated Student Assessment Prize competition organized by Kaggle and the Hewlett Foundation. The system is called Enhanced AI Scoring Engine (EASE). It was ranked 3rd among 154 teams that made it to the final round. It is based on an array of hand-crafted features and statistical methods. A regression model is built using those features of input essays. Two main modules of the system make use of Support Vector Regression (SVR) and Bayesian Linear Ridge Regression (BLRR). Features extracted from input text can be categorized into:

- Text statistics such as average sentence length
- POS tags
- Bag of n-grams
- Similarity with the prompt (e.g. How often are the words from the prompt mentioned in the response?)
- Deterministic spelling and grammar checks

The main ideas used in the EASE system are based on Phandi et. al. [PCN].

1.7 Text Scoring Architecture

The high level diagram of our text scoring architecture is shown in figure 1.21.

Pre-processing phase includes various operations concerning data clean-up. These include replacing named entity with placeholder words, tokenizing words into sequence of lowercase strings, correcting simple spelling errors that would prevent
Figure 1.21: Deep Text Scoring (DTS) architecture diagram
vector embedding lookup, removal of special characters, and other text preprocessing tasks.

Input module is composed of word2vec type embedding, context extraction via a convolutional neural network, and sentence embedding via a recurrent neural network.

The convolutional layer computes a linear mapping of all n-grams in the text. That is, the input to the episodic memory is a sequence of vector embeddings of all n-grams in the input essay, instead of embeddings of each word in sequence. Because the convolution operation ties weights to all n-grams, hence it is representing the essay as a sequence of n-grams instead of a sequence of independent words. This can be thought of as words in context, and is meant to capture some of the local dependencies, as well as reduce the complexity of the problem. In case of a 3-gram model, the vector embeddings produced by the convolutional layer are of the form:

\[ C(x_i) = [w_1 * x_{i-1}, w_2 * x_i, w_3 * x_{i+1}] + b \]

Where \( x_i \) means the vector space embedding of the i-th word in the essay (produced by the embedding layer) and \( b \) is a learned bias vector. \( C(x_i) \) can be thought of as a vector representation of the i-th word in context. For sentence boundary words (i.e. the first and last words in a sentence), we assign all-one vectors to the missing words. Therefore the number of vectors output by the convolutional layer is equal to the number of words in the input text, but each word is now represented as a weighted concatenation of the vector embeddings of words in its n-gram context. The output of the convolutional layer is therefore in a space of n-times (3 in our case) higher dimensionality (i.e. the space of n-gram context embeddings).

Another way of doing a convolution over an input sentence is the following. For each input sentence of length \( m \), we apply a nonlinear function to each n-gram window, to produce a feature map \( c = [c_1, c_2, \ldots, c_{m-n+1}] \), where each \( c_i = C(x_i) \). We then apply a max-ver-time pooling operation to the feature map \([\text{CWB} + 11]\) in order to capture the most important feature for that filter. We do this for several
different filters with different window sizes to produce a fixed array of features for each sentence. This naturally deals with sentences of varying sizes, and produces a fixed feature vector for each sentence.

The remaining modules have been discussed in more detail in other sections of this paper.

1.8 Model Intuition

The intuition behind our model is based on many ideas from neuroscience, cognitive science, linguistics, psychology, computer science, and mathematics. The modularity of our architecture is partially inspired by "society of mind" ideas developed by Martin Minsky [Minsky1986]. The use of convolutional neural networks and vector space embeddings comes from the "manifold hypothesis" in data science, which suggests that data generated by natural phenomena (such as the English language) is distributed in neighborhoods of lower dimensional manifolds within the ambient space [Hinton2006] [Ben-Israel2003] [Minzer2003] [Bengio2003] [Tishby2000] [Jia2000] [Shlens2000] [Schuster1998] [Cayton2005]. The intuition for use of memory and attention mechanisms was discussed in the relevant sections. Various features of our model are a very rough approximation to the operation of human brain (at least our current understanding of it). On the high level of abstraction our model is meant to capture the stages of mental activity, that a human grader would go through while assessing written responses in English. First the input layer produces an internal representation of the words, which corresponds to human understanding of the meaning and syntax of English. The episodic memory layers correspond to a human grader developing an understanding for the arguments presented in the essay, and how the various sentences relate to each other, and to the question posed in the prompt. The scoring module corresponds to human judging the quality of the essay based on having read it multiple times (i.e. the information provided by the final memory vector embedding from the dynamic memory module), paying attention to various features of the writing (extracted by convolutional neural networks). Essay flow, argument coherence, and relevance to the prompt are
captured by the episodic memory, via memories computed by the NAS-cell within the DMN layers.

1.9 Implementation

1.9.1 Neural Computation Frameworks

In order to implement our model, we looked at several popular deep learning frameworks, including TensorFlow, Theano, Torch, mxnet, Caffe, CNTK, and some others. We were looking for an environment that is

- modular (separate front-end and back-end)
- flexible (multiple languages available for the front-end; works on CPU, GPU, cloud clusters)
- active (many resources; corporate support; involved open source community)
- mature (we avoided some of the new frameworks in very early stages of development, because the API changes too drastically between updates)
- static and dynamic (allows for both static and dynamically changing computation graphs)

We have chosen TensorFlow, as it is best suited for our purposes and matches our selection criteria. This section will give a brief overview of the frameworks we experimented with, and their advantages. Next section will introduce TensorFlow.

1.9.2 TensorFlow

Tensorflow is an open source software library for numerical computation using data flow graphs [AAB+16]. Historically, it derived from an older framework called DistBelief, that was used at Google to implement their first deep learning models. It
is especially well suited for machine learning applications, and it was developed at Google Brain for use in deep learning research projects. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. Tensorflow was originally designed as a static computation framework, meaning that the computation graph remains constant regardless of input, once it has been defined. This was a potential obstruction to developing some of the more advanced deep learning architectures, especially those involving recurrent neural networks in the area of natural language processing. However, recent updates to tensorflow introduced tensorflow fold - an extension to the tensorflow libraries that allows for training of neural networks with dynamic computation graphs. It is therefore now possible to define deep-learning models in tensorflow that operate over data of varying size and structure, where the structure of the computation graph depends on the structure of the input data \cite{LHHN17}. Tensorflow is supported on a wide array of devices and architectures, including mobile devices, and cloud computing services. It also comes with an array of visualization tools grouped under tensorboard visualization package. It is a rich visualization tool which can be used in debugging, or to gain inspiration during neural network research.

Tensorflow’s popularity in academic research grew exponentially fast since its release, and it is now the most frequently cloned repository on github in the machine learning category (figures 1.22 and 1.23). Because of this strong community support, especially among academics, and corporate support from Google, I have chosen to learn Tensorflow through this project. Skills acquired in developing complex deep learning architectures in tensorflow will be very useful to me in future research during my PhD dissertation. Because of its popularity among researchers, it is easy to collaborate with people on projects, and there is already a large code base to recycle, which saves time and allows me to focus on novel ideas instead of re-engineering the wheel.
CHAPTER 1. AUTOMATED ESSAY SCORING

Figure 1.22: adoption of tensorflow in academic community

Figure 1.23: tensorflow github popularity
1.10 Data Set

We have obtained a dataset of hand graded essays in English to train our model. The data was collected by the Hewlett Foundation, and contains essays by a large population of students from various backgrounds. We would also like to train on data collected in Japan from Japanese learners of English, as it would make our trained model better at assessing this particular population.

The Hewlett Foundation collected hand graded essays written on a variety of topics. Some characteristics of this dataset are as follows:

- each essay has two or three independent human assigned scores
- there are eight sets of essays with 10686 essays in total
- each essay set contains responses on a different topic, and was assessed on a different grading scale

We have chosen to work with this data, because it is large enough to train a deep model, was written and graded by humans (unlike some of the artificially generated sets), and allows for direct comparison with prior results in the literature, in particular our baseline was developed for the same data set. One drawback to using human data instead of a much larger, algorithmically generated dataset, is that deep architectures require more data to learn, and our model could perform better with a larger dataset. However, using natural data is important to assess performance of our model, because it isn’t clear that learning an algorithmically generated dataset really deals with the problem we’re trying to solve, or is it just...
reverse-engineering the algorithm that generates the dataset. A famous example of an artificially generated dataset is the Facebook bAbI QA dataset. Main parts of the dataset were generated algorithmically, and it is a source of current criticism from the community. Another example, from my own experience, is a robot navigation dataset I’m worked on during a project at the Robot Intelligence through Perception Lab at Toyota Technical Institute Chicago. Previous dataset used in that project was algorithmically generated, and it received criticism for that reason, because it wasn’t clear if our algorithm solves the natural language navigation problem, or if it is simply learning to predict the behavior of the algorithm generating the data. In that parallel project, I’m currently developing methods for deep sequence to sequence learning for robot navigation. We are developing a more human-like dataset by using data available in Google Maps. Because of these experiences, I believe it is worth sacrificing some performance for increased interpretability and confidence when working with natural data. The Hewlett dataset is such a compromise. It is still large enough to train our model, and it was generated by humans in natural setting of school assignments. Therefore, if our system can score Hewlett essays accurately, we will have reasons to believe that our model is actually learning the notion of a "good" essay, and would generalize in education applications.

1.11 Evaluation

We use the metric recommended by Kaggle for the essay scoring competition. Below we include the definition of the evaluation metric as per Kaggle competition documentation. Performance will be evaluated with the quadratic weighted kappa error metric, which measures the agreement between two raters. This metric typically varies from 0 (only random agreement between raters) to 1 (complete agreement between raters). In the event that there is less agreement between the raters than expected by chance, this metric may go below 0. The quadratic weighted kappa is calculated between the automated scores for the essays and the resolved score for human raters on each set of essays. The mean of the quadratic weighted kappa is
then taken across all sets of essays. This mean is calculated after applying the Fisher Transformation to the kappa values. A set of essay responses $E$ has $N$ possible ratings - $1, 2, \ldots, N$ - and two raters, Rater $A$ and Rater $B$. Each essay response $e$ is characterized by a tuple $(e_a, e_b)$, which corresponds to its scores by Rater $A$ (resolved human score) and Rater $B$ (automated score). The quadratic weighted kappa is calculated as follows. First, an $N$-by-$N$ histogram matrix $O$ is constructed over the essay ratings, such that $O_{i,j}$ corresponds to the number of essays that received a rating $i$ by Rater $A$ and a rating $j$ by Rater $B$.

An $N$-by-$N$ matrix of weights, $w$, is calculated based on the difference between raters’ scores:

$$w_{i,j} = \frac{(i - j)^2}{(N - 1)^2}$$

An $N$-by-$N$ histogram matrix of expected ratings, $E$, is calculated, assuming that there is no correlation between rating scores. This is calculated as the outer product between each rater’s histogram vector of ratings, normalized such that $E$ and $O$ have the same sum.

From these three matrices, the quadratic weighted kappa is calculated:

$$\kappa = 1 - \frac{\sum_{i,j} w_{i,j} O_{i,j}}{\sum_{i,j} w_{i,j} E_{i,j}}$$

The Fisher Transformation is approximately a variance-stabilizing transformation and is defined:

$$z = \frac{1}{2} \ln \frac{1 + \kappa}{1 - \kappa}$$

Since this transformation approaches infinity as kappa approaches 1, the maximum kappa value is capped at 0.999. Next the mean of the transformed kappa values is calculated in the $z$-space. For Essay Set 2, which has scores in two different domains, each transformed kappa is weighted by 0.5. This means that each dataset has an equally weighted contribution to the final score. Finally, the reverse
transformation is applied to get the average kappa value:

\[ \kappa = \frac{e^{2z} - 1}{e^{2z} + 1} \]

**Base Line**

The base-line algorithm, which is the state-of-the-art statistical machine learning system, achieved average quadratic weighted kappa scores across all data sets of 0.699 and 0.705 for the support vector regression (SVR) and Baysian linear ridge regression (BLRR) versions respectively.

**DTS - current state**

The research and design of DTS system is complete at this point. I am currently nearing completion of the implementation. I have an 3-month long internship in Tokyo this summer to complete the experimental section and testing of this system. Several Japanese companies are interested in using such systems for real world applications, and I have been invited to complete the implementation in Japan this summer. The company that invited me also owns thousands of schools, and they are collecting a dataset under my instructions to train the DTS system. I will continue working on this project as part of my future research. I am waiting for the new training data, and my summer internship to finish testing DTS. Recently, I have been consumed with a parallel project I have been working on. I plan to publish the results of the second project, and the paper deadline is approaching in a couple weeks. Therefore I have shifted my attention fully to the other project, and will return to DTS during my summer internship in Tokyo. This second project concerns natural language understanding in the context of robotic agents acting within changing environment. This is the topic of the next chapter of this thesis.

Main goals for the coming months include:

- finish full implementation in Tensorflow
• perform a series of experiments with various architectural (types of convolutions used, types of recurrent cells), and data pre-processing choices

• perform hyper-parameter optimization

• design useful visualizations

• assess and interpret the contributions of various modules (e.g. episodic memory) and design choices

• briefly discuss hardware used for training and implementation code details

1.12 Future Directions

Future research directions inspired by this work fall into several categories:

• augmenting the neural network architecture and tuning hyper-parameters to improve performance

• developing additional deep neural network modules to model specific aspects of essays such as topic agreement, argument strength, style, creativity, mechanics, organization, etc.

• experimenting with GAN (Generative Adversarial Network) approach [GPAM+14]

• experimenting with Memory Augmented Neural Networks with Wormhole Connections (shown to outperform LSTM) [GCB17]

• using the current system as a reward module in a DQN type reinforcement learning model [MKS+15]

• developing better feedback mechanisms, that can explain model’s decisions in a pedagogically beneficial way
• visualization

1.13 Conclusion

In this project we have investigated a problem of practical importance, at the intersection of natural language processing and language education. Although the problem is practical in nature, we used it as an opportunity to learn theory and skills that will form the foundation of subsequent research projects in NLP and Computational Linguistics. We formalized the problem of automated essay assessment as an instance of machine learning. We performed a comprehensive literature review of several hundred publications spanning six decades of discourse, which informed us of the current state of research in this area, and was used to introduce the reader to the history of the problem and background theory used to formulate a solution. We then invented and implemented a novel solution to this problem, that based on preliminary experiments and theoretical considerations should outperform the state-of-the-art at the time of writing by a significant margin in terms of score prediction accuracy. Necessary theory and background was introduced, along with discussion of implementation choices (deep learning frameworks, tensorflow), and analysis of empirical results.

Prior approaches to automated essay scoring relied on classical statistical machine learning algorithms using hand-crafted features. Our model employs advanced deep neural network architectures such as CNN, LSTM, DMN, and architecture optimizations obtained via deep reinforcement learning. This makes it more general, and robust, while promising higher performance on the same benchmark datasets.

We believe that there is unexplored potential for these and other newly emerged deep models to explore more fundamental problems in the field of Computational Linguistics. Our experience and theoretical background gained through working on this project prepared us to begin work aimed at developing novel approaches to research on language problems. We are especially interested in morphological analysis of languages with interesting inflectional paradigms, syntactic category induction,
visualization of language (please look at the second writing sample attached below within the same PDF file), and unsupervised language modeling. We plan to develop deep neural architectures to extract various types of structure from raw corpora of text, and then analyze what these models are learning through statistics, mathematics and visualization, in order to answer fundamental questions about the structure of language.
Chapter 2

Robot Intelligence through Perception

Abstract

We propose a new framework for autonomous agents executing natural language instructions in the context of a dynamic, real-world environment. Previous research in this area focused on grounding natural language in pre-determined artificial environments. Here we introduce a framework for robot navigation in real world environments. We introduce a new dataset generated from Google Street View images. Natural language instructions are paired with real-world location and percept sequences corresponding to an autonomous vehicle navigating those environments. We use a deep neural sequence-to-sequence model inspired by neural machine translation \cite{BCB14} to interpret human instructions in the context of environment observation sequences, and relate them to appropriate sequences of agent actions in the environment. We collect data from downtown Chicago area, and test an agent representing an autonomous vehicle navigating the streets according to human instructions in English.
CHAPTER 2. ROBOT INTELLIGENCE THROUGH PERCEPTION

2.1 Introduction

Natural language is the most intuitive and flexible way for humans to communicate. Unfortunately when issuing instructions to machines, humans are forced to master artificial, over-constrained, domain-specific ways of expression. Diverging from those specifications often renders instructions incomprehensible to machines, or worse, leads to unexpected responses.

The goal of my research is to counter that problem by developing systems bringing us closer to effective natural language communication between humans and machines. This fits into a bigger effort at multiple institutions, directed toward natural language understanding.

Natural language provides a rich, intuitive and flexible medium for humans and robots to interact and share information. Apart from convenience, there are many situations in which being able to instruct machines in unconstrained natural language is a necessity.

Robots assisting first responders in search and rescue missions, need to be able to communicate with victims and personnel untrained in domain specific languages. For example, a robot might engage in the following dialog when it encounters a victim:

- Robot: Someone is on the way to get you out of here. Are there any other people around who need help?

- Victim: I saw someone in the main lobby. essays in total

- Robot: Where is the main lobby?

- Person: Exit this room and turn right. Go down the hallway past the elevators. The lobby is straight ahead.

- Robot Understood.
2.1. INTRODUCTION

Another example is instructing a self-driving car which route to take, or teaching it a location not present on its map.

- Passenger: The map is wrong, my house is actually on the other side of this building. Take left, then right, and my house will be next to the large tree.

- Car: Understood.Updating map location.

In this project I am interested in is translating path descriptions given to a robot in natural language, into sequences of actions that robot can follow. The choice to focus on direction understanding is a good starting point for several reasons. First, following directions requires the ability to understand spatial language. Because spatial language is pervasive, this ability is important for almost any application of natural language to robotics. Second, a system that understands directions is useful in many scenarios, including health care, companion robots, search and rescue, and self driving vehicles carrying passengers. Third, it is natural to ask humans to create a set of directions through an environment, yielding an open-ended yet task-constrained corpus of language. Finally, there is a natural correctness metric.
when evaluating a robot’s performance at following natural language directions: did it reach the correct [U+FB01]nal destination? The availability of a corpus and a concrete correctness metric enable an of [U+FB02]ine component-based evaluation of such systems, which is critical for achieving robustness, because we can quickly test new models on linguistic input from many different users.

The goal of this project is to relate a sequence of words representing a natural language command, to a sequence of actions for a robot controller to execute. Grounding of natural language instructions for human-machine interaction, such as directions issued by passengers of self-driving cars, has mostly been approached using graphical models and statistical machine learning approaches with hand-crafted features. The main obstruction to the application of deep neural network models for this purpose is scarcity of quality training data. As part of a bigger effort developing neural network models for natural language direction understanding, we generate a large data set from crawling Google Street View over randomly generated paths in downtown Chicago. We obtain location and image data from Google Maps, and use image classification and language generation techniques to derive a corpus of natural language instructions paired with environment observations and robot action sequences. This data is used to train an alignment-based encoder-decoder model with long short-term memory recurrent neural networks (LSTM-RNN), which translates natural language instructions to action sequences based upon a representation of the observable world state.

2.2 Related Work

Research efforts on algorithms allowing free-form natural language communication with machines [MSK06] [KTRR10] [Che12] [CM11] [KM13] [KLB+14] [HDH+15] [KM12] have mostly focused on the idea of symbol grounding defined in Harnad in the early 1990s [Har90], which concerns associating linguistic terms with the objects (physical or abstract) that they describe. Early solutions to the symbol grounding problem used hand-crafted mappings [TKD+11] [MSK06]. Statistical methods have
2.2. RELATED WORK

also been applied to infer the meaning of words in the context of perceptual inputs [Moo08] [MFK10]. These methods require human engineered features and annotated corpora. Another direction pursued previously is to treat instruction following problem as that of learning a parser defining a mapping of natural language into its formal equivalent, that can be processed by the machine. Chen and Mooney [CM11] use probabilistic context free grammar (PCFG) induction to learn groundings for a learned lexicon, while Artzi et al. [AZ13] [ADP14] use a combinatory categorical grammar (CCG) based semantic parser to map natural language into a lambda calculus representation. Simultaneously, a parallel set of approaches have been developed which can be grouped under a label of probabilistic world model grounding. These techniques map natural language into sets of corresponding objects, locations, and actions within the agent’s world representation under a probabilistic model of symbol to world correspondence. Interpreting instructions in this framework is performing inference in that learned model. Several previous approaches aimed at restricting the complexity of the search space by adopting a probabilistic approach with simplifying assumptions of independence. Kollar et al. [KTRR10] developed a generative model of spacial relations, adverbs, and verbs. In particular graphical models (factor graphs) were used, which represent a factorization of the conditional probability distribution of groundings, given the sequences of words representing commands given by a human. Tellex et al. present a discriminative model that captures hierarchical and compositional structure of language. Such factorizations can be derived automatically by parsing natural language commands into Spatial Description Clauses, and then using the structure SDC decomposition of a command to induce the corresponding factor graph, with random variables representing words, groundings, and correspondence variables (which describe if groundings are correct).

Recently a new array of powerful techniques for working with sequence data have emerged in the neural network community. Deep learning approaches to sequence-to-sequence mapping have been increasingly successful with application
in machine translation [SVL14] [BCB14] [CVMG+14], natural language generation [RCW15] [WGM+15], and image captioning [KSZ14] [MXY+14] [DAHG+15] [VTBE15] [CLZ15] [KFF15]. We propose a deep learning approach to this problem, which does not require contrived formal languages such as SDC. In contrast to those prior methods, the deep learning model uses no specialized linguistic resources (e.g., parsers) or task-specific annotations (e.g., seed lexicons). It is therefore generalizable to a variety of human-machine interaction settings. The mapping is learned automatically in an end-to-end fashion, purely via the means of back-propagation in a deep recurrent neural network.

2.3 Model

In order to allow machines to interpret natural language instructions, we need to map sequence of words in a human language representing the instructions, to the appropriate sequence of actions corresponding to commands that robot’s actuators can execute. On the high level of abstraction, it is therefore an instance of a sequence-to-sequence learning. However, the robot also needs to disambiguate the instructions in the context of the world environment it operates in. Therefore, we also need to consider a third sequence - the temporal sequence of perceptual inputs received by the robot in the process of executing the instructions.

To develop a solution to this problem, we approach instruction following as a form of neural machine translation [BCB14]. In this setting instead of translating from English to Japanese, we translate from English to "machine language", i.e. to the sequence of instructions that the machine can execute. However, in contrast with simple machine translation, we also consider the percept sequence, and use it to alter the interpretation of the input instructions in real time. First, the robot accepts the human instruction, and develops a memory vector representing its general understanding (encoder step). Afterwards, robot takes an action based on that initial understanding, and receives new input from the environment (e.g. if the robot moved, a new image from the cameras will be processed). At each time step,
2.3. MODEL

Figure 2.2: instruction following architecture

Another action is taken and a new world observation received. This iterative process augments the initial memory vector representing the natural language instruction originally given to the robot. Eventually, a stop action is emitted. We can then evaluate how accurately the given instructions were processed. For instance, if we consider directions given to a self-driving car, we can measure the distance of the final location of the car from the desired destination.

Abstractly, the goal is to derive a model over sequences of actions conditioned on world state and natural language instructions: \( P(a_{1:T}|y_{1:T}, x_{1:N}) \). Then extract the maximum probability sequence of actions under the trained model.

\[
a^{*}_{1:T} = \arg \max_{a_{1:T}} P(a_{1:T}|y_{1:T}, x_{1:N}) \\
= \arg \max_{a_{1:T}} \prod_{t=1}^{T} P(a_t|a_{1:t-1}, y_t, x_{1:N})
\]

We approximate this conditional probability using a neural encoder-decoder architecture. During the encoder stage, we use a bidirectional recurrent Long-Short Term Memory neural network similar to that of Hinton et al. [GMH13] to derive a sequence of hidden annotations \( h_{1:N} = (h_1, h_2, \ldots, h_N) \), where we think of annotation \( h_i \) as a summary of the first \( i \) words of the natural language instruction given.
to the robot.

\[
\begin{bmatrix}
i^e_j \\ f^e_j \\ o^e_j \\ g^e_j 
\end{bmatrix} = \begin{bmatrix}
\sigma \\
\sigma \\
\sigma \\
\tanh
\end{bmatrix} T^e \begin{bmatrix}
x_j \\
h_{j-1}
\end{bmatrix}
\]

\[c^e_j = f^e_j \odot c^e_{j-1} + i^e_j \odot g^e_j\]

\[h_j = o^e_j \odot \tanh(c^e_j)\]

Here \(e\) designates that the variables correspond to the encoder phase, \(\sigma\) is the logistic sigmoid function, \(T\) is an affine transform, \(i, f, o\) are the input, forget, and output gates of the bi-LSTM, and \(c\) is the cell state activation vector. The cell memory is updated iteratively based on previous memory and current input under regularization induced by the input and forget gates. The use of a bidirectional LSTM-RNN as the encoder is inspired by its success in speech recognition and machine translation [GMH13] [BCB14] [CVMG+14].

The global view of the architecture is shown in figure 2.2. Initial natural language input is embedded using a bidirectional LSTM (2.3) in the encoder unit. We then concatenate a one-hot embedding of the input words to the memory cell vectors of the LSTM using a linear layer. This is done by the aligner unit. Alignment allows the model to focus on parts of natural language instruction relevant to the current action. Alignment was proven effective in the context of machine translation and machine vision [BCB14] [VHGK14] [BMK14] [HBK+15]. Our model learns to align based not only on the high-level input abstraction, but also on the low-level representation of the input instruction, which improves performance. The aligner is simply a linear layer \(z_t = \sum_j \alpha_{ij} \begin{bmatrix} x_j \\ h_j \end{bmatrix}\), where \(h\) is the encoder embedding, and \(x\) is the original input. The weight \(\alpha\) is computed by a nonlinear function.
\[ \beta_{tj} = v^T \tanh(Ws_{t-1} + Ux_j + Vh_j) \]
\[ \alpha_{tj} = \exp(\beta_{tj}) / \sum_j \exp(\beta_{tj}) \]

Where \( s \) is the hidden state of the decoder, and \( W, V, U \) are learned parameter matrices.

The final memory state of the encoder is used to initialize the decoder. This is another LSTM unit, which translates a sequence of world state observations into a sequence of robot actions. The robot takes those actions in real time, which affects the future world observations. Eventually a stop action terminates the output sequence. In order to choose which action to take, we pass the decoder output through another squashing non-linearity and apply the soft-max layer to obtain a distribution over possible actions. Formally the decoder is another LSTM-RNN, which uses the world state representation \( y_t \), the context of the natural language instruction \( z_t \) (produced by the encoder) and the decoder’s previous memory \( s_{t-1} \) (aligning it with the instruction context). We use a deep output layer [PGCB13] to produce a conditional probability distribution over possible actions to take next
\[ P_{a,t} = P(a_t|a_{1:t-1}, y_t, x_{1:N}). \]
In the equations above, $E$ is an embedding matrix, $L_o, L_s, L_z$ are learned parameters.

For training we use negative log-likelihood of the given action at each time step as the loss function.

$$L = -\log P(a^*_t | y_t, x_{1:N})$$

Inference is done over an ensemble of randomly initialized models, which is shown to have a denoising effect in deep neural networks [SVL14] [ZSV14] [VTBE15]. At each time step we choose the maximum a posteriori action under the trained model.

The model is implemented in Python using the PyTorch computation framework.

### 2.4 Data Set

We are interested of learning the model mapping from corpora of training data of the form $(x^{(i)}, w^{(i)}, a^{(i)})$ for $i \in \{1, 2, \ldots, n\}$ where $x^{(i)}$ is the natural language input (i.e. the instruction given to the robot), $w^{(i)}$ is the sequence of environment observations, and $a^{(i)}$ is the desired sequence of actions that robot should take in the context of the world observations in order to correctly follow the given instruction. After the
model has been trained, it is able to predict a sequence of actions $a^{(i)}$ given a natural language instruction $x^{(i)}$ and environment observations $w^{(i)}$.

We have worked with two data sets. The first data set came from an artificially generated environment 2.4. The environment consists of a maze of hallways characterized by different texture patterns (grass, brick, wood, gravel, blue, flower, yellow, octagons). Various objects are placed within the halls (hat rack, lamp, chair, sofa, barstool, and easel) and the walls can contain several types of paintings (butterfly, fish, or Eiffel Tower). The natural language instructions come from humans who were told to navigate this world in a simulator. The instructions contain noise in form of ambiguities, spelling and grammatical errors, or being incorrect (e.g. using left when meaning right). Some sentences can not be mapped to any actions, and some produce unfeasible paths through the maze.

The second data set was collected using scripts we developed to obtain Google Street View data from real-world environments 2.5. Our code generates paths through downtown Chicago, paired with sequences of geolocation data, environment information (e.g. type of road), and sets of images covering 360 degree view at each location. Every path is paired with natural language instructions, which include references to visible landmarks along the route. There is randomness in the choice of language, as well as landmark references. Because mentioned landmarks are detected using vision API (based on convolutional neural network models), it contains a degree of ambiguity, which makes it more realistic in comparison to the purely virtual dataset.

2.5 Experiments

Our model is meant as a general framework for robot action sequence generation from natural language. However, we first intend to perform experiments with robots available at the Toyota Technological Institute in Chicago (2.6).

I am currently preparing training data and experiments. I will continue working on this project over the summer and hope to extend this model in future research
Place your back against the wall of the “T” intersection. Go forward one segment to the intersection with the blue-tiled hall. This intersection [sic] contains a chair. Turn left. Go forward to the end of the hall. Turn left. Go forward one segment to the intersection with the wooden-floored hall. This intersection contains [sic] an easel. Turn right. Go forward two segments to the end of the hall. Turn left. Go forward one segment to the intersection containing the lamp. Turn right. Go forward one segment to the empty corner.
2.5. EXPERIMENTS

Figure 2.5: Google Street View dataset

Figure 2.6: robots used for experiments
2.6 Conclusion

We presented our ongoing research on applications of deep neural networks and NLP to robotics. The goal of this line of research is to develop deep learning methods allowing free-form communication between humans and machines. In particular we are interested in natural language understanding in the context of changing world state, for the purpose of executing navigation instructions given to autonomous vehicles navigating real world environments. We believe that LSTM-RNN based architectures inspired by neural machine translation research are a promising direction of inquiry aimed at finding robust solutions to this problem. We are lucky to have access to robotics equipment suitable for testing such models. Working on this project was a great learning experience, and it sparked my interests in neural networks and applications of NLP to robotics. I hope to pursue projects relating deep neural networks and natural language data in the context of robotics in my future research endeavors, in addition to my main interests in computational linguistics.
ACKNOWLEDGMENTS

I would like to thank my advisor Prof. John Goldsmith at the University of Chicago for inspiring me to pursue computational approaches to natural language problems. This work was completed in part with resources provided by the University of Chicago Research Computing Center and cloud computing clusters at Amazon.
Bibliography


[CASM15] 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! In *Proceedings of the Fifth International Conference on Learning Analytics And Knowledge*, pages 203–207. ACM, 2015.


Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word bench-


Peter Phandi, Kian Ming A Chai, and Hwee Tou Ng. Flexible domain adaptation for automated essay scoring using correlated linear regression.


