Dataset Augmentation for Aspect Level Sentiment Analysis

by

Nikhil Sapru

A thesis
presented to
the University of Guelph

In partial fulfillment of requirements
for the degree of
Master of Applied Science
in
Engineering

Guelph, Ontario, Canada

ABSTRACT

DATSET AUGMENTATION FOR ASPECT LEVEL SENTIMENT ANALYSIS

Nikhil Sapru
University of Guelph, 2020

Advisor:
Dr. Graham W. Taylor

Deep learning models have been instrumental in achieving state-of-the-art performance on benchmark datasets across various tasks in computer vision as well as natural language processing. The success of these models has come as a result of the availability of large scale labeled datasets along with compute infrastructure to process that data. Deep neural networks perform poorly when there is a lack of labeled data to train them. Data augmentation is one of the methods used to boost model performance by creating new training data from previously labeled data. Improved data augmentation techniques would be an effective tool to expand smaller datasets to match the amount of labeled data required to maximize the performance of deep learning models. This would eventually help to reduce the time and cost to develop these models. However, for data augmentation techniques in the natural language processing domain, most of the developed methods are unique to a particular task or dataset. Therefore, there is a need to develop task agnostic data augmentation techniques in the domain of natural language processing. In this thesis, we the application of two recently known data augmentation techniques to a new task in natural language processing. Manifold mixup is a data augmentation technique for images developed by Verma et al. to act as a regularizer for neural network and SwitchOut is another technique developed by Wang et al. to boost the performance of neural networks used for machine translation. Through carefully designed experimentation, the thesis demonstrates the viability and usability of manifold mixup and SwitchOut as efficient task agnostic data augmentation techniques in natural language processing.

ACKNOWLEDGEMENTS

I would first like to thank my advisor, Dr. Graham W. Taylor, for giving me the opportunity to join his research group despite my background in mechanical engineering with little programming and machine learning experience. Thank you for your patience and giving me the time to try different projects until I felt confident to pursue a research problem for my thesis. I had some really low points during the past few years and your support was instrumental in helping me continue and finish this work. I can't thank you enough for your able guidance and for being patient & supportive when I needed it the most.

I would also like to thank Dr. Fei Song for joining my advisory committee and for being generous with his time whenever there was anything that I needed to discuss.

I am extremely grateful to three friends whose support was crucial at different points of time and I would have probably given up without their support. Joydeep helped me with my master's applications when I was spending long hours completing projects at work and not getting enough time to work on my graduate applications. He was a great friend and advisor during this journey, first helping me during the transition from my industry job to being a full time student and then continuing to help me figure out my way through various challenges in my school and personal life. After my family, he has been the most supportive person during this time. Dhanesh became a friend when I started in the lab and helped me work through various challenges that I faced while coding and debugging my experiments. At one point, he was kind enough to spare one day each weekend to help me out despite having a full time job and other responsibilities towards his family. He made time to help whenever I was unable to debug my code or was dealing with other personal challenges. I remember going over to his place for many lunches and dinners that were nice breaks from my regular routine. Devinder helped me when I was incorporating SwitchOut in my final set of experiments, helping organize my final results and also with regular suggestions to improve my thesis. During the last three years he has consistently been a great friend during both the good and bad times. He held me accountable by regularly following up on my thesis progress, providing the necessary push when needed and helped me stay on track to eventually finish my thesis. He did all this while having a busy schedule himself that involved his PhD research, travel to conferences and working on multiple industry projects. I have spent many days working out of his lab in Waterloo and nights at his apartment while he patiently reviewed my work whenever it was ready and provided feedback to improve it. Thank you guys for helping me persevere when I was ready to give up.

I would like to thank Sangam and Piyush for being great friends and continuing to be there anytime I needed help. Thank you for being there to talk when I felt lost and being supportive when I worked through different issues in my personal, school and professional life. I would also like to thank Mrigank for being a great friend and for the long chats about staying focused and finishing my thesis.

I am also grateful to all the members in the MLRG research group. I felt welcome from the first day and people in the lab were happy to spare time to help out with coding issues and also discuss ideas whenever I had any. I would like to thank Brittany for her support with all the administrative issues and being a great lab manager. I would like to thank Magdalena for all the help in getting ready for my thesis defense, proof-reading the thesis and offering detailed feedback for improving this thesis. I would like to thank Terrance for his help in incorporating manifold mixup in my code-base and also sharing his knowledge about data augmentation. I would like to thank He and Fei for helping me learn about Theano and Theano-MPI during my first project. I would like to thank Matt and Griffin for helping me with different small and big issues when I was new to the lab. I would like to thank everyone else for being great lab mates. I will miss all the fun we had during our camping trip, playing badminton and countless other things that we did together. Thank you guys! I would also like to thank Jacqueline Floyd for her help and support with all administrative matters during the course of my degree.

I would also like to thank Borealis AI for funding my research and giving me the internship opportunity with their research team.

In the end, I would like to thank my sisters Aishu and Khushboo for making things fun at home. I would like to thank my parents, both sets of grandparents, my uncle (Vimal) and aunt (Preevanada) for their support and always being there for me even when you didn't agree with my choices. Thank you for letting me do things that seemed right to me and being patient as I slowly figured things out for myself. I would like to thank all other friends and extended family members for your love, support and all the pleasant memories so far.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Deep learning has helped in building models that have improved performance on benchmark datasets and real world applications. However, the successes in computer vision have outpaced those in natural language processing. One of the major factors that has led to the success of deep learning models is the arrival of large scale datasets with manually annotated labels to train these models. Manually labeling the data is a time intensive endeavour and can become extremely costly when domain experts are needed. For example, datasets for machine translation require at least a few hundred thousand examples and requires experts to provide accurate translations. Data augmentation is one of the tools that can help reduce the number of labeled examples required to train deep learning models.

Data augmentation refers to the practice of expanding labeled data by transforming it to create new examples used at the time of training machine learning models. Simple to execute techniques such as random cropping and noise injection to image data has been demonstrated to improve model performance. Additional training data developed using these techniques has been shown to act as a regularizer [2] that ultimately results in models that generalize better. An effective data augmentation strategy could help reduce the cost and time required to annotate data. Minimizing the amount of data that needs labeling effectively reduces the time and cost in preparing data for the model while providing more time for the model development and optimization.The motivation for this research is to develop task agnostic data augmentation techniques for textual information used by deep learning models. Popular techniques used for textual information, such as back translation used for machine translation [3], are limited to the specific task for which they were originally developed. Hence, such techniques cannot be used on other tasks such as sentiment analysis.

One of the problems in NLP that could benefit from an effective data augmentation technique is the task of aspect level sentiment analysis. Aspect level sentiment analysis refers to the task of analyzing the sentiment expressed towards a particular target or entity. In the sentence *"The food is great but the service is poor"*, the sentiment is positive towards the *food* and negative towards the *service.* Most benchmark datasets [4, 5, 6] for this task have a few thousand to a few ten thousand labeled examples which is not sufficient to build a deep learning model with robust performance. To overcome this issue of data scarcity, it is vital to explore other techniques such as novel data augmentation. Advancing the performance of machine learning models for aspect based sentiment analysis would ultimately help in developing algorithmic tools to automatically analyse data where multiple aspect terms or entities are referred. A large amount of data is generated each day in the form of product reviews, user feedback in the form of email, social media posts about perception of government policy and other information that governments, journalists and businesses analyze. All this information cannot be manually analyzed so it is imperative that we develop automated techniques to analyse such information with a high level of accuracy. We acknowledge that there is potential for abuse as tools get more sophisticated and hope that leadership teams in industry, academia and government would implement necessary checks and balances to prevent abuse and unethical application of machine learning technology.

1.1 Objectives

The main objectives of the research described in this thesis is to develop standard data augmentation techniques that could eventually be added to existing tool-kits used by practitioners working on problems in natural language processing. While developing such a technique, this thesis aims to do the following:

- Explore the current state-of-the-art techniques in data augmentation techniques in the computer vision domain and explore their extensions to the domain of natural language processing.
- Develop a data augmentation technique that is dataset or task agnostic.
- Leverage data augmentation to improve the baseline model performance for aspect level sentiment analysis.

1.2 Contributions

We make the following contributions in this thesis:

- We identified the most effective data augmentation techniques in representational space in the computer vision domain. The first technique that we explored was manifold mixup [7]. We designed a framework for applying manifold mixup from computer vision domain to the NLP domain. We then demonstrated the effectiveness of manifold mixup [7] for the problem of aspect level sentiment analysis. This enabled us to effectively extend a known data augmentation technique from computer vision to a task in natural language processing.
- SwitchOut, proposed by Wang et al. [8], was explored as another study in this thesis. In the seminal paper, SwitchOut was developed specifically for the task of machine translation. In our study, similar to the previous case, we first extended the application of this technique from machine translation to aspect level sentiment analysis and tested its effectiveness by adding SwitchOut to the baseline Long Short-Term Memory (LSTM) model. We then carefully designed experiments to extensively explore optimized hyper-parameters for our task. This enabled us to demonstrate SwitchOut as a task agnostic technique that works for at least one other task beyond machine translation.

- The improvement achieved in classification accuracy by the use of manifold mixup and SwitchOut enabled us to demonstrate the effectiveness of these techniques on a relatively smaller dataset. We carried our experimentation on the benchmark dataset by Dong et al. [4] which consisted of 6,248 labeled examples in the training dataset. Ideally, labeled datasets that are created for deep learning models comprise of hundreds of thousands to a few million examples.

1.3 Organization

The remainder of this thesis is organized as follows:

Chapter 2 provides background information on various neural network architectures used in this thesis. We present Multilayer Perceptron (MLP), a simple neural network, followed by Recurrent Neural Networks (RNNs) and LSTMs.

Chapter 3 provides details regarding the application of deep learning for natural language processing. We specifically cover the required fundamentals for word embeddings and sentiment analysis techniques and how deep learning is used in this context.

Chapter 4 introduces different types of data augmentation in computer vision and natural language processing. The Chapter also presents in detail the two data augmentation techniques that were used in our experiments.

Chapter 5 presents the dataset, experimental design setup and methodology to perform data augmentation for the task of aspect level sentiment analysis.

Chapter 6 provides the results obtained from experiments in Chapter 5. In this Chapter, we also present a detailed discussion on various hyper-parameters and architectural design choices used to obtain our results.

Chapter 2

Background: Deep Learning

Deep learning has made a major impact in fields such as computer vision [9] and natural language processing [10]. Amongst other factors, the success of these models depends on training them using large datasets such as ImageNet [11]. Neural network models and algorithms that have contributed to this success were available since the 1980s; however, the availability of large scale datasets along with economical compute power, specifically from graphics processing units (GPUs), has been instrumental in the widespread success of these models. In this Chapter, we provide an overview of neural networks that is relevant to work presented in this thesis. We first introduce the multilayer perceptrons based neural network architecture, and later build on it to describe Recurrent Neural Networks (RNNs).

2.1 Multilayer Perceptrons

Artificial Neural Networks (ANNs) are a class of models that were inspired by the biological neural network and designed to mimic the information processing in the human brain [12]. ANNs are composed of processing units, referred to as nodes or neurons, that are connected to each other by weighted connections. The processing units are inspired by the neurons in a biological neural network and the weighted connections are analogous to synapses in the human nervous system. In feed-forward neural networks (FFNNs), information flows from the input layer to the output layer in a feed forward manner without forming any cycles between the nodes. A Multilayer perceptron (MLPs) is the most commonly used FFNN comprised of at least three layers: an input layer, a hidden layer and output later (see Figure 2.1). Mathematically, MLPs are function approximators that learn a mapping from the input to the output space.

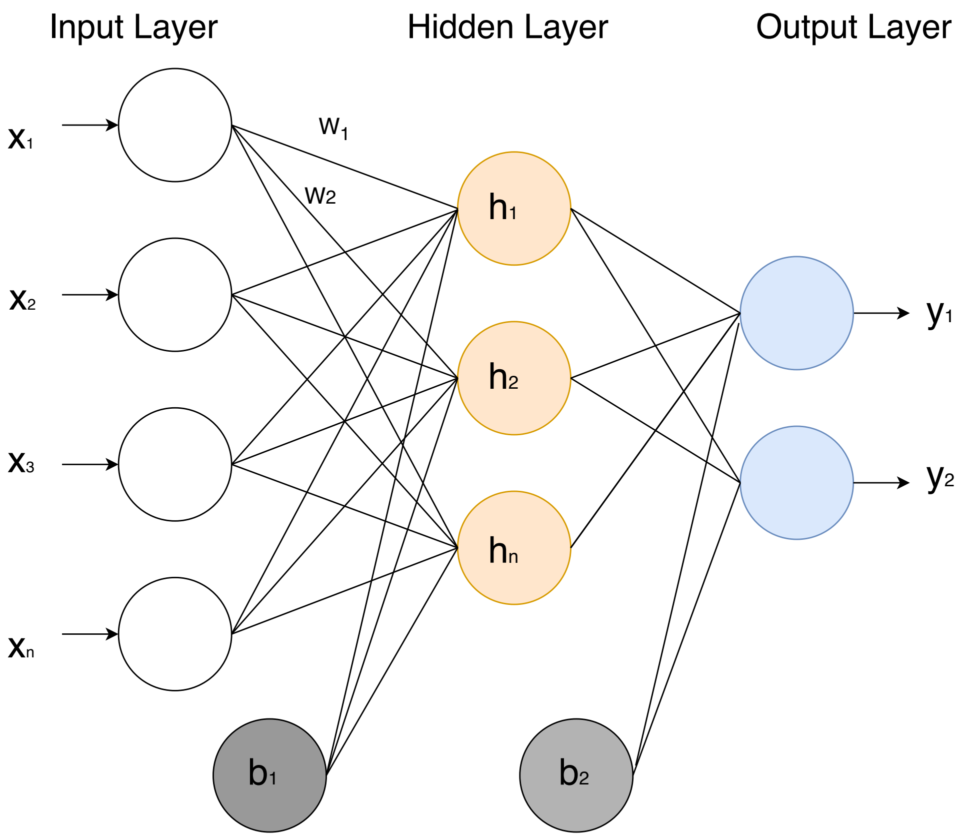


Figure 2.1: Multilayer Perceptron (MLP) with a single hidden layer

As per the Universal Approximation Theorem, a MLP with single hidden layer and sufficient number of neurons can approximate any arbitrary function [13]. In practice, it can be impractical to work with an arbitrarily large number of neurons in a single hidden layer and it is more common to use a MLP with multiple hidden layers. An MLP with multiple hidden layers, as shown in Figure 2.2, is also known as a deep neural network.

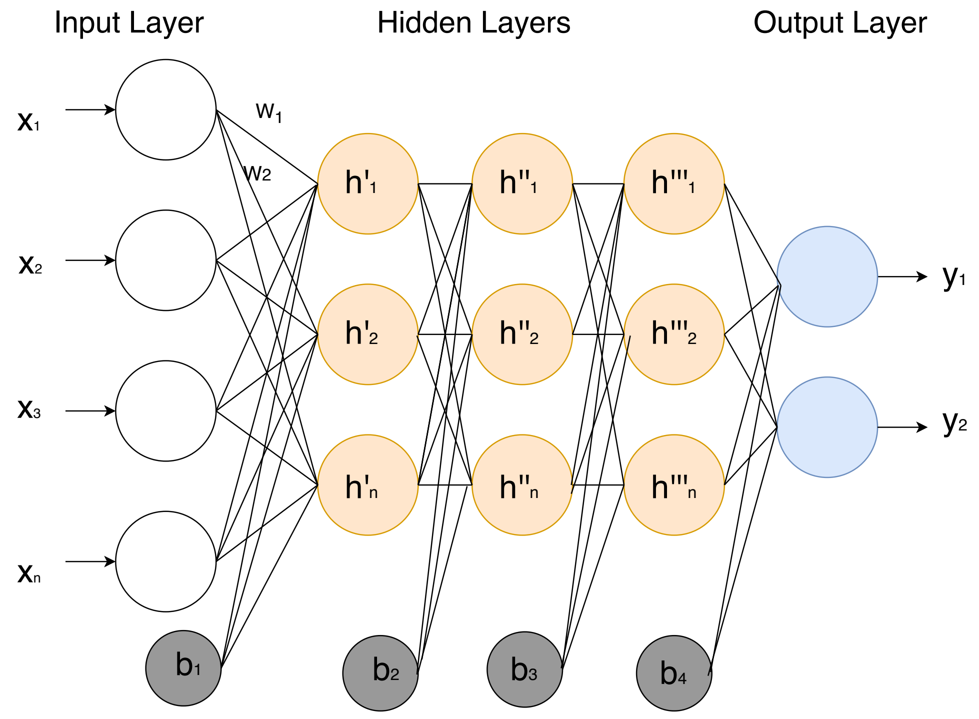


Figure 2.2: MLP with multiple hidden layers , also referred to as a deep neural network

2.2 Recurrent Neural Networks

Standard MLPs do not have the ability to encode sequential information that is necessary for problems in natural language processing (NLP), time series analysis, video generation etc. Recurrent Neural Networks (RNNs) [13] are a popular class of ANNs that provide the ability to process sequential information by factoring in the information from previous computations to calculate the output at any given time step. Figure 2.3 captures the flow of information at different time steps through a RNN.

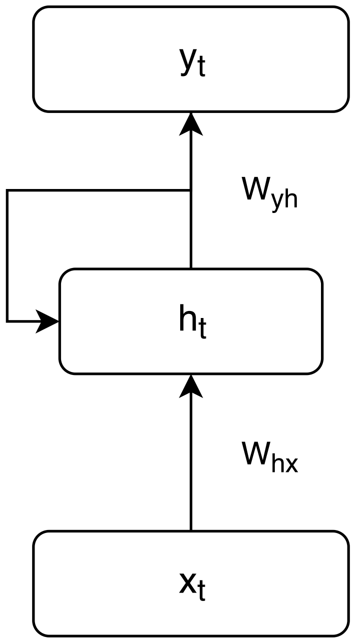


Figure 2.3: Figure demonstrating a typical architecture of a Recurrent Neural Network (RNN).

The hidden units in a RNN act as a *memory* that captures temporal information and propagates it to future time steps. The weight for the hidden unit at each step of a RNN is computed using the value of the hidden state from the previous time step and the input at the current step.

$$h_t = \tanh(W_{h_{t-1}} h_{t-1} + W_{hx} x_t). \tag{2.1}$$

Equation 2.1 describes the formulation used to compute the value of the hidden state, h_t, of a RNN; x_t is the input at the current time step, h_{t-1} is the value of the hidden state at the previous time step $t-1$, $W_{h_{t-1}}$ is the weight matrix at the previous time step, $W_h x$ is the weight matrix at the current time step. We use the *tanh* non linear activation function to squash the value of the hidden state between the bounds: -1 & 1, however a different activation function may also be used, for example a sigmoid or a Rectified Linear Unit (ReLU).

$$y_t = W_{yh} h_t. \tag{2.2}$$

The output, y_t at a given time is then computed using Equation 2.2. Here, W_{yh} refers to the weight matrix at the output state at time t.

The training process of a RNN is similar to that of a MLP except that the derivatives computed at each output step are propagated back to the beginning of the sequence using an algorithm called backpropagation through time (BPTT) [14, 15]. In theory, a RNN could encode a sequence of any length; however, in practice they are limited to a few time steps because the performance degrades on longer sequences as a result of vanishing or exploding gradients [16].

2.3 Long Short Term Memory Networks

Long Short Term Memory Networks (LSTMs) [17] are a class of RNNs designed to overcome the problem of vanishing or exploding gradients in RNNs and to be more effective at learning long term dependencies as compared to a regular RNN [18]. Similar to RNNs, LSTMs use the value of previous hidden unit and the new input to compute the current hidden state. However, LSTMs use a gating mechanism that scales input values between 0 and 1 to control how much of the incoming values influence the current state. In addition, LSTMs have an internal memory unit that controls how much information is *remembered* at the subsequent time steps. Figure 2.4 shows a LSTM unit with all the gates and how the information flows at a given time step.

Equations 2.3 – 2.7 outline how the different gates in the LSTM unit described in Figure 2.4 influence the computation of parameters in the hidden layer and the output at the given time step.

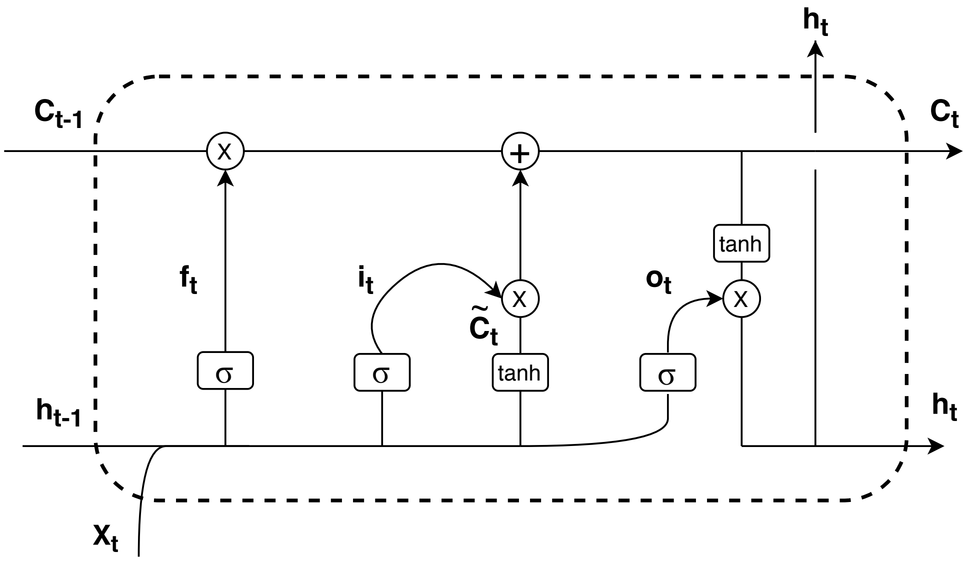


Figure 2.4: Illustration of a typical LSTM module unit.

$$f_t = \sigma(F_w \cdot [h_{t-1}, x_t] + b_f). \tag{2.3}$$

$$i_t = \sigma(I_w \cdot [h_{t-1}, x_t] + b_i). \tag{2.4}$$

$$C_t = \sigma(f_t * C_{t-1} + i_t * \tanh(C_w \cdot [h_{t-1}, x_t] + b_C)). \tag{2.5}$$

$$o_t = \sigma(O_w \cdot [h_{t-1}, x_t] + b_o). \tag{2.6}$$

$$h_t = o_t * \tanh(C_t). \tag{2.7}$$

In the equations above:

- x_t refers to the input vector to the LSTM at time step t
- h_{t-1} refers to the hidden vector from the previous time step t
- C_{t-1} and C_t refers to the cell states of the LSTM at time steps $t-1$ and t
- h_t refers to the hidden state of the LSTM computes at time step t

- f_t, i_t and o_t refers to the values computed by the forget, input and output gates of the LSTM at time step t
- F_w, I_w and O_w refers to the learnt weight matrices corresponding to the forget, input and output gates of the LSTM

In Equation 2.3, the forget gate, f_t, regulates what information from the previous cell state will be forgotten or removed from the cell state using the sigmoid activation function. When the sigmoid is nearly 0 this implies that information from the previous cell state, C_{t-1}, is forgotten whereas the sigmoid close to 1 makes the network retain all the information from the previous state. In Equation 2.4, the input gate, i_t, regulates what information will be stored and eventually used to update the cell state. It also uses the sigmoid function to control the flow of information. In Equation 2.5, we calculate the cell state, C_t, by using the outputs of the forget and input gate. Here, we calculate the product of f_t and C_{t-1} which is then summed with the product of i_t and the output of the *tanh* activation function that processes the input and the previous hidden state. In Equation 2.6, we use a sigmoid activation to process the input and previous hidden state through the output gate. In Equation 2.7, we use the product of the value from the output gate and the *tanh* of the cell state to yield the desired information from the LSTM.

2.4 Summary

In this chapter, we review neural network architectures used in the experiments described in Chapters 5 and 6. We introduced the MLP, a simple neural network, followed by RNNs and LSTMs, a popular class of neural networks for modeling sequential data.

In the next chapter, we provide a general overview of successful applications of deep learning to problems in natural language processing and later focus on sentiment analysis. We also introduce word embeddings which have been an effective tool to provide a vectorized representation of textual information that are used as input by deep learning models.

Chapter 3

Background: Deep Natural Language Processing

Deep learning techniques have shown improved performance over traditional machine learning methods in many natural language processing tasks such as sentiment analysis, machine translation and natural language generation [19, 20, 21, 22]. One of the factors that led to this success has been the effective use of unsupervised learning to generate distributed word representations, also known as word embeddings, that provide a way to encode textual information and act as inputs to deep learning models. In this chapter, we discuss the relevant details about word embeddings and provide an overview of popular word embedding models. We also review some recent successes of deep learning in natural language processing (NLP)and also provide specific examples for the problem of sentiment analysis as well as aspect level sentiment analysis.

3.1 Deep Learning for Natural Language Processing

The success of deep learning based models was first demonstrated by AlexNet winning the ImageNet challenge in 2012 [23]. Machine learning and NLP researchers have continued to develop deep learning based techniques to improve state of the art performance on benchmark datasets. Recurrent Neural Networks (RNNs), and specifically Long Short Term Memory (LSTMs), were the natural choice for NLP problems due to their ability to process sequential information. Early successes were achieved on problems focused on speech recognition [24, 25, 19] as well as machine translation [26, 20]. Over time, researchers developed novel techniques to use purely CNN based architectures to develop solutions for problems in NLP [20, 27]. Most recently, attention based architectures, such as the transformer [22] and the deep bi-directional transformer(BERT) [21], have been the models of choice due to their state-of-the-art performance on multiple benchmark datasets such as the Stanford Question Answering Dataset (SQuAD v2.0) [28] and Multi-Genre Natural Language Inference (MultiNLI) [29]. Both transformer and BERT architectures were designed such that they were easy to parallelize to speed up the model training process. Neither model involves any recurrent or convolutional operations.

The textual data used as input for many of these deep learning models is encoded using dense vector representations, also known as word embeddings. These embeddings, described in the next section, provide a computationally efficient way of representing textual information while capturing semantic and syntactic information to help boost the performance of deep learning models.

3.2 Word Embeddings

The components of natural language, such as words, sentences and punctuation, that act as input to machine learning are discrete in nature. Hence, the first step involved in most machine learning tasks related to language is to encode the text into a vector representation that can be used as input to the model. Text can be encoded in the form one hot vectors where the length of the vector representation of each word corresponds to the size of the vocabulary. For example, the one hot vector representation of each word in a vocabulary of 10,000 words would be a 10,000 dimensional one hot vector. However, one hot vectors are high dimensional which makes them sparse and computationally inefficient to process.

To mitigate the issues caused by high dimensional one hot vectors, word embeddings have become an effective alternative to represent text using dense vector representations. In case of word embeddings, each word in the 10,000 word vocabulary could be represented as an arbitrary length n dimensional vector in a shared n dimensional space which mitigates the dimensionality issue. Word embeddings used by deep learning models are used in one of two ways:

1. The word vector is initialized randomly or from a uniform distribution in the embedding layer of the neural network and the value for the word vector is learnt jointly with the main task that the model is trained for.
2. The word vector in the embedding layer is set using the value of pre-trained word embeddings learnt on another dataset and/or task. The value in the embedding layer is either frozen i.e., stay the same throughout the training process or updated as a part of the neural network training process with the objective to fine tune the word embeddings for a particular task and dataset.

The primary advantage of word embeddings, either pre-trained or those obtained at the end of training a network, comes from their generalization power; a *good* learning algorithm will result in similar features having similar vectors [30]. Another advantage, discussed earlier, is that these dense vectors provide a more computationally efficient alternative to sparse one hot encoding. Below are some of the most efficient techniques for obtaining word embeddings.

3.2.1 Word2Vec

Word2Vec is a neural network model, first introduced by Mikolov et al. [31] that is used for learning word embeddings. Before Word2Vec, count based methods were a popular choice to for obtaining vector representations. These methods computed the statistics on how many times certain words co-occurred in the given vocabulary and mapped these counts to a small dense vector for each word. However, Word2Vec introduced a predictive approach to the problem.

Word2Vec is a computationally efficient predictive model for learning word embeddings from a given corpus of raw text. In Word2Vec, the word embeddings for a given word are predicted based on the neighbors of the words in terms of learned small, dense embedding vectors i.e., as parameters of the model. For each word, the parameters are learned and its embeddings is calculated. If two words are similar in context, then their embeddings should be close in the higher dimensional embedding space. Parameters of the model are thus adjusted in a way to obtain exactly this scenario. Specifically, Word2Vec is a two dimensional neural network model which produces a 300-dim vector for each word. A popular example in the literature demonstrating this similarity in value of learnt embeddings is that the vector corresponding to *king* - *man* + *woman* is approximately equal to the vector embedding of *queen* i.e. *king* - *man* + *woman* $\approx$ *queen*. This is demonstrated in Fig 3.1, which illustrates how some of these vectors would ideally be seen in the embedding space.

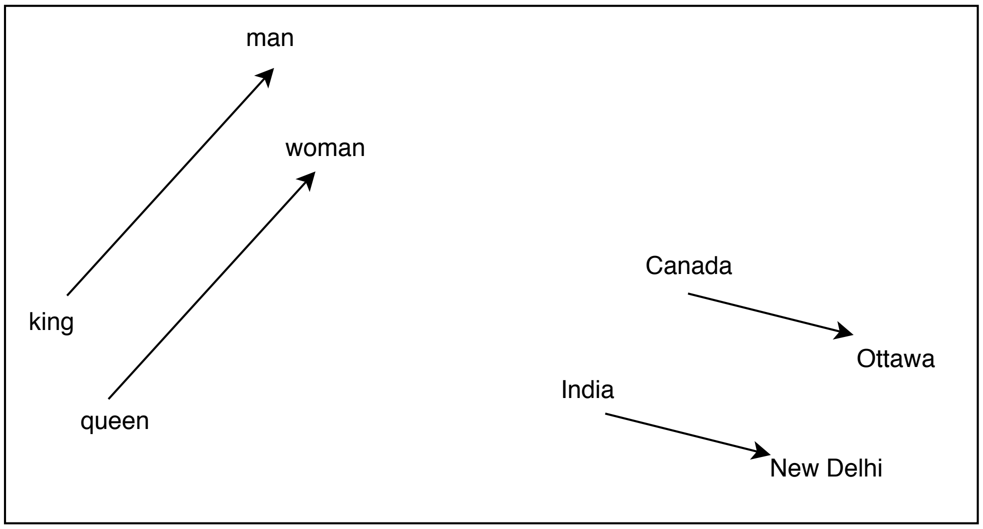


Figure 3.1: An example of Word2Vec vectors in the embedding space. Similar concepts are grouped together in the embedding space.

3.2.2 GloVe

Global Vectors for word representation (GloVe) is an unsupervised learning algorithm for obtaining vector representations for words. In contrast to Word2Vec which relies on the local context information between words, GloVe uses global context via word co-occurrence statistics from the entire given text corpus. The GloVe algorithm consists of the following steps:

- First, an occurrence matrix X is created which contains elements X_{ij} representing how often a word j occurs along with the word i. In practice, to create the co-occurrence matrix elements, a window size is selected first. Given the window size, for a word i, we scan the window size area before and after the word i in the sentence to get context terms. Weights for the particular context terms are decided based on how far they are from the word i using an offset function. The closer the context term to the given word, the higher the weight.

- Next, a soft constraint for each pair of word ij is decided using the weight vector W of main word i and context word j, long with their respectively bias terms b:

$$\log(X_{ij}) = W_i^T W_j + b_i + b_j. \tag{3.1}$$

- Then, the embeddings are trained using the following objective function:

$$\theta = \sum_{i}^{V} \sum_{j}^{V} f(X_{ij})(W_i^T W_j + b_i + b_j - \log(X_{ij}))^2. \tag{3.2}$$

  Here, V is the size of the vocabulary and f is the weighting function that act as a regularizing function to help prevent learning from edge cases. In their seminal paper [32], the creators of GloVe describe the properties that must be obeyed by the weighting function. Equation 3.3 is the weight function used in Equation 3.2 where the authors recommend setting the value of α to 3/4.

$$f(x) = \begin{cases} (\frac{X_{ij}}{x_{max}})\alpha & \text{if } x_{max} \geq X_{ij}, \\ 1 & \text{otherwise} \end{cases} \tag{3.3}$$

Equation 3.1 to 3.3 describe how GloVe is able to incorporate more meaningful embeddings with global context as compared to Word2Vec.

3.2.3 Contextual Word Embeddings

Word2Vec and GloVe both provide a more efficient way to densely encode text into word vectors compared to earlier approaches that used sparse one hot encodings. One of the limitations of these techniques is that there's one, *context independent*, word vector associated with each word i.e. the same word vector is used to encode a word even though the word could be used in a different context. For example, the same word vector would be used to represent the word *broke* in *"I broke a glass"* and *"thieves broke into his house"* even though the word has a different meaning in those sentences. Approaches such as *context2vec* [33] have been introduced in the past to precisely tackle this problem. However the strategy by Peters et al. [34] to learn embeddings described jointly while training a language model has become a popular method to learn context-specific embeddings. The embeddings by Peter et al., referred to as *Embeddings from Language Models (ELMo)*, are functions of the entire input sentence to enable them to encode the entire context of the given sentence. The authors demonstrated the effectiveness of ELMo by adding it to known state-of-the-art techniques on six benchmark tasks and further improving the model's performance on each of these task.

Although contextual word embeddings have been shown to outperform other techniques on multiple benchmark datasets, to the best of our knowledge, there is no theoretical guarantee that using them would give the best model performance on any dataset. Moreover, they are a part of the solution and ultimately the model performance on a given task is also dependent on the choice of neural network architecture as well as the rigor with which hyperparameter optimization is performed. For most deep learning based models used for NLP problems, such as sentiment analysis, the choice of using pre-trained word embeddings or training them from scratch could be seen as a design choice.

In the next two sections, we first provide an overview of sentiment analysis followed by a detailed overview of aspect level sentiment analysis.

3.2.4 Sentiment analysis

Sentiment analysis has been a major topic of interest for researchers focused on linguistics as well as machine learning research. Sentiment is most commonly computed at one of three levels [35]:

1. *Document level sentiment*: Aggregate sentiment refers to overall sentiment expressed in the text that is analyzed. The sentiment here can be computed for text ranging from a few sentences to an entire article or document. In practice, this is useful in scenarios where the information being analyzed refers to a single item or subject. Aggregate sentiment might not be very useful in cases where multiple items or subjects are discussed. For example, a news article discussing a new law that would positively affect certain industries or groups of people and negatively affect some others.
2. *Sentence level sentiment*: Sentence level sentiment analysis refers to the computation of sentiment expressed in a single sentence. A popular example of this can be the case of restaurant or product reviews where customer or users express whether or not they had a good experience.
3. *Aspect level sentiment*: Aspect level sentiment analysis, also known as entity level or fine-grained sentiment analysis, refers to computation of sentiment with respect to a target. For example, in the sentence *"The food was good but the service was poor"*, the sentiment is positive with respect to the target term *food* and negative with respect to the target term *service*.

Each of the three sentiment computation problems above have their own technical challenges and methods successful for one task will not always work on the other. For example, a challenge with document level sentiment analysis is that the amount of text to be processed to compute the sentiment is much larger than that in a single sentence. Hence, most LSTM based approaches for sentence level sentiment fail at the document level due to problems of vanishing or exploding gradients.

In the next section, we provide an overview of aspect level sentiment analysis and some of deep learning methods that have been designed for this problem.

3.2.5 Aspect level sentiment analysis

Aspect level sentiment analysis is the problem of computing sentiment with respect to a target term. It is a two part problem in case of practical applications. The first part of the problem is aspect or target detection and the second is to compute the sentiment with respect to that aspect. In this thesis, we focus on the second part of the problem and use datasets where the entity term is already identified. One of the challenges for this problem is to identify and model the effect of words in the sentence that affect the sentiment expressed towards the target term. For example, a sentence might have two entities and the sentiment expressed towards one might be positive and negative towards the other one; to correctly classify the sentiment, the model would need to identify that the words with negative sentiment are targeted towards one entity and that the words with positive sentiment are targeted towards the other.

A lot of neural network based solutions have been published over the past few years to improve model performance on this problem. Although the performance of deep learning based solutions has improved over time, there is no single technique that could be seen as the dominant choice for aspect level sentiment analysis [35]. Dong et al.'s [4] *adaptive neural network* is amongst the early neural network based solutions used for aspect level sentiment analysis. Subsequent neural network based techniques introduced to tackle the same problem predominantly used end-to-end LSTM based approaches [36]. Some approaches with a greater degree of sophistication also incorporated the use attention and memory [37] in their networks [38].

3.3 Summary

In this Chapter, we first introduced the different types of word embeddings that have been used in the field. Specially, we discussed three major distinctive approaches for obtaining word embeddings that incorporate different level of contextual information in their respective higher dimensional vector space. As the thesis uses aspect level sentiment analysis task for analysis, later in Chapter 5, we also described the various levels of sentiment analysis including aspect level sentiment analysis. The next Chapter will present an overview of data augmentation and review various methods and approaches to data augmentation for problems in computer vision and natural language processing.

Chapter 4

Background: Data Augmentation

Data Augmentation (DA) refers to the expansion of a dataset using different transformations for inducing enough variability in the dataset samples for learning machine learning models that generalize well. Traditionally, this is done by applying domain-specific techniques to examples from the dataset that create new and different training examples. For background completeness, and historical context, we start by introducing data augmentation techniques used in the domain of computer vision.

4.1 DA in Computer Vision

Data augmentation is one of the most widely known domain-specific regularization method in the field of computer vision. Data augmentation within this domain refers to the various transformations performed on images to create a modified images to augment the training dataset. Such transforms typically involve a range of operations including shifts, vertical and horizontal flips, zooms etc. We use Fig 4.1 as a sample image and demonstrate the effect of data augmentation by applying each technique in subsequent images.

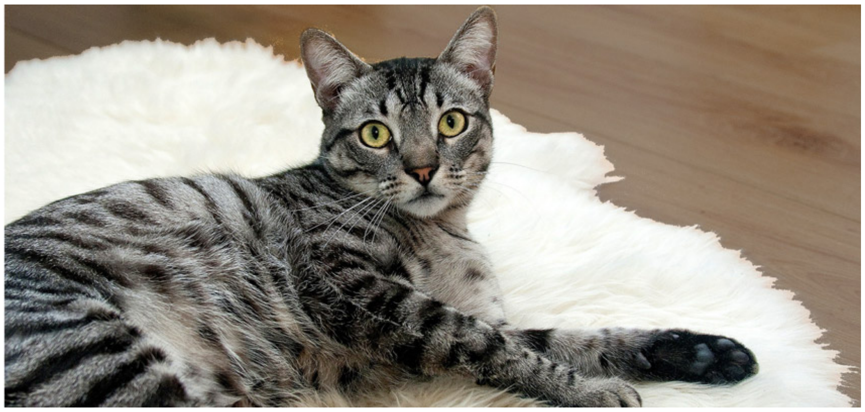


Figure 4.1: Sample image [1] on which we will perform standard data augmentation techniques

- Shift augmentation: In this transformation, all the pixels in a given image are moved in one particular direction. While shifting the pixel in a direction, certain part of the image is cut off and there is a certain region that requires new pixel values to be assigned. When the pixels are shifted in horizontal or vertical direction, it is called horizontal or vertical shift transform respectively. Fig 4.2 and Fig 4.3 demonstrates the effect of horizontal and vertical shift augmentation applied to Fig 4.1.
- Flip augmentation: In this technique, the given image is transformed by flipping the image either horizontally or vertically. Essentially, in this transformation the rows or columns in a given image are flipped. If the rows are flipped, it is known as vertical flip augmentation. In case of columns, it is know as horizontal flip. Fig 4.4 demonstrated the effect of horizontal flip augmentation applied to Fig 4.1.
- Rotation augmentation: In this transformation, the image is rotated by a certain angle in clockwise or anti-clockwise direction. Any area lying outside of the original image dimensions is generally cut-out from the image. Fig 4.5 demonstrates the effect of rotation augmentation applied to Fig 4.1.
- Zoom augmentation: A zoom augmentation allows to zoom in a certain part of the image. This leads to either adding new pixel values around the image or interpolate the pixel values. The amount of zoom is uniformly sampled from the selected region for width and height separately. Fig 4.6 demonstrates the effect of zoom augmentation applied to Fig 4.1.

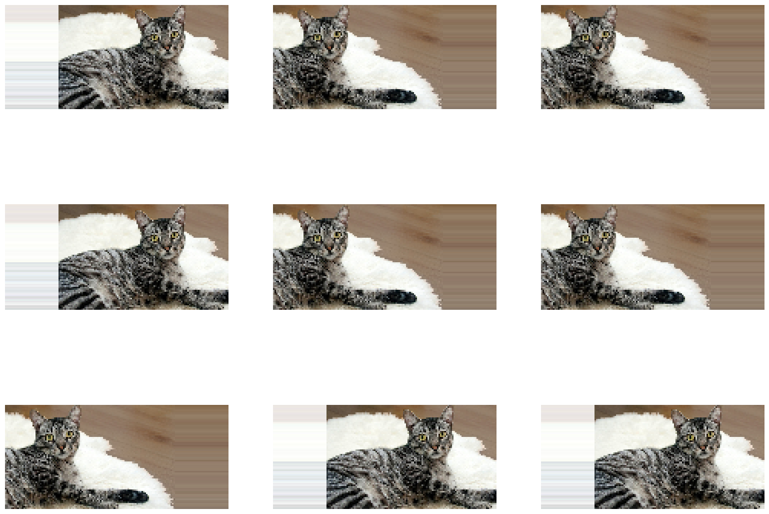


Figure 4.2: Various levels of horizontal shift augmentation applied to Fig 4.1

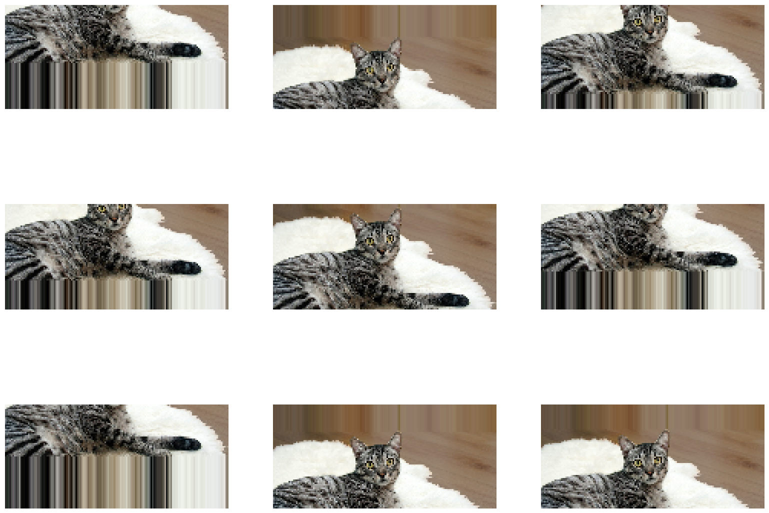


Figure 4.3: Various levels of Vertical shift augmentation applied to Fig 4.1

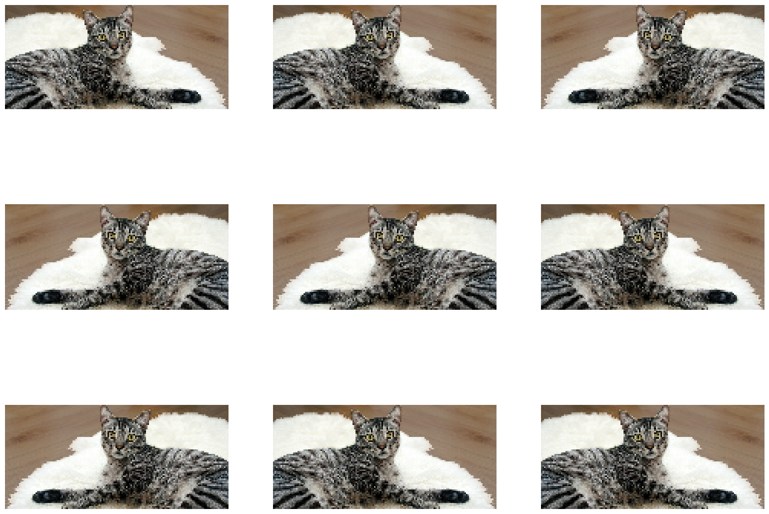


Figure 4.4: Flip augmentation applied to Fig 4.1

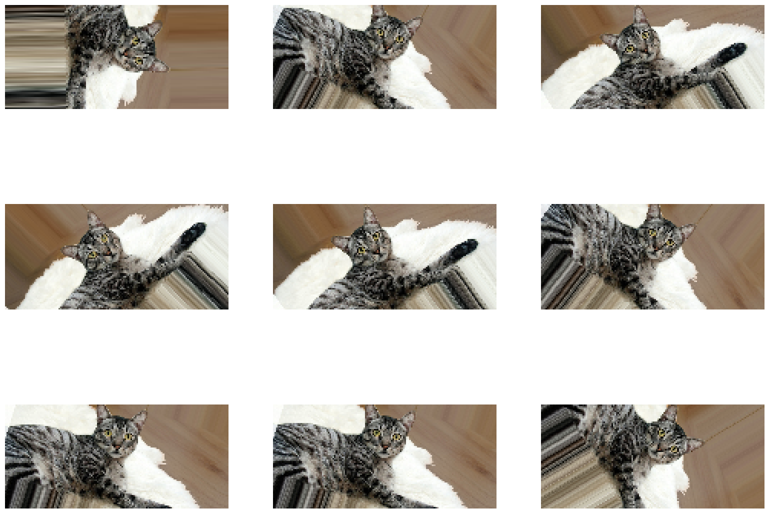


Figure 4.5: Various degrees of rotation augmentation applied to Fig 4.1

- Brightness augmentation: In this data transformation, the pixel values of a given image are changed to make the image appear more or less dark. This is done to allow the model to adjust to different lighting levels. Fig 4.7 demonstrates the effect of brightness augmentation applied to Fig 4.1.
- Cutout [39]: This technique was developed to improve model performance on occluded examples as

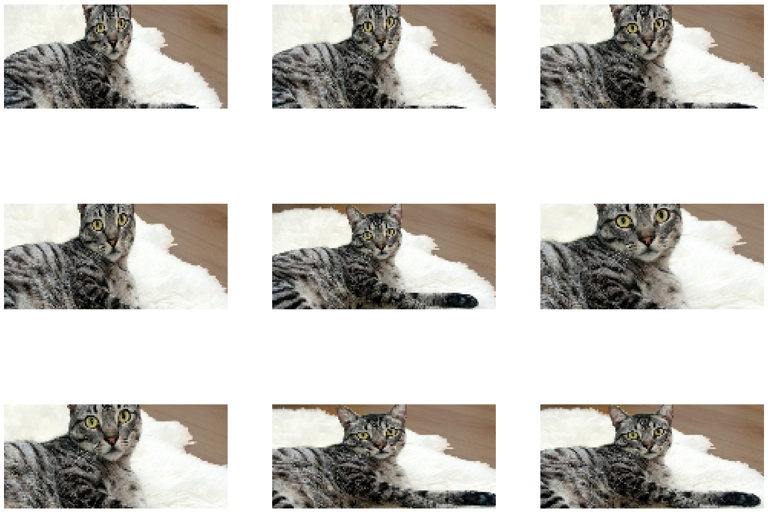


Figure 4.6: Zoom augmentation applied to Fig 4.1

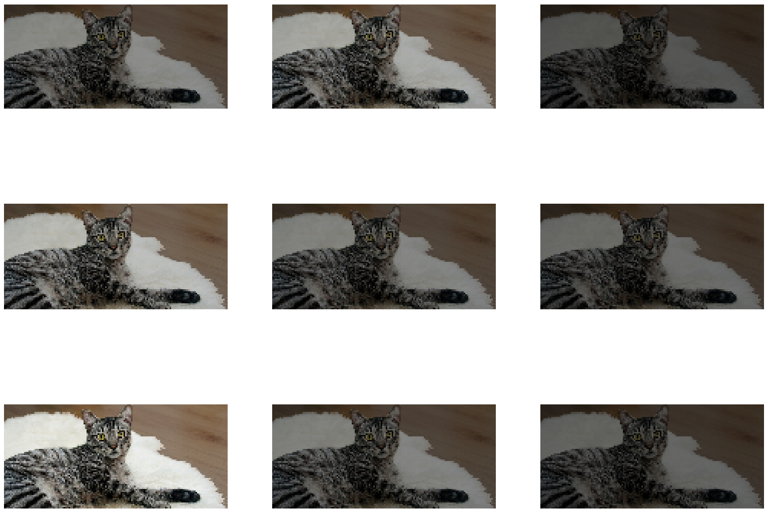


Figure 4.7: Shift augmentation applied to Fig 4.1

a way to regularize and thereby improve model performance simultaneously. Removing contiguous patches from input images and then using the modified images as training data was demonstrated to be an effective training strategy to improve the model performance. Although similar approaches have been used in the past as a technique to corrupt the images, the authors of cutout were the first to demonstrate its effectiveness in improving model performance that were trained in a supervised setting. The authors hypothesized that removing important visual features would be one way to improve performance as it would force the network to focus on the "less important" features and incorporate more of the image and its context when making decisions. This was implemented using targeted cutout and its implementation details are described in [39]. Through more experiments, the authors concluded that randomly applying cutout was equally effective as targeted cutout. Therefore the additional computational overhead associated with targeted cutout didn't seem to add any extra value during training. The authors discovered that that model performance was sensitive to size of the cutout patch so they treated it as a hyperparamter and only used square patches for their cutout masks.

4.2 DA in Natural Language Processing

In the NLP domain, DA has been shown to improve the performance of models trained with DA over the models trained without it. However, in the NLP domain, it is difficult to augment text by simply changing few characters (like pixels in image) due to the complex structure of language. Even changing a word sometimes changes the context of the sentence. Hence, DA in NLP is quite different as compared to computer vision.

Some of the most common DA techniques employed in recent times for different NLP tasks are explained in detail below:

- Thesaurus: Zhang et al. [40] proposed this type of augmentation technique where a particular word or words are replaced by their synonyms. To find the synonyms, a thesaurus is used hence the name thesaurus augmentation. Usually the selection and replacement is done using a particular geometric distribution.
- Back-Translation [3]: In this augmentation, a pre-trained model is used to translate from one language to another and then back translated to the original one by using the same or other pre-trained

model. For example, translating a sentence from English to Korean using a Neural Machine Translation (NMT) model and then back from Korean to English. This usually injects noise in the original sentence thereby creating a new example.

- Word Dropout [41]: In the word dropout data augmentation, word or words in the input sentence are dropped randomly. For example, for the original sentence "This is a cat", using word dropout, a new sentence is generated: "This is cat".
- SwitchOut [8]: Unlike word dropout, a set of words in a given input sentence is randomly replaced instead of dropped with a probability p from other words in the available vocabulary. This creates a new sample with noise that is shown to boost the performance of model. In tasks such as translation, words in both input (source) and output (target) sentences are replaced (switched) from their respective vocabularies. More details regarding this technique is presented in the next Section.

The list above mentions some of the most common data augmentation methods. These methods can be categorized as input augmentation techniques i.e. they create new examples by modifying the input data. However, recently the methods pertaining to augmentation in the higher dimensional representation space such as manifold mixup in computer vision are becoming popular. These techniques have shown to outperform many other methods.In the next section, we cover some popular representation augmentation techniques as well as SwitchOut in detail.

4.3 Input and Representation Augmentation

Data augmentation has traditionally been performed in the input space on given input examples such as method related to back-translation or thesaurus. However, recently methods performing data augmentation of the hidden vector representation i.e. in a higher dimensional representation space are gaining popularity. In these methods, augmentation is performed on the hidden vector representations obtained at various different layers of models such as neural networks. Below, we describe in detail four popular techniques that are relevant to our research. Mixup [42] and SwitchOut [8] are performed in the input space whereas manifold mixup [7], which used mixup, and DeVries and Taylor [2] is performed in the representation space.

4.3.1 DeVries and Taylor: Dataset Augmentation in Feature Space

DeVries and Taylor [2] was motivated by the need to create a task-agnostic data augmentation technique that would not require domain expertise for effective application. The authors hypothesised that manifold unfolding in feature space would result in superior quality of synthetic training examples after performing data augmentation on the feature vectors. This was the first approach to successfully perform data augmentation using higher level representations of training data by manipulating the learned feature vectors corresponding to the training data. In this study, the authors investigated the effect of noise injection, interpolation between feature vectors as well as extrapolation between feature vectors. The feature vectors associated with the training data were learnt in an unsupervised setting by training a LSTM-based sequence autoencoder. The sequence autoencoder is used to obtain feature vectors that mapped the input data into a high dimensional feature space. After training the sequence encoder, the context vector associated with the training data was used as its feature vector. New training examples are then generated by applying transformation to the feature vectors. In their seminal paper, the authors described in detail their training technique and include information about hyperparameters used to perform data augmentation in each of the three cases. DeVries and Taylor discovered that extrapolation between feature vectors was effective in generating training examples that improved model performance on downstream tasks. However, noise injection and interpolation between feature vectors did not help in improving model performance. Mixup [42], described next, is another task-agnostic data augmentation technique that was introduced shortly after DeVries and Taylor.

4.3.2 Mixup

Mixup was introduced by Zhang et al. [42] as a regularization strategy for neural networks by training them on convex combinations of pairs of input examples and their corresponding labels. In mixup, two different input representations are “mixed” by using a λ factor. Given two different input pairs of representation (x_1, y_1) and (x_2, y_2), where x is the input representation (image,speech text) variable and y is the target variable, in mixup we do the following:

1. We first choose a factor λ for random mixing from a Beta distribution
2. Once λ selection is done, an artificial new sample pair is created as:

$$(x^*, y^*) = (\lambda x_1 + (1 - \lambda)x_2, \lambda y_1 + (1 - \lambda y_2)) \tag{4.1}$$

3. Using the original and artificial pairs, a given network is trained using the following loss function:

$$\mathcal{L}(\theta) = \mathbb{E}_{x_1,y_1 \sim p_{train}} \mathbb{E}_{x_2,y_2 \sim p_{train}} \mathbb{E}_{\lambda \sim \beta(0.1)} \ell(\lambda x_1 + (1 - \lambda)x_2, \lambda y_1 + (1 - \lambda y_2)) \tag{4.2}$$

Mixup is performed by interpolating between two different input pairs to create a new data point. The intuition behind this technique is that through such interpolations the network is able to learn the smooth transitions between various data points thereby avoiding sharp transitions.

In the introductory paper, Zhang et al. [42] showed that mixup as a representation data augmentation, was able to boost the performance of the network with few percentage points irrespective of the domain or modality of the input data. From the theoretical side, they also proved that mixup as a data augmentation technique acted implicitly as a empirical risk minimization method that leads to better generalization. Hence, it is an important technique in the representation augmentation space that later inspired other techniques in the this space. One such technique is manifold mixup, which is explained in the next section.

4.3.3 Manifold Mixup: Better Representations by Interpolating Hidden States

Manifold mixup was introduced by Verma et al. [7] that extended the use of mixup [42] to interpolation in the representational feature space as compared to the original technique that performed interpolation in the input space. The authors were motivated to address the problem of poor neural network performance when tested on data that came from a different distribution when compared to the training data. As a result of poor generalization, these neural network models provided incorrect predictions with high confidence when tested on adversarial examples or another example from test sets that came from a different distribution compared to the training dataset. The authors observed that resultant decision boundaries margin were too small and the space on and off the data manifold consisted mostly of high confidence predictions.

The main idea behind this technique consists of leveraging the higher level representations learnt by layers of the network to generate new training data. This is done by creating new training examples that were computed using linear interpolation of feature vectors obtained from intermediate layers of the neural network. For implementation, the first step is to select a network layer where the data augmentation will be performed. During training, two mini batches are processed in the normal way until it reaches the selected layer. Next, input mixup is applied to the feature vectors corresponding to examples from each mini batch. The rest of the network layers use the augmented feature vectors to continue the training process. At the output layer, loss is computed between the predicted output and the augmented label created using manifold mixup. The augmented output labels are computed by interpolating between the one hot labels of the respective training examples.

By means of experiments described in [7], the authors demonstrated improved generalization of the resultant models because:

- The decision boundaries of these models were further away from the training data i.e., they now have larger margins.
- They successfully leveraged high level information by interpolating at the intermediate levels of the network as opposed to the traditional techniques that performed data augmentation at the input level.
- This technique essentially acts as a compression method since the augmentation results in flattened representations that have significantly less variance when compared to the input dimensionality. Thus, mixup is done on "important" features leading to better generalization as proved theoretically

by Tishbhy and collaborators in [43, 44] and extensive experimentation by [45, 46, 47, 48].

4.3.4 SwitchOut: an Efficient Data Augmentation Algorithm for Neural Machine Translation

SwitchOut was introduced by Wang et al. [8] as a data augmentation technique for neural machine translation. This technique can be seen as a natural extension of word dropout where a word or set of words in a given sentence are replaced with a word or set of words that are sampled from the vocabulary. This could be seen as an equivalent of noise injection in training example to improve model performance. In the case of machine translation, words from both input sentences (source) as well as output sentences (target) are replaced (*switched*) with words sampled from their respective vocabularies. This technique was able to achieve performance improvements of 0.5 BLEU on three different datasets: IWSLT'15 English-Vietnamese data with about 131,000 training pairs [49], IWSLT'16 German-English data with about 197,000 training pairs [50] and WMT'15 English-German dataset with about 4.5 million training pairs [51]. Algorithmic sampling using SwitchOut on the source sentences is done as described in Algorithm 1.

Algorithm 1 SwithOut Algorithm

```
Input: sentence $s$, vocabulary $v$ & temperature $\tau$.
Output: switchouted sentence $\hat{s}$
$Sample\, w \sim p(w);\, p(w) \leftarrow e^{-w} / \sum_0^{|s|} e^{-w/\tau}$ for $w = 0, 1, \ldots, |s|$*number of words $w$ to be corrupted
$Sample\ p_i \sim$ Bernoulli$(w/|s|)$*sampling the positions $p$ for replacement
$Sample\ v_i \sim$ Uniform$\{1, 2, \ldots, |v| - 1\}$*sampling replacement words from vocab $v$
for i = 1 to len($s$) do
    if $p_i$ ==1 then  $s_i = v_i$
```

In our experiments for aspect level sentiment analysis, SwitchOut was limited to source sentences and the sentiment labels were left unchanged.

4.4 Summary

This chapter introduced various data augmentation methods that have been used in the domain of computer vision and natural language processing. For the data augmentation related to NLP, we first described the various general techniques used in the literature currently. Table 4.1 summarizes the different data augmentation techniques that are described in this chapter and classifies them based on where the augmentation is performed i.e. input vs. representation augmentation and whether it created new training data by globally transforming, blending samples or performing local substitution. As this thesis aims to explore the effect of data augmentation in representation space, later in the Chapter we also describe in detail techniques related to representation based data augmentation.

Table 4.1: Summary table classifying the type of augmentation performed by each technique discussed in Chapter 4

Data Augmentation Technique	Input			Representation
	Global transform	Local substitution	Example blending	Example blending
Shift Aug.	✓			
Flip Aug.	✓			
Rotation Aug.	✓			
Zoom Aug.	✓			
Brightness Aug.	✓			
Cutout		✓		
DeVries & Taylor				✓
Mixup			✓	
Manifold Mixup				✓
Thesaurus		✓		
Back-Translation	✓			
Word Dropout		✓		
SwitchOut		✓		

In the next Chapter, we describe the experimental setup used to establish the effects of chosen data augmentation methods, manifold mixup and SwitchOut, on the classification accuracy of LSTMs.

Chapter 5

Data Augmentation for Aspect Level Sentiment Analysis

In the previous three chapters, we presented a detailed account of the background and relevant work describing applications of deep learning in natural language processing. We also introduced some popular data augmentation techniques that were developed to boost the performance of deep learning models for different tasks in computer vision and natural language processing. In this chapter, we introduce the dataset that was used for our experiments and implementation details about how relevant baseline results from Tang et al. [36] were replicated. Next, we present carefully designed experiments for two data augmentation techniques, manifold mixup and SwitchOut. The experiments are designed to incorporate these techniques in the training process and analyze their effect on the performance of deep learning models used for aspect level sentiment analysis.

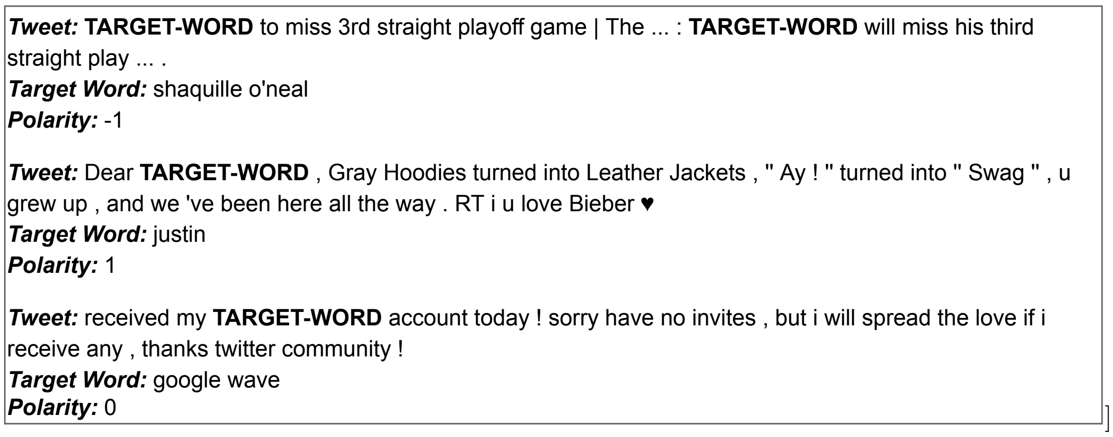
5.1 Dataset

To study the effect of our chosen techniques on the model's classification accuracy, the experiments were conducted on a benchmark dataset [4]. This was the largest manually annotated dataset at the time for aspect level sentiment analysis created from Twitter. For clarity and readability, the dataset is referred to as "Twitter sent API" dataset for the rest of the study. The dataset was collected through the Twitter API and manually annotated to provide sentiment polarity. The tweets were collected by querying the Twitter API with different tags such as celebrity names and product brand names. The tweets were then manually labeled and randomly sampled for different sentiment classes to avoid dataset imbalance. The sentiment in the resulting dataset is comprised of 25% negative, 50% neutral and 25% positive tweets. The training set contained a total of 6,248 tweets while the test set contained 692 tweets for the three different sentiment polarity classes. Table 5.1 summarizes the number of examples for each class in the training and test sets.

Table 5.1: Number of examples in the train test for each of the three classes

	Positive	**Negative**	**Neutral**
Training Set (6248)	1562	1562	3124
Test Set (692)	173	173	346

Each instance in the dataset is split into three lines that include the sentence (tweet), the target term and the polarity of the tweet. The target term in the tweet is replaced with the term, $TARGET - WORD$, to indicate its position in the sentence. The polarity of neutral, positive and negative sentiments with respect to the target term is expressed using numeric values 0, 1 and -1 respectively. Some examples from the dataset are shown in Figure 5.1.



Tweet: **TARGET-WORD** to miss 3rd straight playoff game | The ... : **TARGET-WORD** will miss his third straight play ... .
Target Word: shaquille o'neal
Polarity: -1

Tweet: Dear **TARGET-WORD** , Gray Hoodies turned into Leather Jackets , " Ay ! " turned into " Swag " , u grew up , and we 've been here all the way . RT i u love Bieber ♥
Target Word: justin
Polarity: 1

Tweet: received my **TARGET-WORD** account today ! sorry have no invites , but i will spread the love if i receive any , thanks twitter community !
Target Word: google wave
Polarity: 0

Figure 5.1: Twitter Sent-API examples: Three tweets that represent each of the sentiment polarity present in the dataset.

5.2 Methodology: LSTM Architecture and Baseline Details

The goal of our study was to conduct various experiments to analyze the effects of the selected data augmentation techniques on a given model's classification accuracy for tasks pertaining to sentiment analysis. As shown in Figure 5.2, the first step involved replicating the LSTM baseline from [36]. Tang et al. [36] used a standard LSTM with a single hidden layer for their baselines model. The word embedding for each word in the sentence was used as the input for each time step of the LSTM to compute the hidden vector for each word. The hidden vector at the final step of the LSTM could be seen as the sentence representation. This representation was then normalized by passing it through a softmax to eventually compute the probability of each of the sentiment classes.

Before testing the two effect of SwitchOut and manifold mixup, the first step was to replicate the baseline performance of the LSTM model used in [36]. The authors in [36] only shared partial implementation details of their model as well as the training process that was used to achieve their baseline score. The words in the tweets were encoded using 100 dimensional GloVe embeddings that were learned from Twitter. The hidden units were initialized randomly by sampling from a uniform distribution *U(-0.003, 0.003).* The model was trained end-to-end in a supervised manner using the cross-entropy loss. The data pre-processing and model training process for the LSTM is described in Figure 5.2.

The authors in [36] did not share any details about data pre-processing or the hyper-parameters used to achieve their published results. We decided to use standard techniques from the NLP literature to process the training and test data from the raw files and prepare them for training the LSTM. We used *nltk*, a popular Python library for text processing, to read and tokenize the raw data from [4] and prepare it for training the LSTM. Tokenization refers to splitting the sentence into individual words or *tokens* before they can be used as input to the model. The tokenized sentences and corresponding sentiment labels were saved in a *tsv* file that could be used at the time of training and testing the LSTM model. The training and testing data were kept in separate files to prevent leakage of test data into the training set.

The LSTM model was built and trained using the *PyTorch* framework [52]. *torchtext*, a popular Python library for text processing with *pytorch*, was used to manage the data processing and batching during the training and testing process. *Torchtext* was used to set up the data pre-processing pipeline that managed the flow of data into the model during the training process. *Torchtext* was also used to create a validation set from the training data using a 80-20 split; 80% of the data was used in the training set and 20% was used in the validation set. We also used stratified sampling to ensure the distribution of classes in the training and validation set matched the test set. We used the *BucketIterator* class in *Torchtext* to create minibatches from the tokenized text file. The *BucketIterator* class is designed to batch examples of similar lengths together so as to minimize the amount of padding in the training examples.

The baseline LSTM described in [36] was also missing the specific hyperparamter values that led to model performance described in their paper. We used our own hyper-parameter optimization routine to obtain the best possible setting of the hyper-parameters values through which we could obtain results closer to those reported in [36]. Table 5.2 lists all values that were used in the hyperparameter search space:

During training, we saved the model each time the validation loss improved on the validation set and used early stopping to determine the end of training. The best classification accuracy we achieved was 0.665 ± 0.003 which was the comparable to the accuracy of 0.665 reported in Tang et al. [36]. The authors in [36]

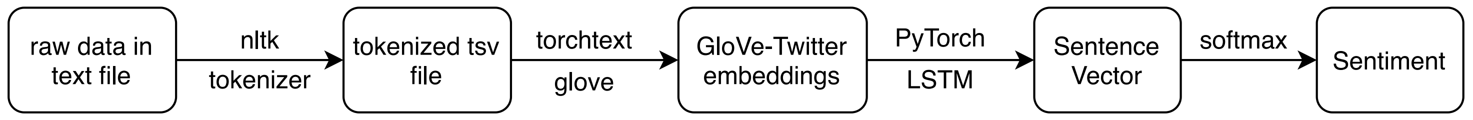


Figure 5.2: LSTM Baseline implementation details: We converted the text file with raw training data into a TSV with tokenized sentences and labels using *Python's nltk* library. The tokens were encoded into their corresponding glove embeddings and input to the LSTM. The hidden vector at the last layer of the LSTM is passed through the softmax to compute the sentiment.

Table 5.2: Table summarizing hyperparameter values used in the search space

Hyperparameter	**Values used in search space**
number of epochs	20, 50, 100, 150, 250, 350, 500, 1000
vocabulary size	10000, 50000, 100000, 100000000
batch size	16, 32, 64, 128, 256,512
learning rate	0.01, 0.001, 0.0001
number of hidden units	64, 128, 256, 512, 1024

did not report the standard deviation in their results.

5.3 Manifold Mixup: Better Representations by Interpolating Hidden States

To demonstrate the effect of manifold mixup [7] on our model's classification accuracy, we implemented manifold mixup in the training loop of our program. At the beginning of each training iteration, the mini-batch was split into two mini batches and the augmented samples are computed by performing manifold mixup on the training examples in the two mini batches. The new *virtual training examples* created with manifold mixup were then used to train the LSTM model along with the regular training examples in the dataset. This resulted in added computational overhead where the new examples generated using manifold mixup were not saved and had to be generated during training any time the experiments are re-run. However, the advantage of implementing manifold mixup in the training loop is that any new examples added to the training set over time could be used to generate new examples whenever the model is retrained on the bigger dataset. Another advantage of creating the examples during training and not saving them is that it eliminated the possibility of accidentally leaking training examples to the test set. This process results in creating an additional examples that increase the size of the training dataset by fifty percent. The schematic in Figure 5.3 describes the training process after we incorporated manifold mixup in the baseline LSTM.

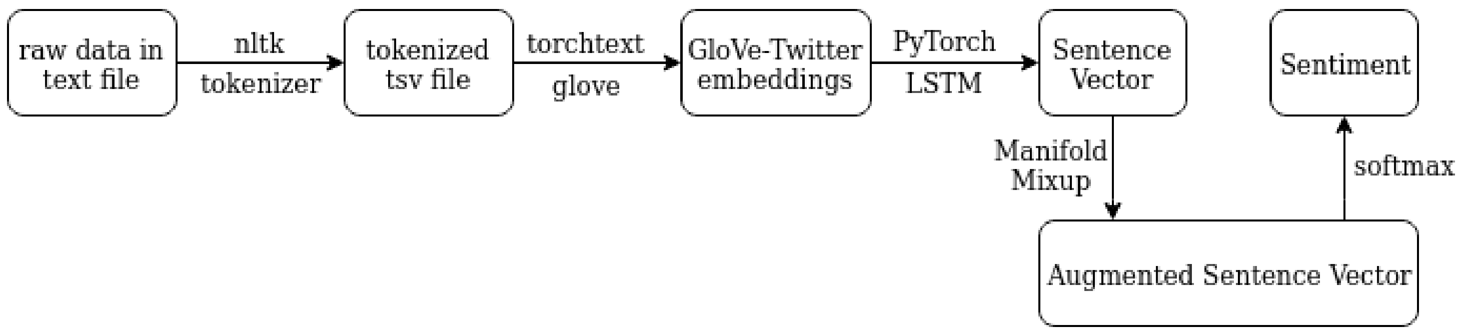


Figure 5.3: Schematic describing how manifold mixup was incorporated in the training process of the baseline LSTM described in figure 5.2. As described above, manifold mixup is applied to hidden vectors obtained at the last step of the LSTM from two different training examples. The newly created hidden vector then acts as a new augmented example created using manifold mixup and sentiment is computed for this vector.

We considered the same hyperparameter search space as in Table 5.2 with the addition of *alpha*; the *alpha* hyperparameter was used to control the strength of interpolation between two vectors. We tested for the following values of *alpha*: 0.1, 0.5, 0.8, 1.0, 2.0, 2.2, 3.0, 4.0, 5.0, 7.0

5.4 SwitchOut: an Efficient Data Augmentation Algorithm for Neural Machine Translation

To demonstrate the effect of SwitchOut on our model's classification accuracy, we re-implemented the SwitchOut function in PyTorch based on the pseudo code available in seminal study [8]. Using our implementation of SwitchOut, we created new training examples for each epoch "on-the-fly" during training of our model for sentiment analysis task. During training, the original training set mini-batch created from the Twitter API dataset was passed as an argument to the separate SwitchOut function. The SwitchOut function then performed random SwitchOut of the words in each example of the referenced mini-batch (Section 4.3.4). As output, an updated mini-batch with new *switched-out* examples was created that were then passed on our LSTM model as input. It is important to note here that all the SwitchOut augmentation happens in real time, making it an efficient process for which no additional pre-processing of the data is required. This process results in creating an additional examples that double the size of the training dataset. The schematic in Figure 5.4 describes the training process after we incorporated manifold mixup in the baseline LSTM.

For the switch-out function, the authors in [8] used *tau* ("temperature") as the hyperparameter to manage the influence of SwitchOut on the training examples. We explored different values of tau to find the optimal choice that gave the best classification accuracy. We considered the same hyperparameter search space as in Table 5.2 with the addition of *tau*. We tested for the following values of *tau*: 0.2, 0.7, 1.0, 2.2, 3.0, 4.0.

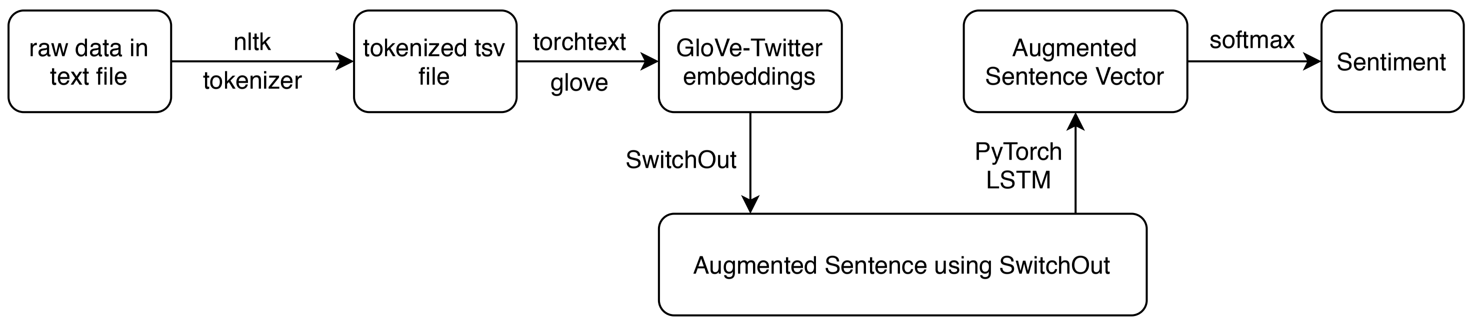


Figure 5.4: Schematic describing how SwitchOut was incorporated in the training process of the baseline LSTM described in figure 5.2. As described in the above, SwitchOut is applied to a training example where we replace one word in the training sentence with another word from the vocabulary creating a new training example. The new sentence is added to a minibatch and used to train the model

5.5 Summary

In this Chapter, we presented the basic experimental framework in which to explore the effect of the two data augmentation techniques on aspect level sentiment analysis. We first described the baseline LSTM architecture and the extensive hyper-parameter search optimization used to match the previously reported performance. We then presented schematics and also described in detail the steps for incorporating manifold mixup and SwitchOut into the training process to further improve the classification accuracy of our LSTM.

In the next Chapter, we discuss and analyze the results obtained through the various experiments described in this Chapter. We present a comprehensive analysis of the obtained results through the effects of the various hyper-parameters involved in the design of the various different experiments.

Chapter 6

Results and Discussion

In this chapter, we summarize the results of the experiments described in the previous section, characterizing the improvement gained in the model's accuracy.

6.1 Training Plots and Analysis

As the first step in our experiments, we replicated the baseline accuracy of the LSTM as presented in Tang et. al. [36]. As the exact model or training code from [36] were unavailable, we trained the LSTM model from scratch. To achieve results at the same level as reported in [36], an extensive hyperparamter search was performed by tuning the following parameters: *number of epochs, vocabulary size, batch size, learning rate and number of hidden units in the LSTM.* Section 5.2 outlines the search space of hyperparameters that were used in during the training process to optimize the LSTM performance. Fig. 6.1 depicts the training curves for optimal hyperparamter combination thus obtained, achieving the 66.5% test accuracy reported in [36].

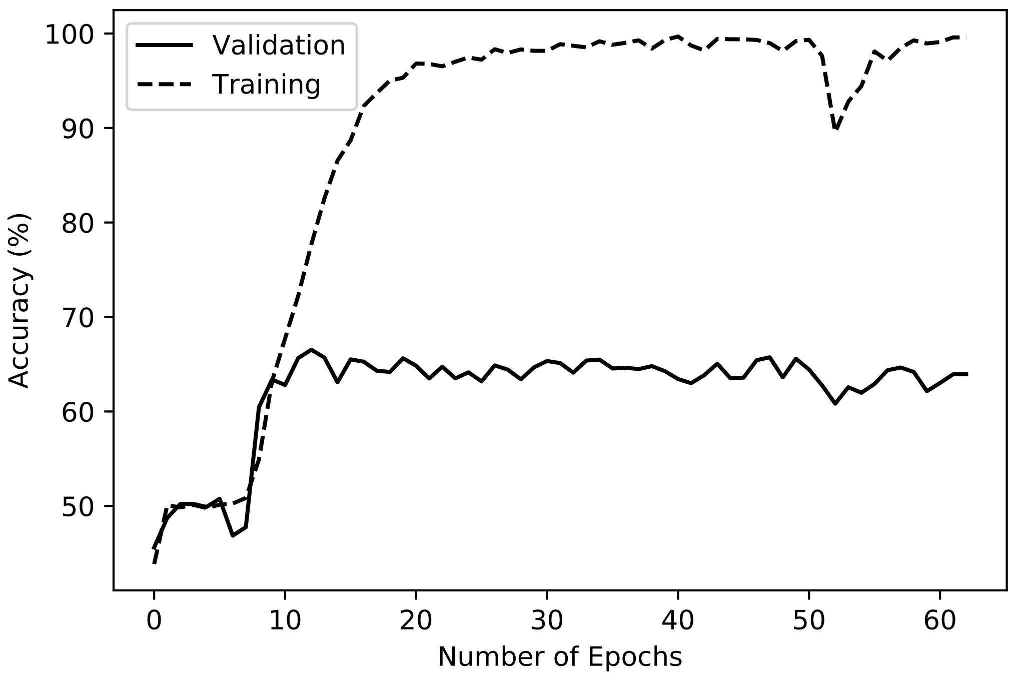
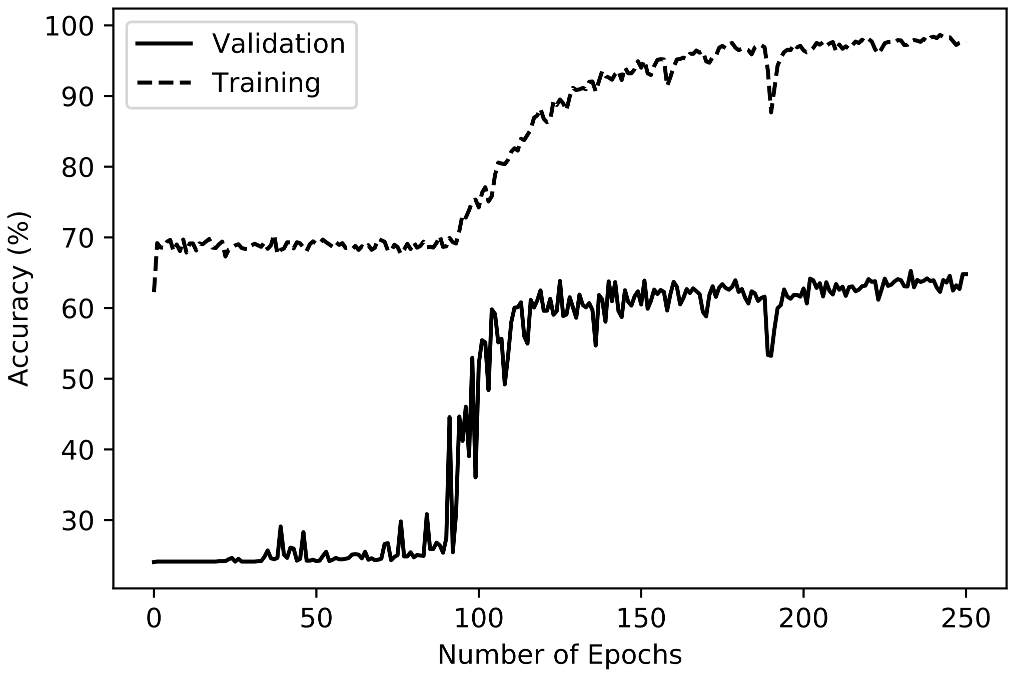


Figure 6.1: Training and validation accuracy curves for the baseline LSTM using the optimal hyper-parameter setting: *number of epochs=100, batch size=256, number of hidden units=256, embedding dimension=100, learning rate=0.001 and vocabulary size=100,000.* The training and validation accuracy increase simultaneously until approximately epoch 10 and then start to diverge. The validation accuracy after epoch 10 plateaus, thus signifying that the network fails to improve its generalization after this point.

6.1.1 Manifold Mixup

Once the baseline LSTM performance was attained, the next step in our experiments involved testing the effect on manifold mixup on model performance. Manifold mixup was incorporated into the training process as depicted in Fig. 5.3. Manifold mixup was applied in the training loop to create new examples using the examples present in the minibatch during each iteration. The new examples created using manifold mixup were then incorporated back into each minibatch, to create a larger set which was eventually used as input to the LSTM. We treated α as a hyperparameter to control the strength of interpolation between two vectors and performed a hyperparameter search to find the optimal value. The details about specific hyperparameters that were used are outlined in the previous chapter. Using the optimal combination of hyperparameters, we were able to improve the model's classification accuracy on the test dataset by 1.3% as compared to the baseline results. The training and validation accuracy curves obtained are shown in Fig. 6.2.

From Fig. 6.2, we made some observations about the training curves. First, we can observe that for models trained with manifold mixup, the classification accuracy did not change during the first 20 to 100 epochs (depending on the hyperparameters). The performance gradually improve after this point with minor fluctuations. The model with an α=4.0 provided a classification accuracy of 0.678 with a standard deviation of 0.002 on the test set when the performance was averaged over ten runs with random initialization. In the best case, we achieved an accuracy of 0.679 with a macro-F1 score of 0.667. The performance of the best case model over different classes is summarized in the confusion matrix presented in Table 6.1. For this particular experiment, we also tested over ten different values of α and Table 6.2 summarizes the model performance thus obtained for different values of α when all other hyperparameters were held constant. In Fig. 6.3, after fixing the values of all hyperparameters other than α, we can see the effect of α on validation accuracy during the training process. For better representation and readability, while forming the performance graph, we only picked four distinct values of α with different intervals ranging from 0.5-5.0. The graph thus obtained with results for each run shown is presented in Fig. 6.3.

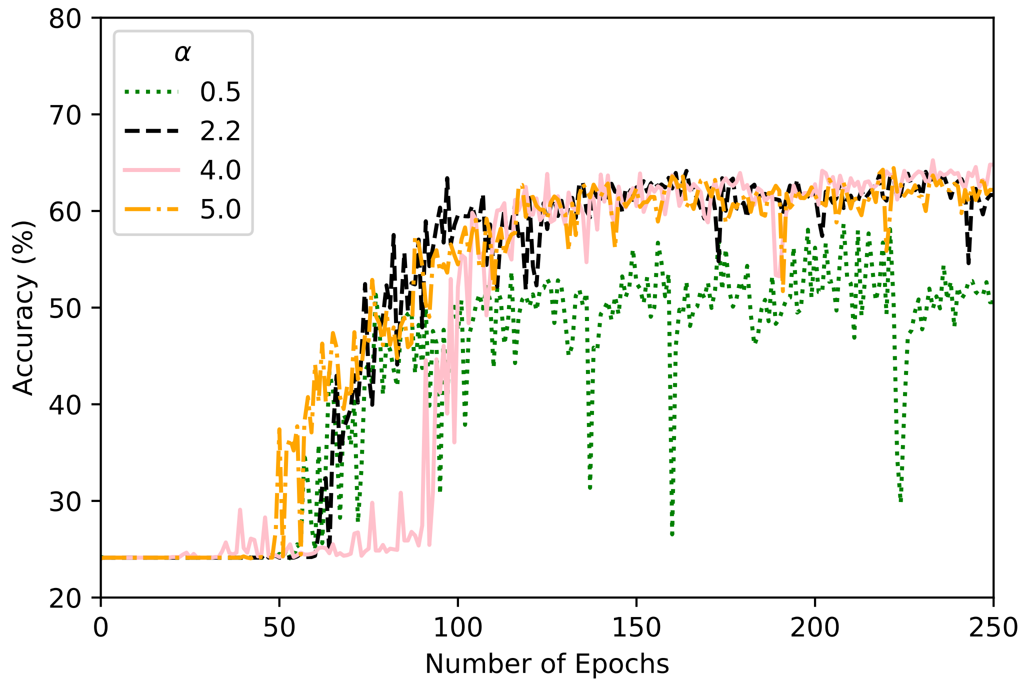


]

Figure 6.2: Training and validation accuracy curves for the LSTM with manifold mixup incorporated in the training process. The optimal hyperparameter setting to achieve the best test accuracy for our model was: *α=4.0, number of epochs=250, batch size=512, number of hidden units=1024, embedding dimension=100, learning rate=0.001 and vocabulary size=50,000.* In can be observed that the validation and training accuracy remain constant form the starting epochs, until around 80-90 epochs. This signifies the network starts learning significantly only after this point and keeps learning until around epochs 150 after which the learning plateaus.

Table 6.1: Confusion matrix for the best case model accuracy using Manifold Mixup

	Actual Negative	**Actual Neutral**	**Actual Positive**
Predicted Negative	122	42	37
Predicted Neutral	40	234	22
Predicted Positive	11	70	114



]

Figure 6.3: Effect of different values of α on training and validation accuracy curves for the LSTM with manifold mixup incorporated in the training process. The following hyperparameter setting was used during the training process: *number of epochs=250, batch size=512, number of hidden units=1024, embedding dimension=100, learning rate=0.001 and vocabulary size=50,000*

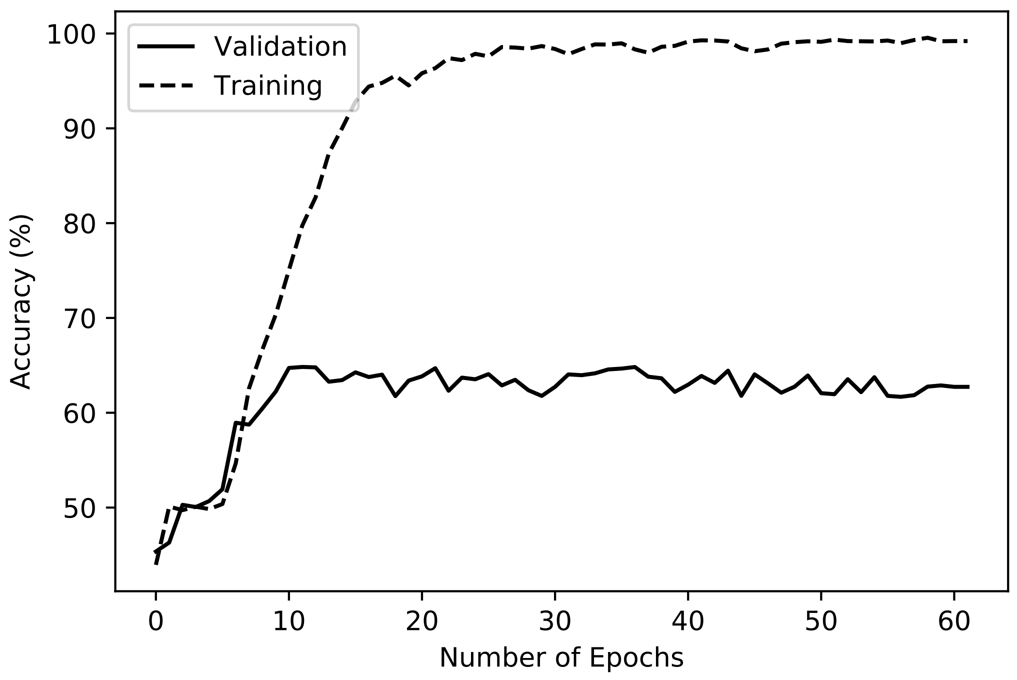
Table 6.2: Table summarizing the effect of different values of α on validation and test accuracy for the LSTM with manifold mixup incorporated in the training process. The following hyperparameter configuration was used during the training process: *number of epochs=250, batch size=512, number of hidden units=1024, embedding dimension=100, learning rate=0.001 and vocabulary size=50,000*

α	**Validation Accuracy**	**Test Accuracy**
0.1	0.572	0.545
0.5	0.587	0.601
0.8	0.630	0.603
1.0	0.641	0.644
2.0	0.649	0.621
2.2	0.642	0.637
3.0	0.648	0.616
4.0	**0.653**	**0.678**
5.0	0.642	0.581
7.0	0.636	0.644

6.1.2 SwitchOut

In addition to the manifold mixup, we performed another set of experiments to test the effect of *SwitchOut* on model performance. In this section, we discuss the results following our experiments to include data augmentation in the input space as well using SwitchOut. A detailed description of how SwitchOut was incorporated into the training process is described in Fig. 5.3. SwitchOut was applied to training examples before creating minibatches for the training process. The new examples were added to the training set effectively increasing the number of examples used to train the model. The details about specific hyperparameters used in this particular experiment are included in *Chapter 5*. Using the optimal combination of hyperparameters, we were able to improve the model's classification accuracy on the test dataset by 0.7% compared to the baseline score of our LSTM model. The training and validation accuracy curves obtained during the training for this model are shown in Fig. 6.4

Unlike manifold mixup, the training accuracy curves in the SwitchOut experiments did not exhibit the same behavior during the training process. The model with an τ=2.2 provided the best classification accuracy of 0.674 with a standard deviation of 0.005 on the test set on the test set when the performance was averaged over ten runs with random initialization. In the best case, we achieved an accuracy of 0.676 with a macro-F1 score of 0.654. The performance of the best case model over different classes is summarized in the confusion



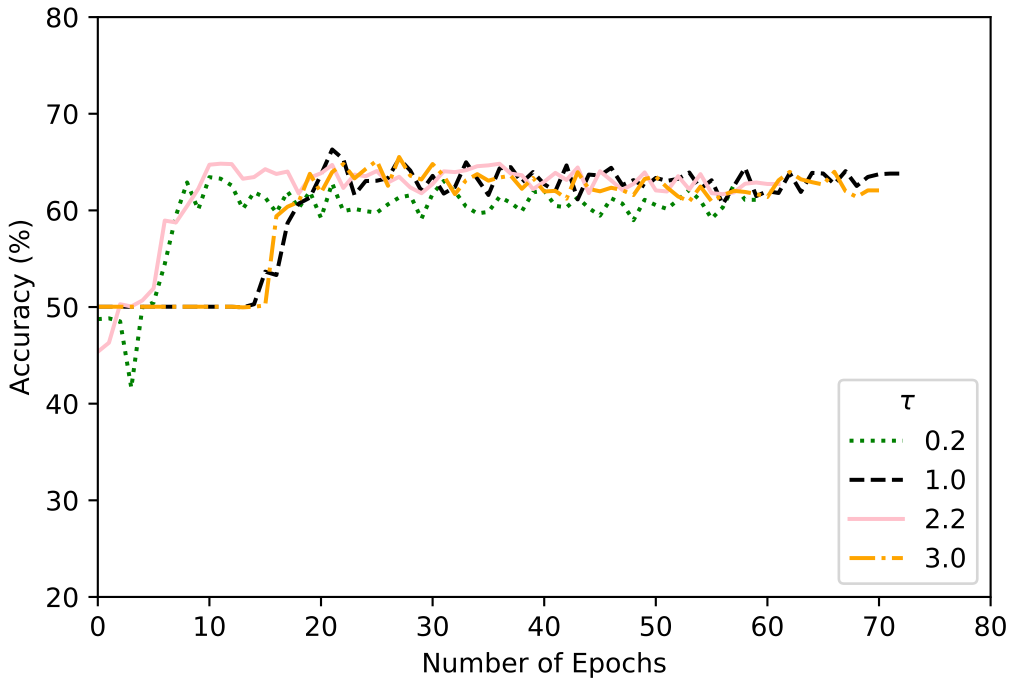
]

Figure 6.4: Training and validation accuracy curves for the LSTM with SwitchOut incorporated in the training process. The optimal hyperparameter setting to achieve the best test accuracy for our model was: *tau=2.2, number of epochs=250, batch size=256, number of hidden units=256, embedding dimension=100, learning rate=0.01 and vocabulary size=100,000.* It can be observed that the training and validation accuracy increases simultaneously until approximately around the epoch 8 and then start to diverge. The validation accuracy after epoch 12 plateaus, thus signifying that the network fails to improve its generalization after this point.

matrix presented in Table 6.3. For some values of τ, there was no change in validation accuracy at first and then there was gradual improvement as training progressed. An improvement can be observed for certain values from the beginning of training until reaching a steady state around 150 epochs. We only picked four distinct values of α to create Fig. 6.5 for better representation and readability. During the training process, we tested over six different values of α which are all listed in Table 6.4. Table 6.4 summarizes the model performance for different values of τ while keeping all other hyperparameters constant.

Table 6.3: Confusion matrix for the best case model accuracy using SwitchOut

	Actual Negative	**Actual Neutral**	**Actual Positive**
Predicted Negative	111	56	22
Predicted Neutral	32	251	46
Predicted Positive	30	39	105



]

Figure 6.5: Effect of different values of τ on training and validation accuracy curves for the LSTM with SwitchOut incorporated in the training process. The following hyperparameter setting was used during the training process: *number of epochs=250, batch size=256, number of hidden units=256, embedding dimension=100, learning rate=0.01 and vocabulary size=100,000*

Table 6.4: Table summarizing the effect of different values of tau on validation and test accuracy for the LSTM with manifold mixup incorporated in the training process. The following hyperparameter setting was used during the training process: *number of epochs=250, batch size=256, number of hidden units=256, embedding dimension=100, learning rate=0.01 and vocabulary size=100,000*

Tau	Validation Accuracy	Test Accuracy
0.2	0.611	0.596
0.7	0.645	0.628
1.0	0.653	0.633
2.2	**0.676**	**0.674**
3.0	0.655	0.639
4.0	0.654	0.655

6.2 Summary of Results

In this Chapter, we demonstrated an improvement in classification accuracy over the baseline LSTM model presented in [36] using manifold mixup and SwitchOut. Manifold mixup creates new training examples by operating in the feature space while SwitchOut creates new training examples in the input space. We observed that both of these techniques improved performance of the baseline LSTM model on the held out test set as compared to the results obtained by Tang et al. [36].

Table 6.5: Comparison of the best results obtained using SwitchOut method with LSTM (baseline), manifold mixup with LSTM and previous methods reported in Tang et al. [36] on the Twitter sent-API dataset.

Method	Accuracy
SVM-indep	0.627
SVM-dep	0.634
Recursive NN	0.630
AdaRNN-w/oE	0.649
AdaRNN-w/E	0.658
AdaRNN-comb	0.663
Target-dep	0.697
Target-dep$^+$	0.711
LSTM (baseline)	**0.665 $\pm$ 0.003**
LSTM + Manifold Mixup	**0.678 $\pm$ 0.002**
LSTM + SwitchOut	**0.674 $\pm$ 0.005**

We observed that both, SwitchOut and manifold mixup, do not always improve model accuracy when compared to the hyperparameter configuration without either of the techniques implemented during training. As per our observation, each of the techniques must be treated as an additional hyperparameter along with the existing ones during the training process to determine the ideal configuration for best accuracy.

6.3 Summary

In this Chapter, we analyzed and discussed the results obtained using the experiments described in Chapter 5. We presented a comprehensive analysis of the obtained results and the effect of hyperparameters on model performance. In the end, we summarized the results in Table 6.5 and compared it to other known results on this dataset.

In the next chapter, we present our conclusions based on the results obtained in this Chapter. We also describe one other technique which, similar to ours, is inspired by mixup and adapts it to the task of sentence classification. In the end, we discuss next steps to potentially extend our work to other tasks and datasets with the goal of making them a standard technique in the deep learning toolkit.

Chapter 7

Conclusion

In this thesis, we demonstrated the effectiveness of manifold mixup and SwitchOut as data augmentation techniques that can help improve the classification accuracy of deep learning models for the task of aspect level sentiment analysis. Our goal was to make progress towards developing task agnostic data augmentation techniques for problems in natural language processing. We were able to demonstrate that manifold mixup, a technique that was developed for tasks in computer vision, could be adapted to the task of aspect level sentiment analysis to improve the classification accuracy of deep learning models. To the best of our knowledge, this was the first successful application of a data augmentation technique for a problem in natural language processing that creates new training examples by performing data augmentation in the feature representational space. In the case of SwitchOut, which was developed and tested for the task of machine translation, we adapted it for the task of aspect level sentiment analysis and demonstrated its ability to improve baseline deep learning model's classification accuracy. The extension of manifold mixup and SwitchOut to the task of aspect level sentiment analysis proved their viability as techniques with the potential of successful application to other problems in natural language processing and eventually becoming a standard technique used by deep learning practitioners.

7.1 Related Works

To the best of our knowledge, our extension of manifold mixup was the first successful application of a data augmentation technique in representation feature space to a problem in natural language processing. Although this will be the first time these results are shared, the experiments extending the application of manifold mixup to the problem of aspect level sentiment analysis were completed in the summer of 2018. In the next sub-section, we would like to acknowledge one other recent work published by Guo et al. [53] in 2019 that builds on the idea of mixup and applies it to the problem of sentence classification.

7.1.1 Augmenting Data with Mixup for Sentence Classification: An Empirical Study

The two methods in Guo et al. [53] is motivated by the use of mixup to improve model performance for the task of image classification. The authors successfully extended the application of mixup to their chosen task in two different ways. In the first method, they perform augmentation at the word level by interpolating between two word embeddings to create a new training example. They refer to this technique as *wordMixup*. In the second method, they perform augmentation using the final "sentence vector" after each word in the sentence has been processed by the neural network. This approach is similar to our extension of manifold mixup, however, their experiments are more extensive since they evaluate their techniques on CNN as well as LSTM-based models. They refer to this second technique as *sentMixup*.

Similar to the improvements demonstrated in this thesis, Guo et al. [53] also reported a significant improvement in performance because of the use of the mixup technique as described above. This reinforces the viability of mixup and mixup inspired techniques, such as manifold mixup, that perform augmentation in representation space as effective data augmentation strategies.

7.2 Next Steps

We selected the hyperparameter values for α, τ and other parameters based on the experiments that were conducted in the seminal papers of manifold mixup and SwitchOut. As seen in the results from the experiments described in Chapter 6, the right combination of hyperparameters has a significant impact on model performance. As a next step, it is recommended to replace our hyperparameter tuning strategy with an automated one to search over a larger space to potentially find a better configuration of hyperparameters. A part of this optimization could also focus on developing a more informed strategy on which words to replace and what to replace them with. Next, it would also be useful to study the effect of number of examples generated using each augmentation technique on the classification accuracy. This could be used to determine the optimal number of augmented examples to be used during training for each technique. Another potential direction to explore could be a hybrid strategy where the training starts off without using manifold mixup or SwitchOut and then the learning process determines the ideal time to incorporate the respective techniques into training. As an extension to our experiments, it would be interesting to observe the effect on training when both techniques are used simultaneously in the training process along with other augmentation techniques that are deemed suitable for the given problem. For our experiments, we chose to use pre-trained GloVe embeddings and froze the embeddings layer during training. The training could also benefit from unfreezing the embedding layer and fine tuning the embeddings for the dataset that's used during training. In the end, we hope that the our work will inspire others to test these data augmentation methods on other problems in NLP and eventually make these techniques available as a Python package for other deep learning practitioners.

References

[1] “Cat picture.” `https://santanvalleyvets.com/wp-content/uploads/2018/11/cat-468232_1280-1080x675.jpg`. Accessed: 2019-12-28.

[2] T. DeVries and G. W. Taylor, “Dataset augmentation in feature space,” *arXiv preprint arXiv:1702.05538*, 2017.

[3] R. Sennrich, B. Haddow, and A. Birch, “Improving neural machine translation models with monolingual data,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Berlin, Germany), pp. 86–96, Association for Computational Linguistics, Aug. 2016.

[4] L. Dong, F. Wei, C. Tan, D. Tang, M. Zhou, and K. Xu, “Adaptive recursive neural network for target-dependent twitter sentiment classification,” in *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 2: Short papers)*, vol. 2, pp. 49–54, 2014.

[5] M. Saeidi, G. Bouchard, M. Liakata, and S. Riedel, “SentiHood: Targeted aspect based sentiment analysis dataset for urban neighbourhoods,” in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, (Osaka, Japan), pp. 1546–1556, The COLING 2016 Organizing Committee, Dec. 2016.

[6] “Task description: Aspect based sentiment analysis (absa).” `http://alt.qcri.org/semeval2014/task4/#`, 2014. [Online; accessed 3-Feb-2020].

[7] V. Verma, A. Lamb, C. Beckham, A. Najafi, I. Mitliagkas, D. Lopez-Paz, and Y. Bengio, "Manifold mixup: Better representations by interpolating hidden states," in *International Conference on Machine Learning*, pp. 6438–6447, 2019.

[8] X. Wang, H. Pham, Z. Dai, and G. Neubig, "Switchout: an efficient data augmentation algorithm for neural machine translation," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 856–861, 2018.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[10] L. Deng and Y. Liu, *Deep Learning in Natural Language Processing*. Springer, 2018.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[12] E. J. Boers and H. Kuiper, "Biological metaphors and the design of modular artificial neural networks," 1992.

[13] B. Lang, "Monotonic multi-layer perceptron networks as universal approximators," in *International conference on artificial neural networks*, pp. 31–37, Springer, 2005.

[14] P. J. Werbos *et al.*, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[15] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[16] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[17] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[18] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," 1999.

[19] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury, *et al.*, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal processing magazine*, vol. 29, 2012.

[20] K. Cho, B. v. M. C. Gulcehre, D. Bahdanau, F. B. H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation,"

[21] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

[22] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, pp. 5998–6008, 2017.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[24] D. Amodei, S. Ananthanarayanan, R. Anubhai, J. Bai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, Q. Cheng, G. Chen, *et al.*, "Deep speech 2: End-to-end speech recognition in english and mandarin," in *International conference on machine learning*, pp. 173–182, 2016.

[25] Q. X. J. C. J. K. G. S. V. L. R. C. Vineel Pratap, Awni Hannun, "wav2letter++: The fastest open-source speech recognition system," *CoRR*, vol. abs/1812.07625, 2018.

[26] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv*, pp. arXiv–1409, 2014.

[27] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, pp. 649–657, 2015.

[28] P. Rajpurkar, R. Jia, and P. Liang, "Know what you don't know: Unanswerable questions for squad," 2018.

[29] A. Williams, N. Nangia, and S. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," in *Proceedings of the 2018 Conference of the North American Chapter*

of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pp. 1112–1122, Association for Computational Linguistics, 2018.

[30] Y. Goldberg, "A primer on neural network models for natural language processing," *Journal of Artificial Intelligence Research*, vol. 57, pp. 345–420, 2016.

[31] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[32] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[33] O. Melamud, J. Goldberger, and I. Dagan, "context2vec: Learning generic context embedding with bidirectional LSTM," in *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, (Berlin, Germany), pp. 51–61, Association for Computational Linguistics, Aug. 2016.

[34] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proc. of NAACL*, 2018.

[35] L. Zhang, S. Wang, and B. Liu, "Deep learning for sentiment analysis: A survey," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 8, no. 4, p. e1253, 2018.

[36] D. Tang, B. Qin, X. Feng, and T. Liu, "Effective lstms for target-dependent sentiment classification," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pp. 3298–3307, 2016.

[37] J. Weston, S. Chopra, and A. Bordes, "Memory networks," *arXiv preprint arXiv:1410.3916*, 2014.

[38] Y. Wang, M. Huang, X. Zhu, and L. Zhao, "Attention-based LSTM for aspect-level sentiment classification," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, (Austin, Texas), pp. 606–615, Association for Computational Linguistics, Nov. 2016.

[39] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *arXiv preprint arXiv:1708.04552*, 2017.

[40] X. Zhang and Y. LeCun, "Text understanding from scratch. corr," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, 2015.

[41] R. Sennrich, B. Haddow, and A. Birch, "Edinburgh neural machine translation systems for WMT 16," in *Proceedings of the First Conference on Machine Translation: Volume 2, Shared Task Papers*, (Berlin, Germany), pp. 371–376, Association for Computational Linguistics, Aug. 2016.

[42] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," 2018.

[43] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," 2015.

[44] R. Shwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," 2017.

[45] A. A. Alemi, I. Fischer, J. V. Dillon, and K. Murphy, "Deep variational information bottleneck," *arXiv preprint arXiv:1612.00410*, 2016.

[46] M. I. Belghazi, A. Baratin, S. Rajeswar, S. Ozair, Y. Bengio, A. Courville, and R. D. Hjelm, "Mine: Mutual information neural estimation," 2018.

[47] A. Goyal, R. Islam, D. Strouse, Z. Ahmed, M. Botvinick, H. Larochelle, Y. Bengio, and S. Levine, "Infobot: Transfer and exploration via the information bottleneck," 2019.

[48] A. Achille and S. Soatto, "Information dropout: Learning optimal representations through noisy computation," 2016.

[49] M. Cettolo, J. Niehues, S. Stüker, L. Bentivogli, R. Cattoni, and M. Federico, "The iwslt 2015 evaluation campaign,"

[50] M. Cettolo, N. Jan, S. Sebastian, L. Bentivogli, R. Cattoni, and M. Federico, "The iwslt 2016 evaluation campaign," in *International Workshop on Spoken Language Translation*, 2016.

[51] O. r. Bojar, R. Chatterjee, C. Federmann, B. Haddow, M. Huck, C. Hokamp, P. Koehn, V. Logacheva, C. Monz, M. Negri, M. Post, C. Scarton, L. Specia, and M. Turchi, "Findings of the 2015 workshop on statistical machine translation," in *Proceedings of the Tenth Workshop on Statistical Machine Translation*, (Lisbon, Portugal), pp. 1–46, Association for Computational Linguistics, September 2015.

[52] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, pp. 8024–8035, 2019.

[53] H. Guo, Y. Mao, and R. Zhang, “Augmenting data with mixup for sentence classification: An empirical study,” 2019.