# Emotion Classification with Natural Language Processing

## (Comparing BERT and Bi-Directional LSTM models for use with Twitter conversations)

Nathaniel Joselson
nathaniel.joselson@gmail.com

Rasmus Hallén
rasmushallen@gmail.com

June 13, 2019

Master's thesis work carried out at
the Department Mathematical Statistics, Lund University.

Supervisor: Pierre Nugues, pierre.nugues@cs.lth.se

Examiner: Magnus Wiktorsson, magnus.wiktorsson@matstat.lu.se

## Abstract

We have constructed a novel neural network architecture called CWE-LSTM (concatenated word-emoji bidirectional long short-term memory) for classifying emotions in Twitter conversations. The architecture is based on a combination of word and emoji embeddings with domain specificity in Twitter data. Its performance is compared to a current state of the art natural language processing model from Google, BERT. We show that CWE-LSTM is more successful at classifying emotions in Twitter conversations than BERT ($F_1$ 73 versus 69). Furthermore, we hypothesize why this type of problem's domain specificity makes it a poor candidate for transfer learning with BERT. This is to further detail the discussion between large, general models and slimmer, domain specific models in the field of natural language processing.

# Acknowledgements

We would like to thank our supervisor, Pierre. Without our regular meetings this thesis would have been more difficult and less fun to write.

We would also like to thank our sources of inspiration and support, Maja Morsing and Amanda Schultz.

# Contents

# Chapter 1
# Introduction

Much of the combined knowledge of the human species is stored through written documents. Words, sentences, paragraphs and even books are endowed with meaning as authors put their thoughts in writing. The process of encoding meaning in text through writing is so revered that some of the most famous and lasting figures in history are writers.

Nearly equally complex is the decoding process that happens as readers interact with written text. There, readers recreate scenes they have never seen, see people they have never met, and are exposed to emotions which are not their own. The written word forms the basis for complex exchange of ideas, the preservation of knowledge and lived experience and the creation of new connections between readers and writers around the globe.

Furthermore, in the past 20 years since the advent of the internet and especially Web 2.0 technologies, the amount of text data in existence has grown immensely. On websites like Facebook or Twitter, there are gigabytes of text published all the time. It doesn't take long to see that digesting this text corpus in its entirety, say for the purpose of content moderation on the platform, would require many thousands of hours of work.

However, despite the fact that language is messy, contextual and cultural, we have come a long way in the past 10 years on the road to teaching computer models to understand and derive meaning from written and spoken language. This field of research is called **natural language processing** (NLP) and will form the basis for this masters thesis where we will specifically focus on applying **artificial neural networks** (ANN) methods to classify Twitter text data based on the emotion contained.

## 1.1 Background

At its core, NLP is concerned with constructing mathematical models to approximate different aspects of language structure and use. According to Allen (1995) the core aspects of language understanding include

- Semantics: What does a word, phrase, sentence or document mean in abstract terms?

- Morphology: What are the building blocks that words or phrases are constructed from?
- Syntax: Where do words occur in sentences or documents and what does that say about which parts of speech they form?

Early NLP attempted to solve problems in these domains through the use of grammar rules and other rigidly defined linguistic structures (Hutchins, 2001). While such rules had some small success, these solutions were limited by the fact that human use of language changes all the time and isn't always strictly correct. It quickly became clear that rule based forms of language modelling were too brittle for real world usage, especially in the domain of semantic evaluation. Furthermore, with the advent of cheaper computing power and faster information storage and retrieval it became norm to think about NLP in terms of data-driven techniques (Manning and Schütze, 1999).

### 1.1.1 Emotion Classification

One specific area of semantic evaluation research is sentiment analysis. This research has become relevant in large part due to the explosion of text on Web 2.0 platforms based on user generated content. As was mentioned before, no human can feasibly consume and synthesize such quantities of text. Thus, for tasks such as content moderation of social media or sorting through reviews to find dissatisfied customers, the improvement of sentiment analysis techniques is important research.

Emotion classification is a recent addition to this field of research. To a human, the emotion behind written language is often clear, but how do we teach a machine to recognize different feelings and intents? One way, which is reminiscent of the old rule-based methods mentioned above, would be to look for emotionally loaded words. For example, if someone writes "happy" multiple times in a sentence, it often means just that. However, one can use negative-loaded sentiment words like swear words in a positive way, for example to intensify one's meaning. For this reason, most emotion classification models now employ a data-driven approach through **machine learning** and especially ANNs.

The idea is that if one sees a word or group of words appearing many times in happy sentences, then it likely bears some correlation with the happy emotion. This type of context based language learning was first proposed by (Firth, 1957) who is quoted as saying, "You shall know a word by the company it keeps." By this he meant that understanding patterns of how words are used is more important than knowing their dictionary definitions. With a machine learning model, we do not have to worry about pointing out which words or patterns to look for, but rather that our data contains sufficiently many examples for our models to learn from.

### 1.1.2 State of the Art and Neural Networks

**Machine learning** and **deep learning** techniques such as ANNs have revolutionized the field of NLP and specifically emotion classification. This is because neural techniques like ANNs are incredibly powerful at finding patterns in large quantities of data. This has been instrumental to NLP research over the past 20 years on two fronts.

Firstly, deep learning models have been revolutionary in modelling core aspects of language such as semantics, syntax and morphology without the need for supervision or rules.

These efforts were helped by the parallel work of collecting, structuring and annotating shareable, standardized bodies of text called **text corpora**. These digitized and annotated text bodies, of which the Penn TreeBank (Marcus et al., 1993) is one of the best known in English, helped feed stochastic language processing methods with high quality language data. This enabled comparison of such methods to previous rule based models on standardized language modelling tasks with well-defined scopes.

However, the same deep learning techniques have also been instrumental in solving applied NLP problems such as machine translation, image captioning, question answering, sentiment analysis and text classification (Otter et al., 2018). Emotion classification is a clear example of where the advances in the state of the art due to ANNs are most pronounced.

Context is incredibly important in correctly classifying emotions since the larger task also contains contextual sub-tasks like decoding metaphors or understanding sarcasm. This means that large amounts of language information need to be processed and remembered by the model to provide enough context for solving these problems (Seyeditabari et al., 2018). Without advances in deep learning, especially in **recurrent neural network**s (RNNs) and **long short-term memory** networks (LSTMs) this classification problem would be computationally infeasible for previous techniques such as Markov chain based language model.

## 1.2 Motivation

Our motivation for writing about NLP and especially emotion classification is many faceted. Firstly, NLP is an interesting and rapidly developing field. Research is being carried out both by private actors and in the academy which makes the innovation climate exciting. Research is also often community driven and many technological advancements are released as open source.

Another central feature of modern NLP research is the competition culture. Competitions where teams from all over the world compete on the same well-defined problem is a both a way of getting one's research recognized, but also helps lower barriers to entry into the research field. By enlisting in one of these many competitions, teams are not only provided with a baseline to measure their results after, but also with valuable, high quality data and often even with useful functions for things like pre-processing and model evaluation. Thus, another motivation for writing this thesis was to gain exposure to the competition culture.

Furthermore, the improvement in this field can be well measured due to the development of many standardized NLP tasks such as the Stanford Question Answering Dataset (Rajpurkar et al., 2018) among many others. These standard tasks allow new technologies to be directly compared against previous innovations. This both provides a backdrop of where the state of the art is in terms of research techniques, but also can highlight where research is lacking. Through writing this thesis we hoped to gain a better understanding of where the state of the art was in NLP and emotion classification.

Finally, a recent trend in machine learning overall is the application of so-called transfer learning. A large model is trained by a company or person who has the computational capabilities. It is then released to the community, who can tune its application to different problems. This technique has had great success in other areas of machine learning however it is important to assess what kind of tasks this concept is applicable to. These models are often complex with long intensive training times. Thus, an ongoing discussion is whether it is possible to produce similar results with less computation. A final motivation to our research therefore is to better understand whether transfer learning is applicable and superior to other modelling techniques in an NLP and emotion classification context.

## 1.3 Objective

This thesis will tackle a real life semantic evaluation problem classifying emotions using Twitter data based on the SemEval 2019 Task 3 competition called EmoContext. We have used a Deep Learning approach based on the Python frameworks Keras and TensorFlow to achieve nearly state of the art results with an original model architecture. We have also implemented BERT, a current state of the art model in natural language understanding to compare with.

We will first detail the background of semantic evaluation and emotion classification. Then we will discuss the theoretical underpinnings of emotion classification models, from principles of deep learning and neural networks. Next, we will discuss the practical details of training our model and BERT and compare their results on unseen test data. We will also attempt to interrogate our findings and discuss differences between these models from a theoretical and practical standpoint. Finally, we will discuss how we could improve our results and the future of semantic evaluation and NLP.

## 1.4 Limitations

We limit the scope of the project by implementing our models in Keras with TensorFlow to take advantage of the large number of ready to use functions in these packages related to rapid optimization, loss function implementation, training tuning and model prediction. Also, we have limited the scope of the project by only applying our learned models to one specific problem from the EmoContext competition. This is instead of attempting to make a highly generalizable model architecture which can solve multiple semantic analysis problems. This is, however, normal practice for competition-based research. Finally, it is also important to mention that while we train our models on four CPU cores, most NLP researches have access to several GPU cores. This limits the amount of engineering optimization we can apply to our model architectures.

# Chapter 2

# Theoretical Background and Related Work

From a theoretical standpoint, there are two important processes in an emotion classification model. Firstly, the language representation process whereby text data is transformed into real valued vectors which contain language information. From there, a classification model can take these vector representations and find features to distinguish between emotion classes.

We have implemented two models, one of our own creation which is a CWE-LSTM model (Concatenated Word and Emoji's LSTM model) and one state of the art model called BERT from Google. In our CWE-LSTM model, the language representation task and the multi-class classification task are separate so we will discuss the theory behind them separately. In the BERT model however, the representation and classification tasks are intermingled into one architecture which is distinct and novel from our model, thus meriting its own theoretical section.

However, all three tasks (the language representation, the classification and the BERT models) are based on machine learning. For that reason, we will begin by describing a core building block of this field, namely artificial neural networks, and their application in classification models.

From there we will turn to language representation models and describe the theoretical background of representing language as real valued vectors, specifically through word embedding models. We discuss the intuition behind the embeddings used in our model, the GloVe twitter embeddings.

Lastly, we present BERT. This is a language representation model that can be used for multiple tasks through transfer learning. In contrast to previous models, which implements recurrence in different forms, this is a language model which relies solely on a concept called attention.

## 2.1 Machine Learning

Machine learning is a modelling approach where utilize algorithms to implicitly find underlying patterns in a data set instead of specifying what features are important to solve a problem. This is accomplished by providing the algorithm with a learning function to optimize and a rich data set to find features in. From there, the algorithm tunes the parameters of the model to optimize the model's predictive performance on unseen data which is used to evaluate the model.

Machine learning problems are called **supervised learning** problems if the dataset also contains the true labels to compare predictions against. For example, in our case, based on a text sentence $\mathbf{x_i}$, a learning model $\mathbb{H}$ should classify the emotional undertone $y_i$ in the sentence. We then compare the predicted emotion labels $\hat{y}_i$ to the actual emotion labels to evaluate our model.

There are two main types of supervised learning problems: classification or regression problems. Classification problems aim to classify data into a finite number of categories, whereas regression problems predict a continuous number as their output. Since we can quantify how well our model does at predicting the correct output values, we can see how changing the parameters of the model changes our predictions. We therefore optimize a function that punishes worsened predictions and rewards improvements. In a machine learning setting, such a measure is called a cost function. The way we minimize this function is through an iterative algorithm called gradient descent.

### 2.1.1 Artificial Neural Networks

One of the fundamental building blocks of machine learning is ANNs first proposed by McCulloch and Pitts (1943) based on the structure of neurons in the brain. These structures have proved applicable to a wide range of problems when combined through different architectures but the underlying idea is the same. We will now present some common architectures in ANNs as well as describing the intuition and mechanics behind how the work to model data.

We first begin with the representation of data. Let $\mathbf{X}$ be a $p$-dimensional dataset of which we have $n$ samples.

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1p} \\ \vdots & \ddots & \dots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{np} \end{bmatrix} \tag{2.1}$$

Here, each row is a vector $\mathbf{x}_i$ where $i = 1, \dots, n$ has a corresponding label denoted $y_i$. Our problem is to model this output vector as well as possible based on the data. This can most simply be modelled as a linear combination of the input data, which gives rise to the well-known linear regression model below.

$$\hat{\mathbf{y}} = V\mathbf{X}^\intercal \tag{2.2}$$

In Equation (2.2), each element of $V \in \mathbb{R}^{1\times p}$ is a real number called a weight.

However, in many applications, the relation between the data and its corresponding label is not linear. This means that no matter how one varies the weights in equation (2.2), the model would still yield bad predictions. One example of this is the problem of category prediction. Instead of predicting an unbounded real number like one does in linear regression, we want to predict a class. More accurately we want to predict a vector of probabilities that the output $y_i$ belongs to a specific class $k = 1, \dots, K$. Ideally, a perfect model would output a vector where the probability of the target class is 1 and the rest are 0. However, in practice, we are satisfied if the probabilities of the target class is higher than the rest.

Instead of only outputting one single value, we can create a model which outputs a vector of $K$ values by increasing the dimension of the original weight vector. From there, we can use the softmax function to scale these outputs to probabilities between 0 and 1, representing the probability that the output belongs to that specific class. Softmax $\phi$ is given by

$$\phi(\mathbf{z})_j = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)} \tag{2.3}$$

For $j = 1, \dots, K$, where $K$ is the number of classes. We can construct predictions by letting $V$ be a matrix of dimensions $(K \times p)$. Then we have the model:

$$\hat{\mathbf{y}}_i = \phi(V\mathbf{x}_i^\intercal) \tag{2.4}$$

Then the $\hat{\mathbf{y}}_i : (1 \times K)$ is the predicted probabilities that the output label belongs to each class. In the next section, we will explain how to tune the weights $V$ to make these predictions reliable.

In equation (2.2), we saw how to formulate the task in a completely linear setting. Then by adding a non-linear function $\phi$ (like softmax), we saw that we can model more complex behaviour. However, the real strength of ANNs come when we create compositions of non-linear functions that all contain weight parameters. This type of combination of functions and weight parameters is where we enter the domain of deep learning. Each function composition is called a hidden layer and is given by equation 2.5.

$$\mathbf{h} = \sigma(U\mathbf{X}^\intercal) \tag{2.5}$$

The elements of the matrix $U$ are also called weights and the function $\sigma$ is referred to as the hidden layer's activation function. We refer to all our weights in the ANN as $\omega$. The predictions of a neural network model, parameterized by $\omega = (U, V)$ with activation functions $\phi$ and $\sigma$, is given by 2.6.

$$\hat{\mathbf{y}} = \phi(V\sigma(U\mathbf{X}^\intercal)) \tag{2.6}$$

As mentioned above, for classification purposes, softmax is a good choice for the output layer. In general, however, any differentiable function may be chosen for activation. Often, non-linear behaviour is desirable for activation functions so popular activation functions include the rectified linear unit (RELU) (Hahnloser et al., 2000) and hyperbolic tangent (tanh) amongst others.

Figure 2.1 shows a neural network parameterized by $\hat{U}$ and $\hat{V}$. This type of neural network is called a **feed-forward neural network** (FNN) since the input data flows forward through the network to the output nodes. If we pass the output layer through the softmax function, we theoretically can train the network to predict the probability of some input $\mathbf{x}_i$ corresponding to a category $y_k$, $k = (1, \dots, 4)$. This relation can be described by the following equation

$$\hat{\mathbf{y}} = f(\mathbf{X}; \omega) = \phi\left(\hat{V}\mathbf{h}\left(\mathbf{X}; \hat{U}\right)\right) \tag{2.7}$$

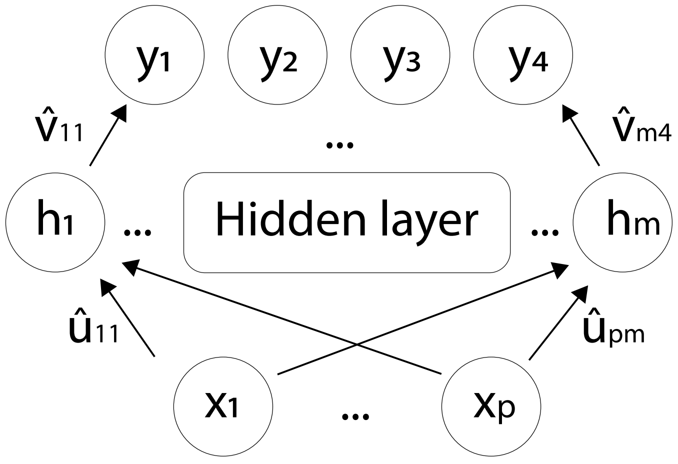


**Figure 2.1:** Standard feed forward network with one hidden layer.

It is clear that $\hat{\mathbf{y}}$ can be seen as either a function of the weights and the input data. Thus, we may change the weights to affect the predictions for some input $\mathbf{x}_i$. In a classification setting, we can ensure that these predictions are good by comparing them to our target vectors. If the weights in $\hat{U}$ and $\hat{V}$ are initialized randomly, chances are that the predictions are faulty. We would then like to update the weights, so that they produce more accurate results. This process called training or learning through gradient descent is generalizable to many different types of networks.

## 2.1.2 Gradient Learning

The underlying idea of training a network is to use an iterative gradient-based optimizer. Such an algorithm requires an objective function to optimize. We call that function a cost function (sometimes error function or loss function), denoted $J$. Different cost functions are applicable to different types of problems. For example, with regression models, the summed square error function is generally a good choice. In our case, we use the the categorical cross-entropy function.

$$J(\omega) = -\sum_{i=1}^{n} y_i \log(\hat{y}_i) \tag{2.8}$$

Here, $y_i$ is the target class and $\hat{y}_i)$ is the models predicted probability of it belonging to the target class, given some parametrization $\omega$ and training examples $i = 1, \ldots, n$.

Utilizing $J$, we can compute an update rule for our weights. As long as the activation function in each layer is differentiable, we are able to calculate the necessary derivatives to see how the cost would change if we changed the weights. Intuitively, if we know the gradients of our optimization problem, we can descend towards the minimum of the cost function.

The size of step we descend in the opposite direction of the gradient is also of importance. In a machine learning context the parameter regulating the step size is called **learning rate** and it is initialized beforehand. If it is too large, we risk overstepping the minimum. On the other hand, if it is too small, the algorithm will converge too slowly. Parameters of this kind which are chosen before training as opposed to as a result of training are referred to as **hyper-parameters**.

The update rule for our weights is given by equation 2.9 where $\eta$ is the learning rate and $\omega^{(t)}$ is the weights after time step $t$ in the optimization process.

$$\omega^{(t+1)} = \omega^{(t)} - \eta \frac{\partial J}{\partial \omega^{(t)}} \tag{2.9}$$

When each training example has acted on the model weights, an **epoch** is completed. Preforming the weight update over multiple epochs is referred to as **backpropagation**. The iterative process of weight updates is called **gradient descent**.

## Variations of Gradient Descent

When training sets are large, calculating the gradient for all weight updates is a computationally infeasible. For this reason, the most common approach to model training is to use a random subset of the training data when updating weights. This technique is called **stochastic gradient descent** (SGD).

When training a neural network with a non-convex cost function using SGD, there is no convergence guarantee. Convergence is sensitive to values of initial parameters such as batch size, learning rate and number of hidden layers. Choosing the correct values is a difficult problem. Luckily, with the expansion of the field of deep learning, variations of SGD have emerged. One such innovation utilizes a dynamic learning rate to increase or decrease our descension speed depending on previous momentum. This makes convergence of the algorithm more likely.

One of the most prominent algorithms is called Adam (Adaptive moment estimation) and it learns based on the update rule below.

$$\omega_{t+1} = \omega_t - \frac{\eta}{\sqrt{\hat{v}_i} + \epsilon} \hat{m}_i \tag{2.10}$$

In equation (2.10), $\hat{m}_t$ and $\hat{v}_t$ are bias corrected estimators of $m_t$ and $v_t$ given by

$$\begin{aligned} \hat{m}_t &= \frac{m_t}{(1-\beta_1)^t} \\ \hat{v}_t &= \frac{v_t}{(1-\beta_2)^t} \end{aligned} \tag{2.11}$$

Where

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial J}{\partial \omega^t} \\ v_t &= \beta_2 m_{t-1} + (1-\beta_2)(\frac{\partial J}{\partial \omega^t})^2 \end{aligned} \tag{2.12}$$

$\beta_1$ and $\beta_2$ are hyper-parameters by default set to of $0.9$ and $0.999$ respectively.

By storing an exponentially decaying average of past squared gradients, as well as past gradients, Adam compares favorably towards other adaptive learning algorithms and is standard for SGD optimization (Ruder, 2016).

## 2.1.3 Recurrent Neural Networks

Language, since we read words in order, can natively be thought of as a sequence. Thus, modelling language through a FNN like above loses some important language information about which words come in which order in the sequence. **Recurrent neural networks** (RNNs), however, proposed by Elman (1990) amongst others, take into account the sequential nature of the input when making output predictions.

RNNs accomplish this by computing a set of feedback weights in a **hidden state vector** at each time step in the sequence that pass information from earlier time points. If we wish to predict whether a sequence belongs to a certain class, we can reformulate our model to incorporate this new time dependency by using the final recurrent hidden state vector to make softmax probability predictions.

$$\hat{\mathbf{y}} = \phi(V\mathbf{h}_p) \tag{2.13}$$

Where

$$\mathbf{h}_p = \sigma(U\mathbf{x}_p + W\mathbf{h}_{p-1}) \tag{2.14}$$

Thus, the parametrization $\omega$ of the model is now comprised of the weight-matrices U, V and W. Furthermore, our hidden layer is now dependent on earlier states.

For clarification of this recurrence mechanism, consider the following figure 2.2. We see

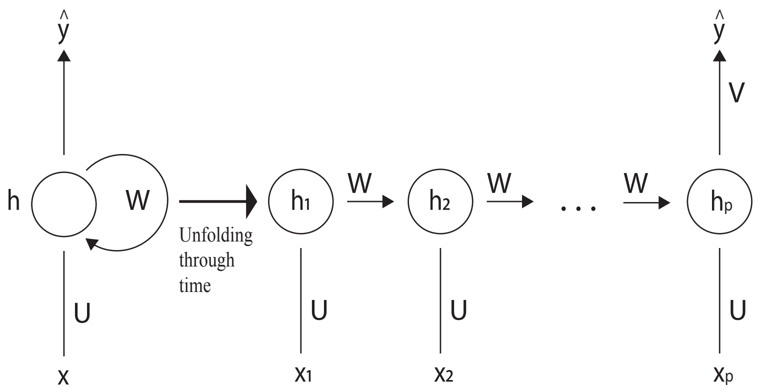


**Figure 2.2:** Recurrent neural network to illustrate the concept of unfolding through time.

that at each time step, a new hidden state vector is computed. This vector can be trained to pass forward the most important information for solving the problem through gradient descent with an appropriate loss function. However, to compute gradients of the weight matrices with respect to our loss function we need to unfold the network through time. If our problem is classification, we can use categorical cross-entropy defined in equation (2.8), and unroll our time dependency through the gradients. Given that $J$ is still differentiable, we also update the network weights according to equation (2.9) for all the weight matrices in $\omega$.

Since we know that $J$ is parametrized by $U, V$ and $W$, to compute the gradient updates, we need to compute the respective partial derivatives. Consider the loss function gradient with respect to the internal state matrix $W$, for example. We can compute this gradient based on the chain rule over the different components of the weight update.

$$\frac{\partial J}{\partial W} = \frac{\partial J}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{h_p}} \frac{\partial \mathbf{h_p}}{\partial \mathbf{h_1}} \frac{\partial \mathbf{h_1}}{\partial W} \tag{2.15}$$

The recurrence in this equation comes from the term $\frac{\partial \mathbf{h_p}}{\partial \mathbf{h_1}}$. However, this recurrence term is also responsible for a main drawback of using RNNs in modelling, that of vanishing gradients.

Vanishing gradients happen when the model is unable to effect change to the loss function based on changes to the weights. The term $\frac{\partial \mathbf{h}_p}{\partial \mathbf{h}_1}$ can be further expanded to reveal the recurrence relationship at work.

$$\begin{aligned} \frac{\partial \mathbf{h}_p}{\partial \mathbf{h}_1} &= \frac{\partial \mathbf{h}_p}{\partial \mathbf{h}_{p-1}} \frac{\partial \mathbf{h}_{p-1}}{\partial \mathbf{h}_{p-2}} \cdots \frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1} \\ &= \prod_{i=1}^{p-1} \frac{\partial \mathbf{h}_{p-i+1}}{\partial \mathbf{h}_{p-i}} \end{aligned} \tag{2.16}$$

For long sequences, the difference $p - 1$ is large. Thus, there is a risk these gradients could all be low and we would be multiplying very small numbers. If that happens, then the gradient updates of W might vanish so $\frac{\partial J}{\partial W} \to 0$. This means that training will not converge optimally. This limits the usefulness of pure RNNs in long dependency datasets in practice.

### 2.1.4 Long Short-Term Memory

To address the issue of vanishing gradients, (Hochreiter and Schmidhuber, 1997) and (Gers et al., 2000) demonstrated how adding gating to RNNs can enable them to learn long-term dependencies. Since gradients vanish due to the continuous multiplication of partial derivatives, we alter the cell structure. This is done by adding elements to each recurrent layer, thus providing a memory of long sequences. The short-term memory capabilities are unchanged in comparison to the simple RNN, giving rise to the **long short-term memory** unit (LSTM).

LSTM networks contain two main innovations from the simple RNN. Firstly, at each time step a hidden state vector and a local context vector are passed to the next recurrent node. Secondly, the LSTM network contains a set of gating mechanisms which allow the model to decide which information to pass forward in recurrence. This allows the LSTM model to more stably learn long-term dependencies in the sequences.

The gating mechanisms are broadly defined as an input gate, a context gate, a so-called forgetting gate and an output gate. Essentially, these are just activated matrix manipulations based on gating weights which are optimally learned through training. These gates, and their associated weights, are defined by the following relationships.

$$\mathbf{f}_p = \sigma(\mathbf{x}_p U^f + \mathbf{h}_{p-1} W^f) \tag{2.17}$$

$$\mathbf{i}_p = \sigma(\mathbf{x}_p U^i + \mathbf{h}_{p-1} W^i) \tag{2.18}$$

$$\mathbf{o}_p = \sigma(\mathbf{x}_p U^o + \mathbf{h}_{p-1} W^o) \tag{2.19}$$

These functions are all recurrent on the previous time step's hidden state $\mathbf{h}_{p-1}$ and the current input data at this time period $\mathbf{x}_p$.

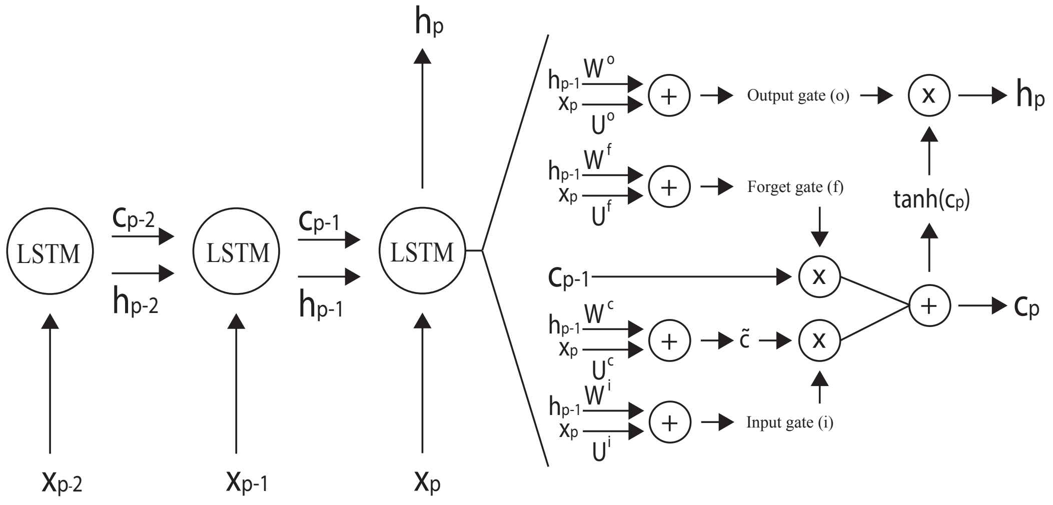


**Figure 2.3:** Long short-term memory network. To the right, the inner workings of a node is shown.

A candidate $\tilde{\mathbf{c}}_p$ for the context state $\mathbf{c}_p$ is computed by.

$$\tilde{\mathbf{c}}_p = \tanh(\mathbf{x}_p U^c + \mathbf{h}_{p-1} W^c) \tag{2.20}$$

The context state is combination of previous context information filtered by the forgetting gate (2.17) and present information from input gate in equation (2.18) filtered through a context gate.

$$\mathbf{c}_p = \mathbf{f}_p \mathbf{c}_{p-1} + \mathbf{i}_p \tilde{\mathbf{c}}_p \tag{2.21}$$

All of this gating machinery allows for the network to optimally determine which long-term and short-term information to filter and pass to the final output representation.

In the final step of the sequence, the hidden state representation of the sequence $\mathbf{h}_p$ is computed as the combination of the current context information combined with the filtered output from the output gate.

$$\mathbf{h}_p = \mathbf{o}_p \tanh(\mathbf{c}_p) \tag{2.22}$$

This output state from (2.22) may then be used for prediction in the same way that hidden states were used in the original RNN architecture in equation (2.13). By keeping an internal memory of past information throughout the chain, LSTMs can represent complex sequences and make stable predictions.

This is also because they can avoid the vanishing gradients seen in standard RNNs. This can be seen by looking at the gradient calculation for model loss with respect to one of the LSTM network's weight matrices.

$$\frac{\partial J}{\partial W^f} = \frac{\partial J}{\partial \hat{\mathbf{y}}_p} \frac{\partial \hat{\mathbf{y}}_p}{\partial \mathbf{h}_p} \frac{\partial \mathbf{h}_p}{\partial \mathbf{c}_p} \frac{\partial \mathbf{c}_p}{\partial \mathbf{f}_p} \frac{\partial \mathbf{f}_p}{\partial \mathbf{h}_{p-1}} \frac{\partial \mathbf{h}_{p-1}}{\partial W^f} \tag{2.23}$$

Clearly, the problematic gradient $\frac{\partial \mathbf{h}_p}{\partial \mathbf{h}_{p-1}}$ is no longer recurrently multiplied due to the intervening forgetting gates and context gates. Thus, if gradients start to vanish, there are other methods of affecting the loss function with respect to the weights and thus the risk of vanishing gradient decreases. However, it has been noted that for long, context heavy sequences, this is still an issue.

### 2.1.5 Bidirectionality

When modelling language, we must recognize that not all words are predictable by the word before. Consider the sequences "river bank" and "bank account", "bank" has a contextual dependency in the two sequences, not only on the previous word, but on the word after. This is a problem that regular recurrent networks can struggle with. This problem can be solved however through **bidirectionality** proposed by Graves and Schmidhuber (2005). This simply means reversing direction of the sequence and feeding this to an independent network and concatenating the resulting hidden states.

For many language models, this is standard practice. In figure 2.4, we have extended the simple RNN from figure 2.2 to work bidirectionally. The states ($h_p$) in figure 2.4 may

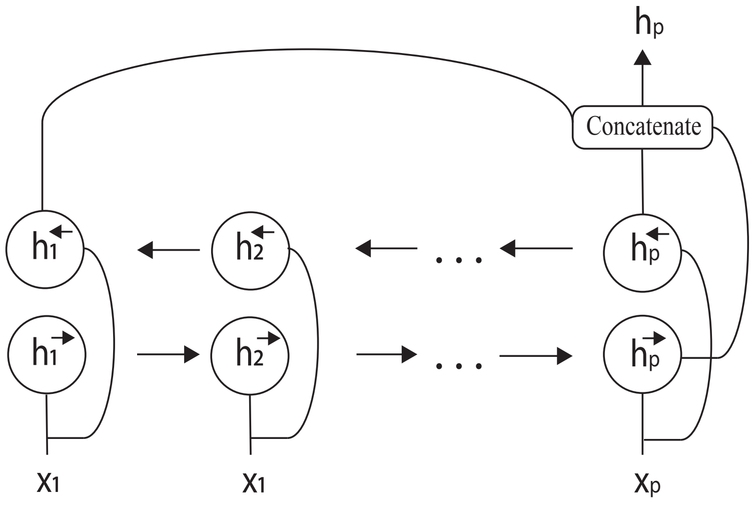


**Figure 2.4:** Bi-directional recurrent neural network

be from regular recurrent nodes or LSTMs. The network with the reversed states is simply an independent copy of the original network. Commonly, the resulting hidden states are concatenated to form

$$\mathbf{h}_p = (\mathbf{h}_p^{\rightarrow}, \mathbf{h}_1^{\leftarrow}) \tag{2.24}$$

This final hidden state vector can then be used in modelling in the same way as for simple RNNs through equation (2.13). By doubling the number of weights, we can now model bidirectional dependencies. In terms of computational capability, constant scalar increases of complexity is usually not a problem. Given that it adds contextual information to language models, this is often a method employed to improve model representation and predictions.

## 2.2 Vector Representation of Language

In all of the above models, the inputs are real valued vectors $\mathbf{x}_i$. This means that an essential step in using neural networks as emotion classifiers is representing the language as vectors. We will now discuss several methods for creating such a vector representation of language and the theory behind them.

We can gather all of the words $w_i$ in our vocabulary to form a bag of words.

$$\mathcal{V} = \{w_i : i \in 1, \dots, N\} \tag{2.25}$$

For example, if the text sequence we were working with was "I love modelling text! 🙂" then the vocabulary would need to include an index for each of the words and symbols.

$$\mathcal{V}=\{\text{I, love, modelling, text, !, 🙂}, \dots\} \tag{2.26}$$

From there, the simplest vector representation $\mathbf{x}_i$ to distinguish between words is one-hot vectors which indicate which word of the vocabulary the vector corresponds to.

$$\begin{aligned} \text{I} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots \end{bmatrix} \\ \text{love} &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & \dots \end{bmatrix} \\ &\vdots \end{aligned}$$

However, simple one hot vectors are not a very useful input to most natural language processing tasks, because they are embedded in a vector space that doesn't contain any extra meaning information about the words being represented. In fact, the only thing the different vector dimensions mean are: Is this word "Love"? Yes or no. Also, as vocabulary size grows, one hot vectors become computationally unusable since each word needs its own separate dimension, so $\mathbf{x}_i \in \mathbb{R}^N$.

### 2.2.1 Statistical Language Models and Word Embeddings

A more efficient method is to pass a lower dimensional vector $\mathbf{x}_i \in \mathbb{R}^d \quad (d < N)$ which also embeds language information about the word $w_i$. This requires less computational power to feed to a classification model as input, and contains more relevant language information than a one-hot vector. This type of lower dimensional representation is called a **word embedding** since they embed words in a multi-dimensional meaning space where groups of semantically and syntactically similar words near to each other in terms of vector distance.

One important language feature which is also good for creating word embeddings is how words are used in conjunction with each other, i.e. their co-occurrence. Through looking at the context that words appear in and the probability of observing a word given the observed context one can gain a lot of insight into their syntactical and semantic use. This is called **statistical language modelling**.

In the example "I love modelling text! 🙂" above, there are clearly some words that are less likely to occur in place of "modelling." For syntax reasons, it would be very strange to see an adverb like "quickly" in that spot, and for semantic reasons it would be strange to see an unrelated word like "gastroenteritis." Important aspects of word meaning, correct syntax and logical relationships between words and phrases can be captured through the probability of these words co-occurring (Jurafsky and Martin, 2000).

Thus, a meaningful language feature is the probability of observing a particular word $w_i$ given the context sequence $\{w_j\}_{i-n}^{i-1}$ of $n$ words that came before. This probability is called an $n$-**gram** since it is based on an $n$-word context.

$$P(w_i|w_{i-1},...,w_{i-n}) \tag{2.27}$$

From a text corpus and with a specific context window, one can calculate the co-occurrence probability of two words $w_i$ and $w_j$. This co-occurrence probability, $P_{ij} = P(w_i|w_j \text{ in context})$ is based on counts of occurrences of $w_i$ with and without $w_j$ in its context window. These probabilities create a co-occurrence probability matrix $P \in \mathbb{R}^{(N \times N)}$ for each word pair in the vocabulary.

The row vectors of $P$ are the co-occurrence probabilities of a word with every other word in the vocabulary and, theoretically, could also be used as word-vector representations like the one-hot vectors above. This would encode more language information than the one-hot vectors. However, it would run into the same computational problems with growing vocabulary size. Also, this matrix is relatively sparse, especially for words that don't occur very often in the text corpus, so not all $N$ dimensions are necessary to efficiently represent this data.

Most traditional word embeddings are based on approximating $P$ in lower dimensional space. A classic dimension reduction technique is **principal component analysis** (PCA) based on the singular value decomposition of the co-occurrence matrix. This approach is appealing because it is unsupervised, meaning that no linguistic rules need to be fed to the system. Also, PCA looks globally at the text corpus for all of the occurrences of a word in all the different context, representing rich language features. However, with large $N$ this technique becomes too computationally expensive.

This problem can still be solved in an unsupervised way however, using neural techniques as shown by Bengio et al. (2003). His breakthrough was to use a feed forward neural network to predict the $n$-gram probabilities of the words in the vocabulary given the context that came before.

For each word in order in the text corpus, the $n$ previous words were used as context and each word was represented by a trainable, $d$-dimensional random initialized vector. These vectors were then concatenated and fed through many hidden layers. From there the output layer was a softmax probability over the vocabulary of all the probabilities of seeing each word, given the context words. This objective function ensured that while training to predict $n$-grams, the network was learning linguistically valuable information in the matrix of $d$-dimensional vectors. Furthermore, it was shown by Collobert et al. (2011) that this type of word embedding wasn't just a placeholder for words, but coded important language information. In fact, he showed that word embeddings could be used as the sole input to other machine learning models for applied NLP tasks with state of the art results. Amongst other things, word embeddings have shown to be useful in finding named entities in phrases (Turian et al., 2010), can be aligned over languages to help machine translation (Zou et al., 2013), and also in semantic sentence parsing (Richard Socher, Alex Perelygin, Jean Y.Wu, Jason Chuang, Christopher D. Manning and Potts, 2013).

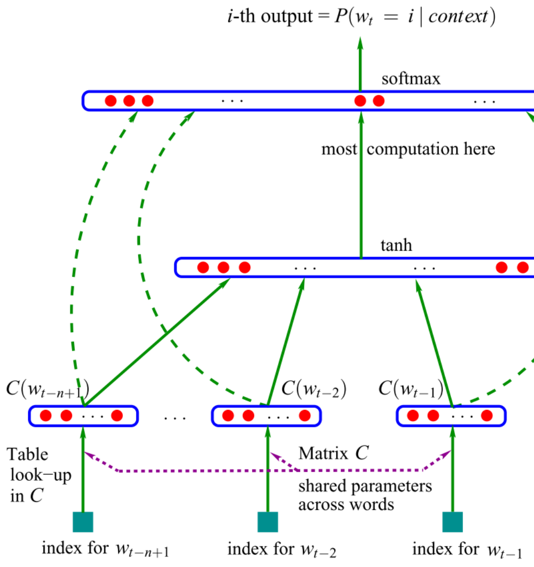


**Figure 2.5:** Feed Forward neural network

Bengio's model had several drawbacks, however. Firstly, it was based on a feed forward neural network so the only way of increasing the context window for the network to learn was by increasing the number of input nodes. This meant that these embeddings were expensive to train for recognizing context heavy features in the data, such as emotional information.

### 2.2.2 Word2Vec and GloVe

To train more efficiently on smaller datasets and with better context information Mikolov et al. (2013a,b) proposed a RNN approach to embedding creation called the **skip-gram** model. RNNs, as we mentioned, can more efficiently model rich context information since they have a memory of the sequence of words they have seen. The skip-gram model starts with the opposite probability to what is predicted in an $n$-gram since it's output is then probability of different context words, based on a target word.

This technique proved to embed these word-vectors in a universe of language meaning which was very context rich and semantically interpretable, especially in terms of relations between words. A famous example of these vectors interpretability is that using vector arithmetic the vector for **King** minus the vector for **Man** was extremely close in vector space to the vector for **Queen** showing that the model has learned to represent words with respect to a dimension which we can understand as gender, and another which we can understand as royalty.

The word embedding we chose for our vector representations, however, is the **GloVe** (Global Vectors) embedding from Pennington et al. (2014). GloVe differs from the skip-gram and feed forward word embedding models due to the fact that it is trained on a loss function which takes into account both local co-occurrences from the $n$-length context windows, but also global count-based co-occurrence probabilities from the text corpus. This is to attempt to encode more of the language features that would be captured in a global method like PCA. This is based on the intuition that including only local context information doesn't

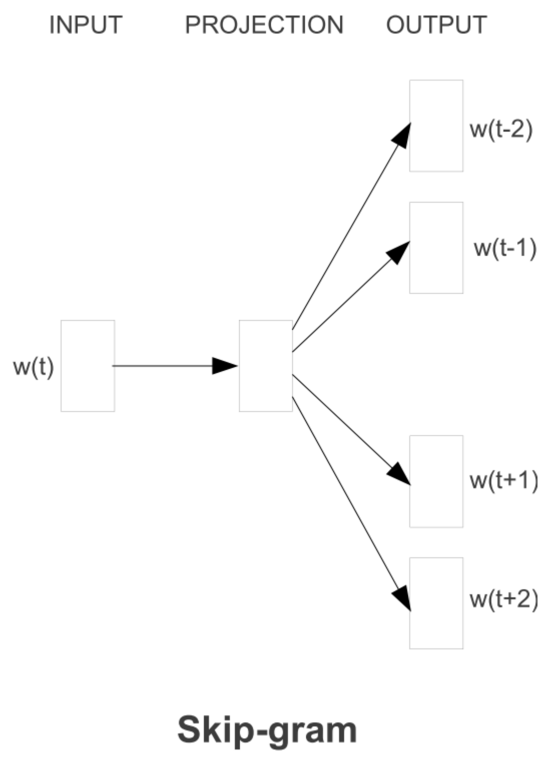


**Figure 2.6:** Feed Forward neural network

give enough information to features about how often words occur in rare contexts. The loss function ends up looking similar to the skip-gram model, but with slightly richer context specific information embedded in the vectors.

Some other popular embeddings models include Latent Semantic Analysis, the FastText embeddings (Joulin et al., 2016) and the ConceptNet embeddings from Speer et al. (2016).

### 2.2.3 Text Corpora and Pre-Training

Nearly more important than the actual word representation model architecture, however is the choice of text corpus that the model will learn from. This has huge importance since different types of language settings have totally different language features and vocabularies. Some common text corpora used for training word embeddings are all the articles on Wikipedia, a corpus of newspaper stories, all text data stored on the web, or all posts on Twitter. The differences in language between Wikipedia, the web at large and Twitter are extremely clear and will create totally different language universes that the vocabulary is embedded in.

Clearly, training a word-embeddings model is a computationally difficult task. Corpora should be as large and varied as possible to give the richest language features. This also means, however, that the loss functions will have lots of local minima which can make training difficult. For that reason, these types of training problems are notorious for requiring lots of engineering tricks and computing power.

Because it is difficult and computationally expensive for all users to need to re-train these embeddings, GloVe has released pre-trained word embeddings for each of these three text corpora mentioned above, Wikipedia, the Common Crawl dataset of websites and all data on Twitter. These can be downloaded and used as inputs to other machine learning models for NLP tasks. We can look at a sample of these word vectors plotted into lower dimensional space to see the clustering structure.

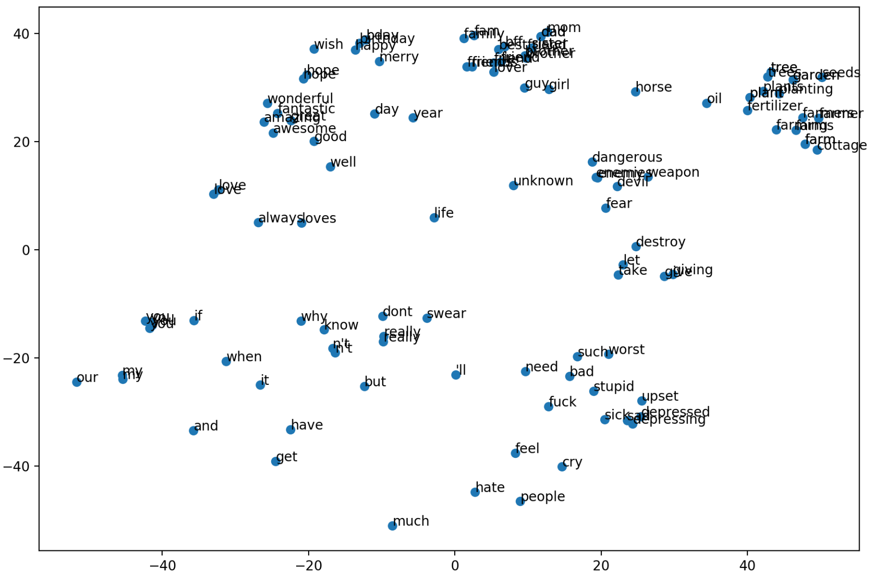


**Figure 2.7:** t-SNE Glove

Every NLP task contains different text data however, and so the language universes are slightly different. For this reason, it is common practice to tune the word-embeddings weights at the same time as training the model. This allows us to better refine the meaning space of vectors to suit the problem context. Also, there will always be some words that are not in our vocabulary of word embeddings. It is most common to initialize these out of vocabulary words' vectors to random values. From there, based on the context that they appear in the problem corpus, we can still train the word embeddings to sort these words into the right part of meaning space.

It should also be mentioned that words are not the only language features that can be embedded in vectors for use as NLP modelling inputs. Especially for social media sites like Twitter, emojis and punctuation play a big roll in communication on the platform. For that reason, researchers have also trained meaning space for emojis such as those used in our sentences and on Twitter in general. Emoji2Vec is a skip-gram style model that learns emoji embeddings the same way that it learns word-embeddings in any other text corpus, with some other tweaks for creating a emoji meaning space (Eisner et al., 2016).

Word embeddings can also be combined with metadata features to aid in problem solving. Examples of this could be combining embeddings with character level data to improve the richness of language understanding (Józefowicz et al., 2016). Similarly, sub-word information contained in longer words has also been used as extra information for embeddings

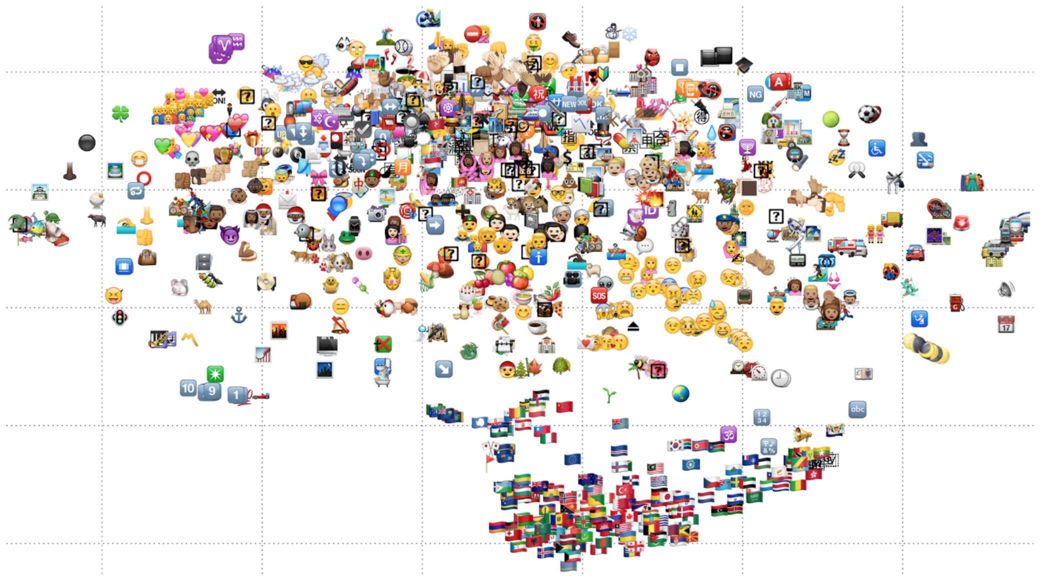


**Figure 2.8:** t-SNE Emoji2Vec

with success especially in multi-lingual contexts (Bojanowski et al., 2016). Document and paragraph embeddings in the meaning space have also been implemented as metadata features with some success in helping semantic classification (Le and Mikolov, 2014). Word Embeddings can even be enriched with problem specific metadata as demonstrated by Tang et al. (2014b,a) with sentiment specific word embeddings for Twitter data.

## 2.3 Bi-Directional Encoder Representations From Transformers (BERT)

We will now discuss the theoretical groundings of BERT, a state of the art language representation and classification from Google. First, we will highlight the background of BERT's creation and the problems with language representation it is supposed to solve. From there we will discuss its neural structure and walk through how the model constructs representations of text data. Finally, we will discuss how BERT's specific input pre-processing and pre-training procedure have made it a widely praised NLP for a range of tasks.

### 2.3.1 Beyond Word Embeddings

Despite the ease and flexibility of using pre-trained word embeddings in language modelling they have one main drawback: each word is represented by just one vector. This means that a word like "set" which has multiple meanings in different contexts is only embedded in one part of meaning space, thus losing language information. This problem is theoretically taken care of by using RNNs or LSTMs which can carry information continuously over sequences to create a context for the current word. However, practically these models can be insufficient to provide rich context data to predict context-heavy language features such as humor or sarcasm (Zhang et al., 2018). This is partially since long term dependencies are formed recurrently so it can be difficult for a model to know which features to pass forward during training.

This problem of insufficient language context from RNNs has been tackled with bidirectional LSTMs (Graves and Schmidhuber, 2005) and attention mechanisms (Bahdanau et al., 2014) for LSTMs with some success. However, one new method for providing context rich information to the network has been the BERT architecture from Google (Devlin et al., 2018) based on the transformer from Vaswani et al. (2017).

BERT (Bi-directional Encoder Representations from Transformers) and its underlying machinery, **transformers**, are based on totally different principles from recurrent neural language models. In contrast to recurrent models, the transformer relies on **attention** for its representations instead of recurrence.

Attention is a technique which was first proposed for RNNs to improve their contextual awareness. Essentially, it works by training a weighted representation of all hidden state vectors at the same time as the recurrence relationships are trained. This allows the network to look backward through the previous hidden states to highlight context information instead of needing to rely solely on the final hidden state representation from the RNN when making predictions. This can allow the model to pick up more subtle and complicated context information.

The transformer architecture builds on this idea of training weighted representations of the previous hidden states. However, instead of applying attention in conjunction with recurrence, the authors show that one can extract relevant language features simply through applying attention to the underlying word vectors from the original sequence. This process is called encoding the text, and while the original transformer paper used this encoded representation for machine translation, it turns out that training contextually encoded language features is useful in many NLP tasks. Furthermore, since the transformer does not use any form of recurrence relations this allows for significantly more parallelization in training.

Transformer based models such as BERT have been able to achieve state of the art results on a range of NLP tasks. Also, the pre-trained weights of the BERT model are freely distributed online to be downloaded used for transfer learning. For that reason, BERT is a perfect model to compare to our CWE-LSTM model in the task of emotion classification. We will now explain the mathematical theory behind our comparison model, BERT. Starting with its structural components, the encoder. Proceeding with attention and its representation of data.

### 2.3.2 Structure of BERT

BERT, as we have said is a state of the art language representation model that can be fine-tuned to many different Natural Language Processing tasks, among them classification. At its most basic, BERT is a sequence to sequence language representation model. It takes as an input a sequence of language information $X = (\mathbf{I}_0, \dots, \mathbf{I}_n)$ and outputs a contextualized vector representation $H = (\mathbf{h}_0, \dots, \mathbf{h}_n)$ of the elements of the input sequence.

Note that there are some slight changes in notation from previous sections. Firstly, we don't pass word vectors directly to the BERT model. Instead we pass enhanced word-parts based on the pre-processing defined in the next section. For that reason, we talk about input vectors $\mathbf{I}$ rather than word vectors $\mathbf{x}$.

Also, in the RNN setting, we talked about a sequence of words $(\mathbf{x}_1, \dots, \mathbf{x}_p)$ however now we talk about a sequence of input vectors which run over the index $\{0, \dots, n : n > p\}$. This is because the BERT pre-processing both splits words into word-parts and inserts placeholder tags before the sequence and after sentences. This will be discussed in more detail in the pre-processing section below, however it means that the input vectors do not directly correspond to the underlying words in the sequence.

Due to the pre-sequence placeholder input $\mathbf{I}_0$ and the design of the BERT model, the output representation $\mathbf{h}_0$ becomes a distributed representation of the underlying sequence in the same way that the final hidden state of a RNN does. This representation can also be used directly for classification in the same way that the hidden states in an RNN can be. By adding an extra hidden layer to the original BERT model and activating it with a softmax function, we obtain a classification model.

$$\hat{\mathbf{y}} = \phi(V\mathbf{h}_0) \tag{2.28}$$

## Encoders

BERT accomplishes the task of being a highly generalizable language representation framework through stacking nodes called **encoders**. Encoders, as one could infer from the name, are a neural network architecture taken from the transformer and used to create encoded representations of text.

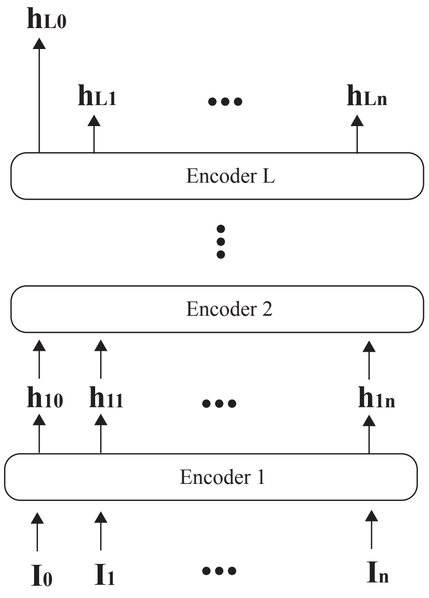


**Figure 2.9:** Bert layers

Each encoder layer is designed to abstract language patterns from an input sequence, forming more nuanced patterns as the information flows up the layers. The input to the first encoder layer is the language inputs and the last outputs are the final encoded language information after being passed through $L$ encoder layers.

$$\begin{aligned} H_1 &= Encoder\,(X) \\ &\vdots \\ H_L &= Encoder\,(H_{L-1}) \end{aligned}$$

The classification task is then based on the first vector $\mathbf{h}_{L0}$ of the final $H_L$ representation.

Each encoder layer is a further combination of two sub-processes. The inputs to the encoder are first passed through a multi-head self attention layer, which uses a series of matrix manipulations to extract language features from the inputs. These manipulations are referred to as **multi-head self attention**. From there, there is a residual connection and normalization process on the outputs before they are passed to the second sub-layer, a feed forward neural network. This second sub-layer extracts the most important features from the attention layer and after normalization and residual connections sends the outputs onward to the next encoder layer.

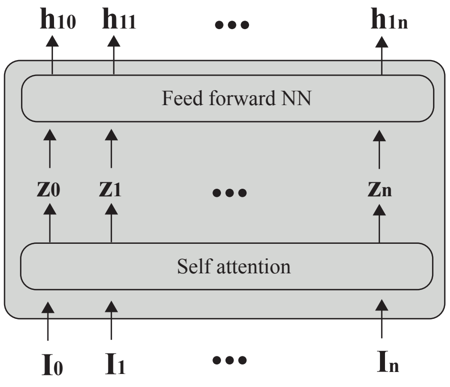


**Figure 2.10:** Encoder 1 (in gray) dissected.

This process can seem convoluted, but it is just the same process repeated over and over. First a series of matrix multiplications extract language features from input sequences which are then weighted together by a FNN to highlight the most important features and combine them.

We will now go through the mathematical flow of the language information through these layers.

## Multi-Head Self Attention

The first sub-layer is a multi-head self attention layer. For the first encoder layer, this input is $X$ but for all the other $l = 2, \ldots, L$ encoder layers this is the output from the previous encoder $H_{l-1}$. We start by extracting a $m$ different representations of this input based on the **scaled dot product attention**. Each of these $i = 1, \ldots, m$ representations is created in the same way but with differently initialized extraction matrices. This is to extract different language features and the technique is called **$m$-head attention**.

The language features that the attention heads find are defined based on three matrices, a so-called Query, Key and Value matrix $(Q_l^{(i)}, K_l^{(i)}, V_l^{(i)})$. These are calculated separately for each of the heads and for each of the encoder layers. They are defined as below as simple matrix multiplications of the inputs and trainable weight matrices $W_{Q_l}^{(i)}, W_{K_l}^{(i)}, W_{V_l}^{(i)}$.

$$Q_l^{(i)} = H_{l-1} \times W_{Q_l}^{(i)} \tag{2.29}$$

$$K_l^{(i)} = H_{l-1} \times W_{K_l}^{(i)} \tag{2.30}$$

$$V_l^{(i)} = H_{l-1} \times W_{V_l}^{(i)} \tag{2.31}$$

These queries, keys and value matrices are combined for each head in a matrix manipulation called scaled dot product attention. This is the manipulation that highlights which input sequences the head pays attention to, and to what extent. This is shown below in equation 2.32 where $\phi$ is the softmax function and $d_k$ is the dimension of the key matrix $K_l^{(i)}$, which in the case of BERT is always 64.

$$Z_l^{(i)} = \phi\left(\frac{Q_l^{(i)} \times K_l^{(i)}}{\sqrt{d_K}}\right) \times V_l^{(i)} \tag{2.32}$$

To gain information from all of the $m$ attention heads, they need to be combined efficiently. These $m$ attention head representations are therefore concatenated and compressed with a matrix multiplication with a layer specific trainable weighting matrix as below.

$$Z_l = Concat\left(Z_l^{(1)}, \ldots, Z_l^{(m)}\right) \times W_l \tag{2.33}$$

These internal encoder representations $Z_l$ now have the same size as the input sequences so they can be residually connected to the input and layer normalized to improve training and prevent vanishing gradients. This is shown below for the first vector of the $Z_l$ matrix.

$$\mathbf{z}'_{l,0} = LayerNorm\left(\mathbf{z}_{l,0} + \mathbf{h}_{(l-1),0}\right) \tag{2.34}$$

Layer normalization is a technique from Ba et al. (2016) which normalizes the input values to the later layers to solve the problem of covariate shift where the data changes distribution over the layers of the network. Residual connections are added to prevent vanishing gradients and because it makes training more stable by reducing the number of weight matrix multiplications in deep neural networks. Both methods are commonly used to stabilize training of large networks like BERT however the theory behind them is out of the scope of this project.

This internal representation is then passed through a feed forward neural network parameterized by the weight matrices $U_l, \tilde{U}_l$. This is to efficiently summarize the most important components of the self attention for passage to the next encoder layer.

$$H'_l = \sigma\left(Z'_l \times U_l\right) \times \tilde{U}_l \tag{2.35}$$

Finally, this final representation is normalized and residually connected again, row by row.

$$\mathbf{h}_{l,0} = LayerNorm\left(\mathbf{z}'_{l,0} + \mathbf{h}'_{l,0}\right) \tag{2.36}$$

This matrix of row vectors is then the final output of the encoder layer $l$. This is a summary of relevant language and context information, ready to be passed to the next encoder level or to be used for classification.

### 2.3.3 Training and Fine-Tuning Bert

Much of the above matrix manipulation is taken directly from the original transformer implementation. However, BERT proposes some novel ideas for sequence pre-processing and pre-training which also have helped it set state of the art results.

#### Input Sequences

As was mentioned before, BERT proposes two novel ideas for coding input data. Firstly, it uses word-piece tokenization and embeddings from Wu et al. (2016) which splits parts of words to get richer word information and to decrease vocabulary size. They also introduced several BERT specific tokens including the classification token [CLS] and the sentence end token [SEP].

The classification token is especially important for us since it always comes at position zero and thus is encoded to the representation vector $\mathbf{h}_{L,0}$ which we can use for classification. This works because this vector has access to all language information in each self attention layer but doesn't encode any information itself, thus it becomes a distributed summary of the language information in the rest of the sentence. The sentence end tokens are also important since they allow the model treat different sentences differently.

Secondly, the BERT combines word-piece embeddings with segment embeddings and positional embeddings.

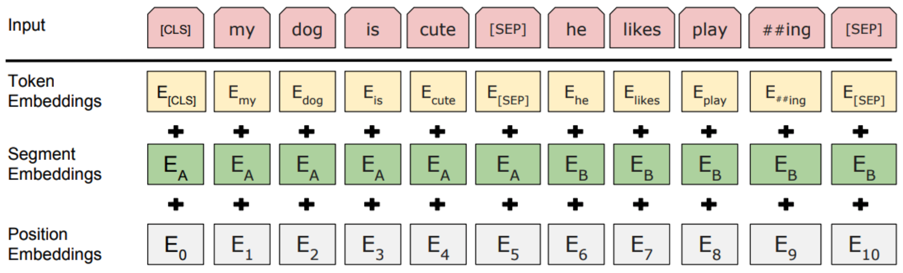

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ $E_{my}$ $E_{dog}$ $E_{is}$ $E_{cute}$ $E_{[SEP]}$ $E_{he}$ $E_{likes}$ $E_{play}$ $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
$E_A$ $E_A$ $E_A$ $E_A$ $E_A$ $E_A$ $E_B$ $E_B$ $E_B$ $E_B$ $E_B$
Position Embeddings
$E_0$ $E_1$ $E_2$ $E_3$ $E_4$ $E_5$ $E_6$ $E_7$ $E_8$ $E_9$ $E_{10}$

**Figure 2.11:** Embeddings for BERT

This gives the model extra information about which sentence the words belong to and where in the sequence the words come. These positional embeddings are especially important since without them, it is not possible for the scaled dot product attention to differentiate between sentences with the same words in a different order, for example "The cat sat on the dog" compared to "The dog sat on the cat."

The positional embeddings are based on sinusoidal curves. These embeddings are of the same dimension as the word vectors and thus are defined for position $i$ in the text sequence and embedding dimensions $2j$ and $2j + 1$ in the embedding vector. We define the values of the position embeddings as below where $d_{embeddings}$ is the dimension of the word embeddings used in word representation, for BERT this is 512.

$$PE_{i,2j} = \sin(i/1000^{2j/d_{embeddings}}) \tag{2.37}$$
$$PE_{i,2j+1} = \cos(i/1000^{2j/d_{embeddings}}) \tag{2.38}$$

These values of positional embeddings described in Equation 2.38 are desirable because they have the property that for any positional difference $k$ in the word sequence, we can represent $PE_{i+k}$ as a linear function of $PE_i$ (Gehring et al., 2017). This allows the model to gain information about the position of words in relation to each other.

Finally, the segment embeddings are a learned embedding vector based on the sentence end tokens to allow the model to represent different sentences in different embedding spaces. This can be helpful to allow the sentences to be divergent in meaning or emotion, which is very relevant to our dataset.

### Pre-Training BERT

The other key to BERT's general success in representing language is in its training procedure. The pre-trained BERT weights that we use have been trained on two different language representation tasks to make the attention weighting more generalizable. These tasks are next sentence prediction which is a binary classification problem, and word prediction from context which is a multi-class classification problem. These tasks work well together due to a novel masking approach that the authors use.

The training works through the following training procedure. Two sentences are sampled from the corpus either following each other or from different parts of the corpus. At the same time, several of the words in sentences A and B are masked with a [MSK] token so the model can't see the words, or changed to a random word in the vocabulary to confuse the algorithm. The model then learns based both on the task of predicting whether the sentences are consecutive or unrelated and the word prediction task based on context. The two loss functions are added together and minimized for a large sample set.

The theory behind this is that learning contextual information is helped by the next sentence prediction task, while the masking task forces the model to keep a higher amount of word-level information in its representations. This also helps the multi-head attention from converging to finding too similar representation features in the encoding since the model is trained also on word prediction, thus the encoding needs to still be grounded in word meaning features.

BERT has advanced the state of the art on several natural language processing benchmarks including beating human performance for the first time on the highly contextual Stanford Question Answering Dataset (Rajpurkar et al., 2018). This should help make BERT perform well on emotion classification tasks and thus makes it a good candidate to train our problem on.

## 2.4 Combating overfitting

One of the important theoretical concepts in training neural models is **overfitting**. Since model training often involves millions of parameters, neural models can often become overfitted to the training data. This means that during training, the model learned the distribution of the training data too specifically to be able to generalize its knowledge to unseen examples. Many methods to prevent overfitting exist, however we primarily utilize **dropout** and **early stopping**.

Early stopping is based on constantly testing the model performance on unseen data after training epochs to see if it is finding ungeneralizable patterns. By splitting the data into train, test and validation sets, we may condition training on performance on the validation set. Specifically, we use the training set for weight updates in our model, but after every epoch we evaluate the model's performance on the validation set. In doing so, we receive an indication of the predictive capabilities of our model and can stop training if the updates our training is making are not translating to increased performance on the validation set.

Early stopping works to prevent overfitting by stopping training when the network shows signs of overfitting, however it doesn't change the actual weight training procedure. Another way of preventing overfitting is to affect the underlying weights of the model. A common cause for overfitting is that the weights grow too large or too small during training. There are a lot of ways to impose regularization constraints on the values of the weights to avoid this. However, dropout, as described by Srivastava et al. (2014), works more simply. Instead of applying limits directly to weight values, it zeros them with a pre-specified probability. Thus, at every epoch, a certain probability of the total number of weights are not used. With less weights in the model during training there is less possibility to learn super specific patterns in the data, and thus we reduce the risk of overfitting.

## 2.5 Evaluating the Model

To know how well our model predicts we must have a measure of model performance. There are several ways of measuring how well a classification model performs in predicting the data. These metrics will be discussed below.

### 2.5.1 F1, Precision, Recall and Accuracy

Accuracy, defined as the percentage of correctly classified samples is possibly the most intuitive measure however not the most useful. In a scenario where the class sizes are uneven, then the largest class dominates accuracy and the model could get a pretty good accuracy score just by guessing the largest class every time. Furthermore, we want to know specifically how well our model is at distinguishing between each emotion class in order to see where it is going wrong. For that reason, in a multi-class setting, most model performance is measured against other metrics such as Precision, Recall and the $F_\beta$ score.

Each time that a model predicts that a data example is a member of of a certain class, it is by default predicting that this data is not a member of the other $k - 1$ classes. This means that for each class we can talk about the number of true and false positive (and negative) predictions the model made on the test data set. This is called the confusion matrix for the model for this particular class.

We can then directly define our three metrics: Precision, Recall and $F_\beta$ score.

$$\begin{aligned}
\text{Precision} &= \frac{TP}{TP + FP} \\
\text{Recall} &= \frac{TP}{TP + FN} \\
F_\beta &= \left(1 + \beta^2\right) \times \frac{\text{Precision} \times \text{Recall}}{(\beta^2 \times \text{Precision}) + \text{Recall}}
\end{aligned} \tag{2.39}$$

The Precision of a specific class therefore is how well the model discriminates between true and false positives, i.e. the percentage of times when the model guessed Happy that that prediction was correct. The Recall is then how well the model finds all the occurrences of a specific class, i.e. the percentage of conversations with a specific emotion label that the model could find. $F_\beta$ is then a harmonic mean of the precision and recall where if $\beta < 1$ then Precision is weighted more highly then Recall in evaluation and if $\beta > 1$ then Recall is weighted more highly than Precision (Rijsbergen, 1979). This trade-off is context specific and should be based on the expected cost of misclassification. However for our context these Precision and Recall metrics are balanced to create the $F_1 = 2 \times \frac{\text{Precision}\times\text{Recall}}{\text{Precision}+\text{Recall}}$ score.

Each class now has a balanced measure of the model's ability to correctly classify that class. For a multi-class data set we can now define the micro- Precision and Recall over all $k$ classes.

$$\begin{aligned}
\text{micro-Precision} &= \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k} TP_i + FP_i} \\
\text{micro-Recall} &= \frac{\sum_{i=1}^{k} TP_i}{\sum_{i=1}^{k} TP_i + FN_i} \\
\text{micro-}F_\beta &= \left(1+\beta^2\right) \times \frac{\text{micro-Precision} \times \text{micro-Recall}}{(\beta^2 \times \text{micro-Precision}) + \text{micro-Recall}}
\end{aligned} \tag{2.40}$$

This score therefore is a way of balancing the performance of the model on the different classes during evaluation.

# Chapter 3
# Approach

In this section, we present the competition from which the data is retrieved. We proceed to discuss pre-processing of this data. Lastly, we motivate our methodology and explain how to implement our model.

It is common practice to test new techniques and algorithms on well-established datasets where benchmark and state of the art results are documented and corroborated. This provides a clear evaluation method for new techniques and architectures and allows for consistency of results despite differences in technique. Similarly, common is to release well structured, labeled datasets in conjunction with starting a competition about which team can create the best predictive model for this dataset on an unseen test set.

In these competitions, it is common to see both academic teams and industrial teams competing for the leader-board which naturally drives results upwards. A final feature of using these types of competition datasets and more established bench-marked datasets is that breakthrough results are encouraged to generate academic papers detailing their techniques and providing access to their code. This open-source style community sharing results around machine learning breakthroughs further facilitates faster improvement and reproducible research.

## 3.1 SemEval

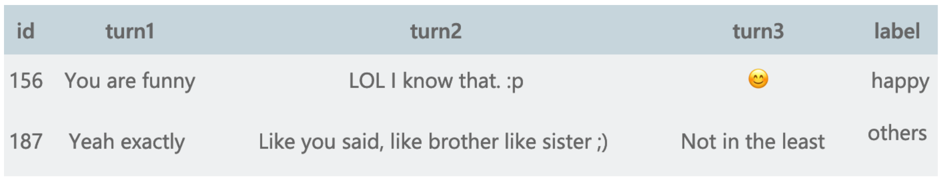
The International Workshop on Semantic Evaluation (SemEval) conference which is organized each year to promote NLP research and innovation is one such competition. Each year teams compete in several tasks having to do with different aspects of natural language understanding with the best teams and models being asked to present their work at the conference. This year the tasks included translation and parsing, fake news detection, hate speech detection, math question answering and emotion detection.

The participants had around six months to create models to solve the problems in any of these tasks with evaluation and submission taking place through the Codalab.

### 3.1.1 EmoContext

We chose to participate in the EmoContext task where the goal was to create a model for classifying the last phrase of three-turn conversations on Twitter into one of four emotions: **Happy**, **Sad**, **Angry** or **Others**. These datasets were labelled with emotions based on the votes of ten linguists and NLP researchers to create high quality data. This is important because classification models are a supervised area of machine learning so without trustworthy labels, there is no point in training such models.

Examples of the dataset and structure are provided below:



| id | turn1 | turn2 | turn3 | label |
|---|---|---|---|---|
| 156 | You are funny | LOL I know that. :p | 😊 | happy |
| 187 | Yeah exactly | Like you said, like brother like sister ;) | Not in the least | others |

**Figure 3.1:** Sample data

The goal of the competition was to create an NLP model to best classify the emotion present in the last turn of the conversation based on the context of the previous two turns. Participants were provided with a labelled training data set, a validation set and a baseline classification model.

This competition and evaluation was based on the competition kick-off paper by Gupta et al. (2017) which proposed a novel LSTM based model called the Sentiment and Semantics LSTM model (SS-LSTM). They combined sentiment specific word embeddings for Twitter from Tang et al. (2014b), mentioned earlier, with GloVe embeddings for semantic information to create their model. These researchers created the dataset that the competition was based on and decided on the evaluation metric to be used in this competition. Thus, this model and research served as a starting point for the competition and forms another baseline model to evaluate our results against.

Model performance in the competition was evaluated based on micro-averaged $F_1$ scores from the emotion classes Happy, Sad and Angry. This means that the task of correctly recognizing others was not rewarded in the evaluation, except in that affects the $F_1$ scores of the other emotion categories. The task was further complicated however, through having an unbalanced training set where the Others category was 15,000 samples and each emotion category was only 5,000. This means that 50% of the training examples were Others. On the other hand, however, the test set was 85% Others, so was even more unbalanced, meaning that the model needed to be able to distinguish well between emotions, but also be able to discriminate between Others and emotion categories.

## 3.2 Method

As has been stated, the objective of the EmoContext competition is to classify which emotion, if any is present in the final turn of a set of conversations. To achieve this, we created a neural network architecture which we call CWE-LSTM which is able to classify conversations into the four emotion categories. Furthermore, we have tuned a state of the art language representation model, BERT, to the same classification task. This provides a way of comparing the two models' performance on the problem. The dataset was provided by the EmoContext competition, as well as a baseline model, some pre-processing functions and an evaluation script.

We will first discuss the text pre-processing we performed on the text sequences to increase coverage and raise the data quality. Then will explain the baseline model provided by EmoContext, before describing our iterative improvements to this model which resulted in our final CWE-LSTM model. Finally, we will discuss BERT and our implementation of these comparison models.

### 3.2.1 Pre-processing

Classifying text is a context heavy task where high quality data is extremely important. Helping the model to gain meaning from the input text sequence relies heavily on data pre-processing. Our text pre-processing can be divided into two steps: data cleaning and tokenization. Data cleaning is the process of fixing data irregularities, and tokenization is mathematical mapping of this text data into a form that can be fed to our classification models.

#### Data Cleaning

Data cleaning is the most involved of these steps and is important for two main reasons, boosting coverage and clarifying meaning in the text. We measure the **coverage** of our text by our pre-trained embeddings as the number of words for which pre-trained embeddings exist divided by the total number of unique words in our problem. Clearly, since the embedding matrix from GloVe is meant to be generalizable but our dictionary is problem dependent, it is not certain that the embedding matrix has a vector representation of a given word. Thus, we perform several data cleaning steps to boost coverage.

The original data contains many irregularities easily identifiable by humans, but which would confuse the embedding model during tokenization. Examples include not putting spaces between entities like emojis or the inclusion of repeating letters or symbols in long sequences. Our data cleaning therefore includes rules to add spaces between entities like emojis and removing repeated letters or symbols.

At the same time, data cleaning can also be a way of clarifying meaning in the text. We understand that symbol combinations like ":)" refer to a happy emoji, however our embedding matrices don't know that. For that reason, we correct well known symbol emojis to Unicode emojis. Also, to improve meaning recognition we decided to separate abbreviations like "you'll" to "you will." This was found to improve model performance by improving sentence representation in meaning space.

An example sentence that would be unrecognizable for many embeddings would be: "bye:):)". After pre-processing we obtain "bye 🙂 🙂," which our embeddings entirely recognize. It is important to note that we did manage to create a general rule to fix misspellings like "byeeeee" to "bye". Regrettably, this is quite a common feature of Twitter communication. With these steps, we raised our coverage to 83% of our words recognized, from 72% with only baseline pre-processing.

## Tokenization and Other Pre-Processing

After the data cleaning words need to be vectorized and prepared for passing into classification models. This process is called tokenization. This involves separating all words found in the text corpus and mapping them to a unique number so they can be recognized later. Many Python libraries (including Keras) have built in functionality to deal with this step.

These tokenized texts are then ready to be mapped to with our embedding matrices which represents our sequences in vector space. Notably, the vector represented sequences passed to the LSTM models as input must all be of equal length. This is to make sure we are training the weights of the model to recognize features at the same places in the different sequences. This is solved with zero-padding the vectors to the same length. To not induce too much computational difficulties, the vector lengths are constrained to 100. Thus, a maximum of 100 words per sentence is considered. The figure below shows the distribution of word length in our training data.

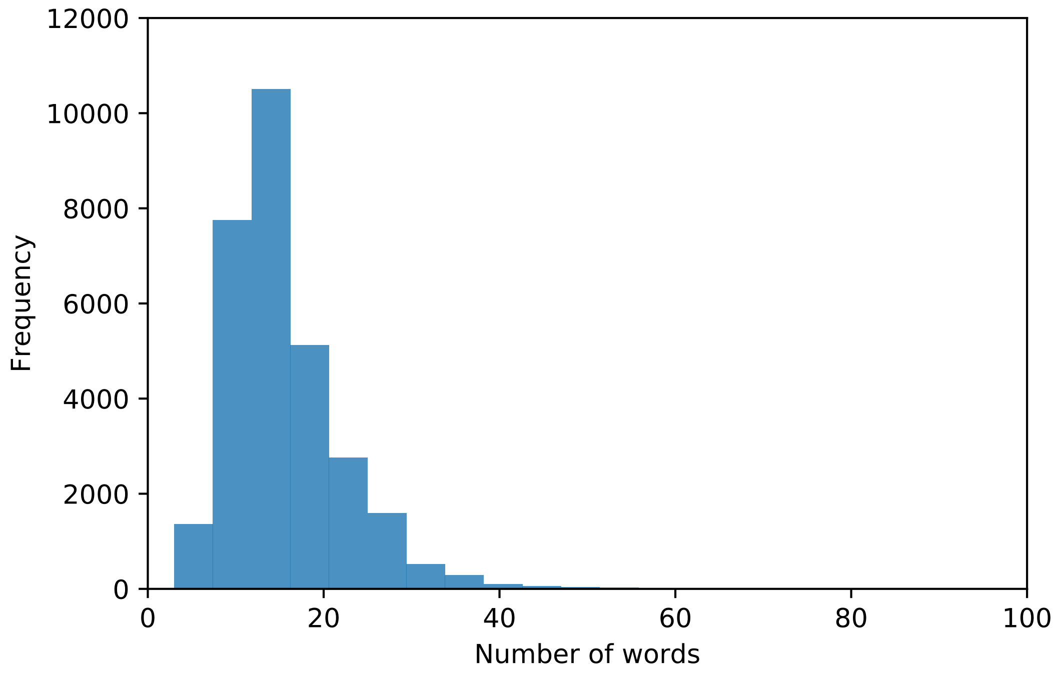


**Figure 3.2:** Histogram of word lengths per sentence. The maximum sentence length is 167.

After the pre-processing, each turn in a conversation is a matrix with elements of padded vectors representing the sentences. Their corresponding labels are one-hot vectors, where the index of the element 1 represent what emotion the training example is connected to. The indices are translated as follows: 0 - Others, 1 - Happy, 2 - Sad, 3 - Angry (pythonic indexing starts from 0). We now have a dataset ready to be processed by our models.

### 3.2.2 Baseline Model

The starter set from the EmoContext included the framework of a model which we modified to our baseline classification model. A baseline model is important because it gives us something to compare to in evaluating our final model.

For the baseline model, we have a standard LSTM that outputs straight to a prediction layer, filtered through a softmax function to get a probability vector. The embeddings we use for our baseline model are also GloVe embeddings, however the text corpus they are trained on is from Wikipedia data. This model uses word embeddings of dimension 100 and a LSTM

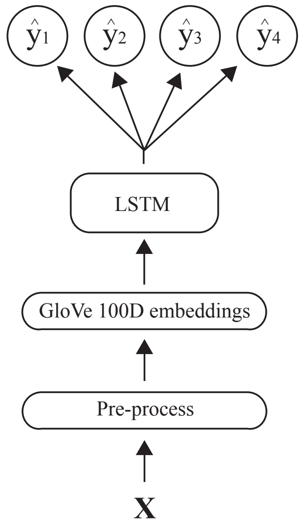


**Figure 3.3:** Baseline model architecture.

layer of dimension 128 with built in dropout. In contrast to the CWE-LSTM, the input turns are not separated. Instead, an end-of-sentence tag (<eos>), is used to mark the sentences in the conversation. That means that each conversation is fed to the model as one single input sequence: "That was a joke btw... <eos> it was <eos> Yes :D".

### 3.2.3 CWE-LSTM Model

The baseline model provides a framework on which we could build to make the predictions more accurate. By iteratively making small changes to our pre-processing and our model architecture, we could isolate features with high impact on model performance, as measured by $F_1$ score. In doing so, we were able to create simple networks consisting only of the most important improvements. A complete description of our architecture, Concatenated Word-Emojis bidirectional-LSTM (CWE-LSTM), is shown in Figure 3.4 below.

In Figure 3.4, the architecture for our original architecture, the CWE-LSTM is displayed. Clearly, it is more complex than the baseline model.

We will now detail all of the significant improvements we made over our baseline model when creating the CWE-LSTM model.

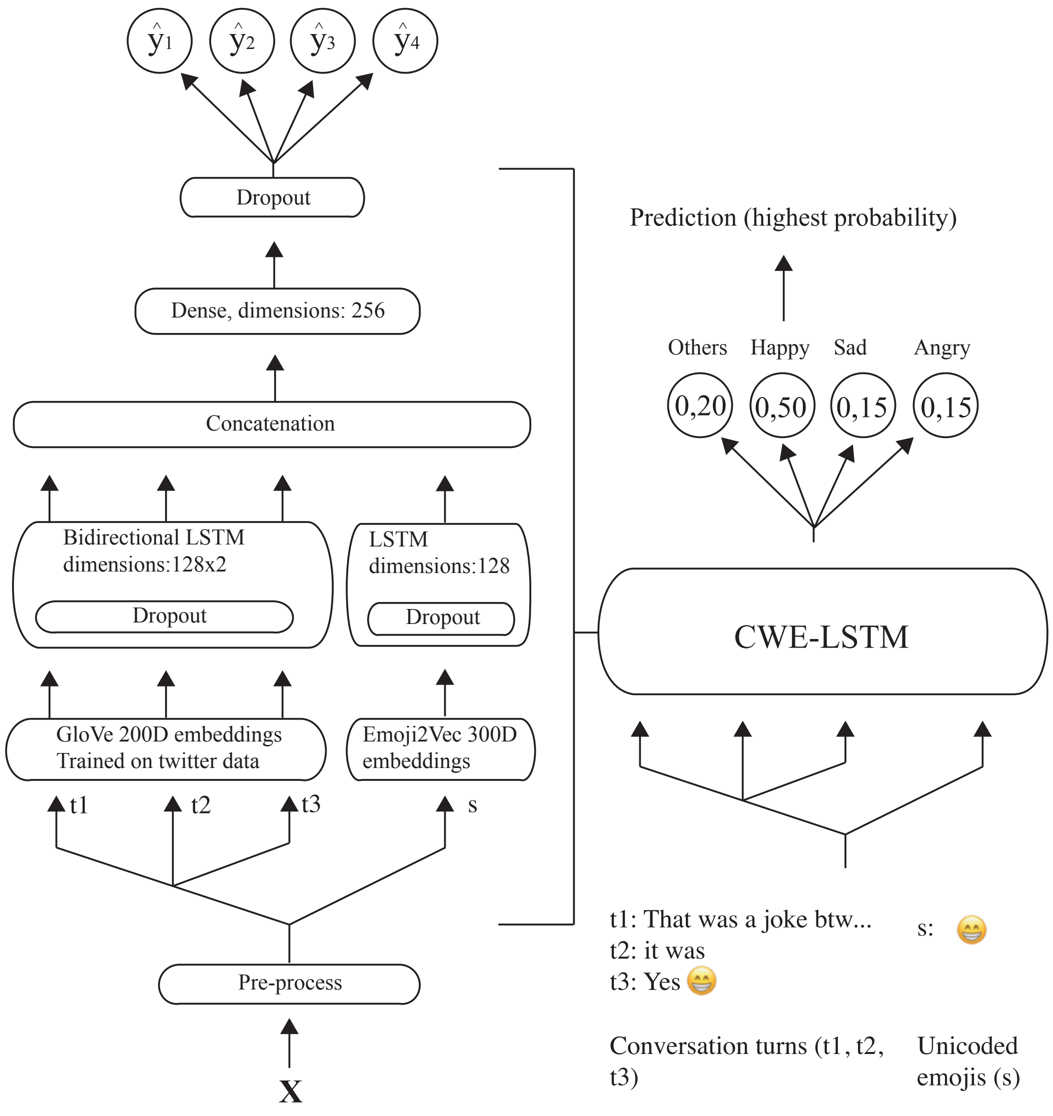


**Figure 3.4:** CWE-LSTM model architecture. To the left, the layers of the model are shown. To the right, an example conversation.

The first improvement over our baseline which made a significant difference in $F_1$ score was separating the turns of the conversation and passing each one to the LSTM node separately. This allows the model to more clearly see the differences in sentences in the conversation compared to the end of sentence tags in the baseline model. This was further helped by our improved pre-processing described in the previous section.

Also significant was the decision to use 200-dimensional GloVe embeddings trained on the Twitter data corpus instead of 300 dimensional embeddings trained on Wikipedia. These embeddings are closer to the domain of our conversations since they were also created on Twitter. Furthermore, we found that the increased complexity from higher embedding dimensions didn't demonstrate better performance.

Another highly effective improvement was extracting smileys from text and processing them separately. This allows us to represent emojis with other embeddings, thus providing new language information to the model. As has been stated, the smileys are embedded with Emoji2Vec embeddings, a 300-dimensional representation of different Unicode smileys. Furthermore, we then concatenate the information from smileys with the information from the rest of the model and pass it through a final dense layer. This helps the model train its attention to weight the relative importance of the emojis and the three turns of the conversation in classifying the emotion present in the text. The activation function in this dense layer is the rectified linear unit.

Next, we found that a bidirectional LSTM node instead of a simple LSTM node for classifying our conversations performed better on our data. Since each conversation contains fewer smileys than words, we concluded that the network processing smileys need not be as large. For that reason, and since the smileys were already taken out of context, we found that bidirectionality was redundant for the smiley layers.

Another difference to the baseline is that our model utilizes trainable embeddings. This means that inside the embedding layer, some tweaking of the pre-trained embeddings is performed during training. This aims to catch more problem specific behavior and align the embeddings better with our problem domain.

Finally, and also extremely importantly, the LSTM layers employ dropout in the connections between input $X$ and weights. Another dropout layer is added after the dense layer. Lastly, the output layer transforms weights to probabilities through softmax.

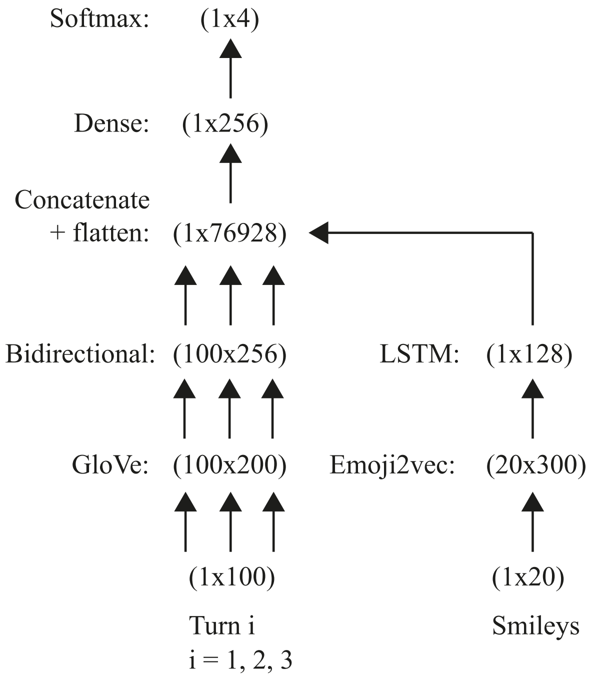


**Figure 3.5:** How data is reshaped throughout our model.

The figure 3.5 above shows how data is reshaped and transformed through different stages of CWE-LSTM. The bidirectional LSTM layer outputs the return sequences, one for every conversation turn. Since these are 100 dimensional, so are their hidden representations.

In total the model consists of 27,557,896 trainable weights. In many applications, this would be quite a large number, however it only took us 1 hour and 24 minutes to train.

This is a stark contrast to BERT's fine-tuning, which took 26 hours and 45 minutes. We will now discuss our implementation of BERT and its application to our emotion classification problem.

### 3.2.4 BERT

The core strength of BERT is that it is applicable to a wide range of problems. The model is hosted on TensorFlow Hub, a pre-trained machine learning model repository hosted by Google. There one can choose to download a small or a large implementation which has to do with the number of encoder layers, the number of heads in the multi-head attention, the number of dense layers in the feed forward network, and the number of training samples used.

It should be noted that since BERT uses word-piece embeddings rather than GloVe embeddings, it follows different pre-processing and tokenizing steps than those described above. These can also be downloaded with the pre-trained model and used out of the box. This means, that we did not perform any pre-processing on the sentences we passed as input to BERT.

Like most transfer learning models, BERT is a large model, and we add a final classification layer, but allow the whole model to be trainable. This is to allow the training procedure to tune the relevant attention weights, as well as training the final output layer.

We have used the hyper-parameters recommended in the BERT-small implementation. This means that we have 6 encoder layers with 8 attention heads and 512 length word embeddings. Our input sequences are padded to a maximum length of 100 words, in the same way as for the CWE-LSTM model. This results in a model which contains approximately 110 million trainable parameters.

We have also used the warm-up training procedure also recommended, where the learning rate starts small and then gradually increases as the model trains.

Finally, since the BERT model only allows two sentence inputs, we have concatenated the first and second turn of the conversation as sentence A, and then present the final turn as sentence B. This forces the model to treat the sentences differently since it adds a different segment embedding to the sentences. This was found to improve performance slightly, since the task is to predict the emotion in the final sentence.

## 3.3 Implementation

As mentioned in the theory section above, using neural architectures to solve problems often results in difficult optimization problems. For that reason, lots of ready to use tools are available to implement and train this type of neural network. Python has become the most popular programming language for deep learning and many important machine learning frameworks are native to Python, thus the decision to implement our model in Python was most logical.

One of the most important frameworks for machine learning in python is the TensorFlow library. This is a highly customizable group of tools both for linear algebra manipulations and for stochastic optimization. TensorFlow is an open source tool, so much of the code has been optimized and tweaked by many dedicated engineers to produce a product that would be extremely hard to replicate from scratch today. At the same time, TensorFlow has put a lot of effort into creating an open source model repository which they call TensorFlow Hub where pre-trained models can be downloaded and used directly for transfer learning applications. That said, TensorFlow is very technical to use and not necessarily optimal for all Python coders.

For that reason, the Keras framework has emerged as the most popular environment for coding deep learning models. Keras is built on TensorFlow but is much easier to use straight out of the box, taking care of implementation of loss functions, passing matrices of dynamic size through the various weight layers correctly, and providing a myriad of other tools to combat overfitting, help model explanation and speed up development.

We have implemented our own CWE-LSTM model in Keras based on their functional API style. The code for this model is presented in the appendix. Our implementation of BERT is a transfer learning retraining based on the pre-trained BERT-small model on TensorFlow Hub. Both models were trained with the ADAM optimizer locally on our four core CPU computers.

### 3.3.1 Hyper-Parameter Optimization

One of the more difficult engineering challenges in implementing and training Deep Learning models of this kind is choosing the hyper parameters to use in training and in the model architecture. As was discussed in the theory section the most important training parameter is the Learning Rate. The learning rate is a number less than 1.0 which shrinks the size of the steps that the gradient descent algorithm takes during each weight update. After experimentation, we decided on a learning rate of 0.001.

The batch size is also extremely important since it determines how many samples the model sees before it updates its weights each time. A larger batch size gives slower, more stable learning, however we found best performance at a batch size of 200 samples.

Also, to combat overfitting we have implemented an early-stopping rule that looks after every epoch at the validation dataset (unseen during training) and calculates the loss function value on this validation set. We stop when the validation loss reaches a minimum however to allow for local minima, we implement a patience parameter of 3 in our early stopping such that if the loss function doesn't improve for 3 epochs then the model stops training. This means that we never reach the maximum number of epochs of 100.

Finally, also to combat overfitting we have chosen a dropout parameter of 0.2. This means that at the points in our model where we have implemented dropout, 20% of the networks nodes and their respective weights are zeroed out during training. As we have said, dropout is implemented in the LSTM layers as well as after the dense layer before prediction.

Our model architecture is also strongly influenced by the hyper parameters that steer the shape of the loss function. These include the dimensions of our LSTM layers which was decided to be 128 hidden layers. Also, the dimensions of the word embeddings we used play a role in the type of features that can be found by the LSTM models. We have used pre-trained GloVe word embeddings of dimension 200 and pre-trained Emoji2Vec embeddings of dimension 300 to capture all of the contextual information in our sentences. These embeddings are also trainable during the gradient descent optimization to allow the word embeddings to be better suited to distinguishing patterns in our sentences.

Other important parameters that steer which input values the model has are the maximum sequence length of words per sentence. This is set at 100 words so we don't lose much training information. Similarly, our vocabulary is set at a maximum of 20,000 words, which in this training set is more than enough. These hyper-parameter values are found in a configuration file which is saved alongside our model and is also lined to in the appendix.

# Chapter 4
# Evaluation

## 4.1 Results

The comparative micro-averaged $F_1$ results for the models on the emotion categories of Happy, Sad and Angry are presented below in table 4.1. As has been noted, this evaluation metric ignores the model performance explicitly on the emotion class Others.

We begin presenting the results by presenting our baseline model. As described above this was a single LSTM model with minimal pre-processing and word embeddings from GloVe trained on Wikipedia data. This baseline model achieved overall micro-averaged $F_1$ on the emotions of .6089. It achieved best results on the Angry emotions and worst results on Happy conversations. The majority of the confusion in the model comes from an inability to discern whether an emotion is present or whether the conversation should be classified as Other.

| | Emotion $F_1$ | | | |
|---|---|---|---|---|
| Model | Happy | Sad | Angry | Micro $F_1$ |
| Baseline-LSTM | 0.558 | 0.599 | 0.670 | 0.6089 |
| BERT | 0.628 | 0.715 | 0.727 | 0.6915 |
| SS-LSTM | 0.597 | **0.808** | 0.736 | 0.7134 |
| CWE-LSTM | **0.705** | 0.746 | **0.740** | **0.7302** |

**Table 4.1:** $F_1$ scores

BERT achieves better results right out of the box when using transfer learning to adapt to our problem. The extra context information helps increase the macro $F_1$ by over 8 points from the baseline model. These gains are especially in the sad category where BERT increased by more than 11 $F_1$ points over the baseline. The Angry category is still the best predicted by the model with a category $F_1$ of .727.

However, comparing to the competition kick-off model from Gupta et al. (2017) we see that their SS-LSTM model performs better than BERT on the categories of Sad and Angry emotions, but is confused by Happy emotions. This translates to a marginally better micro-averaged $F_1$ score for the SS-LSTM model overall compared to BERT.

Our CWE-LSTM model however, performs the best in the Happy and Angry emotion categories, with especially large gains over BERT, SS-LSTM and the baseline in the Happy category. This results in a micro $F_1$ score for the CWE-LSTM model of .7302 which is the best overall score for the compared models. Notably it doesn't perform nearly as well as the SS-LSTM model in the Sad emotion category.

The other important factor in evaluating model performance is training time as shown in table 4.2. Of the three models we implemented and trained locally, the quickest was our Baseline-LSTM model. This makes sense since it was the least complex both in terms of pre-processing and model architecture. This resulted in a training time of just over 37 minutes, locally on a four-CPU computer.

The next fastest training time was by our CWE-LSTM model at under 90 minutes. Our CWE-LSTM model was the next most complex in terms of the number of training parameters, as well, so this number is in line with that added complexity. Finally, the training times for BERT were by far the highest at over a day. This was the most complex model in terms of number of parameters, but the model wasn't twenty times more complex than the CWE-LSTM model. This is also a surprisingly high number given that this is a transfer learning application, so that the weights of the BERT model have been pre-trained to be relevant for representing language. However, it should be noted that BERT's training is parallelizable on GPUs not on CPU's, so it is likely that if we were training on GPUs this difference would decrease.

| Model | Training Time Rounded to (hrs:mins) |
|---|---|
| Baseline-LSTM | 00:37 |
| BERT | 26:45 |
| CWE-LSTM | 01:24 |

**Table 4.2:** Training Times on 4 core Computer

We can also look at the confusion matrix for our best model, the CWE-LSTM model to see where it predicts incorrectly. This confusion matrix, shown in table 4.3, shows clearly the uneven distribution between the emotion labels in the test set. Others is by far the largest emotion category and most of the confusion comes from predicting Others when it should be an emotion, or the other way around. Perhaps not so surprisingly, there is very little overlap between the emotion categories where the model predicts the wrong emotion. The model never predicts the labels Sad or Angry for a sentence that is actually in the Happy category. There is some small confusion between Sad and Angry but that is small in comparison to the confusion between others and emotions. This suggests that one of the better ways to improve the model would be to focus on trying to improve the features that distinguish between an emotion and not. This will be discussed at more length in the following section.

We can also look at the loss function over the training epochs to get an idea of whether our early stopping rule works to prevent overfitting. The training loss decreases sharply over

| | Predicted | | | |
|---|---|---|---|---|
| True | Others | Happy | Sad | Angry |
| Others | 4403 | 91 | 85 | 98 |
| Happy | 77 | 204 | 1 | 2 |
| Sad | 37 | 0 | 204 | 9 |
| Angry | 52 | 0 | 7 | 239 |

**Table 4.3:** Confusion Matrix

the epochs however the validation loss is more unevenly decreasing. In fact, it reaches a minimum after 3 epochs and then rises afterwards.

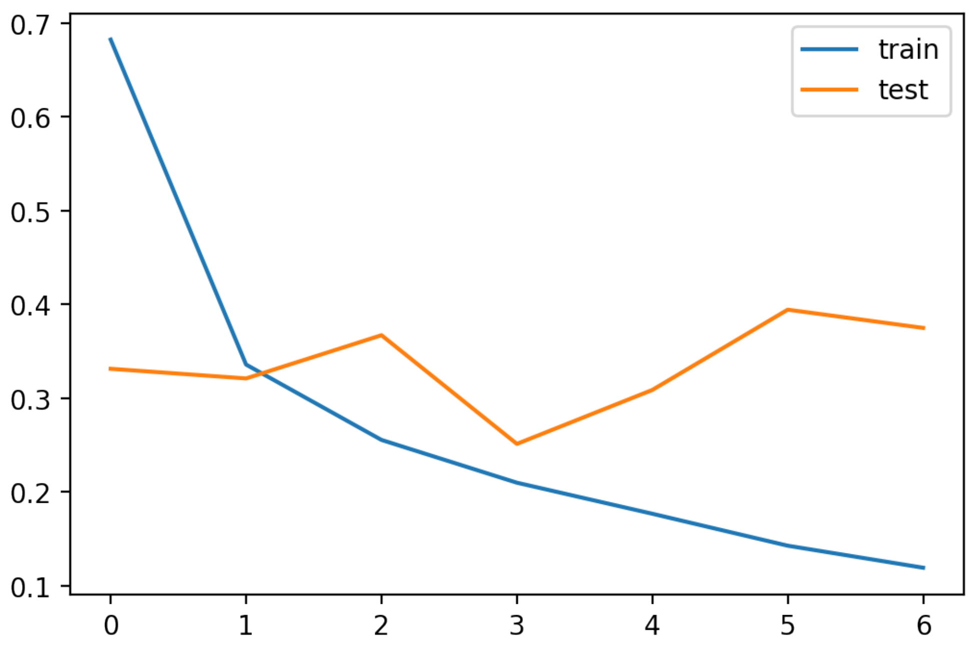


**Figure 4.1:** Training Loss for Training and Validation Sets

This can partially be explained by the extremely different distributions of emotions in the training and validation sets as shown below in table 4.4. The gradients are updated based on loss from the Training set, but the same gradient updates change the validation loss in other ways. There are also far fewer data points in the validation set compared to the training set, so this also contributes to higher variance in the effects of updating the loss. This also highlights the importance of early stopping since clearly after epoch 3 the improvements in training loss are not being translated to improvements in validation loss and thus it isn't worth using computing power to keep training.

| | Others | Happy | Sad | Angry | Size |
|---|---|---|---|---|---|
| Train | .496 | .141 | .183 | .181 | 30160 |
| Validation | .849 | .052 | .054 | .045 | 2755 |
| Test | .857 | .043 | .056 | .045 | 5509 |

**Table 4.4:** Emotion Class Distribution

## 4.2 Discussion

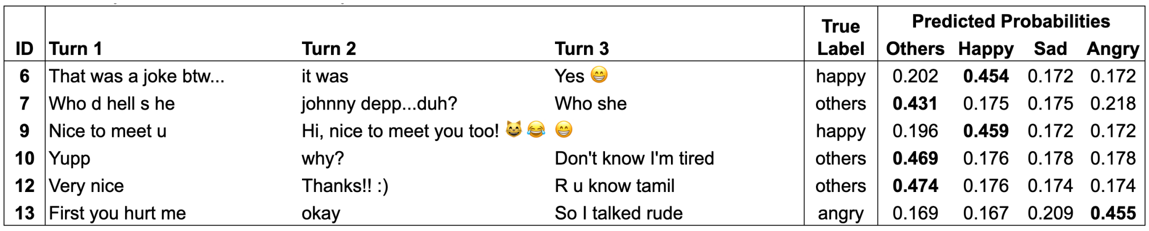
We will now discuss and attempt to explain the results we obtained above. We will do this both through looking at some example sentences and the predicted output probabilities of the different emotion classes. With this we can begin to see which features the model is finding important do distinguish between emotion categories. First, we will look at outputs of the CWE-LSTM model and then at the BERT model. We will also introduce some new conversations of our own creation to see if we can interrogate how well these results generalize outside the context of this dataset.

### 4.2.1 Model Interpretation

We will now look at example conversations and the models' predictions for a variety of situations. In the first three columns of each example we can see the first, second and third turns of the conversation, where the first and third turn are written by the same person. The next column is the correct emotion category label of the conversation and the four columns after are our predicted probabilities of the four Emotion categories.

#### CWE-LSTM Model Predictions

We can start by looking at some examples where our CWE-LSTM model predicts correctly, shown below in figure 4.2. Overall, it seems that the model finds a lot of true language features in the text and is quite good at discriminating between the Others category and the Emotion categories.

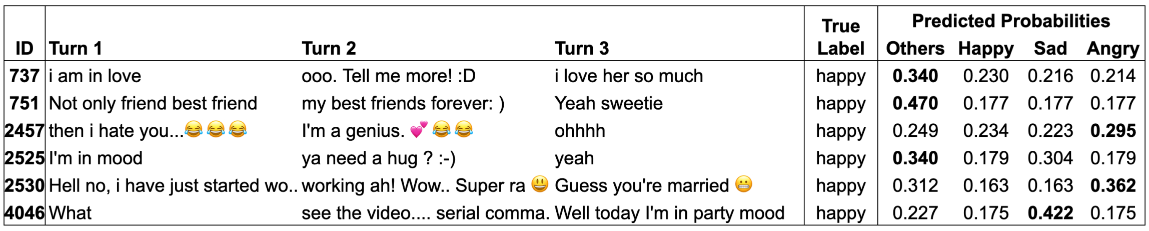


| | | | | True | Predicted Probabilities | | | |
|---|---|---|---|---|---|---|---|---|
| **ID** | **Turn 1** | **Turn 2** | **Turn 3** | **Label** | **Others** | **Happy** | **Sad** | **Angry** |
| **6** | That was a joke btw... | it was | Yes 😁 | happy | 0.202 | **0.454** | 0.172 | 0.172 |
| **7** | Who d hell s he | johnny depp...duh? | Who she | others | **0.431** | 0.175 | 0.175 | 0.218 |
| **9** | Nice to meet u | Hi, nice to meet you too! 😺 😂 | 😁 | happy | 0.196 | **0.459** | 0.172 | 0.172 |
| **10** | Yupp | why? | Don't know I'm tired | others | **0.469** | 0.176 | 0.178 | 0.178 |
| **12** | Very nice | Thanks!! :) | R u know tamil | others | **0.474** | 0.176 | 0.174 | 0.174 |
| **13** | First you hurt me | okay | So I talked rude | angry | 0.169 | 0.167 | 0.209 | **0.455** |

**Figure 4.2:** Correctly Classified Conversations by CWE-LSTM

However, just by looking at correctly classified conversations, we don't know exactly which features the model is using to predict emotion categories. Instead, to get a feeling for how the model is making predictions, it is often helpful to look where it is going wrong. We will first look at a selection of sentences where the true label was Happy, Sad, and Angry, but our CWE-LSTM model didn't predict correctly.

The first conversations are examples of where the true emotion category label was Happy but our model predicted another state. These are shown in figure 4.3. Firstly we can notice that the model seems to get caught up on certain negative sentiment words like "hate" in line 2457 and "hell" in line 2530, choosing to classify those conversations as Angry. Also, the model misclassifies line 4046 and almost misclassifies line 2525 as Sad. This could be because they both contain the word "mood" which the model may have learned often corresponds to a negative mood, however this is hard to tell definitively. Otherwise, it is just failing to read emotions into these conversations and classifying them as Others. Given the lack of context for lines 2525 and 737, this doesn't always seem like a bad choice even from the perspective of a human observer.



| ID | Turn 1 | Turn 2 | Turn 3 | True Label | Predicted Probabilities | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Others | Happy | Sad | Angry |
| **737** | i am in love | ooo. Tell me more! :D | i love her so much | happy | **0.340** | 0.230 | 0.216 | 0.214 |
| **751** | Not only friend best friend | my best friends forever: ) | Yeah sweetie | happy | **0.470** | 0.177 | 0.177 | 0.177 |
| **2457** | then i hate you...😂😂😂 | I'm a genius. 💕😂😂 | ohhhh | happy | 0.249 | 0.234 | 0.223 | **0.295** |
| **2525** | I'm in mood | ya need a hug ? :-) | yeah | happy | **0.340** | 0.179 | 0.304 | 0.179 |
| **2530** | Hell no, i have just started wo.. | working ah! Wow.. Super ra 😃 | Guess you're married 😬 | happy | 0.312 | 0.163 | 0.163 | **0.362** |
| **4046** | What | see the video.... serial comma. | Well today I'm in party mood | happy | 0.227 | 0.175 | **0.422** | 0.175 |

**Figure 4.3:** Misclassified Happy Conversations by CWE-LSTM

It is also interesting, that the probability of the Happy class is the second most likely choice for the model for line 737, but not for any of the other conversations. Looking back at figure 4.2 we see that the two Happy labelled conversations both included a happy emoji, which was a clear signal for our model since emojis encode a lot of emotion information. However, for less clear happy conversations, our model seems to have difficulty predicting the Happy emotion class. This is backed up by the results in figure 4.1 where the Happy emotion class has the worst performance for all of the models. Our CWE-LSTM model performed best out of all of the models in terms of classifying Happy conversations, but this is still a challenge for the model.

For the misclassified Angry labelled conversations in figure 4.4, it is also difficult as a human observer to set an emotion to these texts. From the examples before there seem to be some words that our model has associated with anger, like swear words and strong dislike words. This seems to have served our model well in general, since Angry is our model's best performing emotion category in the evaluation set. However, it looks like the model hasn't learned disappointment and defensiveness, which is what as a human we read into most these texts. These two soft emotions are similar to the emotion category Sad, which is the label predicted in lines 2679 and 3499.

Sad conversations also are difficult to classify correctly, as shown in figure 4.5. Lines 1755, 2133 and 3472 use words like "rude" and "annoying" which may have valence to anger in the embeddings and are misclassified as Angry. This shows that the model may be getting too hung up on individual words instead of stringing them together to sequences with more complex meanings. That said, the conversations in lines 1755, 2171, 4066 and 4177 have Sad as the second most likely label, so the model has clearly found some features to associate with this emotion category, though they aren't strong enough to change the model's predictions. These conversations seem to show that helplessness and loneliness are more difficult language

| ID | Turn 1 | Turn 2 | Turn 3 | True Label | Predicted Probabilities | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Others | Happy | Sad | Angry |
| **2679** | Ur cheater | no I'm not | U cheated me | angry | 0.176 | 0.174 | **0.473** | 0.176 |
| **3355** | What?😂 | what say what what la alamak | What is this nonsense | angry | **0.425** | 0.166 | 0.166 | 0.243 |
| **3499** | I'm seriously disappointed | Flush it :'( | Wtf | angry | 0.174 | 0.172 | **0.469** | 0.185 |
| **3515** | sarcasm? | Glad you liked it :) | what? I dont like it | angry | **0.442** | 0.169 | 0.169 | 0.220 |
| **3530** | Dont know | why? | What why...dont know means.. | angry | **0.402** | 0.148 | 0.148 | 0.301 |
| **3651** | I don't need your idea | I just want to tell you all... | I didn't want to hear that | angry | **0.432** | 0.165 | 0.169 | 0.235 |

**Figure 4.4:** Misclassified Angry Conversations by CWE-LSTM

features to identify with sadness than more clear forms of sadness.

| ID | Turn 1 | Turn 2 | Turn 3 | True Label | Predicted Probabilities | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Others | Happy | Sad | Angry |
| **1755** | My boyfriend is rude | U can always share | He doesn't care about me | sad | 0.181 | 0.168 | 0.316 | **0.336** |
| **2133** | Sry | Sorry for what?? | For annoying you | sad | 0.182 | 0.173 | 0.173 | **0.471** |
| **2171** | My life is so confusing | hats off t'ya | I wnt to share | sad | **0.311** | 0.198 | 0.290 | 0.200 |
| **3472** | Don't u find me hot | i will never find out anyway | So rude | sad | 0.179 | 0.174 | 0.174 | **0.473** |
| **4066** | Sorry dear | sorry for what...???? =/ | I'm leaving this world | sad | **0.398** | 0.190 | 0.223 | 0.190 |
| **4177** | Not fine | why? Tell me | Please call me | sad | **0.353** | 0.172 | 0.304 | 0.172 |

**Figure 4.5:** Misclassified Sad Conversations by CWE-LSTM

If we look in more detail at one emotion, we can start to narrow in on which language features the model is finding. Since the Angry class is the class where our model performs best, we can start in this emotion class. One of the recurring patterns in the Angry class is the use of swear words. Perhaps then it isn't strange that the model has picked up on this pattern and classifies these conversations as Angry with such high probability. The conversation in line 1262 for example, which doesn't include any clear swear words or dislike words is much closer to being classified as other. Here, however, the model has an angry emoji to use as a clue to classify the conversation as Angry.

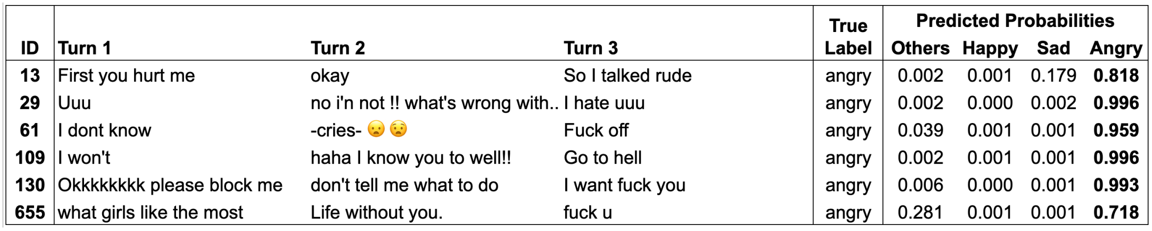
| ID | Turn 1 | Turn 2 | Turn 3 | True Label | Predicted Probabilities | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Others | Happy | Sad | Angry |
| **655** | what girls like the most | Life without you. | fuck u | angry | 0.286 | 0.152 | 0.152 | **0.410** |
| **675** | Why? | Because I'd rather go to the ... | I hate you | angry | 0.188 | 0.172 | 0.172 | **0.468** |
| **707** | I m breaking up with u | Rude. | Oh bitch pls | angry | 0.189 | 0.171 | 0.178 | **0.461** |
| **770** | Teach me doh | can u teach me french? | Fuck u | angry | 0.304 | 0.152 | 0.152 | **0.390** |
| **1262** | Keep it private | Will do. | 😡 | angry | 0.225 | 0.193 | 0.214 | **0.367** |
| **1301** | Then dobt talk to me | About what? | You are impossible.. 😡 bye | angry | 0.198 | 0.171 | 0.172 | **0.459** |

**Figure 4.6:** Correct Classified Angry Conversations by CWE-LSTM

## BERT Model Predictions

We now turn to evaluating the BERT model performance in more detail. We can start by looking at conversations which the BERT model correctly classified as Angry. This is the emotion class where the performance of BERT and our CWE-LSTM model were most comparable, with only a 0.013 difference in $F_1$ scores between the models. However, when we evaluate BERT's predictions on this same task, we can see some interesting differences between the two models. This is shown below in figure 4.7.

Firstly, we can see that the probabilities of the different emotion classes are much less evenly distributed, even if these conversations are often ambiguous. The probabilities of the Angry emotion class in lines 29, 109 and 130 are over 99%. These sentences are classified correctly, but none of the predictions from the CWE-LSTM model are nearly this high. This indicates that the BERT model is finding patterns that it thinks are much more predictive than they actually are.

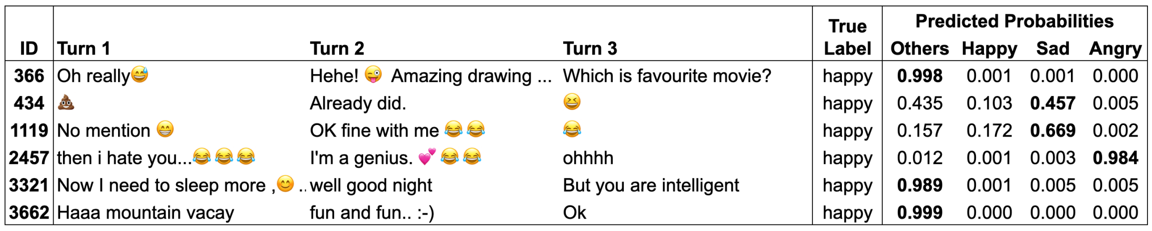


| | | | | True | Predicted Probabilities | | | |
|---|---|---|---|---|---|---|---|---|
| **ID** | **Turn 1** | **Turn 2** | **Turn 3** | **Label** | **Others** | **Happy** | **Sad** | **Angry** |
| **13** | First you hurt me | okay | So I talked rude | angry | 0.002 | 0.001 | 0.179 | **0.818** |
| **29** | Uuu | no i'n not !! what's wrong with.. | I hate uuu | angry | 0.002 | 0.000 | 0.002 | **0.996** |
| **61** | I dont know | -cries- 😦 😧 | Fuck off | angry | 0.039 | 0.001 | 0.001 | **0.959** |
| **109** | I won't | haha I know you to well!! | Go to hell | angry | 0.002 | 0.001 | 0.001 | **0.996** |
| **130** | Okkkkkkkk please block me | don't tell me what to do | I want fuck you | angry | 0.006 | 0.000 | 0.001 | **0.993** |
| **655** | what girls like the most | Life without you. | fuck u | angry | 0.281 | 0.001 | 0.001 | **0.718** |

**Figure 4.7:** Correct angry Conversations by BERT

If we look to where the BERT model has misclassified emotions, this pattern becomes even clearer. We can look at Happy labelled conversations which BERT has classified incorrectly. This is shown in figure 4.8.

One of the things that seems clear to a human reading these examples, is that the BERT model is not good at reading emotions into the emojis in these conversations. Lines 366, 434, 1119, 2457 and 3321 all contain context emojis in the first turn that point to a Happy label, but the BERT model hasn't learned these connections. That said, it does seem like the BERT model has learned that emojis carry emotional valence, since the three conversations that contain the most emojis, lines 434, 1119 and 2457 are the only example conversations that BERT has not classified as Others. Other than that, however, it is hard to see what the differences are in the conversations where the BERT model is 99.9% sure of its prediction like in line 3662, and the conversations where the prediction probabilities are under 50% like line 434.



| | | | | True | Predicted Probabilities | | | |
|---|---|---|---|---|---|---|---|---|
| **ID** | **Turn 1** | **Turn 2** | **Turn 3** | **Label** | **Others** | **Happy** | **Sad** | **Angry** |
| **366** | Oh really😅 | Hehe! 😜 Amazing drawing ... | Which is favourite movie? | happy | **0.998** | 0.001 | 0.001 | 0.000 |
| **434** | 💩 | Already did. | 😆 | happy | 0.435 | 0.103 | **0.457** | 0.005 |
| **1119** | No mention 😁 | OK fine with me 😂 😂 | 😂 | happy | 0.157 | 0.172 | **0.669** | 0.002 |
| **2457** | then i hate you...😂 😂 😂 | I'm a genius. 💕 😂 😂 | ohhhh | happy | 0.012 | 0.001 | 0.003 | **0.984** |
| **3321** | Now I need to sleep more ,😊 .. | well good night | But you are intelligent | happy | **0.989** | 0.001 | 0.005 | 0.005 |
| **3662** | Haaa mountain vacay | fun and fun.. :-) | Ok | happy | **0.999** | 0.000 | 0.000 | 0.000 |

**Figure 4.8:** Misclassified Happy Conversations by BERT

Misclassified Angry conversations by BERT are shown in figure 4.9. Lines 449, 1640 and 1903 contain emojis and have been classified as emotion categories instead of the Others category, though it is hard to say exactly why. Also, compared to the CWE-LSTM model, BERT doesn't seem to attach as much negative Angry weight to swear words like "fuck" in line 324. Other than that, it is still difficult to pick out exact features of the conversations that tip the BERT predictions towards one emotion class or another.

Misclassified Sad conversations are shown in figure 4.10. Here there are some conversations that are very close to being correctly classified as Sad, like line 121 and 1745. On the other hand, lines 1159 and 1949 are misclassified as Happy from Sad. Both lines, however,

| ID | Turn 1 | Turn 2 | Turn 3 | True Label | Predicted Probabilities Others | Happy | Sad | Angry |
|---|---|---|---|---|---|---|---|---|
| **307** | Why??? | cause it's annoying! | Ok leave it | angry | **0.977** | 0.000 | 0.000 | 0.022 |
| **324** | Favorite team in ipl | Bajaj Allianz owns the team. | Fuck you | angry | **0.966** | 0.001 | 0.000 | 0.033 |
| **449** | Ur waste | it's not wasted time | Im not gonna talk with u😡 😡 | angry | 0.001 | 0.003 | **0.565** | 0.431 |
| **1640** | You don't have it from outside | You have form the inside? Ar... | 😡 | angry | 0.042 | **0.908** | 0.033 | 0.017 |
| **1903** | Wow i didn't expect this frm u | no i didn't mean dere was s... | I thought you meant it😡 😡 😡 | angry | 0.079 | **0.482** | 0.372 | 0.067 |
| **2679** | Ur cheater | no I'm not | U cheated me | angry | 0.001 | 0.001 | **0.978** | 0.021 |

**Figure 4.9:** Conversations Incorrectly Predicted as Angry by BERT

include some combinations of words, that could possibly occur in other contexts. Being busy, like in line 1159 could also be a good thing, and doesn't necessarily point to a Sad emotion, and same with calling someone "Bro" like in line 1949. These word combinations are taken totally out of context, however, if this is truly what is making the model predict the Happy emotion class.

| ID | Turn 1 | Turn 2 | Turn 3 | True Label | Predicted Probabilities Others | Happy | Sad | Angry |
|---|---|---|---|---|---|---|---|---|
| **121** | Yes | Do you have help? | I don't have anyone.. | sad | **0.559** | 0.001 | 0.436 | 0.004 |
| **1159** | Yeah...I have been pretty busy. | busy being busy | 😞 | sad | 0.139 | **0.841** | 0.019 | 0.001 |
| **1249** | you are not listening | Sorry, what was that? | leave it | sad | 0.013 | 0.000 | 0.002 | **0.984** |
| **1745** | I don't like..that's it | you are garbage. | 😭 | sad | 0.019 | 0.048 | 0.452 | **0.480** |
| **1949** | Yoo too bro | U AT HOME | Yeah😞 | sad | 0.096 | **0.900** | 0.003 | 0.001 |
| **2171** | My life is so confusing | hats off t'ya | I wnt to share | sad | **0.999** | 0.001 | 0.000 | 0.000 |

**Figure 4.10:** Misclassified Sad Conversations by BERT

### 4.2.2 Generalization Capability

Another way of interrogating the predictions of these two models is to see what we can give them as input sentences that confuses them. We start by giving the CWE-LSTM model three conversations which it predicts correctly, this is shown in figure 4.11.

| ID | Turn 1 | Turn 2 | Turn 3 | True Label | Predicted Probabilities Others | Happy | Sad | Angry |
|---|---|---|---|---|---|---|---|---|
| **0** | You are such a good friend | Thanks! | 😁 | happy | 0.221 | **0.445** | 0.167 | 0.167 |
| **1** | Why don't you ever write me back? | Cause you're a creep. | I know... I am... 😔 | sad | 0.248 | 0.193 | **0.345** | 0.213 |
| **2** | I don't get it... why can't I ever get a date? | 🙃 too ugly? | Screw you! | angry | 0.192 | 0.176 | 0.176 | **0.458** |
| **3** | Too much pollution in the world | Or too many people… | Or both… | others | **0.475** | 0.175 | 0.175 | 0.175 |

**Figure 4.11:** Newly Created Conversation Classification

Then from there we can tweak these sentences until the model guesses wrong, to see where the predictive features are. This has been accomplished in figure 4.12.

For the Happy conversation, we noted that the model seems to have a difficulty distinguishing between swearing as an intensity word and swearing in terms of negative sentiment. This might stem all the way from the word embeddings where one can hypothesize that words like "fuck" end up in a negative sentiment part of the vector space. With that said however, we can easily get the model to predict this happy sentence incorrectly by adding a swear word to the sentence.

| ID | Turn 1 | Turn 2 | Turn 3 | True Label | Predicted Probabilities | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Others | Happy | Sad | Angry |
| 4 | You are such a good friend | Thanks! | I really fucking mean it! 😁 | happy | 0.264 | 0.165 | 0.159 | **0.411** |
| 5 | Why don't you ever write me back? | Cause you're a creep. | I know... I am... | sad | **0.446** | 0.178 | 0.178 | 0.198 |
| 6 | I don't get it… why can't I ever get a date? 🙃 | too ugly? | Screw you! 😭 | angry | 0.190 | 0.188 | **0.419** | 0.202 |
| 7 | Too much pollution in the world 😭 | Or too many people… | Or both… | others | 0.294 | 0.173 | **0.360** | 0.173 |

**Figure 4.12:** Tricking CWE-LSTM

Similarly, we noted that the model has some difficulty distinguishing between emotional and not emotional conversations. We hypothesize that in the Sad conversation the model is putting a lot of weight on the emoji in the final sentence. This is proven true, since if we remove the emoji that looks like it has given up, then we get a missclassification to Others. It is interesting to note that the second ranked prediction is actually Angry, which is perhaps understandable given the slightly accusatory tone in the first turn of the conversation.

The Angry sentence is tricked by adding a crying face emoji to the final sentence. This change the prediction from Angry to Sad, which makes sense, but means that the model has only associated this crying emoji with sad tears, not with tears of anger or frustration. Angry is still the second choice of the model however, indicating that it sees that there are angry features there, but doesn't think they are as important as the crying face. This is possibly because the emoji strengthens the feelings of sadness and loneliness that are present in the first turn of the conversation as well. Another hypothesis is that since the word "screw" has different meanings depending on the context, it is a difficult word to encode in word embeddings. This could mean that "Screw you!" doesn't get the negative sentiment weight in the CWE-LSTM model that we would assign to it as a human observer.

Finally, the Others class conversation can be tricked by adding in an emoji as well. This is interesting since we have simply added a crying emoji to the first turn, but this is enough to tip the classification over to an emotion category. This demonstrates that the model is looking backwards in context to gain information about emotion, but again is adding too much weight to emojis.

To conclude this section, we can also look at BERT's predictions for the same sentences in figure 4.13 and attempts to trick the model in figure 4.14. It is interesting to note that the BERT predictions on this set actually are more accurate than the CWE-LSTM model, though harder to interpret. As before, the BERT predictions are surer of themselves than the CWE-LSTM predictions. Also, as before the BERT predictions don't seem to put as much weight on emojis compared to the CWE-LSTM model, so changing the emojis doesn't trick the BERT predictions as much. It also seems that the model is similarly sensitive to swear words as the CWE-LSTM model, so that is still a way to trick BERT into guessing Angry with high probability.

| ID | Turn 1 | Turn 2 | Turn 3 | True Label | Predicted Probabilities | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | Others | Happy | Sad | Angry |
| 0 | You are such a good friend | Thanks! | 😁 | happy | 0.216 | **0.770** | 0.013 | 0.002 |
| 1 | Why don't you ever write me back? | Cause you're a creep. | I know... I am... 😔 | sad | 0.022 | 0.043 | **0.916** | 0.019 |
| 2 | I don't get it… why can't I ever get a date? 🙃 | too ugly? | Screw you! | angry | 0.003 | 0.000 | 0.002 | **0.994** |
| 3 | Too much pollution in the world | Or too many people… | Or both… | others | **0.997** | 0.000 | 0.001 | 0.001 |

**Figure 4.13:** BERT Comparison Correct

| ID | Turn 1 | Turn 2 | Turn 3 | True Label | Predicted Probabilities | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | **Others** | **Happy** | **Sad** | **Angry** |
| **4** | You are such a good friend | Thanks! | I really fucking mean it! 😁 | happy | 0.011 | 0.030 | 0.021 | **0.938** |
| **5** | Why don't you ever write me back? | Cause you're a creep. | I know... I am... | sad | 0.450 | 0.001 | 0.010 | **0.539** |
| **6** | I don't get it… why can't I ever get a date? | 🙃 too ugly? | Screw you! 😭 | angry | 0.004 | 0.006 | 0.017 | **0.973** |
| **7** | Too much pollution in the world 😭 | Or too many people… | Or both… | others | **0.833** | 0.016 | 0.146 | 0.005 |

**Figure 4.14:** BERT Comparison after Tricking

### 4.2.3 Model Comparison and Reflection

From the figures above it is clear that there are problems with both models when it comes to their predictions. This section will detail the failings of both models in turn. From there we will turn to comparing the models in terms of their performance, structure, and ease of use. Finally, we will propose improvements to the models along with providing other discussion around the problem as a whole.

#### Model Failings

Even though BERT is a state of the art language representation model, it performed worse overall than the CWE-LSTM model. This was based on several failings that have been hinted at in the discussion above. Firstly, the BERT model's predictions seem overfitted to the training data. We suspect this due to the high certainty of the model's predictions. This points to the idea that the model is learning language patterns that aren't predictive of emotion category, but were present in the training data set. This can be compared to the BERT model hallucinating that it has found strong patterns but in actuality these patterns are not important. The BERT model has over 110 million weights to be tuned, however, so it makes sense that it can find very specific patterns.

This leads into another downside of the BERT model and its predictions, that they are hard to interrogate and understand. Due to the probable overfitting, the model seems to have learned strong patterns, but both because of the automatic pre-processing and the large number of weights, it is very hard to get an idea of where the model is going wrong. As was mentioned above, there is very little to distinguish a conversation where the model predicts an output probability of one of the classes as over 99%, compared to an example where the model is less sure. Furthermore, since the BERT model takes so long to train on CPU's, it was very difficult to attempt to iteratively improve the model's predictions. This meant we weren't able to tune the model training in any meaningful way.

Also, the BERT model seems to not have any generalized knowledge of emojis or even a robust method of learning emotion information from the training set examples containing emojis. The extent of the model's knowledge about emojis seems to be that the presence of emojis probably raised the likelihood that the model would predict an emotion category instead of Others. The model's inability to learn more from these highly emotionally loaded symbols poses an interesting question about which type of context information BERT can be tuned to learn from.

Finally, it also seemed that BERT had trouble with shorter sequences and more broken communication. Thus, another possible weakness of the BERT model is dealing with shorter, less context rich language information. Attention and the machinery of BERT is designed to solve problems of long term dependence, so perhaps this is not the context where BERT can shine.

The CWE-LSTM model also had some notable patterns of failure in this emotion classification task. Firstly, the CWE-LSTM model is clearly too focused on individual words and symbols to which it attaches large emotional meaning. Examples of this are emojis, where we showed that changing one emoji in the conversation changes a lot of the emotional information that the CWE-LSTM model picks up. Another example is the use of swear words being connected to the Angry emotion category. This may be true in general, but also points to an easy way of tricking the predictions of the CWE-LSTM model. The CWE-LSTM model places disproportionate weight on these types of individual, negative context words without getting a better idea of the context of the sentence.

This leads to another clear failing of the model in dealing with subtlety in emotions. The examples of misclassified emotions almost all fell into emotion categories adjacent to the main emotion classes of Happy, Sad or Angry. It should be noted that some of these sentences are hard for humans to accurately classify, as well, but the model was disproportionately missing edge cases of despair, loneliness, frustration, amusement and other less clear emotions. This could be due to a lack of training examples for these types of emotional expressions, or could be due to our model not understanding ambiguity in these contexts.

Finally, a large but unquantified failing of our CWE-LSTM model is that it is so domain specific to this problem context. We have not tested this in any meaningful way, but due to its structure, embeddings and pre-processing it can be hypothesized that the model would need substantial work to bring it up to the same level of performance on another dataset with similar classification goals.

## Model Comparison

With that said, however, the CWE-LSTM model did perform well on this specific dataset and problem, especially compared to the BERT model. This can be traced to three main sources of difference between the models. These are: differences in domain knowledge and problem specificity, differences in architecture and differences in ease of iterative improvement.

Firstly, the CWE-LSTM model is a domain specific model with embeddings trained specifically on Twitter data and emoji representations. This is a completely contrast compared to the BERT model with word-piece embeddings which is trained on perfectly written English from Wikipedia and newspaper text. Twitter data is typified by broken grammar, domain specific abbreviations and the use of symbols like emojis and punctuation in unique ways. BERT's suitability for this type of language data may be improved in the future but for the moment, the tokenizer and vocabulary isn't optimal for this data. For these reasons, it is not surprising that the more domain specific model performs better for this data.

Next, the CWE-LSTM model architecture was developed directly for this application as opposed to adapting the existing BERT model which was developed for totally differnt applications. BERT was developed for tasks like question answering, machine translation and other very information intensive tasks. This means that its attention machinery is very large and powerful for long context dependencies, but perhaps is too large and powerful for three turn Twitter conversations. Tweets in general must be less than 280 characters, so there it is a totally different task to understand this language information compared to reading several paragraphs for question answering. Thus, BERT can easily become overfitted on this type of data with so few training samples and create unnecessarily specific representations of the training data. This in turn leads to the type of uninterpretable predictions we obtained in our implementation.

Furthermore, there are structural differences in the inputs between the CWE-LSTM model and BERT. The CWE-LSTM model has four separate inputs (3 conversation turns and the emojis) to highlight for the model the different sources of language information in the conversations. BERT on the other hand takes a concatenation of the first and second turn of the conversation as its first input, and the third turn as its second input. This artificial splitting of the conversation is also probably a source of error for the BERT mode. This is because there are many parts of the BERT implementation that cannot be changed without writing a lot of new code.

This leads to the final significant difference between the models, the ease with which the models can be adapted and improved. The CWE-LSTM model was created through a process of iterative improvement over the baseline, however this was possible because of the modularity and ease of training this model. Pieces of the CWE-LSTM code can be added and taken away from the implementation with ease. This is a contrast to using a pre-trained model like BERT where a lot functionality like its tokenizer, its pre-processing, and the encoder structure is essentially a black box. Even changing hyperparameters like the learning rate, or making training decisions about whether all model weights should be trainable or only a subset become difficult decisions. Since the model takes so long to train, it was unreasonable to test a large range of learning rate values or transfer learning training depths. For that reason, it is nearly sure that our particular implementation of BERT is not the optimal implementation for this problem setting, but it is the best we could do based on the training resources at hand.

### 4.2.4 Reflection and Model Improvements

Twitter, as we have said, is not a communication platform that is natively designed for conversations. Nor is it designed to foster correct, rich language information. Even for human observers, the emotional content of these conversations are not always clear. In the original competition kick-off paper, there was discussion about how difficult it is to create high quality data from this domain. This is both because it is difficult to find a representative sample of conversations on the platform, but also because it was difficult to get the human judges to agree on which labels to apply to these conversations. For this reason, we can probably question some of the dataset labels, as well as the overall data quality.

However, despite these difficulties, we were able to create a CWE-LSTM model that performed well on this emotion classification task, and demonstrated its effectiveness over a larger, state of the art language model in BERT. This is a valuable task in itself and has helped us nuance the field of NLP and specifically emotion classification for ourselves.

Not surprisingly, there are many ways of improving our work to get a better $F_1$ score in evaluation, or to make the models more interpretable. We have not included any models which utilize character sequence information instead of word sequence information, and this is potentially a way of boosting our performance in the CWE-LSTM model. Since we only had 83% coverage of our dataset, there is clearly more language information present in our text than what could be found in embeddings and this could be tapped into through character sequence information. This problem could possibly be solved through more involved pre-processing.

Another idea is to attempt to first "translate" Twitter text to full conversations before attempting emotion classification. This could perhaps be accomplished by converting the conversations to sounds to get an understanding of what the writers were trying to say. Otherwise, a spell check applied to the text could work well in pre-processing. However, as likely as not, this could just inject more uncertainty into the task and raise the variance of the predictions.

Also, it should be noted that we have not used any model averaging or ensembling techniques which are often features of winning competition entries. Furthermore, since we saw an example of a model in the SS-LSTM model which performed very well on Sad labelled conversations in comparison to our CWE-LSTM model, there is clearly something to be learned from the inclusion of sentiment specific Twitter information.

The BERT model meanwhile, has many areas of improvement over its current implementation in this thesis. The word-piece embeddings inability to deal with emojis is a clear point where small improvements might make a large difference in model performance. As of writing, these discussions are underway on the BERT GitHub page, so we can expect this type of functionality to be implemented sometime in the future.

However, most simple ways of improving this model would require more computing power to be able to test. This could be in the form of access to GPU's for training, or access to a computer with more computing cores. This would allow us to test questions about the transfer learning procedure including tests of different learning rates, and the optimal number of trainable weights in the parameter that are necessary to tune BERT to our classification problem. Since the model became so overfitted when tuning all the weights, it is likely that only tuning a subset of weights in the last encoder layers would perform better on our task.

In fact, the BERT model's size is problematic for other reasons. As we have said, BERT is a general model which performs well on a range of tasks outside of the context of emotion classification in these Twitter conversations. Due to its generality, the BERT model needs to be quite large, however in this case the model could possibly perform better through some form of pruning to take away unimportant parts of the architecture for this problem. This raises the question about whether large, general models are always better than smaller, more specific models. A hypothesis which should be verified is whether in specific modelling situations where BERT does well, a smaller model could do just as well with less computing power. In the age of climate awareness, it is also a point of contention about whether it is worth spending the energy to train such large models, since such training processes are extremely resource intensive.

A final piece of reflection on the use of deep learning models of this kind is that the $F_1$ score is maybe not the best evaluation metric for all situations. If content moderation was the task, then maybe the loss function should penalize Angry conversations which the model missed as more important than Happy conversations. Then it might not be such a large problem that the CWE-LSTM model gets hung up on swear words and negative valence words in its predictions.

# Chapter 5
# Conclusions

We have studied two different models for language interpretation. BERT, a recent transfer learning model from Google and the CWE-LSTM model. We find that the latter outperforms BERT on the task of classifying emotions in Twitter data. The purpose of general representation models such as BERT is to be applicable on a wide range of problems. However, when dealing with very domain specific data, a more direct approach may be advisable. The CWE-LSTM is a good example of how a slimmer model, more specialized model can be advantageous over tuning a generalized model with transfer learning.

Previously, the idea of finding large general networks was where many researchers put emphasis. Now, we see a trend of more nimble and problem specific networks. Creation of this type of massive generalized model is expensive, both with respect to time and resources. Thus, understanding when to use what tool is an important skill in applications of machine learning.

To summarize, there are many decisions relevant in constructing a neural model for a specific natural language processing problem. From choosing word embeddings and training methods to designing a model architecture, there are many options and factors to consider. The advantage of using a transfer learning model is to skip these steps. However, there are limits to where and when such approaches are relevant.

Despite rapid development, machine learning models are still not near human performance for many tasks. We show that neural language models can gain an understanding of the emotion categories in this report, however miss understanding more subtle meaning in text, like sarcasm, despair or amusement. To correctly classify the complex emotions in such sentences, we still have a way to go. Hopefully, in solving this problem, we have learned important features and characteristics that are generalizable to other types of problems.

Language is constantly evolving and dynamic. Whether it be chatting with emojis or purposely misspelling words to create new meanings in a written sentence, language nuances abound. The task of capturing and modelling such nuances in different language domains be it on Twitter or elsewhere is interesting and important future work.

This field is rapidly evolving and new techniques are proposed every month. This is an exciting time to be a natural language processing researcher and hopefully it will only get more exciting as more of these questions are answered. This paper forms a piece of the chain of understanding that will hopefully link us towards better language understanding by machine intelligence systems.

# Bibliography

James Allen. *Natural Language Understanding (2Nd Ed.)*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1995. ISBN 0-8053-0334-0.

Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv e-prints*, abs/1409.0473, September 2014. URL `https://arxiv.org/abs/1409.0473`.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, March 2003. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=944919.944966`.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016. URL `http://arxiv.org/abs/1607.04606`.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel P. Kuksa. Natural language processing (almost) from scratch. *CoRR*, abs/1103.0398, 2011. URL `http://arxiv.org/abs/1103.0398`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL `http://arxiv.org/abs/1810.04805`.

Ben Eisner, Tim Rocktäschel, Isabelle Augenstein, Matko Bosnjak, and Sebastian Riedel. emoji2vec: Learning emoji representations from their description. *CoRR*, abs/1609.08359, 2016. URL `http://arxiv.org/abs/1609.08359`.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. doi: 10.1207/s15516709cog1402_1. URL `https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1`.

J. R. Firth. Applications of general linguistics. *Transactions of the Philological Society*, 56(1): 1–14, 1957. doi: 10.1111/j.1467-968X.1957.tb00568.x. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-968X.1957.tb00568.x`.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017. URL `http://arxiv.org/abs/1705.03122`.

Felix A. Gers, Jürgen Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471, 2000.

Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602 – 610, 2005. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2005.06.042. URL `http://www.sciencedirect.com/science/article/pii/S0893608005001206`. IJCNN 2005.

Umang Gupta, Ankush Chatterjee, Radhakrishnan Srikanth, and Puneet Agrawal. A Sentiment-and-Semantics-Based Approach for Emotion Detection in Textual Conversations. 2017.

Richard H R Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951, 2000. ISSN 1476-4687. doi: 10.1038/35016072. URL `https://doi.org/10.1038/35016072`.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9 (8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

W. John Hutchins. Machine Translation over fifty years. *Histoire Épistémologie Langage*, 22(1): 7–31, 2001. ISSN 0750-8069. doi: 10.3406/hel.2001.2815.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759, 2016. URL `http://arxiv.org/abs/1607.01759`.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *CoRR*, abs/1602.02410, 2016. URL `http://arxiv.org/abs/1602.02410`.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000. ISBN 0130950696.

Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014. URL `http://arxiv.org/abs/1405.4053`.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-13360-1.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: the penn treebank. *Computational Linguistics*, 19(2):313—-330, 1993. ISSN 0891-2017.

Warren Sturgis McCulloch and Warwick Pitts. A logical calculus of the ideas immanent in nervous activity. 1943. *Bulletin of mathematical biology*, 52 1-2:99–115; discussion 73–97, 1943.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013a. URL `http://arxiv.org/abs/1301.3781`.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013b. URL `http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionali` `pdf`.

Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. A survey of the usages of deep learning in natural language processing. *CoRR*, abs/1807.10854, 2018. URL `http://arxiv.org/abs/1807.10854`.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822, 2018. URL `http://arxiv.org/abs/1806.03822`.

Andrew Y. Ng Richard Socher, Alex Perelygin, Jean Y.Wu, Jason Chuang, Christopher D. Manning and Christopher Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. *PLoS ONE*, 8(9): 1631–1642, 2013. ISSN 19326203. doi: 10.1371/journal.pone.0073791. URL `http://nlp.stanford.edu/{~}socherr/EMNLP2013{_}RNTN.pdf{%}5Cnhttp://www.aclweb.org/anthology/D13-1170{%}5Cnhttp://aclweb.org/supplementals/D/D13/D13-1170.Attachment.pdf{%}5Cnhttp://oldsite.aclweb.org/anthology-new/D/D13/D13-1170.pdf`.

C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979. ISBN 0408709294.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL `http://arxiv.org/abs/1609.04747`.

Armin Seyeditabari, Narges Tabari, and Wlodek Zadrozny. Emotion detection in text: a review. *CoRR*, abs/1806.00674, 2018. URL `http://arxiv.org/abs/1806.00674`.

Robert Speer, Joshua Chin, and Catherine Havasi. Conceptnet 5.5: An open multilingual graph of general knowledge. *CoRR*, abs/1612.03975, 2016. URL `http://arxiv.org/abs/1612.03975`.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Duyu Tang, Furu Wei, Bing Qin, Ting Liu, and Ming Zhou. Coooolll: A deep learning system for twitter sentiment classification. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 208–212, Dublin, Ireland, August 2014a. Association for Computational Linguistics. doi: 10.3115/v1/S14-2033. URL `https://www.aclweb.org/anthology/S14-2033`.

Duyu Tang, Furu Wei, Nan Yang, Ming Zhou, Ting Liu, and Bing Qin. Learning sentiment-specific word embedding for twitter sentiment classification. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1555–1565, Baltimore, Maryland, June 2014b. Association for Computational Linguistics. doi: 10.3115/v1/P14-1146. URL `https://www.aclweb.org/anthology/P14-1146`.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 384–394, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=1858681.1858721`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL `http://arxiv.org/abs/1706.03762`.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Gregory S. Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

Lei Zhang, Shuai Wang, and Bing Liu. Deep learning for sentiment analysis : A survey. *CoRR*, abs/1801.07883, 2018. URL `http://arxiv.org/abs/1801.07883`.

Will Y. Zou, Richard Socher, Daniel Cer, and Christopher D. Manning. Bilingual word embeddings for phrase-based machine translation. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1393–1398, Seattle, Washington, USA, October 2013. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/D13-1141`.

# Appendices

# Appendix A
# Important Links

- Github link to our code repository: `https://github.com/Njoselson/natemusMasters`
- BERT download link from TensorFlow Hub: `https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1`
- GloVe embeddings download: `https://nlp.stanford.edu/projects/glove/`
- Emoji2Vec embeddings repository: `https://github.com/uclmr/emoji2vec`
- Codalab EmoContext competition link: `https://competitions.codalab.org/competitions/19790#learn_the_details`
- SemEval conference webpage: `http://alt.qcri.org/semeval2019/`