## Acknowledgements

All sentences or passages quoted in this report from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations which are not the work of the authors of this report are not have been used (where possible) with the explicit permission of the originator and are specifically acknowledged.

We would like to thank our mentors Dr.Mydhili K Nair and Nihal Nayak for guiding us throughout the project and enlightened us with great knowledge.We would also like to thank Yahoo Research and Joel Tetreault for making their data available.

## Abstract

Any given expression of thought can be conveyed in a variety of styles without changing its meaning. The different styles include colloquial/professional, personal/impersonal, polite/impolite, formal/informal etc and these variations are commonly used to adapt the content to a specific context, audience, or purpose. However, applying these stylistic variations is a manual process and Style transfer is an emerging field in computational linguistics which deals with automating the above-mentioned process. In this project, we aim to translate a given sentence in an informal style to a more formal style.

Informal Language is more casual and spontaneous. It is more generally used more when communicating with friends and family. The tone is more personal and colloquial with greater usage of slangs. They need not always follow grammar rules and is often delivered in a passive voice. Formal Language is more used in a professional or in an academic setting. They are less personal than informal and always follow grammar rules. It is usually delivered in an active voice with a third person perspective.

With the ever-increasing usage of social media, people tend to develop writing skills which are very skewed towards the informal style and they often find it really difficult to express things in a formal setting. Our tool can be used to tackle the above problem where users can improve their writing style to deliver content in a professional tone.

## Table of Contents

## List of Figures

## List of Tables

# CHAPTER 1

# Introduction

# 1. Introduction

In the process of communication, it is necessary that the integrity of the message communicated must not be tampered with as well as misunderstood. The communication style and tone play an important role. Today we have artificial conversation agents that are transforming our lives as their applications are limitless and are soon to become a major part of our personal life. When subjected to different scenarios the style and tone of communication delivered by these agents must match the expected sentiment. For example, when a business letter of rejection is translated from formal style to an informal style and delivered the integrity of the message as well as the tone of rejection must be present.

## 1.1 Motivation

Our project is based on style transfer of English sentences from informal to formal using Neural Machine Translation(NMT). Today a major part of the population of the world are second language English learners and due to the influence of social media, the authenticity of English is slowly deprecating. The advent of social media has introduced the world to grammatical errors, short-forms, slangs, unknown abbreviations, and even emoticons are being exploited in a similar fashion.

This has affected many second language English learners and has hindered their writing and speaking skills which impacts their career growth negatively.

To address these problems our project comes in the picture. It is an educative tool that helps you translate informal sentences to formal sentences and also will help you improve your linguistic skills and make you sound professional.

## 1.2 Constraints and Requirements

In our project, we implemented the best techniques and employed the best practices followed in the field of NMT. The major requirements for this project were :

- A proper parallel Corpus that had formal and informal sentences.
- Good evaluation metrics.
- High Computation Power like the use of GPU’s.
- Need for proper neural sequence modeling
- Huge storage (approx. 40GB of data, embeddings, and models)

The major constraints of the project were:

- Due to low computation power, the training of NMT models was a hardship.
- Small Corpus size, GYAFC dataset has 1Lakh pair of sentences which is very low for NMT.
- Evaluation metrics are still an open research problem and hence we had to use different metrics to evaluate the model correctly.

## 1.3 Problem Statement

Applying Style Transfer to translate Informal sentence to Formal sentence using Neural Machine Translation. Then to identify the style of a sentence and provide the degree of formality and informality. Also calculating the fluency and meaning preservation scores of the translated sentences

## 1.4 Scope and  Objectives

- To use GYAFC dataset as a benchmark for style transfer
- To build good heuristics to preprocess the datasets
- To aim at solving out of vocabulary problems by using BPE
- To use different neural modeling sequences to improve output
- To be able to translate informal to formal sentences
- To calculate the degree of formality of translated sentences
- To build a user-friendly web application for easy use
- To be able to guide second language English learners
- To build a educative tool to help in English proficiency

## 1.5 Proposed Model

- There are two main ways (or) styles of writing text in the English language
    - Formal style
    - Informal style

- **Formal** language is less personal and is often used when writing for professional and academic purposes.

- **Informal** language is more casual and spontaneous and is used when communicating with friends or family either in writing or in conversation. It uses a more personal tone.

- They are mainly distinguished by :
    - Contractions
    - Phrasal verbs
    - Slang/Colloquialism
    - Greater usage of First person pronouns

- We approach the task of Style transfer with machine translation. There are totally 3 main phases:
    - Preprocess
    - Train
    - Translate

- The GYAFC dataset is preprocessed using 6 basic steps
    - Capitalization
    - Lowercase words with all upper cases
    - Expand contractions
    - Replace slang words
    - Replace swear words
    - Remove character repetition

- In the training phase, the preprocessed sentences are tokenized and passed through our neural sequencing model and weights are obtained.Finally, the output sentence is generated in the translation phase where we use beam search.

## 1.6 Organization of Report

**Chapter-2** contains literature review and also covers the topics such as style transfer with parallel data, creation of parallel data and attention based NMT.

**Chapter-3** covers the system design and analysis which describes the overall process flow of our project and the important modules in them.

**Chapter-4** covers the modelling and implementation that describes the important modules in the process flow in detail like Dataset Description, Neural Machine Translation models and simple architectures, encoders and decoders, preprocessing steps like rule based techniques and BPE, training steps and finally translation process.

**Chapter-5** covers the testing, results and discussion. Here we have described in detail the different evaluation metrics we have implemented to present the effectiveness of our approach as well as the results obtained.

**Chapter-6** contains the Conclusion and Future works. In this project, the scope for future work is very vast and a huge research in the trend.

**Chapter-7** contains Bibliography that lists all the important papers we have referred to get a complete understanding of the previous works done in this field.

# Chapter 2

# Literature Review

## 2. Literature Review

### 2.1 Style transfer with parallel data

One of the very first works in style transfer related to formal and informal sentences was attempted by (**Sheikha and Inkpen (2011)**). Here they approach the task as a natural language generation task. They first do a detailed study of the characteristics of each type of sentences and use them as rules when generating sentences. They build a system which takes two inputs, a sentence and a parameter indicating the desired style in which output is required, and perform style transfer by replacing certain words based on the rules learnt earlier.

(**Xu et al. (2012)**) were the very first in approaching transfer as a sequence to sequence task. Their work deals with trying to translate sentences in Shakespeare's style to that of a modern English and vice versa. Here they create their own parallel corpus of around 30K. They do so by scraping pre-existing translations of Shakespeare's plays from Wikipedia. They apply machine translation system to the corpus collected above and perform

### 2.2 Style transfer without parallel data

Another direction of research directly controls certain attributes of the generated text withoutusing parallel data.

(**Hu et al. (2017)**) They propose a new neural generative model which combines variational autoencoders (VAEs) and holistic attribute discriminators for effective imposition of semantic structures. The model can alternatively be seen as enhancing VAEs with the wake-sleep algorithm for leveraging fake samples as extra training data. With differentiable approximation to discrete text samples, explicit constraints on independent attribute controls, and efficient collaborative learning of generator and discriminators, our model learns interpretable representations from even only word annotations, and produces sentences with desired attributes of sentiment and tenses. Quantitative experiments using trained classifiers as evaluators validate the accuracy of short sentence and attribute generation. Here they control the sentiment and the tense of the generated text by learning a disentangled latent representation in a neural generative model.

**(Ficler and Goldberg (2017) )** Most work on neural natural language generation (NNLG) focus on controlling the content of the generated text. Here they experiment with controlling several stylistic aspects of the generated text, in addition to its content. The method is based on conditioned RNN language model, where the desired content as well as the stylistic parameters serve as conditioning contexts. They also demonstrate the approach on the movie reviews domain and show that it is successful in generating coherent sentences corresponding to the required linguistic style and content. Their main aim is to control several linguistic style aspects simultaneously by conditioning a recurrent neural network language model on specific style (professional, personal, length) and content (theme, sentiment) parameters.

**(Sennrich et al. (2016a))** Many languages use honorifics to express politeness, social distance, or the relative social status between the speaker and their addressee(s). In machine translation from a language without honorifics such as English, it is difficult to predict the appropriate honorific, but users may want to control the level of politeness in the output. The authors here, perform a pilot study to control honorifics in neural machine translation (NMT) via side constraints, focusing on English→German. We show that by marking up the (English) source side of the training data with a feature that encodes the use of honorifics on the (German) target side, we can control the honorifics produced at test time. Experiments show that the choice of honorifics has a big impact on translation quality as measured by BLEU, and oracle experiments show that substantial improvements are possible by constraining the translation to the desired level of politeness. Their main aim is to control the politeness of the translated text via side constraints.

**(Niu et al. (2017))** They propose to use lexical formality models to control the formality level of machine translation output. They also demonstrate the effectiveness of the proposed approach in empirical evaluations, as measured by automatic metrics and human assessments. Their main aim is to control the level of formality of MT output by selecting phrases of a requisite formality level from the k-best list during decoding.

**(Kajiwara and Komachi (2016))** Here they propose an unsupervised method that automatically builds the monolingual parallel corpus for text simplification using sentence similarity based on word embeddings. For any sentence pair comprising a complex sentence and its simple counterpart, we employ a many-to-one method of aligning each word in the complex sentence with the most similar word in the simple sentence and compute sentence similarity by averaging these word similarities. Their experimental results demonstrate the excellent performance of the proposed method in a monolingual parallel corpus construction task for English text simplification. The results show superior accuracy in text simplification that use the framework of statistical machine translation trained using the corpus built by the proposed method to that using the existing corpora.

## 2.3 - Identifying and evaluating formality

**(Brooke and Hirst, 2014)**They apply information from large mixed genre corpora, demonstrating that significant improvement is possible over simple word-length metrics, particularly when multiple sources of information, i.e. word length, word counts, and word association, are integrated. Their main aim is to identify and propose a new method to get sentence level formality scores

**(Pavlick and Nenkova, 2015)** Here the authors present an intuitive and effective method for inducing style scores on words and phrases. They mainly exploit signal in a phrase's rate of occurrence across stylistically contrasting corpora, making our method simple to implement and efficient to scale. They show strong results both intrinsically, by correlation with human judgements, and extrinsically, in applications to genre analysis and paraphrasing. Again this work is also based on evaluating sentence level formality

# Chapter 3

# System Analysis and Design

## 3. System Analysis and Design

In our project, we follow the below process flow.

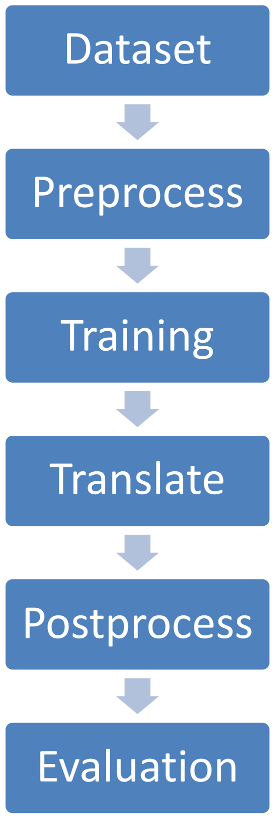


**Fig 3.1 Complete Process flow**

Each module comprises different subprocess:

- Preprocess has rule based heuristics and applying BPE.
- Training follows the NMT model which has encoders and decoders.
- Translation process uses beam search and generates a sentence.
- This sentence is postprocessed by decoding the BPE.
- Finally, the evaluation metrics like formality, fluency and meaning are calculated.
- The Evaluation process is a very complex process flow where each metric is calculated independently.
- Each metric has its own model for predicting the scores.

# Chapter 4

# Modeling and Implementation

# 4. Modeling and Implementation

## 4.1 Dataset description

Our primary dataset for this project is GYAFC. This was built by Grammarly primarily using yahoo's question answer dataset. Yahoo's question answer dataset is a raw text corpus containing questions and their answers curated from Yahoo's very own blogs from various domains examples - entertainment and music, sports, politics, family relationships etc. Grammarly chose 2 domains namely entertainment and music and family relationships. Since the corpus mainly consisted of text from blogs, they were mostly informal and had to be manually translated to its more formal variant. They randomly sampled around 0.15M sentences from the above-mentioned domains and asked experts in linguistics to translate the sentences for them using the Amazon's Mechanical Turk platform. During the process, Grammarly consistently monitored the sentence quality. Hence the very first parallel corpus for informal to formal sentences were constructed. The statistics of the corpus are given below.

| | | **Informal to Formal** | | **Formal to Informal** | |
|---|---|---|---|---|---|
| | **Train** | **Tune** | **Test** | **Tune** | **Test** |
| **Entertainment and music** | 52,595 | 2,877 | 1,416 | 2,356 | 1,082 |
| **Family Relationships** | 51,967 | 2,788 | 1,332 | 2,247 | 1,019 |

**Table 4.1 GYAFC metadata**

The numbers represented in the above table show indicate pairs of sentences. For example, the number 52595 that there are 52595 sentences in formal and 52595 sentences in informal.

## 4.2 Neural Machine Translation

We would like to describe our project in 3 main phases

1. Preparation
2. Training
3. Translation

Before we explain in detail about the different phases in our project, we would like to give a brief explanation about neural machine translation since that is the main concept our project is built on.

The basic idea behind neural machine translation( NMT) is that if we can convert any sentence in any given language into a universal representation( mostly in the form of a vector) then we can use that universal representation to decode it into any required language. Two main components of an NMT model which helps us achieve this are

1. Encoder
2. Decoder

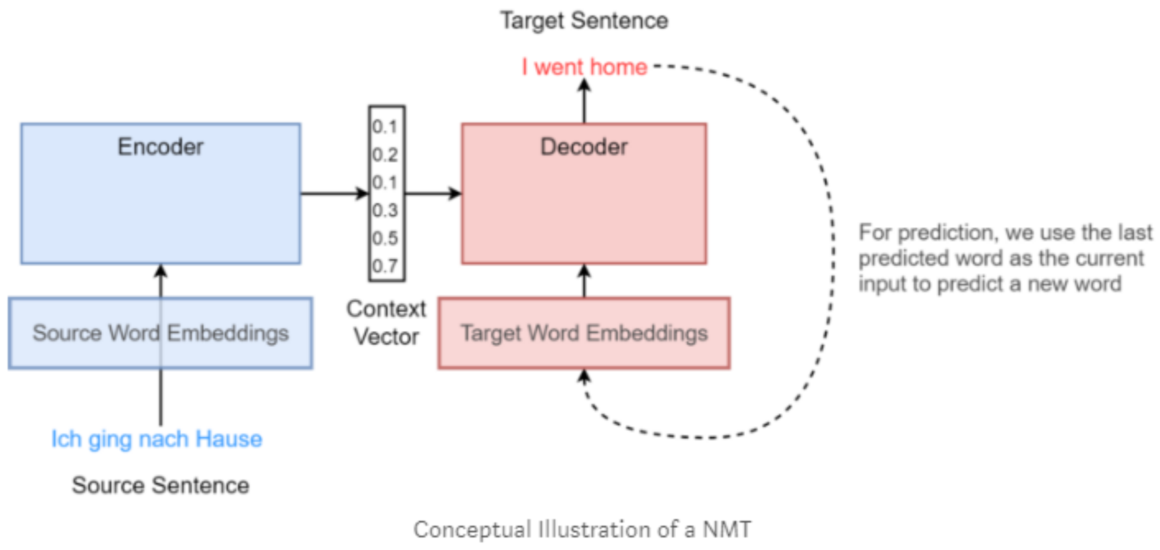


**Fig 4.2.1 Simple NMT architecture**

The image above shows the working of an NMT model. Here first an input sentence in plain raw text form is fed into the encoder component, whose functionality is to convert that text into a universal representation, also known as the context vector of a fixed length. The decoder then uses this context vector as input to convert it to a textual format of the required output language. The obvious question is how does the network convert plain text into numbers. The answer to this is word embeddings, notice in the figure that input text is passed into an embedding layer first and later fed into the encoder. Let's go through this in a bit more detail.

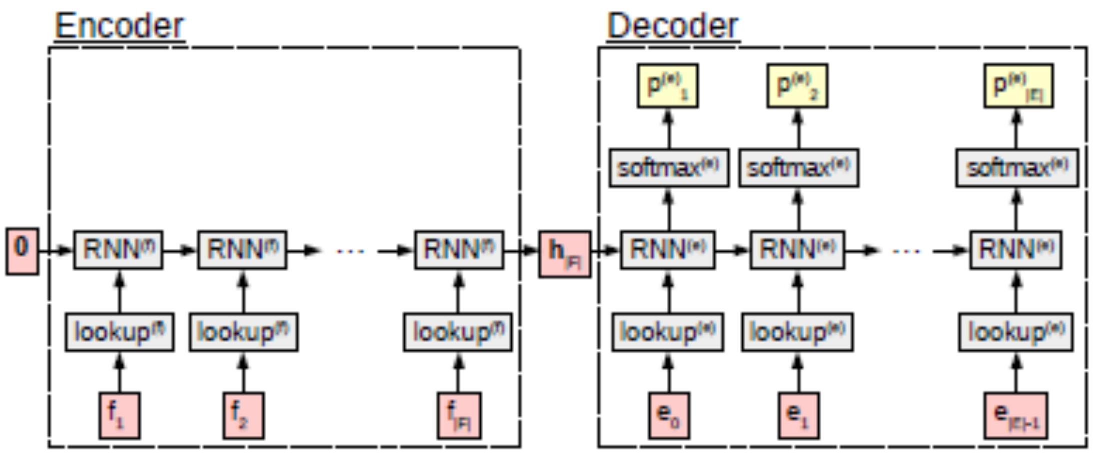


**Fig 4.2.2 Encoder(left) and Decoder(right)**

So what are word embeddings ?. Word embeddings are distributed representations of text in an n-dimensional space, what it means is that each word in vocabulary can be represented by a unique vector of dimension n such that they are semantically related. What semantically related here means is that 2 words which have similar meanings should be represented by vectors whose distance be a least as possible. For example, the distance between word vector King and the distance between word vector Emperor should be less as both almost convey the same meaning.
Now the encoder accepts one word, converted into its respective embedding, at a time from a sentence and its previous output as inputs and produces an output which is later as used as an

input in the next time step. Thus after all the words in a sentence are passed through the encoder the final output vector what is get is known as the context vector( also sometimes referred to sentence embedding). This is then used as an initial input to the decoder. The decoder accepts this context vector and a start token as inputs and iteratively goes on to predict the output words as shown in the diagram above. NOTE, in the decoder at each time step the inputs to the decoder are the context vector and output word from the previous time step. Since recurrent neural networks( RNN) or its variants are the heart of almost any neural machine translation systems. They are usually trained with backpropagation through time. The loss commonly function used is KL divergence or cross entropy. They are given by

$$D(E||F) = H(E,F) - H(E)$$

KL divergence or cross entropy usually measures how different two distributions are. Notice in the diagram above that the output of a decoder is a softmax function which is a probability distribution of vocabulary in a language. Thus at each time step, KL divergence is calculated against the ground truth. Now since we have a basic understanding of a neural machine translation model, we will now look at the 3 phases in our project.

### 4.2.1 Phase 1: Preparation

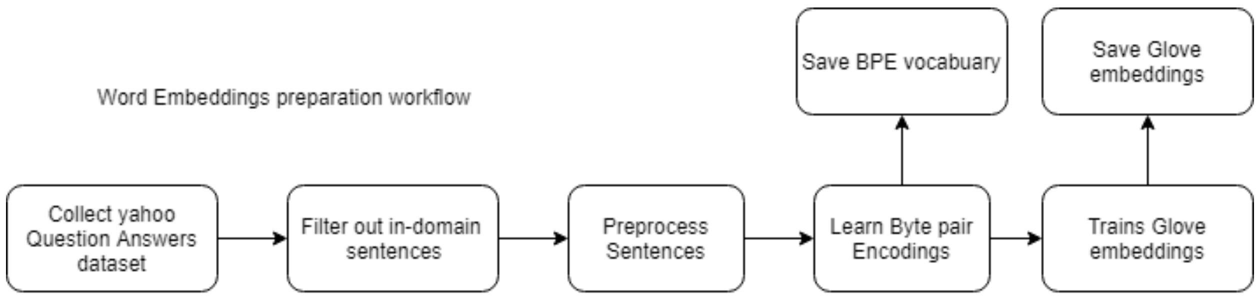

Word Embeddings preparation workflow
Collect yahoo Question Answers dataset
Filter out in-domain sentences
Preprocess Sentences
Learn Byte pair Encodings
Save BPE vocabuary
Trains Glove embeddings
Save Glove embeddings

**Fig 4.2.3 Flowchart of phase-1 preparation**

In phase 1, we concentrate on preparing word embeddings and use transform our raw text into byte pair encoding format which greatly helps to reduce the vocabulary of our corpora, we will explain it step by step.

First, in order to prepare word embeddings, we will need in-domain data. It is empirically proved that to get good embeddings we will always need to re-train it on an in-domain corpus, by in-domain we mean that word embeddings need to be trained on a corpus which is from the same distribution as that of our parallel corpus. As mentioned earlier in our dataset description Grammarly developed the parallel corpus using question answers corpus from yahoo from domains entertainment and music and family relationship. We had to train embeddings from the same distribution. We first requested dataset from Yahoo, which is available for free if used for research purposes. After getting the corpus we filter out data from the two domains mentioned earlier. The number of sentences from each domain is given below

| Domain | No of Sentences |
|---|---|
| Entertainment and Music | 1,20,00,000 |
| Family Relationships | 80,00,000 |

**Table 4.2 Total size of GYAFC dataset**

The sentences obtained were raw and need to be pre-processed before training word embeddings. The steps we took pre-process the sentences are described below;

**Step 1 Capitalization:**
We capitalize the first letter of a sentence, we capitalize the pronoun 'I' and we capitalize proper nouns by identifying words with parts of speech NNP or NNPS.

**Step 2 Lowercase words with all upper cases:**
In several informal sentences, words are often capitalized for emphasis, e.g. "ARE YOU KIDDING ME????" We lowercase such sentences or words.

**Step 3 Expand contractions:**
Informal sentences contain contractions like 'wasn't', 'haven't', etc. We handcraft a list of expansions for all such contractions and maintain it in a file.

**Step 4 Replace slang words:**
Informal sentences contain slang word usage like 'juz', 'wanna', etc. We handcraft a list of slang replacements and maintain it in a text file.

**Step 5 Replace swear words:**
Informal sentences frequently contain swear words. We handcraft a list of swear words and replace all but their first character with asterisks. Example, 'suck' is replaced
with 's***'.

**Step 6 Remove character repetition:**
Informal sentences contain several instances of repeated characters for emphasis. For example, 'nooooo','yayyyyy', '!!!!', '???'. We use regular expressions to replace such repeated occurrences with a single occurrence.
Once our corpus is preprocessed we apply Byte pair encodings to them. Let's look at what it means

**Byte pair encodings( BPE)**
Byte pair encodings are more of a compression algorithm which is used in machine translation to reduce vocabulary in a corpus. The complexity of an NMT model greatly reduces by reducing the number of vocabulary in the train set. Let's understand BPE with examples. In BPE we first consider every alphabet individually as words. For example, consider corpus to have 5 words and their frequencies as given below. Vocabulary will contain every unique alphabet

**Step 1**

| Frequency | words | vocabulary |
|---|---|---|
| 5 | l o w | l, o, w, e, r, n, w, s, t, i, d |
| 2 | l o w e r | |
| 6 | n e w e s t | |
| 3 | w i d e s t | |

From the above, we will look at which sets of consecutive characters are most frequent and thus combine them and add them to our vocabulary. Here **es** is the most frequent and thus we combine them.

**Step 2**

| frequency | words | vocabulary |
|---|---|---|
| 5 | l o w | l, o, w, e, r, n, w, s, t, i, d,es |
| 2 | l o w e r | |
| 6 | n e w **es** t | |
| 3 | w i d **es** t | |

Now we find **est** is the most frequent and add that to our corpus

**Step 3**

| frequency | words | vocabulary |
|---|---|---|
| 5 | l o w | l, o, w, e, r, n, w, s, t, i, d, es, est |
| 2 | l o w e r | |
| 6 | n e w **est** | |
| 3 | w i d **est** | |

Now we find **lo** is the next most frequently occurring sequence of characters. We combine them and add it to our vocabulary

**Step 4**

| frequency | words | vocabulary |
|---|---|---|
| 5 | **lo** w | l, o, w, e, r, n, w, s, t, i, d,es,est,lo |
| 2 | **lo** w e r | |
| 6 | n e w **est** | |
| 3 | w i d **est** | |

Thus we move on until only a required number of vocabulary is reached and stop it at that point. In our case, we chose the vocabulary size to be 35000. Once the vocabulary is fixed we can update the corpus like the example given below. Consider a sentence

“It is lower”

Let's say lower is not resent in our vocabulary, however **lo** and **wer** present. We transform our sentence as.

”It is lo@@ @@wer”

Where @@ is a special marker to later determine where was our words were split. Once all these changes are applied to the corpus, we can now train our word embeddings. We choose Glove method to train our word embeddings. We will now explain how Glove embeddings work.

**Glove**

The approach of global word representation is used to capture the meaning of one word embedding with the structure of the whole observed corpus; word frequency and co-occurrence counts are the main measures on which the majority of unsupervised algorithms are based on. GloVe model trains on global co-occurrence counts of words and makes sufficient use of statistics by minimizing least-squares error and, as a result, producing a word vector space with meaningful substructure. Such an outline sufficiently preserves words similarities with vector distance. To store this information we use co-occurrence matrix X, each entry of which corresponds to the number of times word j occurs in the context of the word i. As a consequence:

$$P_{ij} = P(j|i) = \frac{X_{ij}}{X_i}$$

is the probability that word with index j occurs in the context of the word i.

| | the | cat | sat | on | mat |
|---|---|---|---|---|---|
| the | 2 | 1 | 2 | 1 | 1 |
| cat | 1 | 1 | 1 | 1 | 0 |
| sat | 2 | 1 | 1 | 1 | 0 |
| on | 1 | 1 | 1 | 1 | 1 |
| mat | 1 | 0 | 0 | 1 | 1 |

**Table 4.3 Example of a co-occurrence matrix**

Ratios of co-occurrence probabilities are the appropriate starting point to begin word embedding learning. We first define a function F as:

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

which is dependent on 2-word vectors with indexes i and j and separate context vector with index k. F encodes the information, present in the ratio; the most intuitive way to represent this difference in vector form is to subtract one vector from another:

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

Now in the equation, the left-hand side is the vector, while the right-hand side is the scalar. To avoid this we can calculate the product of 2 terms (product operation still allows us to capture the information we need):

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

As long as in word-word co-occurrence matrix the distinction between context words and standard words is arbitrary, we can replace the probabilities ratio with, and solve the equation:

$$F((w_i - w_j)^T \widetilde{w}_k) = \frac{F(w_i^T \widetilde{w}_k)}{F(w_j^T \widetilde{w}_k)}$$

and solve the equation:

$$F(w_i^T \widetilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

If we assume that F function is exp(), then the solution becomes:

$$w_i^T \widetilde{w}_k = \log P_{ik} = \log X_{ik} - \log X_i$$

This equation does not preserve symmetry, so we absorb 2 of the terms into biases:

$$w_i^T \widetilde{w}_k + b_i + \widetilde{b_k} = \log X_{ik}$$

Now our loss function we're trying to minimize is the linear regression function with some of the modifications:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + b_j - \log X_{ij})^2$$

where f is the weighting function, which is defined manually. We train using the formulas on the preprocessed yahoo corpus of an embedding size 500. This concludes our phase 1. We can now move on to phase 2.

### 4.2.2 Phase 2: Training

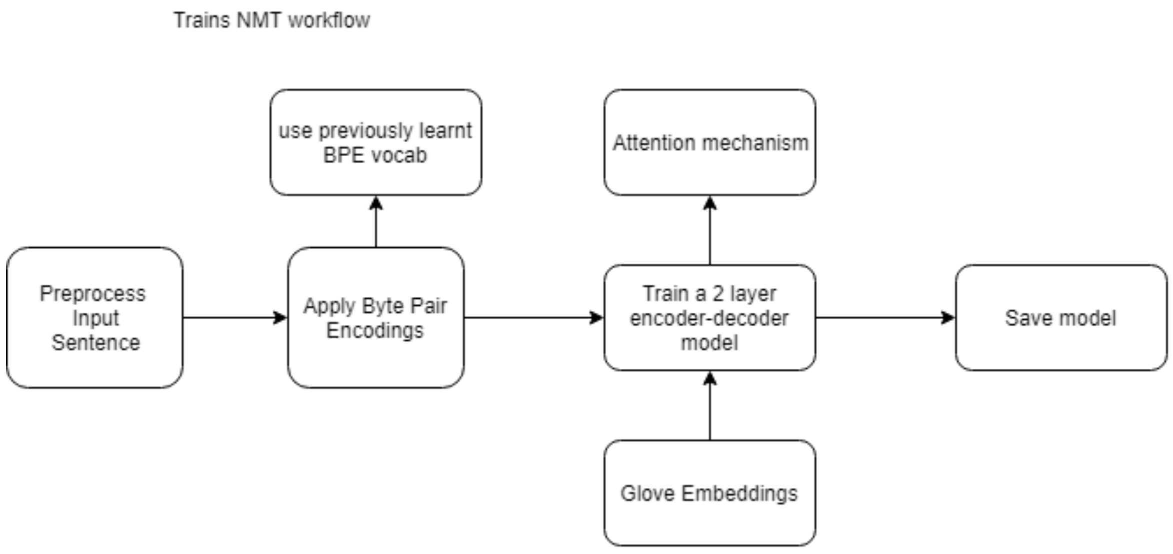

Trains NMT workflow
use previously learnt BPE vocab
Attention mechanism
Preprocess Input Sentence
Apply Byte Pair Encodings
Train a 2 layer encoder-decoder model
Save model
Glove Embeddings

**Fig 4.2.4 Flowchart for phase-2 Training**

Once we have prepared our word embeddings and byte pair encodings, we proceed to train our neural network. Before training, we need to first prepare our parallel corpus and these steps will again include the once we had described earlier. We first preprocess our parallel corpus using the earlier mentioned 6 steps of heuristics and then later apply our byte pair encodings. The number of sentences in our parallel corpus was mentioned earlier in our dataset sectionOur neural network is a 2 layer encoder-decoder bi-lstm model with attention mechanism which can be roughly visualized as given below.

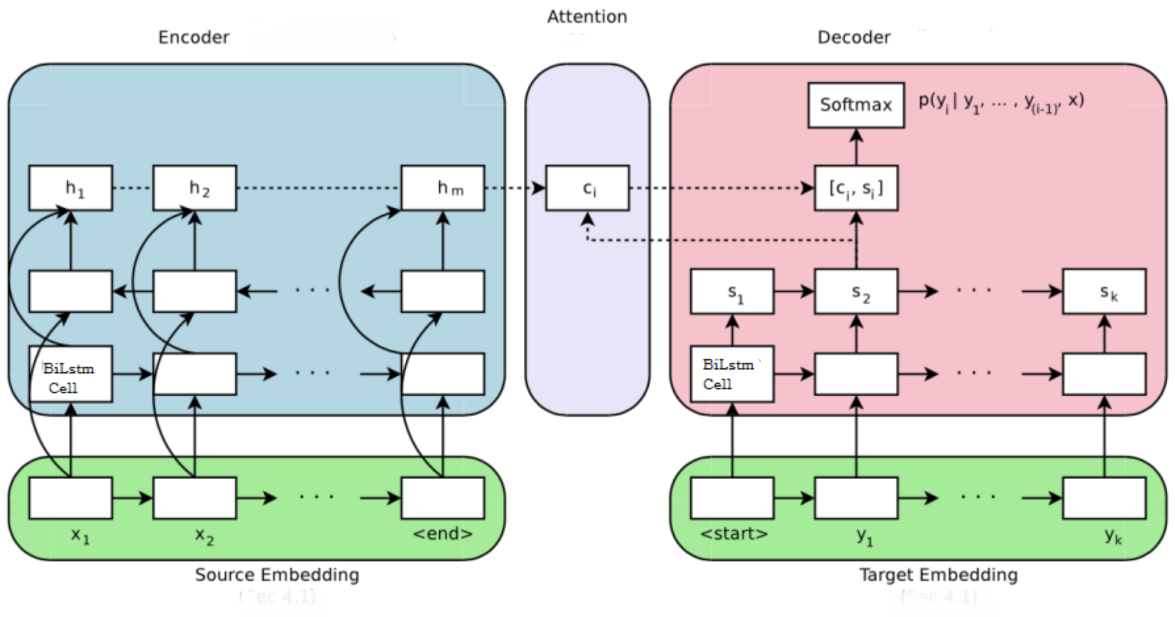


**Fig 4.2.5 Use of Attention mechanism in NMT**

Before we look at how we train our network we will first understand what is attention and why is it required.

**The problem with regular NMT models**

Theoretically, a sufficiently large and well-trained encoder-decoder model should be able to perform machine translation perfectly. Neural networks are universal function approximators, meaning that they can express any function that we wish to model, including a function that accurately predicts our predictive probability for the next word P(e | F). However, in practice, it is necessary to learn these functions from limited data, and when we do so, it is important to have a proper inductive bias - an appropriate model structure that allows the network to learn to model accurately with a reasonable amount of data. There are two things that are worrying about standard encoder-decoder architecture.

**First,** there are long-distance dependencies between the words that need to be translated into each other. This can be alleviated to some extent by reversing the direction of the encoder to bootstrap training, but still, a large number of long-distance dependencies remain, and it is hard to guarantee that we will learn to handle these properly.

**Second**, and perhaps more, the worrying aspect of the encoder-decoder is that it attempts to store information sentences of any arbitrary length in a hidden vector of a fixed size. In other words, even if our machine translation system is expected to translate sentences of lengths from 1 word to 100 words, it will still use the same intermediate representation to store all of the information about the input sentence. If our network is too small, it will not be able to encode all of the information in the longer sentences that we will be expected to translate. On the other hand, even if we make the network large enough to handle the largest sentences in our inputs when processing shorter sentences, this may be overkill, using needlessly large amounts of memory and computation time. In addition, because these networks will have large numbers of parameters, it will be more difficult to learn them in the face of limited data without encountering problems such as overfitting. The remainder of this section discusses a more natural way to solve the translation problem with neural networks: attention.

**Solution: Attention mechanism**

The basic idea of attention is that instead of attempting to learn a single vector representation for each sentence, we instead keep around vectors for every word in the input sentence, and reference these vectors at each decoding step. Because the number of vectors available to reference is equivalent to the number of words in the input sentence, long sentences will have many vectors and short sentences will have few vectors. As a result, we can express input sentences in a much more efficient way, avoiding the problems of inefficient representations for encoder-decoders mentioned. First we create a set of vectors that we will be using as this variably-lengthed representation. To do so, we calculate a vector for every word in the source sentence by running a BiLSTM in both directions:

$$\overrightarrow{\boldsymbol{h}}_j^{(f)} = \mathrm{RNN}(\mathrm{embed}(f_j), \overrightarrow{\boldsymbol{h}}_{j-1}^{(f)})$$
$$\overleftarrow{\boldsymbol{h}}_j^{(f)} = \mathrm{RNN}(\mathrm{embed}(f_j), \overleftarrow{\boldsymbol{h}}_{j+1}^{(f)}).$$

Then we concatenate our two vectors into a bi-directional representation

$$\boldsymbol{h}_j^{(f)} = [\overleftarrow{\boldsymbol{h}}_j^{(f)}; \overrightarrow{\boldsymbol{h}}_j^{(f)}].$$

We can further concatenate these vectors into a matrix:

$$H^{(f)} = \mathrm{concat\_col}(\boldsymbol{h}_1^{(f)}, \ldots, \boldsymbol{h}_{|F|}^{(f)}).$$

This will give us a matrix where every column corresponds to one word in the input sentence. However, we are now faced with a difficulty. We have a matrix H(f) with a variable number of columns depending on the length of the source sentence but would like to use this to compute, for example, the probabilities over the output vocabulary, which we only know how to do (directly) for the case where we have a vector of input. The key insight of attention is that we calculate a vector αt that can be used to combine together the columns of H into

$$c_t = H^{(f)}\alpha_t.$$

a vector ct.

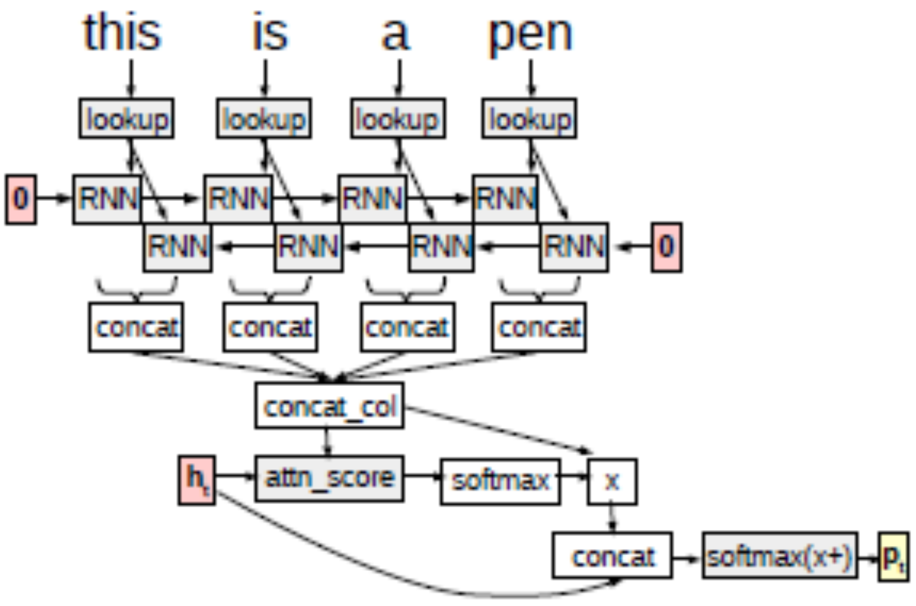


**Fig 4.2.6 A computation graph for attention.**

Now that we have a good understanding of attention, we can explain the architecture and the hyperparameters in our model. As mentioned earlier we use a 2 layer BiLSTM encoder-decoder model, which means that we have 2 layers of BiLSTM on the encoder and 2 layers of BiLSTM on the decoder. We use the glove embeddings on both encoder and decoder as style transfer is basically monolingual, that is both our input and our output language is the same, thus removing the need of different embeddings on encoder and decoder. Considering all this into account our model reported around 22457897 parameters to train. We train our model for 100000 epochs, where the source is an informal sentence and target is a formal sentence taken from GYAFC parallel corpus. Our model also uses the above-explained attention mechanism while training. Thus to summarize.

| | |
|---|---|
| Epochs | 100000 |
| Word embedding size: | 500 |
| Batch size: | 1024 |
| No of parameters to train: | 22457897 |
| Attention: | enabled |
| No of Hidden layers in encoder: | 2 |
| No of Hidden layers in decoder: | 2 |
| Context vector size: | 2048 |
| Attention vector size: | same as that of the length of the input sentence |

**Table 4.4 Hyperparameters of our model**

Once the training is done we save the model as a .pt file. With this, we finish our train phase.

### 4.2.3 Phase 3: Translation

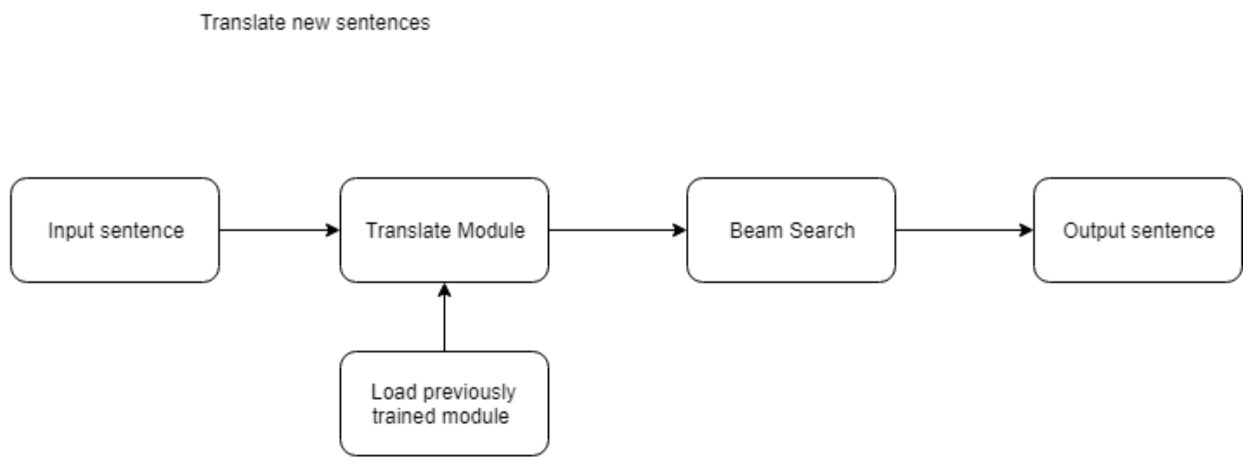


**Fig 4.2.7 Flowchart of phase-3 Translation**

The flow chart above pretty much explains our workflow for translation of new and unseen sentences. We first load our previously trained model and preprocess our sentences as per the previously explained requirements and use it for translation. However, during translating we use one additional step for getting better quality sentences and that technique is known as Beam Search. The idea behind using Beam search is that rather than predicting the next word while knowing the previous word with the highest probability we maintain a Beam of a chosen value which can consider top n words instead of just the best word. The intuition is that, instead of using a greedy approach and choosing the best word at every iteration of the translation step, we decide on having top n words with an aim that the entire sentence probability will increase. Let us look at it step by step.

Let us say the decoder has just started its first iteration and let’s assume that our beam size is 2

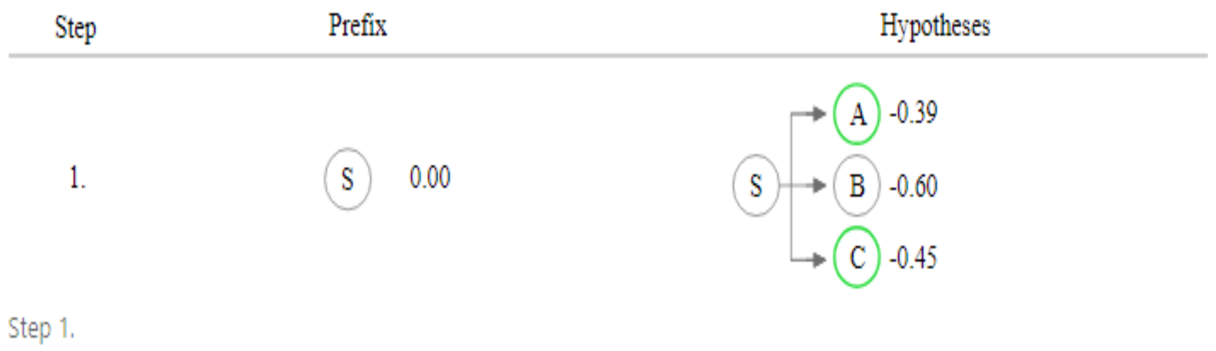


**Fig 4.2.8 Beam search in decoder step 1**

Looking at the diagram above let’s say that our decoder has 3 choices with various probabilities. From the above diagram it’s evident that according to our beam size we will consider A and C as the top 2 probabilities instead of choosing just A. Let’s move to step 2.

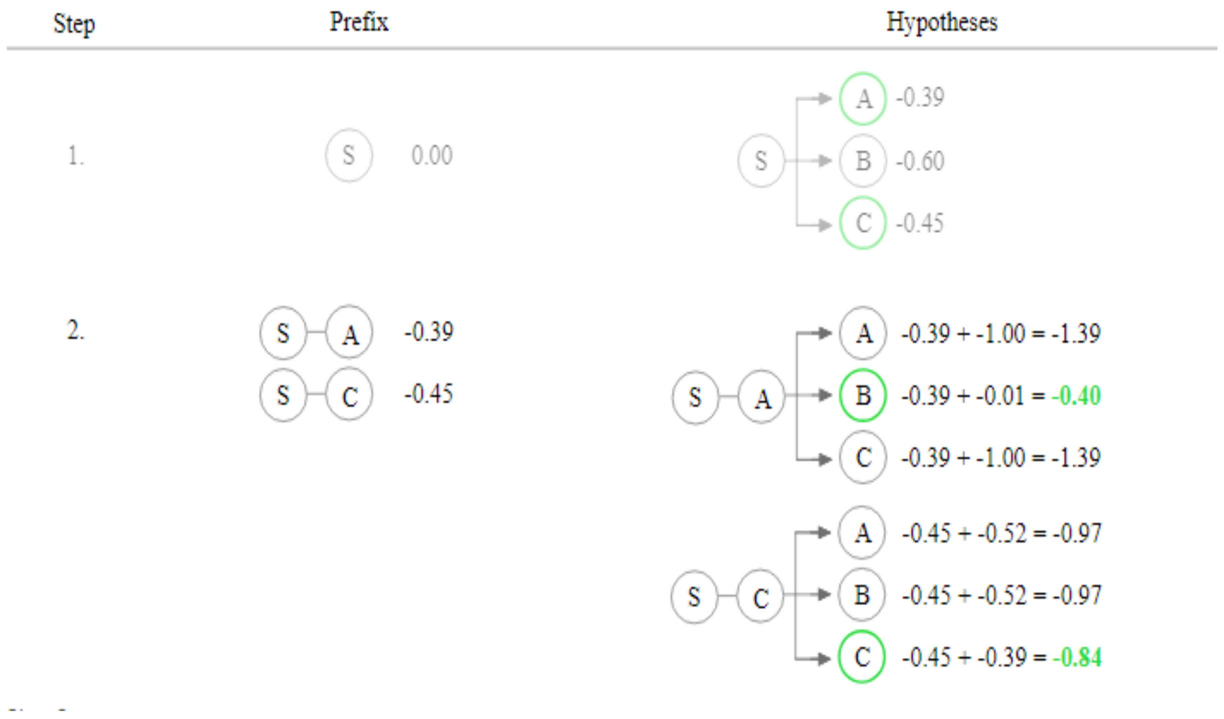


**Fig 4.2.9 Beam search in decoder step 2**

Now looking at step 2 with A and C as input, we have several other combinations of probabilities. Here again we consider characters with the top 2 probabilities as our beam size is 2. From the diagram it is evident that B and C have the highest probabilities. We add that to our Beam and move on to the next step.

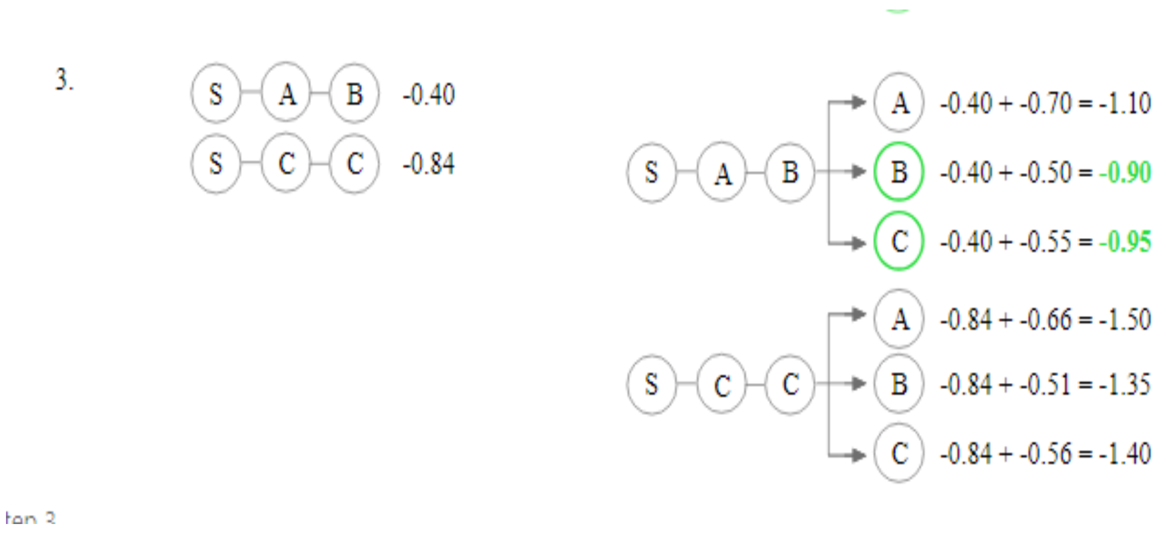


**Fig 4.2.10 Beam search in decoder step 3**

The same process continues in step 3 as well, we consider the best 2 predictions rather than one.

This process continues till the decoder finds an end of sentence <eos> token which marks the end of decoding phase and the sentence we get is taken as the output. In our project we take our Beam size as 5. This marks the end of our translation phase. We should however note that we had given our input in Byte pair encoding format and thus the output file will have @@ markers in between, thus it will need a little postprocessing. This is easily done by using **SED** command. For example

```
Cat <output_file> | sed -r ‘s/\@\@ //g’ > <postprocessed_output filet>
```

# Chapter 5

# Testing, Results and Discussion

## 5. Testing, Results and Discussion

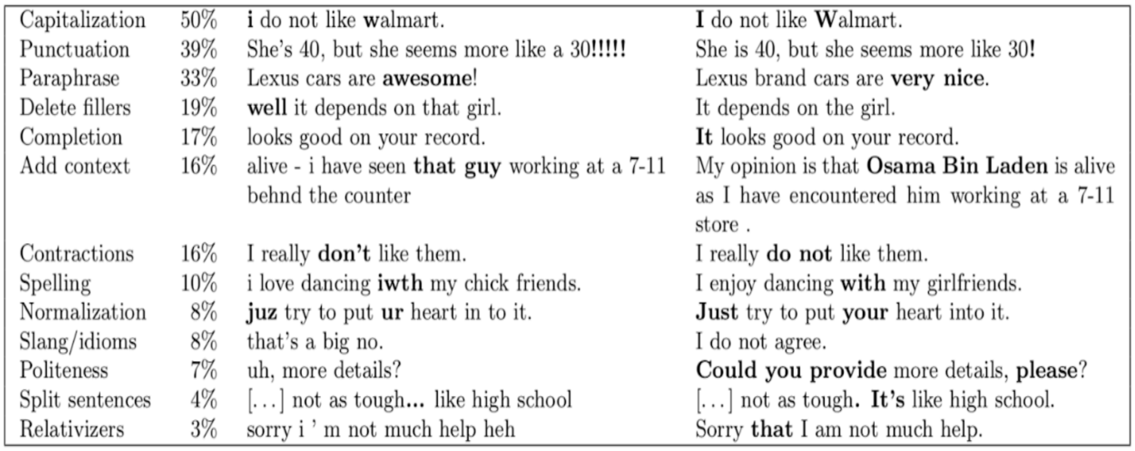
In Machine translation, the input and output are both sentences and to find proper evaluation metrics is still an open research problem. BLEU was originally developed to measure machine translation. It's fast and easy to calculate, especially compared to having human translators rate model output. It's ubiquitous. This makes it easy to compare your model to benchmarks on the same task.

BLEU does not measure meaning. It only rewards systems for n-grams that have exact matches in the reference system. That means that a difference in a function word (like "an" or "on") is penalized as heavily as a difference in a more important content word. It also means that a translation that had a perfectly valid synonym that just didn't happen to show up in the reference translation will be penalized.

BLEU does not consider sentence structure. If the words in reference sentence are partially or completely present in the translated sentence despite the order the scores are always the same for both ordered and unordered sentences.

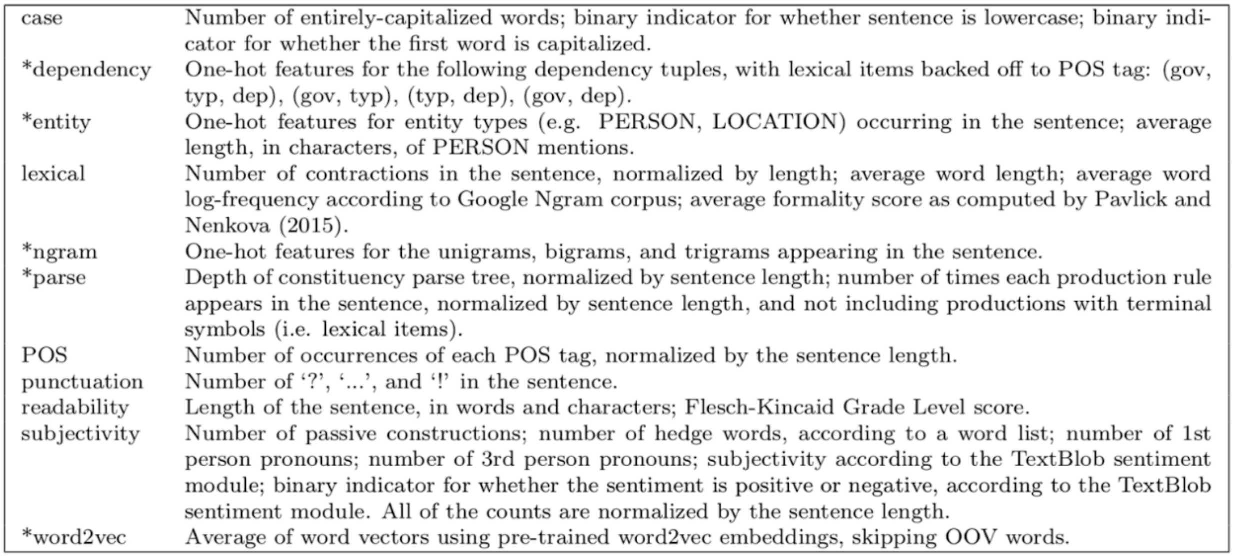
BLEU does not handle morphologically-rich languages well. In English cat is singular and cats are plural here 's' is the morpheme that provides additional information about that word's morphology. BLEU matches the sentences in word level and skips morphologically enhanced words and scores it low.

In our project, we use different evaluating metrics that overcome the demerits of BLEU like formality, fluency and meaning preservation. Formality scores are predicted using Pavlick and Tetreault Classifier(PT16) that shows a statistical model for predicting formality, which is evaluated under different feature settings and genres.



| | | | |
|---|---|---|---|
| Capitalization | 50% | **i** do not like **w**almart. | **I** do not like **W**almart. |
| Punctuation | 39% | She's 40, but she seems more like a 30**!!!!!** | She is 40, but she seems more like 30**!** |
| Paraphrase | 33% | Lexus cars are **awesome**! | Lexus brand cars are **very nice**. |
| Delete fillers | 19% | **well** it depends on that girl. | It depends on the girl. |
| Completion | 17% | looks good on your record. | **It** looks good on your record. |
| Add context | 16% | alive - i have seen **that guy** working at a 7-11 behnd the counter | My opinion is that **Osama Bin Laden** is alive as I have encountered him working at a 7-11 store . |
| Contractions | 16% | I really **don't** like them. | I really **do not** like them. |
| Spelling | 10% | i love dancing **iwth** my chick friends. | I enjoy dancing **with** my girlfriends. |
| Normalization | 8% | **juz** try to put **ur** heart in to it. | **Just** try to put **your** heart into it. |
| Slang/idioms | 8% | that's a big no. | I do not agree. |
| Politeness | 7% | uh, more details? | **Could you provide** more details, **please**? |
| Split sentences | 4% | [...] not as tough**...** like high school | [...] not as tough**. It's** like high school. |
| Relativizers | 3% | sorry i ' m not much help heh | Sorry **that** I am not much help. |

**Fig 5.1 Frequency of types of edits/changes made**



| | |
|---|---|
| case | Number of entirely-capitalized words; binary indicator for whether sentence is lowercase; binary indicator for whether the first word is capitalized. |
| *dependency | One-hot features for the following dependency tuples, with lexical items backed off to POS tag: (gov, typ, dep), (gov, typ), (typ, dep), (gov, dep). |
| *entity | One-hot features for entity types (e.g. PERSON, LOCATION) occurring in the sentence; average length, in characters, of PERSON mentions. |
| lexical | Number of contractions in the sentence, normalized by length; average word length; average word log-frequency according to Google Ngram corpus; average formality score as computed by Pavlick and Nenkova (2015). |
| *ngram | One-hot features for the unigrams, bigrams, and trigrams appearing in the sentence. |
| *parse | Depth of constituency parse tree, normalized by sentence length; number of times each production rule appears in the sentence, normalized by sentence length, and not including productions with terminal symbols (i.e. lexical items). |
| POS | Number of occurrences of each POS tag, normalized by the sentence length. |
| punctuation | Number of '?', '...', and '!' in the sentence. |
| readability | Length of the sentence, in words and characters; Flesch-Kincaid Grade Level score. |
| subjectivity | Number of passive constructions; number of hedge words, according to a word list; number of 1st person pronouns; number of 3rd person pronouns; subjectivity according to the TextBlob sentiment module; binary indicator for whether the sentiment is positive or negative, according to the TextBlob sentiment module. All of the counts are normalized by the sentence length. |
| *word2vec | Average of word vectors using pre-trained word2vec embeddings, skipping OOV words. |

**Fig 5.2 Summary of feature groups**

Baselines. We measure the performance of our model using Spearman ρ with human labels. We compare against the following baselines:

- Sentence length: We measure the length in characters, as this performed slightly better than the length in words.

- Flesch-Kincaid grade level: FK grade level is a function of word count and syllable count, designed to measure readability. We expect higher grade levels to correspond to more formal text.

- F-score: Heylighen and Dewaele's formality score (F-score) is a function of POS tag frequency which is designed to measure formality at the document- and genre-level. We expect higher F -score to correspond to more formal text.

- LM perplexity: We report the perplexity according to a 3-gram language model trained on the English Gigaword with a vocabulary of 64K words. We hypothesize that sentences with lower perplexity (i.e. sentences which look more similar to edited news text) will tend to be more formal. We also explored using the ratio of the perplexity according to an "informal" language model over the perplexity according to a "formal" language model as a baseline, but the results of this baseline were not competitive, and so, for brevity, we do not include them here.

- Formality lexicons: We compare against the average word formality score according to the formality lexicon released by Brooke and Hirst. We compute this score in the same way as Sidhaye and Cheung, who used it to measure the formality of tweets.

- Ngram classifier: As our final baseline, we train a ridge regression model which uses only ngrams (unigrams, bigrams, and trigrams) as features.

The Ngram classifier proved to be a very strong baseline model and with the highest correlation to the human judgment and make improvements on all the 11 features described above.

Fluency scores are predicted by evaluating grammatical error correction without reference and then finally correlating that scores with references or human judgment scores. Existing metrics depend on gold-standard corrections and therefore have a notable weakness: systems are penalized for making corrections that do not appear in the references. While grammaticality-based, reference-less metrics have been effective in estimating the quality of machine translation (MT) output. GLEU, I-measure, and M2 are calculated based on comparison to reference corrections.

These Reference-Based Metrics (*RBMs*) credit corrections seen in the references and penalize systems for ignoring errors and making bad changes (changing a span of text in an ungrammatical way or introducing errors to grammatical text). RBMs penalize output that has a valid correction that is not present in the references or that addresses an error not corrected in the references.

We use three metrics to evaluate the grammaticality of output without comparing to the input or a gold-standard sentence (Grammaticality-Based Metrics, or *GBMs*). The first two metrics are scored by counting the errors found by existing grammatical error detection tools : e-rater is a large-scale, robust tool that detects more errors than Language Tool, it is proprietary whereas Language Tool is publicly available and open sourced. The third metric is a linguistic feature-based model (LFM).
The LFM score is a ridge regression over a variety of linguistic features related to grammaticality, including the number of misspellings, language model scores, OOV counts, and PCFG and link grammar features. It has been shown to effectively assess the grammaticality of learner writing. LFM predicts a score for each sentence while ER and LT, like the RBMs, can be calculated with either sentence- or document-level statistics.

To harness the advantage of RBMs (adequacy) and GBMs (fluency), we build combined metrics, interpolating each RBM with each GBM. For a sentence of system output, the interpolated score (SI) of the GBM score (SG) and RBM score (SR) is computed as follows:

$$SI = (1-\lambda)SG + \lambda SR$$

All values of SG and SR are in the interval [0, 1], except for I-measure, which falls between [−1, 1], and the distribution varies for each metric. I-measure and GLEU have stronger correlations with the expert human ranking when using the mean sentence-level score. The system score is the average SI of all generated sentences.

Modeling sentence similarity is complicated by the ambiguity and variability of linguistic expression and to cope with these challenges, a model for comparing sentences that use a multiplicity of perspectives was applied. Each sentence was modeled using a convolutional neural network that extracts features at multiple levels of granularity and uses multiple types of pooling.

The model consists of 2 main components:

1. A sentence model for converting a sentence into a representation for similarity measurement; we use a convolutional neural network architecture with multiple types of convolution and pooling in order to capture different granularities of information in the inputs.

2. A similarity measurement layer using multiple similarity measurements, which compare local regions of the sentence representations from the sentence model.

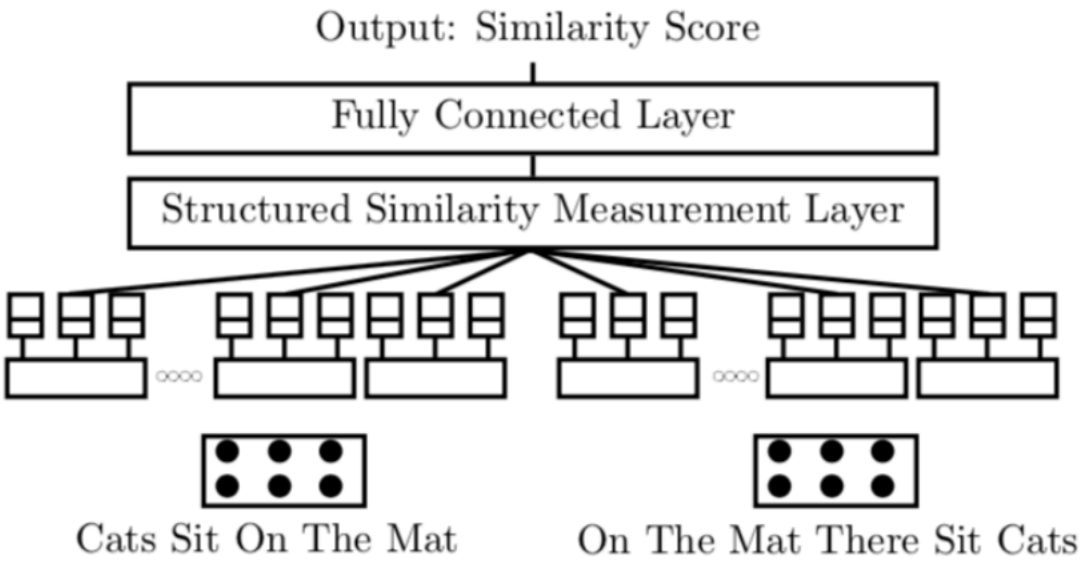

Output: Similarity Score
Fully Connected Layer
Structured Similarity Measurement Layer
Cats Sit On The Mat
On The Mat There Sit Cats

**Fig 5.3 Meaning Score model overview**

Model overview. Two input sentences (on the bottom) are processed in parallel by identical neural networks, outputting sentence representations. The sentence representations are compared by the structured similarity measurement layer. The similarity features are then passed to a fully-connected layer for computing the similarity score (top). Importantly, the model does not require resources like WordNet or syntactic parsers for the language of interest; it only uses optional part-of-speech tags and pretrained word embeddings. The main difference from prior work lies in our use of multiple types of convolution, pooling, and structured similarity measurement over local regions.

For Sentence modeling, the model has two types of convolution filters defined on different perspectives of the input and also use multiple types of pooling. The inputs are streams of tokens, which can be interpreted as a temporal sequence where nearby words are likely to be correlated. Let sent ∈ Rlen×Dim be a sequence of length input words represented by Dim-dimensional word embeddings, where senti ∈ RDim is the embedding of the i-th word in the sequence and senti:j represents the concatenation of embeddings from word i up to and including word j.

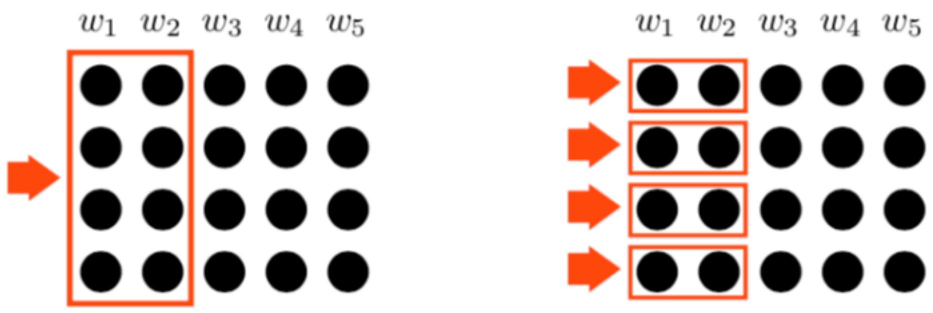


**Fig 5.4 Horizontal and vertical filter comparisons**

Left: a holistic filter matches entire word vectors (here, ws = 2). Right: per-dimension filters match against each dimension of the word embeddings independently.
We define a convolution filter F as a tuple ⟨ws, wF, bF, hF ⟩, where ws is the sliding window width, wF ∈ Rws×Dim is the weight vector for the filter, bF ∈ R is the bias, and hF is the activation function (a nonlinear function such as tanh). When filter F is applied to sequence sent, the inner product is computed between wF and each possible window of word embeddings of length ws in sent, then the bias is added and the activation function is applied. This results in an output vector out F ∈ R1+len −ws where entry i equals

outF [i] = hF (wF · senti:i+ws−1 + bF ) (1)

where i ∈ [1,1 + len − ws]. This filter can be viewed as performing "temporal" convolution, as it matches against regions of the word sequence. Since these filters consider the entirety of each word embedding at each position, we call them holistic filters; see the left half of Figure 2.

In addition, we target information at a finer granularity by constructing per-dimension filters F[k] for each dimension k of the word embeddings, where wF [k] ∈ Rws . See the right half of Figure 2. The per-dimension filters are similar to "spatial convolution" filters except that we limit each to a single, predefined dimension. We include separate per-dimension filters for each dimension of the input word embeddings.

Applying a per-dimension filter F [k] = ⟨ws,wF[k],bF[k],hF[k]⟩ for dimension k re- sults in an output vector outF[k] ∈ R1+len−ws where entry i (for i ∈ [1,1+len−ws]) equals

Out F[k] [i]=h [k](w [k] ·sent[k] +b [k]) F F F i:i+ws−1 F

Our use of word embeddings in both ways allows more information to be extracted for richer sentence modeling. While we typically do not expect individual dimensions of neural word embeddings to be interpretable to humans, there may still be distinct information captured by the different dimensions that our model could exploit. Furthermore, if we update the word embeddings during learning, different dimensions could be encouraged further to capture distinct information.

We define a convolution layer as a set of convolution filters that share the same type (holistic or per-dimension), activation function, and width ws. The type, width, activation function, and a number of filters numFilter in the layer are chosen by the modeler and the weights of each filter ($w_F$ and $b_F$ ) are learned.

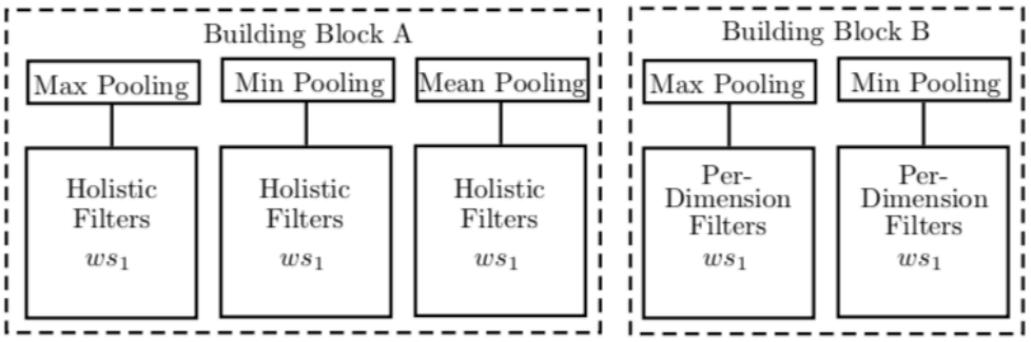


**Fig 5.5 Different Filters and pooling combinations**

Each building block consists of multiple independent pooling layers and convolution layers with width ws1. Left: blockA operates on entire vectors of word embeddings. Right: blockB operates on individual dimensions of word vectors to capture information of finer granularity.

The output vector outF of a convolution filter F is typically converted to a scalar for subsequent use by the model using some method of pooling. We use 3 types of pooling max,min and mean. We define blockA as {groupA(wsa, p, sent) : p ∈ {max, min, mean}}. That is, an instance of blockA has three convolution layers, one corresponding to each of the three pooling functions; all have the same window size wsa .

We use blocks of type A for all holistic convolution layers.

We define blockB as {groupB(wsb,p,sent) : p ∈ {max,min}}. That is, blockB contains two groups of convolution layers of width wsb, one with max-pooling and one with min-pooling. Each groupB (∗) con- tains a convolution layer with Dim per-dimension convolution filters. That is, we use blocks of type B for convolution layers that operate on individual dimensions of word vectors. We use these multiple types of pooling to extract different types of information from each type of filter. The design of each group(∗) allows a pooling function to interact with its own underlying convolution layers independently, so each convolution layer can learn to recognize distinct phenomena of the input for richer sentence modeling.

Similar to traditional n-gram-based models, we use multiple window sizes ws in our building blocks in order to learn features of different lengths.

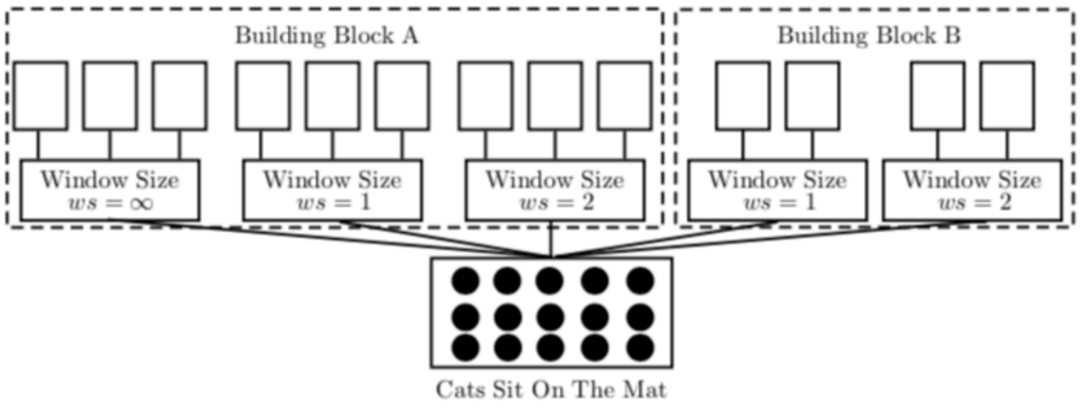


**Fig 5.6 Example neural network architecture for a single sentence**

Example neural network architecture for a single sentence, containing 3 instances of blockA (with 3 types of pooling) and 2 instances of blockB (with 2 types) on varying window sizes ws = 1, 2 and ws = ∞ .

Given two input sentences, the first part of our model computes sentence representations for each of them in parallel. One straightforward way to compare them is to flatten the sentence representations into two vectors, then use standard metrics like cosine similarity. However, this may not be optimal because different regions of the flattened sentence representations are from underlying sources (e.g., groups of different widths, types of pooling, dimensions of word vectors, etc.). Flattening might discard useful compositional information for computing similarity. Therefore, we perform structured comparisons over particular regions of the sentence representations.

One important consideration is how to identify suitable local regions for comparison so that we can best utilize the compositional information in the sentence representations. There are many possible ways to group local comparison regions. In doing so, we consider the following four aspects: whether from the same building block; whether from convolutional layers with the same window size; whether from the same pooling layer; whether from the same filter of the underlying convolution layers. We focus on comparing regions that share at least two of these conditions.

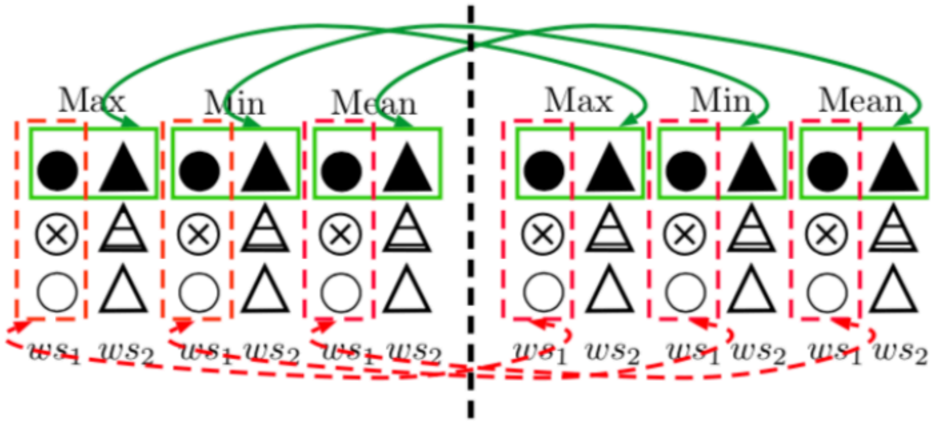


**Fig 5.7 Pooling Computation example**

A simplified example of local region comparisons over two sentence representations that use blockA only. The “horizontal comparison” is shown with green solid lines and “vertical comparison” with red dotted lines. Each sentence representation uses window sizes ws and ws with max/min/mean pooling and numFilterA = 3 filters.
Output Fully-Connected Layer. On top of the similarity measurement layer (which outputs a vector containing all fea∗), we stack two linear layers with an activation layer in between, followed by a log-softmax layer as the final output layer, which outputs the similarity score.
Activation Layers. We used element-wise tanh as the activation function for all convolution filters and for the activation layer placed between the final two layers.

Sentences Involving Compositional (SICK) dataset. This data was collected for the 2014 SemEval competition (Marelli et al., 2014) and consists of 9,927 sentence pairs, with 4,500 for training, 500 as a development set, and the remaining 4,927 in the test set. The sentences are drawn from image and video descriptions. Each sentence pair is annotated with a relatedness score ∈ [1, 5], with higher scores indicating the two sentences are more closely related.

We use regularized KL-divergence loss for the semantic relatedness tasks (SICK and MSRVID), since the goal is to predict the similarity of the two sentences. The training objective is to minimize the KL-divergence loss plus an $L_2$ regularizer:

$$loss(\theta) = \frac{1}{m}\sum_{k=1}^{m} KL\left(f^k \,||\, \widehat{f_\theta^k}\right) + \frac{\lambda}{2}||\theta||_2^2$$

where fθ is the predicted distribution with model weight vector θ, f is the ground truth, m is the number of training examples, and λ is the regularization parameter.
Finally, compare sentence representations at several granularities using multiple similarity metrics like Pearson’s and Spearman's correlation. But for getting meaning preservation scores we trained it for SICK dataset and then obtained the KL divergence loss value for it. If the sentence similarity score predicted matches the ground truth the lesser the loss value.

We used all three metrics at the sentence level.

## 5.1 Results

Sample Sentences were taken from Entertainment and Music, and Family and Relationship genres

| **Original Sentence(Informal)** | **Translated Sentence(Formal)** |
|---|---|
| IM GLAD THEY PASSED THE TOILETS | I am just glad they did not show us the restrooms. |
| Hello Guys wassup!! | Greetings everyone. How are you? |
| It ain't that easy. | It is not that easy |

**Table 5.1 Input and output for sample test set**

| **Formality** | **Fluency** | **Meaning** |
|---|---|---|
| 82.5% | 3.09 | 4.27 |

**Table 5.2 Overall average scores**

# Chapter 6
# Conclusion and Future Work

## 6. Conclusion and Future Work

In this project, we have explored existing techniques in the field of Machine translation and were successful in improving the scores by using additional preprocessing techniques like rule-based heuristics and BPE. The use of training glove on our corpus gave us the extra boost and also the fine tuning of hyperparameters played key role as well. The use of attention in our neural network architecture helped us to overcome limited word representation problems. The use of BPE helped us to split word into subunits and overcome the out of vocabulary word problems as well. The field of style transfer still is an open research area and our project has scope for future works like:

- Enhance the decoder to provide a more fluent output
- Parallel corpus size is extremely low which make it fall under low resource machine translation category and therefore finding clever ways to increase corpus size helps( ex- unsupervised SMT)
- Improve the Evaluation metrics by finding one standard metric for NMT that overcomes BLEU.
- Introduce unsupervised learning approach in style transfer for low resource corpus.
- Identifying abusive sentences in different styles.

# Chapter 7

# Bibliography

## 7. Bibliography

[1]Fadi Abu Sheikha, Diana Inkpen," Generation of Formal and Informal Sentences ", Proceedings of the 13th European Workshop on Natural Language Generation (ENLG), pages 187–193, Nancy, France, September 2011

[2] Sudha Rao, Joel Tetreault , "Dear Sir or Madam, May I Introduce the GYAFC Dataset: Corpus, Benchmarks and Metrics for Formality Style Transfer", Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)

[3]Shibamouli Lahiri,"SQUINKY! A Corpus of Sentence-level Formality, Informativeness, and Implicature" joint proceedings of CoNLL 2015

[4] Fadi Abu Sheikha, Diana Inkpen," Learning to Classify Documents According to Formal and Informal Style " Linguistic Issues in Language Technology LiLT Submitted, March 2012

[5] Luong, T., Pham, H., and Manning, C. D. Effective Approaches to Attention-based Neural Machine Translation. In Proceedings of the 2015 Conference on Empiri- cal Methods in Natural Language Processing, pages 1412–1421, Lisbon, Portugal, September 2015a.

[6] Luong, T., Sutskever, I., Le, Q., Vinyals, O., and Zaremba, W. Addressing the Rare Word Problem in Neural Machine Translation. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, pages 11–19, Beijing, China, July 2015b.

[7] Sennrich, R., Haddow, B., and Birch, A. Neural Machine Translation of Rare Words with Subword Units. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pages 1715–1725, Berlin, Germany, August 2016.

[8] Ellie Pavlick and Joel Tetreault. 2016. An empirical analysis of formality in online communication. *Transactions of the Association for Computational Linguistics* 4:61–74.

[9] Courtney Napoles, Keisuke Sakaguchi, and Joel Tetreault. 2016. There's no comparison: Reference- less evaluation metrics in grammatical error correction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 2109–2115.

[10] Hua He, Kevin Gimpel, and Jimmy J Lin. 2015. Multi-perspective sentence similarity modeling with convolutional neural networks. In *EMNLP*. pages 1576–1586.