Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Master's Thesis

Evidence Extraction for Fact Validation using Neural Network Architectures

Ramesh Kumar

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Paul G.Plöger
Prof. Dr. Jens Lehmann
Dr. Diego Esteves

September 16, 2019

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Ramesh Kumar

New Thesis Title

We would like to change the thesis title from *"Evidence Extraction for Fact Validation using Neural Network Architectures"* to *"Benchmarking Evidence Extraction and Claim Classification Methods for Fact Validation"*. At the time of proposal submission, we were not completely certain about exploring the claim classification approaches also, rather than evidence extraction only. Furthermore, we also employed classical approaches along with the Neural Network architecture. Thus, we propose the new title, as it describes the research conducted during this thesis.

Abstract

With the increase of misleading information on the internet, fact validation is the prominent and difficult task that helps the people to determine the veracity of the facts by verifying through the reliable knowledge sources such as Wikipedia. To accomplish this, generally, three steps are considered: Document retrieval, Evidence retrieval, and Claim classification. Recently largest fact validation datasets known as Fact Extraction and VERification (FEVER) original and FEVER simple claims were introduced to benchmark the state of the art approaches, in which system has to assign the label as well as retrieve evidences related to the claim/fact.

In this work, we explored the classical and deep learning sentence retrieval and claim classification approaches to enhance the performance of DefactoNLP [51] (proposed to solve this task) for FEVER original dataset. Currently, DefactoNLP employs Term Frequency - Inverse Document Frequency (TF-IDF) for evidence retrieval and hand-crafted features along with the textual-entailment model to assign the final label to the claim. The main drawback of TF-IDF is, it relies on string similarity and does not account for synonyms. Furthermore, we also evaluate different claim classification approaches on FEVER simple claims dataset and compare the results to the state of the art.

We evaluated sentence retrieval and claim classical approaches on FEVER original and FEVER simple claims dataset. As a result, we observed that simple LSTM using BERT outperforms state of the art on FEVER simple claim dataset. While, on FEVER original dataset, after integration with DefactoNLP, we obtained comparable performance to the state of the art.

Acknowledgements

This work would have been incomplete without the guidance and help of some great people. I would like to express my gratitude to Prof. Dr. Paul Plöger and Prof. Dr. Jens Lehmann for being my supervisors. Special thanks to Dr. Diego Esteves for his countless support and guidance throughout my Master thesis. I would also like to thank Piyush Chawla for helping during this work. I am very grateful to Livin Natious, Rajjat Dadwal, Chaitali Prabhu, Akhilesh Vyas, Ashutosh Sao, and Mohammad Wasil for reviewing the report.

My sincere thanks to my friends Rubanraj Ravichandran, Pardeep Kumar Naik, Livin Natious, Ashutosh Sao, and Rajjat Dadwal for their continuous encouragement and brotherly environemnt.

Last but not least, I would like to thank my family for their endless love, help, and support.

Contents

List of Figures

List of Tables

1

Introduction

1.1 Motivation

Once upon a time, the information broad-casted through the sources such as newspapers, magazines, radio, television, and books were examined and verified by the publishers and broadcasters that prevented a lot of problems regarding false claims. However, with the arrival of World Wide Web (WWW), information started to flow through various channels, such as blogs, social media, emails, forums, etc. and we are exposed to the information that is not validated.

For instance, there can be information related to a particular event or topic that is produced by multiple sources that may contain conflicting claims. If two authors make contradictory claims, such as "William Shakespeare was born in Stratford-upon-Avon, United Kingdom on April 23rd, 1564" and "William Shakespeare was born in Stratford-upon-Avon, United Kingdom on April 26th, 1564", which one to consider correct? Certainly, an easier solution would be to determine based on majority voting. Like, if "William Shakespeare was born in Stratford-upon-Avon, United Kingdom on April 26th, 1564" is published by majority of authors then we consider it correct. But, it is not necessary that all the authors are trustworthy [41].

Therefore, in this context, there is a requirement of a system that is not biased and determines the authenticity as well as validates the claims based on external knowledge sources rather than the majority voting approach.

1.2 Problem Statement

Many Researchers have been proposing different techniques to automate the process of fact checking. Several datasets corresponding to this task have been released such as Fake News Challenge [19] comprised of 300 claims, Snopes [48] has around 4856 claims, Wikipedia [47] consists of 100 claims, and Vlachos and Riedel [68] contains 106 claims. All the aforementioned datasets are limited to few hundred to thousand claims, which are insufficient to solve this problem with satisfactory performance. However, recently largest fact checking dataset was presented by Thorne et al. [60] known as Fact Extraction and VERification (FEVER) (explained in section 5.1.1) consisting of 185,445 claims, that can be useful for Deep learning and Natural Language Processing (NLP) approaches. More specifically, *FEVER shared task 2018* [59] was initiated to solve task of automating the fact extraction as well as verification using FEVER dataset (in this report, we addressed as FEVER original dataset). For a given input claim, system fetches the related documents, selects the relevant sentences and assigns the corresponding labels (*supporting*, *refuting*, *Not Enough Info (NEI)*). Figure 1.1 shows an example taken from FEVER shared task. In addition to this, FEVER simple claims dataset [46], extracted from FEVER original, is also proposed to mainly focus on claim classification task.

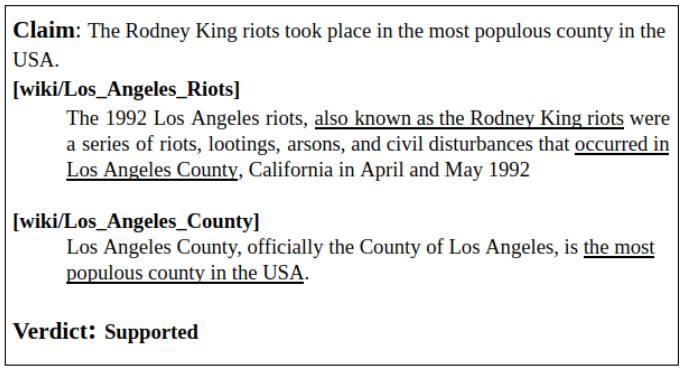


Figure 1.1: Example of FEVER shared task. For a given claim, system fetches the related documents, selects potential related sentences, and determines the final label based on the extracted evidences [60].

Various approaches were introduced during FEVER shared task, such as UNC-

NLP [39], UCL Machine Reading Group [71], Athene UKP TU Darmstadt [24], Papelo [32], Columbia NLP, DeFactoNLP [51], FEVER baseline [60], and so on. Most of the approaches perform fact validation in three-steps: Document retrieval, sentence/evidence retrieval, and textual entailment/claim classification. For a given claim, first k-top documents related to the claim are retrieved from the database, then l-top sentences are extracted from each document and lastly textual entailment is performed to determine whether each sentence is entailing, refuting, or does not have information about the claim.

In this project, we explore the different *sentence retrieval and claim classification* approaches to enhance the performance of DeFactoNLP for FEVER original dataset. In a nutshell, currently DeFactoNLP uses the TF-IDF vectors to retrieve the relevant sentences and employs hand-crafted features to assign the final label to the claim. But TF-IDF vectors have a limitation as they mainly rely on the string similarity and make no use of synonyms. Thus, we assume that the improvement of evidence retrieval and the claim labeling component would significantly boost the overall performance of the DeFactoNLP project. Furthermore, we also evaluate claim classification approaches on *FEVER simple claims* dataset and compare its performance with the state-of-the-art (SOTA) approaches.

1.3 Contribution

- Explore and evaluate different evidence retrieval and claim classification approaches (section 4.1) on *FEVER simple claims* and *original* datasets.
- Integrate the best approach evaluated on *FEVER original dataset* with the DeFactoNLP project (section 5.3.2).
- Compare the performance of DeFactoNLP with other SOTA approaches (section 5.3.2).
- Compare the performance of claim classification approaches on *FEVER simple claims* dataset with the existing approaches (section 5.3.1).

1.4 Thesis Outline

The outline of this thesis is structured as the following:

Chapter 2 introduces the theoretical concepts that are used throughout the work. **Chapter 3** reviews SOTA approaches used for fact validation. **Chapter 4** presents the details about the classical as well as deep learning approaches that are used for evidence retrieval and claim classification. **Chapter 5** describes an evaluation on five different datasets using all the approaches and compares the result. **Chapter 6** discuss the conclusion. Lastly, **Chapter 7** reveals the future directions of this work.

2

Theoretical Background

This chapter discusses the theoretical concepts that are employed during our work. We start with Artificial Neural Network (ANN), followed by Recurrent NN and its variant such as LSTM. Next, concepts of attention mechanism, Transformers, and embeddings are presented. Lastly, we end this chapter with the basic machine learning concepts such as loss functions and optimization algorithms.

2.1 Artificial Neural Network (ANN)

ANN commonly called as neural network (NN), consists of a set of connected neurons with the ability to accumulate the knowledge and making it accessible for later use [25]. Figure 2.1 shows a single neuron which comprises of inputs, set of synapses or weights, summation function that calculates the weighted sum of inputs plus bias, and the activation function. Here, the weights are the trainable parameters, that are adjusted based on the error between the actual output and predicted output. Activation functions are employed to introduce the non-linearity, although linear functions are easier in terms of the computations, but they are not useful for complex tasks such as Speech Recognition, Text Generation, Image and Video Captioning, etc. Furthermore, there are various activation functions that are used to decide the final output based on value of summing function. Some of them are shown in figure 2.2. Although single neuron is useful for solving the simpler problems such as AND, OR and NOT gate, which are linearly separable, when it comes to complicated problems such as XOR gate, Image Recognition, Time Series prediction, Speech Recognition, etc, we need to add more number of succeeding layers in our network. This is also

known as Deep Neural Network (DNN). In the coming section, we will discuss different types of DNN.

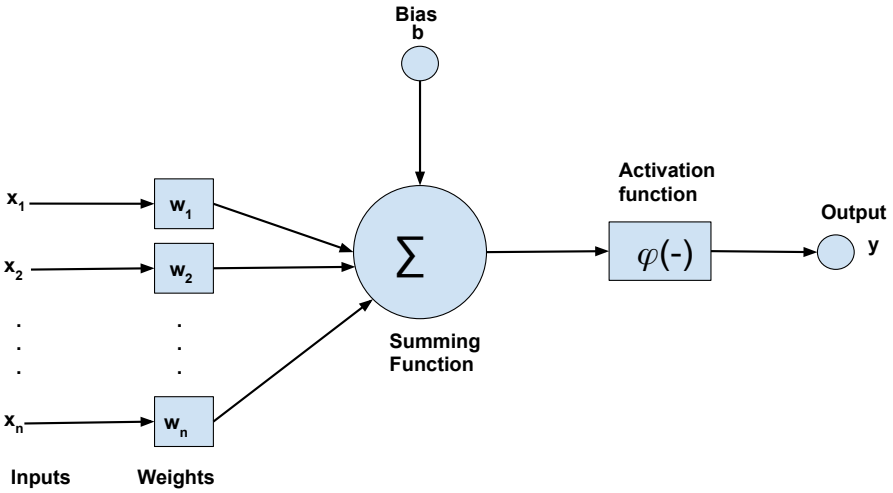


Figure 2.1: Simple artificial neural network model [25].

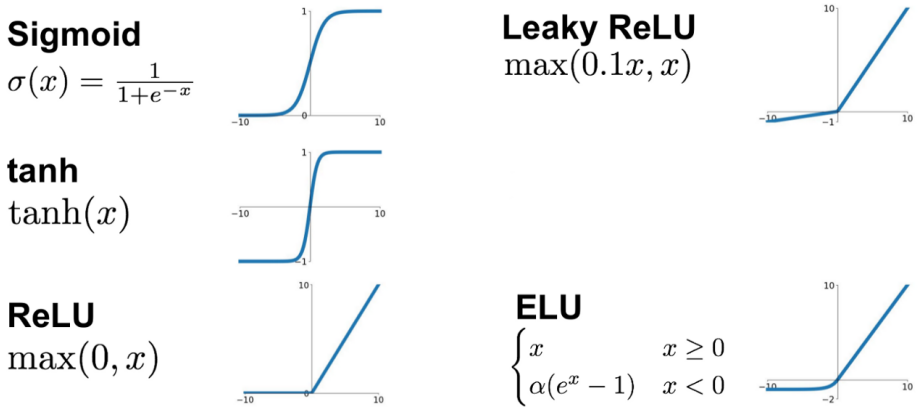


Figure 2.2: List of activation functions [63].

2.2 Recurrent Neural Networks (RNNs)

Unlike Feed-Forward NNs (FFNNs), the output of these networks is based on the information from the current inputs as well as the prior inputs as shown in figure 2.3. This property allows RNNs to be useful for the sequential data where inputs are

related to each other. Some examples of sequential data are Speech Recognition, Text Generation, Time Series, and so on. Furthermore, unlike FFNNs, these networks have the ability to process the sequences of variable lengths as shown in figure 2.4, where, On the extreme left, *one to one* uses FFNNs, while the remaining use RNNs. Boxes in red, green and blue colors are the inputs, hidden states, and outputs respectively. *One to one* considers a fixed size input vector and yields a single output vector of fixed size. E.g: Image classification (input is image and output is the probability of classes). *One to many* depicts RNN that considers a single input image or sequence and produces sequences of words or images. E.g: image captioning (input is image and output is a sequence of words). *Many to one* takes sequence as input and generates a single label. E.g: sentiment classification. *Many to many* is RNN that is used for machine translation problem, where the network outputs the corresponding translated sequence, once the input sequence is completely processed. Lastly, synchronized input and output sequence is the video classification problem, where RNN outputs the respective label for each input frame.

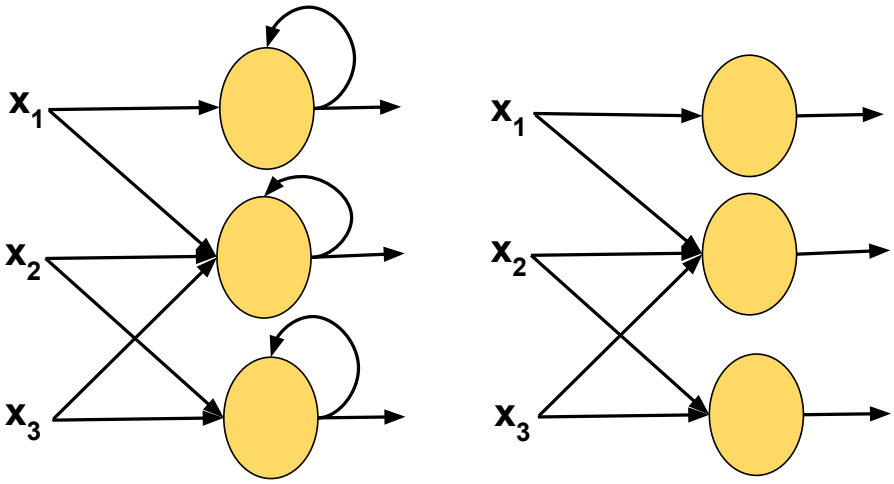


Figure 2.3: RNN vs FFNN [1].

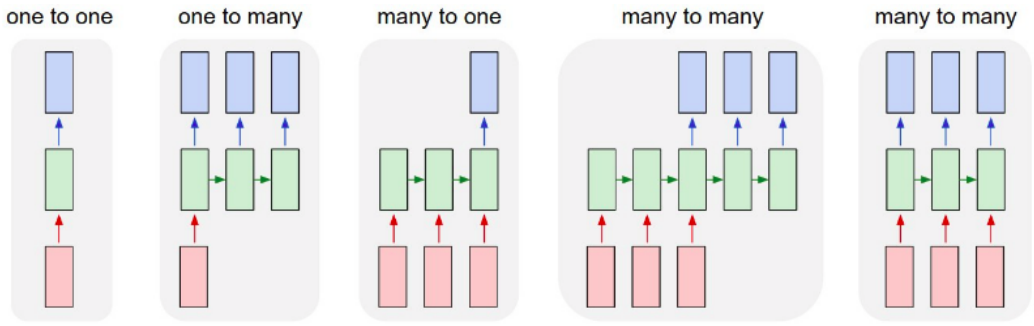


Figure 2.4: FFNNs vs RNNs and its variants [7].

Figure 2.5 shows the rolled and unrolled version of RNN, we can see how information is passed from one time step to the next and the total number of trainable parameters involved from input to output. Following equations show the computations of the hidden and output state:

$$h_t = g(W_{hh}h_{t-1} + W_{ax}x_t + b_t) \tag{2.1}$$
$$y_t = f(W_{yh}h_t + b_t)$$

where: f and g are the activation functions.

All the weights are the trainable parameters that are learned using forward and backward propagation. During forward propagation, weights are initialized randomly in the first iteration, and error between actual and predicted output is calculated. To perform backward propagation, the partial derivative of the error with respect to weights is computed and weights are adjusted in the direction where overall error decreases. This process continues until the algorithm reaches the global minima. Since during forward propagation, the algorithm moves in the increasing time direction, and in backward propagation, it goes in the decreasing time direction, the process is named as Backpropagation Through Time (BTT).

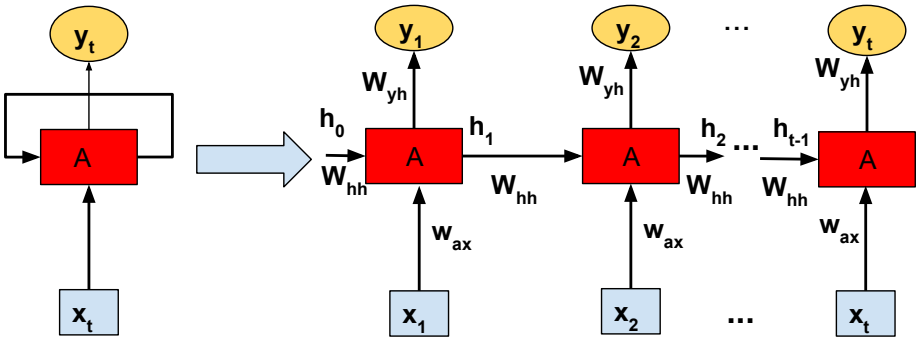


Figure 2.5: The images on the left and right of the main arrow represent a rolled and an unrolled RNN respectively. Here, x is the input, y is the output at any time step, and h represents the hidden state output being fed to the next state. Weights from input to the hidden states are represented as w_{ax}, weights from the hidden to hidden states are shown as w_{hh}, and weights from hidden state to the output are W_{yh} [58].

Although RNNs can handle sequential data, they cannot capture long term dependencies due to the vanishing gradient problem. Assume we have two examples E1: *The boy, who ate apples, mangoes,..., was absolutely full* and E2: *The boys, who ate apples, mangoes,..., were absolutely full.* In these examples if network is intended to predict *were* or *was*, it needs to remember whether noun i.e. *boys* or *boy* is plural or singular. However, it turns out that during backpropagation, the final predicted output cannot affect the weights present in the earlier layers of the network due to smaller values of gradients. As a result, the algorithm fails to converge to the optimal solution.

Furthermore, another problem that occurs in RNN is exploding gradient problem. During backpropagation, there is also a possibility that gradients can get larger values, which can explode or crash the model. This problem is not severe as it can be easily tackled by clipping the gradients to some thresholds. To address the limitations of simple/Vanilla RNNs, Long Short Term Memory (LSTM) networks were introduced, which are explained in the next section.

2.2.1 Long Short Term Memory (LSTM) Networks

LSTM-Ns are a special type of recurrent networks, that have the advantage of solving the long term dependencies efficiently. The main feature of LSTM-Ns is to memorize the important content present in the data and ignore the irrelevant information. This is done with the help of gates and cell state that can control the information between the LSTM units. Figures 2.6 and 2.7 illustrate the structure and difference between basic LSTM and vanilla RNNs. Unlike RNNs, LSTM consists of three gates i.e. forget, input and output.

The primary function of forget gate is to decide which information needs to be neglected. This is determined based on the output of sigmoid function, i.e. if the output is 1, it keeps the information and if it is 0, then data is thrown away. Output of forget gate is computed as:

$$f_t = \sigma(W_f \cdot [x_t, h_{t-1}] + b_f) \tag{2.2}$$

where: σ represents sigmoid function, x_t is input sequence, h_{t-1} is previous hidden state, W_f is a weight matrix, and b_f is a bias of the forget gate.

The main focus of the input gate is to select the relevant information from the sequence. Input sequence (x_t) and previous state output h_{t-1} are concatenated and fed to the sigmoid function, which is further multiplied with the output of tanh function ($\tilde{C}_t$) to assign preference to the important information as shown in equation 2.7.

$$i_t = \sigma(W_x^{(i)} * x_t + W_h^{(i)} * h_{t-1} + b_i) \tag{2.3}$$

$$\tilde{C}_t = \tanh\left(W_x^{(C)} * x_t + W_h^{(C)} * h_{t-1}\right) + b_C \tag{2.4}$$

where: tanh represents Hyperbolic tangent function. W_x and W_h are the input and hidden weight matrices respectively, and b_i and b_C are biases of input gate and temporary cell state $\tilde{C}_t$ respectively.

Output gate determines the information that is to be passed to the next hidden state, after forgetting the irrelevant and considering important information. Following

equations are used to compute output o_t and next hidden state h_t:

$$o_t = \sigma(W_x^{(o)} * x_t + W_h^{(o)} * h_{t-1} + b_o) \tag{2.5}$$

$$h_t = o_t * \tanh C_t \tag{2.6}$$

where: C_t is updated cell state, and b_o is bias of output gate.

Lastly, Cell state is introduced to carry the information between the gates present in each LSTM unit as well as to other LSTM units. Information is added and removed in the cell state using forget and input gate vectors respectively. It is computed as:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{2.7}$$

where: C_{t-1} is the previous cell state, $\tilde{C}_t$ is the temporary cell state, i_t is input gate equation (2.3), and f_t is forget gate equation (2.2)

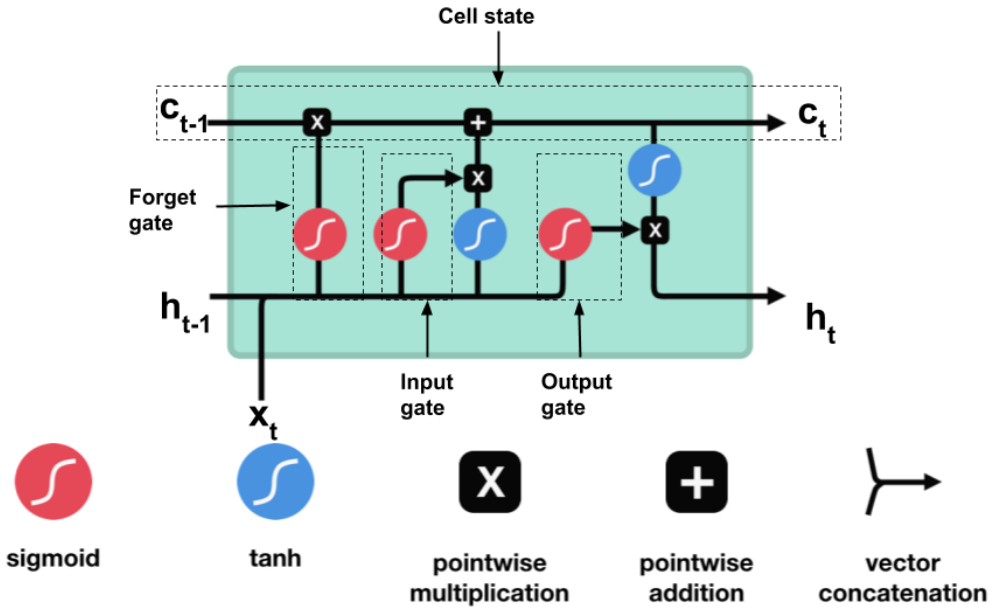


Figure 2.6: Set of operations involved in basic LSTM cell [35].

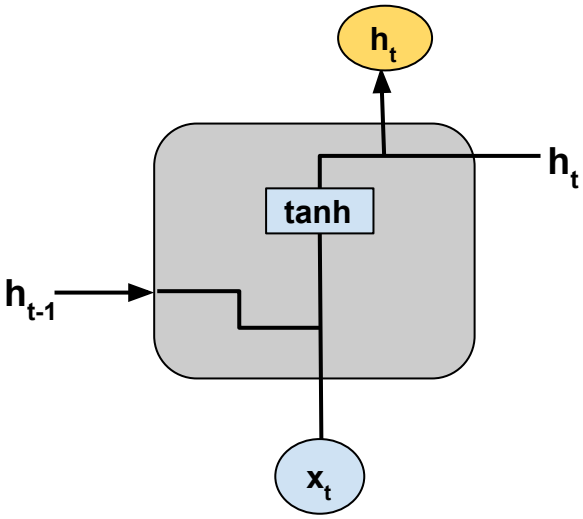


Figure 2.7: Vanilla RNN Cell [29].

All these parameters are trained using Backpropagation Through Time (BTT) as in vanilla RNNs [35].

2.3 Attention

Attention is a way of determining the correlation between words present in a sentence. As a human, we can easily deduce the relation between words present in a single sentence. For example, when we see the word "drinking", we immediately anticipate a word representing a "drink" word and pay higher attention to it (in this case to "coffee") as shown in figure 2.8.

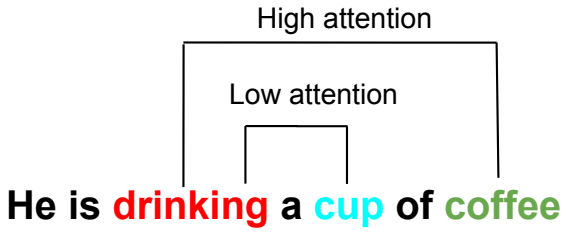


Figure 2.8: Attention Example.

The concept of attention was introduced to overcome the limitations of the Sequence

to Sequence (Seq2Seq) models. As shown in figure 2.9, Seq2Seq consists of three components: Encoder, Decoder, and an Encoder/Context vector. Encoder contains set of RNN/LSTM units, where each unit accepts a single word/element $(x_1, ...x_n)$ of the input sequence and the hidden state from the previous unit. In the end of the sequence, the final hidden state contains the summarized information of the input sequence. This summarized vector of information is called as context vector. This vector acts as the initial hidden state of the Decoder module. Like Encoder, Decoder also contains a stack of RNNs/LSTMs units, where each unit predicts output y_t at time step t based on the previous hidden state. However, the limitation of this approach is that the context vector does not memorize the long sentences once processing of input is completed, as only the last hidden state of the Encoder is fed to the Decoder. As a result, it degrades the performance in tasks such as Machine Translation, Video Captioning, and so on.

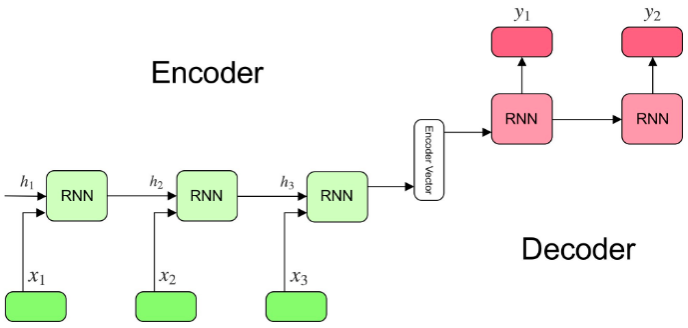


Figure 2.9: Seq2Seq general architecture [31].

Figure 2.10 shows the Encoder-Decoder model with an attention mechanism. Instead of considering the single context vector from Encoder's last state, Attention creates the direct connection between the Abstract/Context vector and the complete input sequence. As a result, forgetting problem is eliminated. In a nutshell, context vector takes following information:

- Encoder and Decoder hidden states.
- Alignment between input (source) and output (target) sequence.

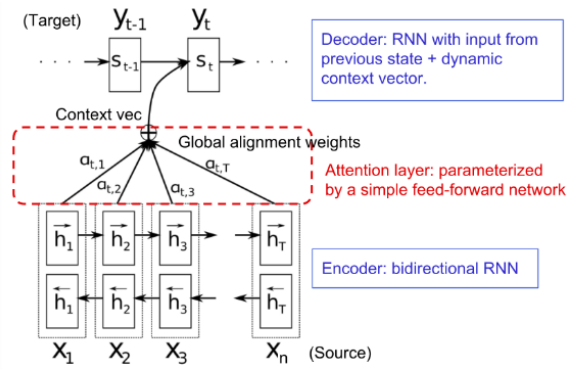


Figure 2.10: Attention mechanism [31].

To understand the attention mechanism in details, consider that we have an input sequence x of length n, and target output sequence y of length m given as:

$$x = [x_1, x_2, ..., x_n]$$

$$y = [y_1, y_2, ..., y_m]$$

On the Encoder side, bidirectional RNN is employed that uses forward and backward hidden state ($\overrightarrow{h_i}$ and $\overleftarrow{h_i}$). Concatenation of both the states yields the representation of single word that is shown as :

$$h_i = [\overrightarrow{h_i}, \overleftarrow{h_i}]$$

where i = 1,...,T

On the Decoder side, hidden state (s_t) for target word at position t is computed as:

$$s_t = f(s_{t-1}, y_t, c_t)$$

where t = 1,...,m

Context vector c_t is computed as the summation of hidden states of the input sentence and the alignment scores ($\alpha_{t,i}$):

$$c_t = \sum_{i=1}^{n} \alpha_{t,i} h_i$$

Alignment score ($\alpha_{t,i}$) is calculated as:

$$\alpha_{t,i} = align(y_t, x_i) = \frac{exp(score(s_{t-1}, h_i))}{\sum_{i'=1}^{n} exp(score(s_{t-1}, h_{i'}))}$$

It is a word alignment model that assigns score ($\alpha_{t,i}$) based on the matching of input at position i and targeted output word at t. The alignment score ($\alpha_{t,i}$) is later multiplied with each hidden state present in Encoder to determine how much attention, the model has to pay to predict the target word.

Furthermore, Bahdanau et al. [9] jointly trains the model parameters and alignment score (α) which is parameterized by a FFNN. The score function is given as:

$$score(s_t, h_i) = v_a^T tanh(W_a[s_t; h_i])$$

where v_a and W_a are learning parameters for word alignment model [31].

Note that we use additive attention mechanism here. There are other types of attention mechanism that are differentiated based on the alignment score function as shown in figure 2.11. We provide brief details of self-attention and multi-head attention in the following section, as it is an important concept for the Transformer network.

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

Figure 2.11: List of different types of attention mechanisms [31].

2.3.1 Self-Attention

It is also known as intra-attention. The idea of this mechanism is to compute the representation of the words by relating it with the other words present in the same sequence as shown in figure 2.12. This helps in determining the correlation between the current word and the prior words present in the sequence. Instead of different source and target sequence, here we consider the same input sentence at the Encoder as well as Decoder side. This approach is quite useful in tasks such as Coreference Resolution, Machine Reading Comprehension, Image Caption Generation, and so on [31].

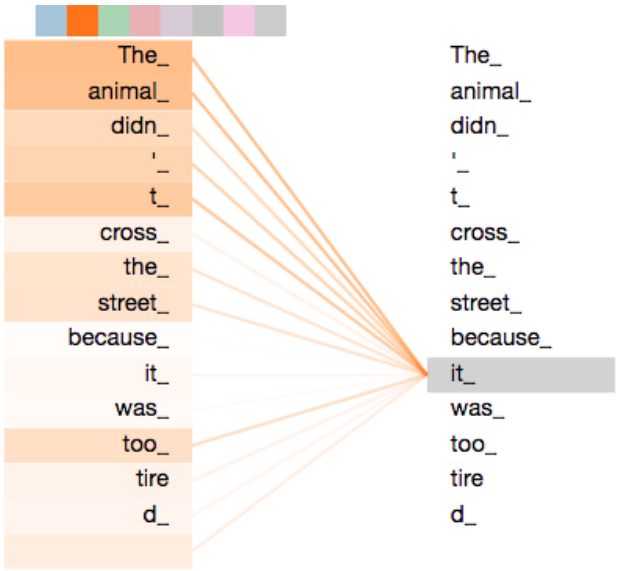


Figure 2.12: For a given sequence "*The animal didn't cross the street because it was too tired*", to encode the word "*it*" efficiently, algorithm tries to obtain the clue from the other words. Like humans, model tries to guess to which word "*it*" is referring to [28].

The calculations of the self-attention mechanism used in the Transformer architecture are shown in figure 2.13. The following section explains the steps:

- Given an input sequence (*Thinking Machines*), first, it is transformed into vectors using word embeddings (such as word2vec). Next, using each word

vector, Query (q), Key (k), and a Value (v) vectors are obtained by multiplying the embeddings with the three trained matrices (named as W^Q, W^K, and W^V respectively). Note: dimension of these trained matrices is 64, and embeddings' dimension is 512 (as shown in [64]).

- After that, we calculate score of each word (lets assume *Thinking*) against all the words (*Thinking Machines*) present in the same sentence. This is done by multiplying the query vector (q1) of word (*Thinking*) with the key vector (k) of all the words as shown in figure 2.13.

- Lastly, to get the final attention score, scaled dot-product attention (as shown in 2.16) is employed:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.8}$$

where d_k is scaling factor.

Although, final vector (z_1) uses the information of the other words (such as *Machines*) to represent the word (*Thinking*), it might be influenced by the vector of itself. This may contribute in performance degradation.

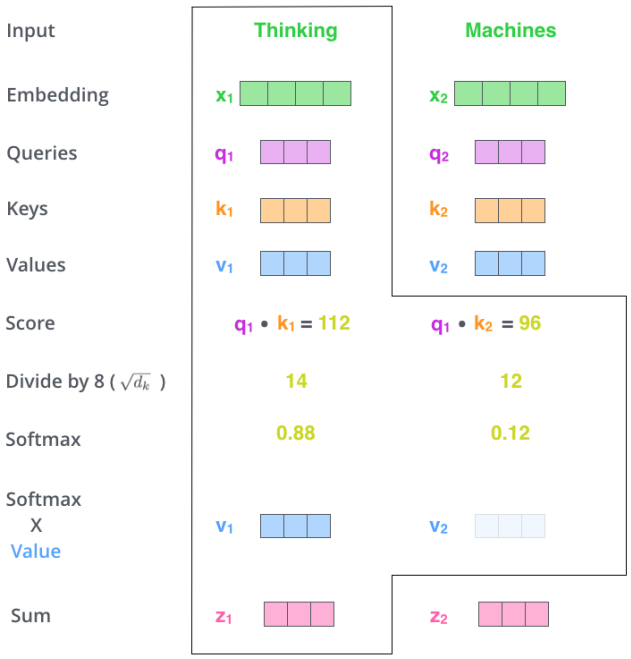


Figure 2.13: Input sequence is *Thinking Machines*. Queries, Keys, and Values are computed by multiplying the input embeddings (x_1 and x_2) with the trained weights (W^Q, W^K, and W^V). A score of each word is computed using query and key vectors of all the words in the sequence. Later, the softmax activation function is used for the scaled dot-product attention to compute the probabilities, that denotes how much each word contributes in the representation of the particular word (here: "Thinking"). In the end, weighted value vectors are summed up to generate the final representation of the word [28].
Note: Value of scaling factor d_k is taken from the original paper [64]. This helps in obtaining stable gradients.

To overcome this problem, authors in [64] introduced the concept of multi-head attention. First, it forces the model to concentrate on different words of the sequence. Second, instead of single attention head (single Query/Key/Value matrices), it uses multiple sets of Q/K/V matrices as shown in figure 2.16. All these matrices are the trainable parameters. In the end, all the different representations based on each set of Q/K/V are concatenated and fed to the FFNN in the transformer network as shown in figure 2.15. Multi-head attention is mathematically represented as:

$$MultiHead(Q, K, V) = Concat(head_1, head_2, ..., head_h)W^O \tag{2.9}$$
$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

2.4 Transformer

Apart from vanilla RNNs and LSTMs, Transformer is another architecture that is well-suited for various tasks including Language Modeling, Machine Translation (translating one language to another), Constituency Parsing, and so on. Although simple LSTM-Ns are also useful to solve these problems, this architecture has significantly better performance on these tasks [64] [34]. Furthermore, it relies only on attention mechanism and does not incorporate recurrent or convolutional NN models.

The architecture of transformer (shown in figure 2.14) mainly consists of two parts: Encoder unit, that contains self-attention mechanism followed by the feed-forward network, and Decoder unit that employs self-attention, Encoder-Decoder attention, and the feed-forward network.

Figure 2.15 shows the detailed architecture of transformer networks. Since these networks do not use the concept of RNNs, therefore to preserve the order of the words, positional encoding is introduced. Initially, input sequences are converted to n-dimensional vectors using word embeddings and added with positional encoding. This is fed to the Encoder, where each unit consists of two sub-layers: multi-head self-attention and feed-forward layer. Multi-head self-attention is composed of various parallel attention layers. In addition to this, each sub-layer has a residual connection along with the layer normalization.

On Decoder side, each unit consists of three sub-layers: two sub-layers are multi-head attention and one is feed-forward. Like Encoder, Decoder also uses a residual connection with normalization for each sub-layer. For both Encoder and Decoder N_x = 6 is considered, this means 6 Encoders and Decoders are stacked on top of each other.

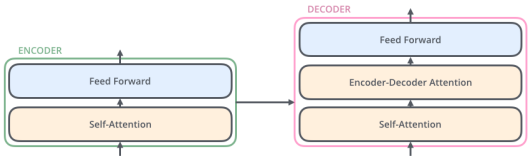


Figure 2.14: Transformer model general architecture [27].

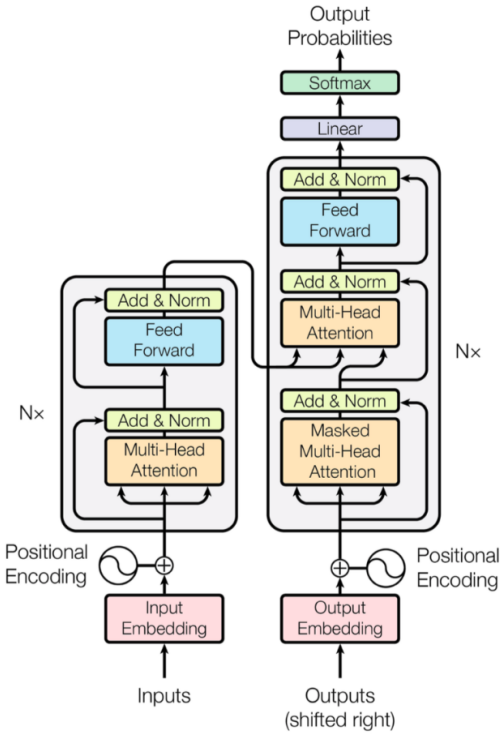


Figure 2.15: Transformer model detailed architecture. Encoder and Decoder consist of multiple modules that are stacked sequentially on top of each other. Each Encoder unit is composed of Multi-head attention mechanism and feed-forward network. Each Decoder contains two Multi-head attention sub-layers and feed-forward network. The final layer of Decoder part is linear layer with a softmax activation function. N_x represents the number of Encoder/Decoder units stacked on top of each other [64].

2.5 Embeddings

Word embedding is the method of converting words into vectors of real numbers, in order to feed the information to the machine learning algorithms efficiently. These

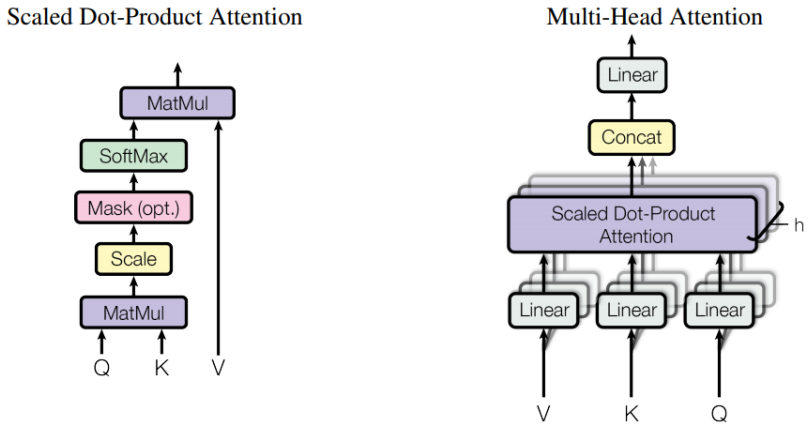


Figure 2.16: (Left) Scaled Dot-Product Attention (Right) Multi-Head Attention. In transformer architecture, h=8 parallel attention layers are employed [64].

vectors are densely distributed representation of words, having 10 to 100 dimensions. This is much effective than the one-hot encoding representation, where vectors are very sparse and computationally expensive. In addition to this, these embeddings also capture the semantic relationship between the words, unlike one-hot encoding, where even order of the words is not preserved. This efficient word representation helps algorithms to achieve better results on various tasks such as Text Classification, Sentence Similarity, Document Clustering, Machine Translation, and so on.

There are several ways to generate the word embeddings such as word2vec (obtained using single layer neural network), GloVe (word co-occurrence matrix) [42], fastText [12], ELMo [44], and so on. All these approaches are trained using huge text corpus such as Wikipedia, books, articles, etc. without labeling them. In the next section, we will briefly explain a few of them.

2.5.1 Word2Vec

One way of obtaining word embeddings is word2vec. It is a shallow neural network consisting of a single hidden layer, that takes text corpus (such as Wikipedia) as an input and generates the n-dimensional vectors for each word present in the corpus. It is an unsupervised method, which means it does not require human intervention to label the words. However, to generate the samples for training, a window of size m

(consider 5 words) is used against the words present in the text.

To train the word2vec as shown in figure 2.17, two approaches are introduced:

Continuous bag-of-words (CBOW) Model

This model consists of input, projection, and an output layer. Given the context words as input, it predicts the target word. Consider an example, a text corpus is *The quick brown fox jumps over the lazy dog*, the window size is 5 and the target word is *brown*. So the input to the model is context words which are *the quick fox jumps* and targeted output is *brown*. To train the model, during the first iteration, weights are initialized randomly and error is computed based on the predicted and targeted output. This error is reduced by adjusting the weights using backpropagation. According to Mikolov et al. [36], since CBOW model finds the output word with maximum probability, therefore, prediction of rare occurring words may not be so accurate.

Continuous skip-gram

This model is the reverse of the continuous bag of words. It takes the center word as an input and generates the context or neighboring words. Considering the same example, *The quick brown fox jumps over the lazy dog*, the window size is 5, the center word is *brown*. Here input is *brown* and targeted outputs are *[the, quick, fox, jumps]*. Similar to CBOW, this model is also trained using backpropagation and weights are adjusted to minimize the overall error.

In general, the vector representation of words obtained using both methods is efficient. This is verified by performing the operations between vectors of words. Figure 2.18 shows that when a vector of any country name such as *Germany* is added with word *capital*, it yields vector that is similar to their capitals such as *Berlin*.

Though word embeddings obtained using word2vec and GloVe perform better than one-hot encoding, these embeddings are the context-independent. For example, given a sentence *I am going to the bank to withdraw money, then I will come to the bank of river*. Here the context of word *bank* is different in the first and second part of the sentence. However, these methods produce same embeddings for the word *bank* [5]. Due to this problem, Deep learning methods such as ELMo and BERT are introduced to encode the information based on the context of the text.

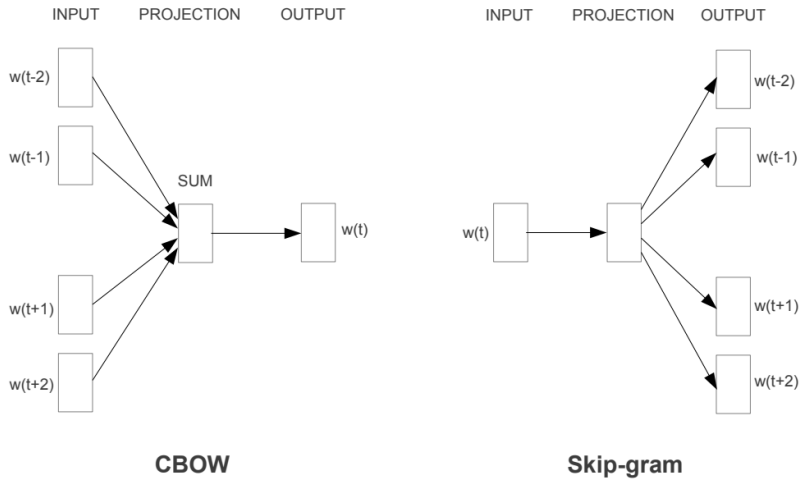


Figure 2.17: CBOW predicts the word based on its neighboring words. Skip-gram predicts the surrounding words based on the center word [36].

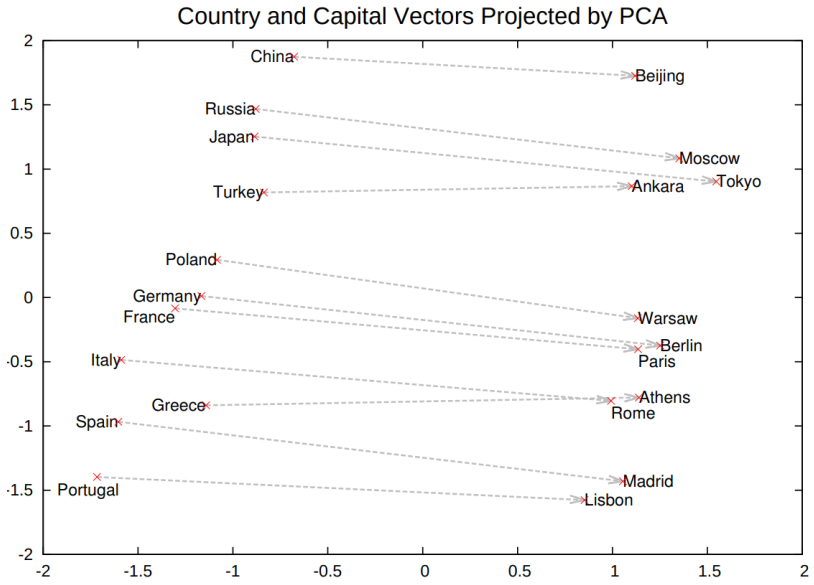


Figure 2.18: PCA projection of countries with their capitals vectors. This plot is reduced to 2D from 1000-D vectors obtained using skip-gram model [37].

2.5.2 Embeddings from Language Models (ELMo)

ELMo encodes the representation of words by considering the syntax, semantics, and their usage based on the context. Unlike word2vec, that generates embeddings

using the shallow feed-forward network, ELMo obtains the vectors for word representation using RNNs. More specifically bidirectional LSTMs (BiLSTMs) trained on huge text corpus are employed. Apart from this, it also considers character-based input representation, to handle the out of vocabulary words.

BiLSTMs is a set of LSTM units that encodes the information in forward (t+1, t+2, t+3,...) and backward (t-1, t-2, t-3,...) direction. This approach is efficient and performs better than uni-directional LSTMs as it uses the past as well as future information. Consider the sentences, *He said, Bill Gates is the founder of Microsoft*, and *He said, bill for the electricity is still pending*. Here to understand the word *Bill*, we need information from both the directions. Since, BiLSTMs are able to capture this information robustly. Thus, embeddings obtained using ELMo are context-dependent and their results are significantly better in tasks such as Question Answering, Textual Entailment, Co-reference Resolution, etc. Furthermore, forward (Left to Right) and Backward (Right to Left) LSTM units are trained independently and concatenated later to produce the final embeddings. Final word representation is a linear function of all hidden layers as shown in figure 2.19.

To understand the mathematical expressions, consider each token is represented as x_k, where k is the token number, and each layer *(L)* determines *2L+1* representations. Here *2L* shows that there are two embeddings obtained using forward and backward LSTM units. All these representations are combined into single vector (R_k), which is given as:

$$R_k = \{\mathbf{x}_k^{LM}, \overrightarrow{\mathbf{h}}_{k,j}^{LM}, \overleftarrow{\mathbf{h}}_{k,j}^{LM} | j = 1, ..., L\} \tag{2.10}$$

$$= \{\mathbf{h}_{k,j}^{LM} | j = 0, ..., L\} \tag{2.11}$$

where: $\overrightarrow{\mathbf{h}}_{k,j}^{LM}$ and $\overleftarrow{\mathbf{h}}_{k,j}^{LM}$ are the forward and backward LSTM units, that are represented as single LSTM unit $\mathbf{h}_{k,j}^{LM}$. j is the layer number, and j = 0 denotes the token layer.

In addition to this, task-specific embeddings are generated based on the linear composition of all the hidden layer representations. These embeddings are computed

as:

$$\mathbf{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^{L} s_j^{task} \mathbf{h}_{k,j}^{LM} \tag{2.12}$$

where: Θ^{task} is token representation of particular task such as question answering, sentence similarity, and so on. γ^{task} is used to scale the ELMo representations for the particular task. s_j^{task} are softmax-normalized weights [44].

Lastly, authors conducted several experiments on various tasks such as Question Answering, Textual Entailment, Named Entity Recognition (NER), and so on. As a result, embeddings obtained using ELMo performs better as compare to the previously existing approaches.

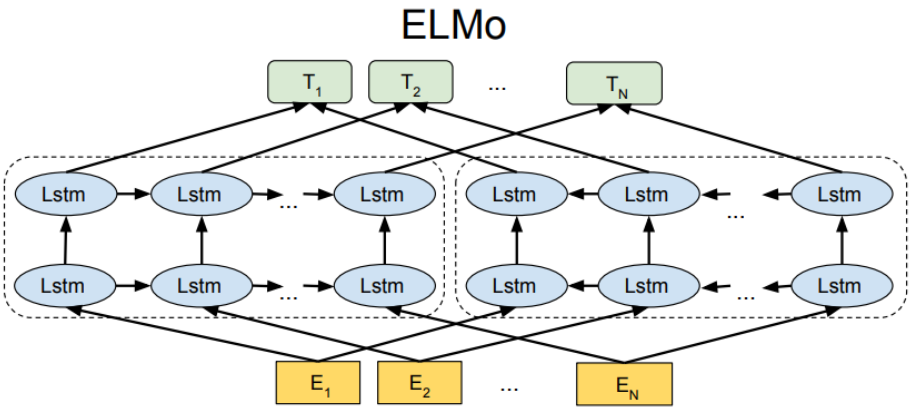


Figure 2.19: Embeddings obtained using ELMo are functions of the hidden BiLSTM layers. Here, E_i represents token embeddings, and T_i is final representation of token [21].

2.5.3 Bidirectional Encoder Representations from Transformers (BERT)

In [21], authors proposed another method used for language modeling. Instead of RNNs, it uses Transformer network (as explained in section 2.4) that completely omits the need for recurrent as well as convolutions. Unlike ELMo, that trains the forward

(Left to Right) and Backward directional LSTMs independently and concatenates their output, BERT is modeled to obtain pre-trained bidirectional word representations that are jointly dependent on the left and right context as shown in figure 2.21. Another feature that differentiates it from ELMo is that it uses two different training strategies: *Masked Language Model (MLM)* and *Next Sentence Prediction (NSP)*. In the following section, we explain the training methods used by BERT.

Masked Language Model (MLM)

The idea is that it randomly masks some of the words present in the sequence, and the task is to predict the words in the vocabulary list based on the context of other non-masked words. This approach is adopted to avoid the problem of standard conditional language models, in which during bidirectional modeling each word can "see itself" in a multi-layered architecture.

Next Sentence Prediction (NSP)

In this method, authors feed the pairs of the sentences as input along with the masked words. During the training process, 50% are the actual adjacent sentences labeled as 1, while another 50% next sentences are randomly selected and labeled as 0. Figure 2.20 shows how the training inputs are encoded during training. Initially, both the input sentences are represented using token, segment (sentence), and the position embeddings. In the input, [CLS] is added at the start of the sentence, and [SEP] is the token inserted to show the ending of each sequence. Further, Positional embedding is added to preserve the position of each word as we explained in the Transformer.

Lastly, the authors jointly train both strategies to reduce the overall loss.

Although BERT is used as the method of feature extraction, the authors also fine-tune it for the various Natural Language Processing (NLP) tasks such as Question Answering, Sentence Similarity, Named Entity Recognition (NER), and so on.
Note: Not all the tasks can be used for the fine-tuning as Transformers have strong requirements of the input representation.

Model Architecture

The architecture of BERT uses multi-layer bidirectional Transformer Encoder. Although, Transformer consists of Encoder and Decoder. However, for language modeling, only Encoder part is used. Further, authors proposed two models: $BERT_{BASE}$

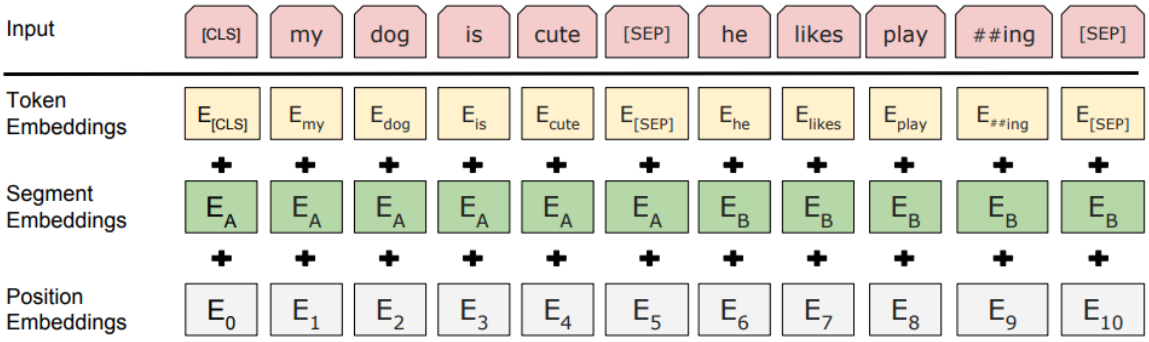


Figure 2.20: Input representation is addition of word, sentence and position embeddings. Two sentences are separated by constant identifier ([SEP]) [21].

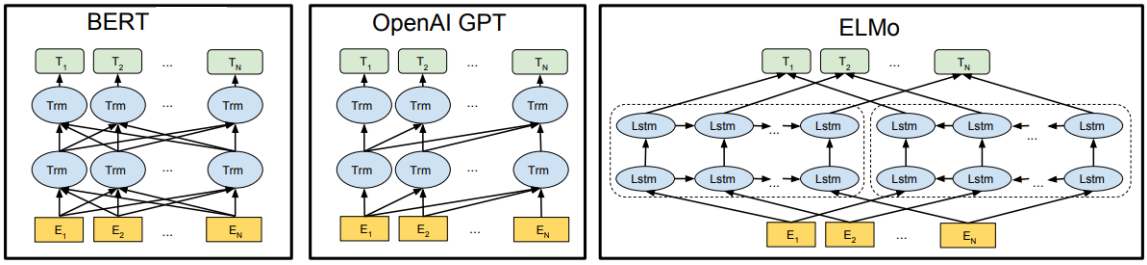


Figure 2.21: Comparison among BERT, OPENAI GPT, and ELMo architectures. ELMo uses Bidirectional LSTM units that are trained independently and concatenated later. OPENAI GPT uses Transformer that obtains word representation using context from only single direction. BERT employs bidirectional Transformer to produce the word embeddings based on both directions [21].

and $BERT_{LARGE}$ with specifications as ("L"=12, "H"=768, "A"=12, Total Parameters=110M) and ("L"=24, "H"=1024, "A"=16, Total Parameters=340M) respectively, where "L" is Transformer blocks, "H" is hidden size, "A" is the number of attention heads. Here, authors introduced $BERT_{BASE}$ to have the same size to compare the performance with Generative Pre-trained Transformer (OPENAI GPT) [6] model. Although OPENAI GPT also uses the concept of the Transformer, it is uni-directional as shown in figure 2.21.

Lastly, authors evaluated this model on eleven NLP tasks, as a result, it secures new SOTA results on datasets such as SQuAD v1.1, v2.0[1], Microsoft Research Paraphrase

[1]https://rajpurkar.github.io/SQuAD-explorer/

Corpus (MRPC)[2], Quora Question Pairs (QQP)[3], and so on. Due to this reason, we aim to use this approach in the domain of fact validation.

2.6 Basic Machine Learning Concepts

2.6.1 Loss Functions

Loss function also known as error/cost function is a vital concept in the neural network. It determines how far model's predicted values ($\hat{y}$) are from true values (y). In other words, it is the way of summarizing the performance of the model based on a single value. The efficiency of the model increases as the error/loss reduces as shown in figure 2.22. The general equation of loss function (J) is given as:

$$J = y - \hat{y} \tag{2.13}$$

Different loss functions are used for different tasks. Loss functions must be differentiable so that optimizers can reduce the loss. In neural networks, supervised tasks are divided into two categories: Classification and Regression. While Classification tries to map the function (f) from input (X) to categorical output variables ($\hat{y}$), Regression maps the function (f) from input (X) to numerical output values ($\hat{y}$). Examples of classification models are email classification (spam/not spam) and customer churn prediction. Examples of regression models are house price prediction and temperature forecasting. As we use classification losses in our work, we explain it briefly in the following section.

Loss Functions - Classification

In classification, there are several types of losses, but most commonly used is cross-entropy, which is explained briefly in the following section.

Cross-entropy loss is also called log loss. It is a way of determining the difference between the probability distribution of the model's predicted and actual values. In

[2]https://www.microsoft.com/en-us/download/details.aspx?id=52398
[3]https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs

other words, as the difference between both the probabilities increases, the performance of the model reduces. So, an ideal model should have loss as 0.

Cross-entropy is defined as:

$$L(y, \hat{y}) = -\sum_{i}^{C} y_i log(\hat{y}_i) \tag{2.14}$$

where: y_i and $\hat{y}_i$ are true and predicted scores by model for each class i present in C.

It is further divided into two types, that are illustrated below:

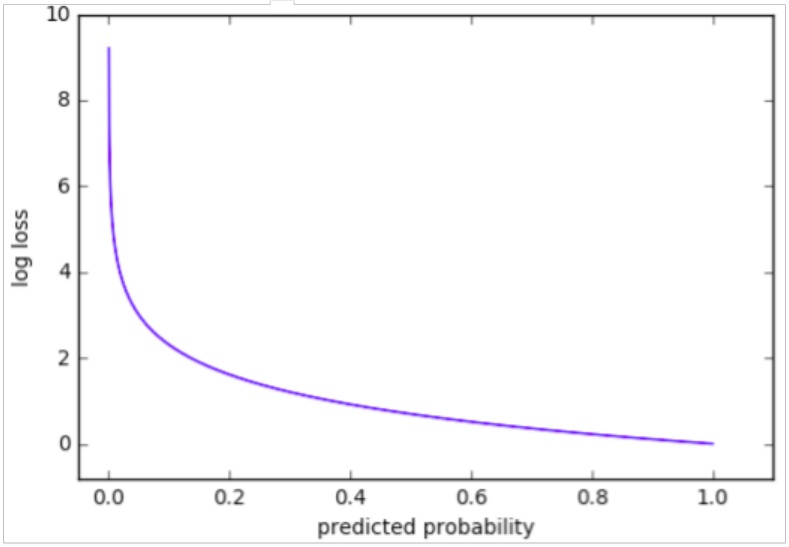


Figure 2.22: Relation between log loss and predicted probability [33]. As the loss decreases, predicted probability increases.

- **Binary Cross-Entropy**: It is used to compute the loss for the binary problems i.e: where only two classes are involved such as (1 (dog) vs 0 (not dog)). This loss is used with a sigmoid activation function. Therefore, it is also called as sigmoid cross-entropy loss. The mathematical expression is given as [2]:

$$L(y, \hat{y}) = -\sum_{i}^{C=2} y_i log(\hat{y}_i) = -y_i log(\hat{y}_i) - (1 - y_i) log(1 - \hat{y}_i) \tag{2.15}$$

where: $\hat{y}$ is predicted output computed based using sigmoid activation function and y_i is actual output.

- **Categorical Cross-Entropy**: It computes the losses when there are more than two classes. This loss function is also known as softmax loss. It is a combination of softmax activation and cross-entropy loss. It is defined as [2]:

$$L(y, \hat{y}) = -\sum_{i}^{N} \sum_{c}^{C} y_i^c log(\hat{y_i^c}) \tag{2.16}$$

where: N and C are total number of examples and classes respectively. For this loss, true labels (y^c) should be one-hot encoded. Here $\hat{y}$ is computed using softmax activation function.

2.6.2 Optimization Algorithms

It is the way of optimizing the training parameters (such as weights and biases) to reduce the overall loss of the model. These algorithms generally come under one of the categories [8]:

First Order Optimization Algorithms

These algorithms compute the first-order derivative of the loss/error function with respect to the adjustable parameters. This first derivative reveals the direction in which the loss is decreasing or increasing. Although several methods fall under the first-order optimization algorithms, Gradient Descent is widely used.

- **Gradient Descent**: In this algorithm, during each iteration, gradient of the cost/loss function ($J(\theta)$) is computed, and parameters are adjusted in the direc-

tion where overall error decreases. Following equation illustrates the parameters adjustment:

$$\theta = \theta - \eta \nabla J(\theta) \tag{2.17}$$

where: θ is trainable parameter, η is learning rate, which determines the step size to reach a local minima, and J is the cost function

- **Gradient Descent Variants**

  Gradient descent variants are differentiated based on the amount of data that is used for computation of the gradient of the loss function. Based on the quantity of data, there is a trade-off between the accuracy of the parameters' adjustment and computation time. Following section explains few important variants [56]:

  Batch Gradient Descent: It is also called as Vanilla gradient descent. It considers all the training examples to update the parameters once. The downside of this method is that it is slow and memory inefficient due to a large number of examples.

  Stochastic Gradient Descent: This method resolves the parameters' update time by performing adjustment of each training example individually. However, it uses the same learning rate (η) for all the training examples, which is a problem when we have a huge number of examples [49].

  Adaptive Moment Estimation (Adam): This is the most widely used optimization algorithm in Deep learning. It fuses the advantages of both momentum and Root Mean Square Propagation (RMSProp) [38]. Momentum helps the algorithm to advance in the direction of the global minima. Instead of using the same learning rate (η), RMSProp uses a different rate for each learning parameter e.g: weight and bias. These learning rates are adjusted automatically. As a result, oscillations are reduced to obtain the global minima [67].

Second Order Optimization Algorithms

Algorithms in this category determine the direction of increasing/decreasing error function by computing second-order derivative (also called Hessian) of the error with

respect to the trainable parameters. These algorithms are computationally expensive in terms of time as well as the memory, therefore mostly first-order algorithms are employed during training of the neural networks. For our work, we do not use these algorithms. Therefore, further details are not discussed.

3

State of the Art

In this chapter, we describe the few SOTA approaches used to solve FEVER shared task (1.0) [59] and FEVER simple claims task proposed by Chawla et al. [46]. In a nutshell, FEVER shared task involves all three steps for fact validation: document retrieval, sentence/evidence selection, and claim classification/labeling. On the other hand, FEVER simple claims require only prediction of claim labels i.e. claim classification.

3.1 FEVER Shared Task

The problem statement is to classify the claim whether it is entailing, contradicting or documents does not contain any information related to the claim. Apart from the correct claim labels, the system also has to determine the related documents, and find accurate potential evidences. To perform this, authors in [59] introduced the FEVER dataset that is consists of 185,445 annotated claims. In this work, we call it as *FEVER original* dataset. To initiate this challenge, authors in [60] proposed a baseline approach that obtained the F1 score of 18.26%, label accuracy of 48.84% and the FEVER score as 27.45%. Figure 3.1 depicts the result of the teams that participated in the first FEVER shared task. Following section briefly explain the SOTA approaches.

Rank	Team Name	Evidence (%)			Label	FEVER
		Precision	Recall	F1	Accuracy (%)	Score (%)
1	UNC-NLP	42.27	70.91	52.96	**68.21**	**64.21**
2	UCL Machine Reading Group	22.16	82.84	34.97	67.62	62.52
3	Athene UKP TU Darmstadt	23.61	**85.19**	36.97	65.46	61.58
4	Papelo	**92.18**	50.02	**64.85**	61.08	57.36
5	SWEEPer	18.48	75.39	29.69	59.72	49.94
6	Columbia NLP	23.02	75.89	35.33	57.45	49.06
7	Ohio State University	77.23	47.12	58.53	50.12	43.42
8	GESIS Cologne	12.09	51.69	19.60	54.15	40.77
9	FujiXerox	11.37	29.99	16.49	47.13	38.81
10	*withdrawn*	46.60	51.94	49.12	51.25	38.59
11	Uni-DuE Student Team	50.65	36.02	42.10	50.02	38.50
12	Directed Acyclic Graph	51.91	36.36	42.77	51.36	38.33
13	*withdrawn*	12.90	54.58	20.87	53.97	37.13
14	Py.ro	21.15	49.38	29.62	43.48	36.58
15	SIRIUS-LTG-UIO	19.19	70.82	30.19	48.87	36.55
16	*withdrawn*	0.00	0.01	0.00	33.45	30.20
17	BUPT-NLPer	45.18	35.45	39.73	45.37	29.22
18	*withdrawn*	23.75	86.07	37.22	33.33	28.67
19	*withdrawn*	7.69	32.11	12.41	50.80	28.40
20	FEVER Baseline	11.28	47.87	18.26	48.84	27.45
21	*withdrawn*	49.01	29.66	36.95	44.89	23.76
22	*withdrawn*	26.81	12.08	16.65	57.32	22.89
23	*withdrawn*	26.33	12.20	16.68	55.42	21.71
24	University of Arizona	11.28	47.87	18.26	36.94	19.00

Figure 3.1: Results of teams participated for FEVER Shared Task [59].

3.1.1 FEVER Baseline

In [60], authors proposed the FEVER baseline approach that mainly consists of 3 components: document retrieval, sentence selection, and textual entailment. To find the documents related to claim, authors consider DrQA [15] system that computes the cosine similarity between the TF-IDF vectors based on binned unigram and bigram of input claim and related articles. After retrieving the "k" nearest articles, "l" sentences from each article are selected that are closer to input claims by using Modified DrQA with binned TF-IDF vectors. Authors did a comparison between two models to find textual entailment and used one with better performance. The first model was introduced by Riedel et al. [52] during fake news challenge [19], uses multi-layer perceptron (MLP) that consider features as cosine similarity between TF-IDF vectors of input claim and evidence. Second model was based on decomposable

attention (DA) proposed by Parikh et al. [40] that takes claim and evidence as input, outputs whether evidences "entails", "contradicts" or are "neutral". Authors used decomposable attention (DA) model as its performance was better than other SOTA models on Stanford Natural Language Inference (SNLI) [13] dataset.

However, according to [71], this approach has two drawbacks: first, it retrieves complete evidence set for only 55% of input claims, and TF-IDF is a general approach to compute the similarity, a system would perform better if task-specific features are extracted and a simple linear model is used. Second, the textual entailment module is not robust, in case of providing multiple sentences as evidence, it just concatenates all the evidence sentences into one paragraph.

3.1.2 UNC-NLP

UNC-NLP team [39] introduced the system that obtained 1^{st} position in FEVER shared task. Authors considered similar neural semantic matching networks (NSMN) for all three units; document retrieval, sentence selection, and the claim verification. To retrieve relevant documents, initially, all the articles are selected based on keyword matching and later on ranked and filtered by using the semantic matching network. Next, the top 5 sentences are selected from the retrieved articles. This is done by using the same network i.e. NSMN to compute the semantic similarity between claim and sentences extracted from all ranked documents. After that, token-level features, ontological relations from WordNet, embeddings of numerical values, and semantic relatedness scores are extracted from extracted sentences and fed to the NSMN to predict the relation between claim and the evidences.

3.1.3 Athene UKP TU Darmstadt

In [24], authors proposed the system to solve the FEVER shared task. To retrieve the related documents to input claim, entity linking method along with constituency parser and handcrafted features is used, as in Wikipedia corpus, each article represents an entity. In order to find the potential evidences for claims and to label them, the authors used the Enhanced Sequential Inference Model (ESIM). ESIM was presented by Chen et al. [17] to determine the textual entailment on SNLI dataset. It uses

Bidirectional LSTM (BiLSTM) and multi-layer perceptron to find the entailment relation. Furthermore, authors modify ESIM architecture, to rank the top k sentences in each article related to claim. After retrieving the top k sentences similar to the claim, ESIM is further modified, and entailment relation is predicted between the input claim and top k sentences.

3.1.4 DeFactoNLP

This approach is present in table 3.1 as *Directed Acyclic Graph*. Reddy et al. [51] introduced the system for the FEVER shared task which is consists of four steps: document retrieval, sentences selection in form of evidences, textual entailment recognition, and the claim classification. For document retrieval, authors use two methods: one by using the TF-IDF vector method, which also finds relevant evidence supporting/refuting the claim based on a cosine similarity between input claim and sentences present in each document. The second method used for document retrieval is named entity recognition (NER). In case of later method, all the sentences present in documents are returned as they might represent the possible evidence.

Relevant sentences extracted from both the document retrieval methods are concatenated and applied as an input to the textual entailment recognition component, where it calculates the probabilities of sentences entailing, contradicting, or are uninformative with respect to the claim. To compute the probabilities, authors considered decomposable attention [40] model for textual entailment as it has higher performance as compared to other SOTA approaches on SNLI [13] task. However, it was observed that this model did not perform well on FEVER dataset, therefore, decomposable attention [40] model was trained on SNLI dataset and later on fined tuned on the FEVER SNLI-style dataset [50]. Lastly, twelve features were extracted from the probabilities obtained using textual entailment module and fed to the random forest classifier to predict the label of the claim.

For the future work, authors are interested to enhance the sentence retrieval and identification of claim evidence by incorporating the triple extraction methods in a format of subject-predicate-object rather than textual claims. Along with this, named entity disambiguation can be performed by applying entity linking methods.

3.1.5 Pepelo

In [32], another system is developed to solve the FEVER task. It is also comprised of three stages: document retrieval, sentence retrieval, and textual entailment. Though it relies more on Transformer network used to recognize textual entailment. Apart from this, the authors modified the sentence and document retrieval approach presented in FEVER baselines. As the first adjustment, during the computation of TF-IDF vectors between the claim and premise statement, authors attach the title of the article to each premise sentence to make sure even if in premise sentence subject is not replicated, the relevant article still gets credit. This improves the performance of evidence retrieval as compared to the baseline approach. The second modification uses named entity recognition (NER) to retrieve Wikipedia articles, which improves evidence retrieval from 68.3% to 80.8% . Since most of the questions present in FEVER Development set are related to films, therefore, authors also retrieve the page with title "X (film)" if it exists in a claim. This method slightly enhances evidence retrieval from 80.8% to 81.2%. Furthermore, instead of retrieving five sentences based on TF-IDF cosine similarity from entire articles. Authors extend to first fifty sentences based on the highest cosine similarity from each article. These sentences are fed to the Transformer network to perform the textual entailment.

Moreover, this work considers the entailment model that uses a transformer network (introduced by Vaswani et al. [64]). More specifically, the transformer network proposed by OpenAI is used, which is trained for language modeling. This network is comprised of twelve units and two branches, each unit contains multi-head attention, feedforward network, and two normalization layers. Out of two branches, one branch predicts the entailment classification and second one predicts the next token. For FEVER task, authors use the pre-trained model on the Book-corpus dataset [73] and re-train both branches on FEVER dataset.

3.1.6 UCL Machine Reading Group

In [71], authors describe a system consists of four modules: document retrieval, sentence selection, recognizing textual entailment (or natural language inference (NLI)), and predictions aggregation. In the document retrieval phase, authors construct a

dictionary containing article titles from Wikipedia, as most of the claims present in FEVER dataset [60] have Wikipedia documents title. So, for each input claim related articles are ranked by training logistic regression based on features such as sentence position, stop words, token matching between input claim and first sentence present in each article, and capitalization. For Sentence retrieval, authors also use logistic regression model that is trained on features such as document retrieval score, token matching between input claim and sentence present in each article, location of a sentence, and length of sentence. As a result, k-ranked sentences from this module are fed to the natural language inference model. To recognize textual entailment, ESIM (proposed by Chen et al. [17]) is trained on SNLI corpus [13], later on, tuned on dataset presented by FEVER. Lastly, during predictions aggregation stage, all the predicted labels and evidence are combined to predict the final output.

For the future, this work can be extended to include ELMo embeddings(proposed by [44]) to deal with complicated sentences. Furthermore, numerical expressions in claims such as years (1970s vs 70s) can be handled efficiently, as these expressions have similar word embeddings due to which performance of NLI decreases.

3.1.7 SWEEPer

Another system was proposed by SWEEPer team [26]. Like other SOTA approaches, the proposed approach is also comprised of three units: document retrieval, sentence retrieval, and textual entailment. But, are merged into two stages: (1) find relevant articles (2) Mutually extracting related sentences from the top k-ranked articles and classify the entailment relation between the input claim and each sentence. In the first stage, to fetch relevant Wikipedia articles, the classifier is trained based on lexical and syntactic features. All the features are extracted between the top two sentences present in each article and input claim. In the second stage, based on a weighted combination of cosine and Jaccard similarity of claim and sentences average word embeddings, top “m” sentences are selected. Afterward, memory network such as ESIM (proposed by Chen et al. [17]) is used to generate the embeddings of input claim and each selected sentence, and pointer networks [66] are used to identify only related sentences. Lastly, to recognize the final label of the claim based on the evidence sentences, a multi-layer perceptron is trained and used for the prediction.

For future work, the authors consider to enhance the performance of document retrieval and explore different sentence representations.

3.1.8 Columbia NLP

Columbia NLP [14] team introduced the end-to-end approach for the FEVER shared task, that is consists of three components: document retrieval, sentence selection, and textual entailment. To fetch relevant document related to the claim, three methods are proposed: Google custom search API (proposed by Wang et al. [69]), named entity recognition, and dependency parsing. Authors provide claim as an input to the Google custom search API and retrieve the top two documents. In the second method, pre-trained bidirectional language model [45] presented by AllenNLP [23] is considered to recognize the named entities in the claim, after that, with the help Wikipedia python API, top k-Wikipedia documents are retrieved related to entities in the claim. In dependency parsing, authors compute the dependency tree of the claim and query only words using Wikipedia API that are present before the first lowercase verb phrase, this decrease the chances of missing the relevant entities in input claims. Lastly, documents retrieved by all three methods are combined and fed to the sentence selection component. As a result, the proposed document retrieval method performs better as compare to FEVER baseline [60] approach for document retrieval.

In order to select the relevant sentences, authors consider document retrieval DrQA component proposed by Chen et al. [15] and choose relevant sentence using binned bigram TF-IDF vectors as presented in [60]. Although this sentence selection approach performs better on the blind test set provided by FEVER, it introduces additional noise when sentences were fed to the textual entailment module. Therefore, to tackle this problem, ELMo embeddings (proposed by Peters et al. [44]) are used to transform the input claim and relevant sentences to vectors. Afterward, cosine similarity between both the vectors is computed and the top 3 sentences are extracted according to the scores. To recognize textual entailment, a model proposed by Conneau et al. [18] is used. Initially, input claim and set of evidence are encoded in vectors (u,v) using bidirectional LSTMs along with max-pooling layer. To extract the relations between vectors (u and v) following methods are introduced: (i) concatenation of (u, v)

representations (ii) element-wise multiplication (u*v) and (iii) absolute element-wise subtraction $|u-v|$. Results are fed to the 3-way classifier formed using fully connected layered network. To predict the final output of the system, boosted tree classifier is trained that considers confidence scores of claim evidence pairs as well as their location in the document.

3.1.9 Conclusion

To summarize all the approaches, most of the authors divide this task into three steps: document retrieval, sentence selection, and natural language inference (NLI) or claim classification. Some of them also jointly solve sentence retrieval as well as claim labeling and train a single model for both the tasks. However, others consider an extra step after performing the textual entailment, to remove false sentences such as the use of decision trees i.e. prediction aggregation step. Table 3.1 illustrate the different methods that SOTA approaches consider for sentence retrieval and claim classification. We can observe that the majority of the approaches use TF-IDF for sentence selection, while few of them employs deep learning approaches such as NSMN, ESIM, and Pointer networks. On the other hand, to solve claim labeling task, majority of the approaches apply deep learning approaches with different word embeddings such as Glove, wordNet, and FastText.

Most of the approaches use word embeddings to represent the text, but there are limitations such as these embeddings are not contextual as compare to recently introduced embeddings such as ELMo and BERT. Also using these representations, the authors have achieved SOTA results on various tasks such as Question Answering, Paraphrase detection, Named Entity Recognition, Natural Language Inference, and so on. Thus, in this work, we aim to compare the performance of classical NLP techniques (such as TF-IDF, Vector Space, and word mover's distance) and deep learning approaches (LSTM using word2vec, ELMo, and BERT embeddings) for sentence retrieval and claim classification in the domain of fact validation. At the end, we also integrate the sentence retrieval and claim labeling components with the DeFactoNLP to compare the performance with the SOTA approaches.

In the following chapters, we discuss and show the application of mentioned deep learning and classical approaches for the fact validation.

Approach	**Sentence Selection**	**NLI/Claim classification**	**Aggregation**
FEVER-Baseline	Modified DrQA with binned TF-IDF vectors	Decomposable Attention with Glove embeddings	-
UNC-NLP	NSMN	NSMN along with wordNet embeddings	-
Athene UKP	Modified ESIM	ESIM with Glove embeddings	Multi-Layer Perceptron (MLP)
DeFactoNLP	Modified DrQA with binned TF-IDF vectors + all sents from NER docs.	Decomposable Attention with Glove embeddings	Random forest with hand crafted features
Pepelo	TF-IDF	Transformer network	-
UCL	Logistic Regression	ESIM with Glove embeddings	MLPs
SWEEPer	Word embeddings + Pointer nets	MLP	-
ColumbiaNLP	TF-IDF + ELMo embeddings	LSTM with FastText embeddings	Majority voting

Table 3.1: Approaches used for sentence selection and claim classification for FEVER shared task. "-" in aggregation column shows that those approaches perform aggregation and claim classification jointly in a single step.

3.2 FEVER Simple Claims Task

In this section, we describe the approaches that are used to solve the FEVER simple claims task. This task was proposed by Chawla et al. [46]. Here, the problem is limited to claim classification only. So this task does not involve fetching of documents and potential evidences step. As far as we know, to solve this task, most of the approaches extract features from the claims and sentences, later on, use supervised models to determine the final label. However, only single approach considered deep learning methods such as LSTM and decomposable attention for this task.

In [61], authors employed hand crafted features to solve the stance detection. More specifically, authors considered set of 20 features such as distance between vectors of claims and sentences, similarity measures, overlap between token attributes, sentiment

scores between evidences and claims, and so on to train gradient boosted ensemble classifier to solve the task. In [46], authors evaluated different set of approaches such as feature-based approaches mainly extracting features from the claims and sentences to train the supervised model such as Random Forest or XGBOOST [61] and Textual Entailment (TE) approach that use decomposable attention [40] by AllenNLP [23] to solve this task. Furthermore, authors also proposed new feature based and LSTM model using word2vec embeddings. As a result, LSTM model outperformed SOTA by 11%, 10.2%, and 18.7% on the FEVER simple claims dataset.

To summarize this, we can observe that deep learning approach outperforms the feature based methods. Furthermore, authors have used LSTM with word2vec embeddings for this task. We believe that by incorporating new contextual embeddings such as ELMo and BERT, performance on this dataset can be significantly improved. More specifically, in this project, we aim to explore different classical as well as deep learning claim classification approaches and determine the performance on FEVER simple claims datasets. At the end, we compare the results with the SOTA methods.

4

Methodology

In this chapter, we describe evidence retrieval (sentence selection) and claim classification approaches that are employed during our work. Apart from this, we also explain how independently trained models for evidence retrieval and claim classification are connected in a fact validation pipeline to determine the final label of a claim with the help of majority vote classifier.

Fact validation generally consists of three components: relevant document retrieval, evidence retrieval, and claim classification. For a given claim, related documents are fetched from the database. Next, potential evidences related to the claim are selected. After that, each evidence is assigned a label whether it is supporting, refuting or does not contain any information related to the claim (Not Enough Info (NEI)). Lastly, the final label is decided by aggregating the labels of all the selected evidences as shown in figure 4.1. However, in some other contexts, fourth components apply, where task is to determine source credibility score.

In this work, we mainly focus on evidence retrieval and claim classification. So, we use a document retrieval component from DefactoNLP project. For each task, six approaches: Vector space, Term Frequency-Inverse Document Frequency (TF-IDF), Word Mover's Distance (WMD), Simple LSTM using word2vec, ELMo, and BERT embeddings are employed. Later on, all the approaches are bench-marked on two datasets: FEVER original (explained in section 5.1.1) and simple claims (details in 5.1.2), which is further divided into four sub-datasets. On FEVER original set, the task involves all three components. However, on FEVER simple claims dataset, the problem is only limited to claim classification. So, evidences related to the claim are

already present.

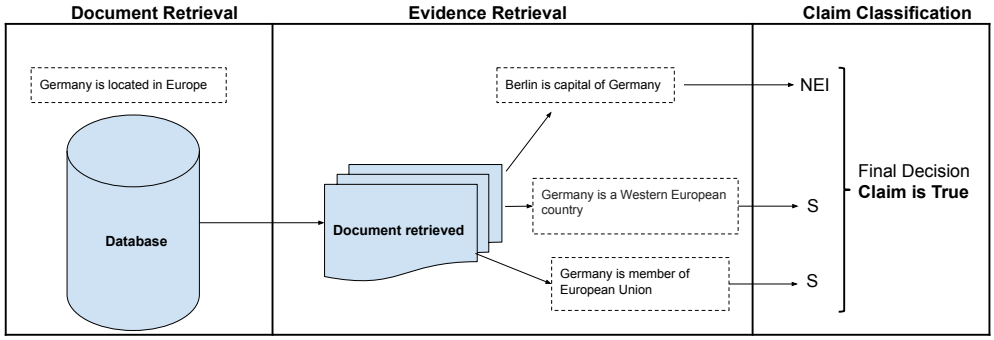


Figure 4.1: General steps required for validation of the fact.

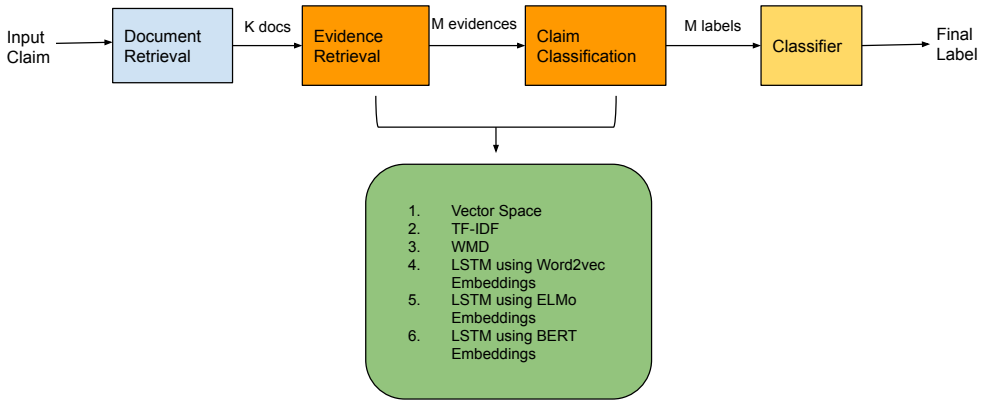


Figure 4.2: Complete pipeline of the Fact Validation. It is consists of document retrieval, evidence/sentence retrieval, claim classification, and the meta classifier.

Figure 4.2 illustrates the complete pipeline to determine the final label of the claim. To solve the evidence retrieval, we treat it as a binary problem: "related" and "not related". This helps us to determine the top n sentences associated with the claim. Also, this step reduces the search space of the sentences. These results are fed to the claim classification module. This component considers claim labeling as multi-class classification, where labels are "SUPPORTS", "REFUTES", and "NEI". It assigns the "m" labels to the "m" related sentences. So, to determine the final label for the claim, meta classifier is added at the end of the pipeline.

All the employed approaches train the models for evidence retrieval and claim classification independently. So, each component assumes that previous component outputs are 100% correct. Later on, trained models outcomes are combined to evaluate both components on the test set. The following section explains all six different approaches that are considered in this work:

4.1 Evidence Retrieval and Claim Classification Approaches

4.1.1 Vector Space

In this approach, the text is converted in the form of vectors. Initially, a dictionary of words is constructed based on the words present in the text. In each vector, words are represented based on their counts. The dimension of the vector depends upon the vocabulary size of the corpus. For example, given the text of two sentences: ['The quick brown fox jumps over the lazy dog', 'fox is lazy']. Dictionary of words constructed as {'brown': 0, 'dog': 1, 'fox': 2, 'jumps': 1, 'lazy': 2, 'over': 1, 'quick': 1, 'the': 2}. After representing the sentences in the form of vectors, cosine similarity is computed to determine how similar two vectors are.

The limitation of this approach is that it cannot handle synonyms and semantics of the word in different contexts. Furthermore, the size of the vector depends upon the vocabulary size due to which computation problems occur. Also, there is no way to determine the important terms present in the text.

In the coming section, we explain how this approach is used in the context of sentence retrieval and claim classification.

How it is used for evidence retrieval and claim classification?

To use this approach for evidence retrieval, initially, claims and "m" sentences are converted into vectors based on the occurrence of each word. After that, cosine similarity is computed between claim and each sentence. Top "n" sentences are ranked based on their similarity scores and considered as closely relevant to the claim. Now these "n" sentences are fed to the claim classification model to determine the labels (SUPPORTS, REFUTES, NEI) based on the range of similarities scores between claim and n sentences. All the scores lie in the range of 0 to 1. We choose thresholds

manually to assign the label. For example, if score < 0.1, then label is "Not Enough Info", else if $0.2 <$ score < 0.6, then label is "REFUTES", else "SUPPORTS".

$$label = \begin{cases} NEI & score < 0.1 \\ REFUTES & 0.1 \leq score \leq 0.6 \\ SUPPORTS & score > 0.6 \end{cases} \tag{4.1}$$

Note: For now, thresholds in 4.1 are chosen randomly. Later on, we discuss how thresholds are selected for different datasets.

Claim classification module assigns "n" labels to "n" sentences. Since idea is to assign a single label to the claim based on the evidences. Therefore, the majority vote classifier is employed.

4.1.2 Term Frequency - Inverse Document Frequency (TF-IDF)

Another way of finding similarity between claim and sentences is TF-IDF. It is a way of assigning weights to the words present in the document and is computed as the product of TF and IDF of the word. This product defines how frequent is the occurrence of each term in the corpus. Higher the product, rarer the term's frequency and more it is important. To use TF-IDF to compute similarity, first TF-IDF of each word present in the sentence and claim is calculated and then cosine similarity between TF-IDF vector of a claim and each sentence is computed. The formula of TF-IDF is [4]:

$$TF(t) = \frac{Count\ of\ term\ t\ in\ a\ doc}{count\ of\ total\ terms\ in\ doc}$$

$$IDF(t) = \log \frac{total\ number\ of\ docs}{No.\ of\ docs\ that\ contains\ term\ t}$$

$$W(t) = TF(t)\ IDF(t)$$

The limitation of this method is that it relies on string similarity which means it

cannot handle synonyms, and it also uses a bag of words model, therefore, does not capture the semantics, unlike other word embedding techniques [62].

Furthermore, we employed this approach for sentence retrieval and claim labeling that is explained in the following section.

How it is used for evidence retrieval and claim classification?

For evidence retrieval, sentences and claims are converted in the form of vectors, where each term is encoded using weights computed by TF-IDF. After that, cosine similarity scores are computed between claim and each sentence present in document relevant to the claim. Top "n" sentences are fetched with highest similarity scores, which shows that evidences are related.

Next, the results of evidence retrieval component are considered for the classification of the claim. Again, top n sentences and claim are transformed into vector representation and similarity is calculated between claim and each sentence. As a result, claim classifier associates n labels to the n sentences based on the scores. Similar to the previous approach, meta classifier is considered to determine whether relevant evidences are entailing, contradicting or does not contain any information related to the claim.

4.1.3 Word Mover's Distance

It is another way of determining the similarity between two or more documents/sentences. It measures similarity based on the minimum distance between words as well as sentences. For a given two sentences, first, it computes the minimum distance between the words present in both the sentences. Then, it adds the distances to find how similar both the sentences are.

Figure 4.3 shows the example that considers the three documents (D_0, D_1, and D_2). Initially, the algorithm transforms each word into vectors. This is done by using word2vec or GloVe embeddings. After that, it computes the euclidean distance between all the word vectors. Words having least distance between two docs. are considered similar (E.g: *Obama* and *President* in D_1 and D_0 respectively). Later on, word distances are accumulated to determine the similarity between documents. As a result, sentences having minimum distances are considered as similar. Furthermore,

this method does not considers stop words (such as “to”, “the”, “in”) as they do not have any meaning.

In addition, this approach is considered as hyper-parameter free and by using word embeddings, it overcomes with synonym problem unlike other sentence similarity approaches such as bag-of-words and TF-IDF vectors. Along with this, outputs of this approach are easily interpretable and individual distances between words are also accessible. However, similar to previous methods, this approach does not preserve the order of the words [22].

In the paper [30], authors have evaluated proposed approach on eight document classification datasets such as REUTERS [3], TWITTER [55], AMAZON [11], etc. and compared performances with seven sentence/document similarity methods such as bag-of-words (BOW), Term Frequency-Inverse Document Frequency (TF-IDF) [54], BM25 Okapi [53], Latent Semantic Indexing (LSI) [20], Latent Dirichlet Allocation (LDA) [10], Marginalized Stacked Denoising Autoencoder (MSDA) [16], and Componential Counting Grid (CCG) [43]. As a result, the proposed approach has the lowest error as compared to other SOTA methods.

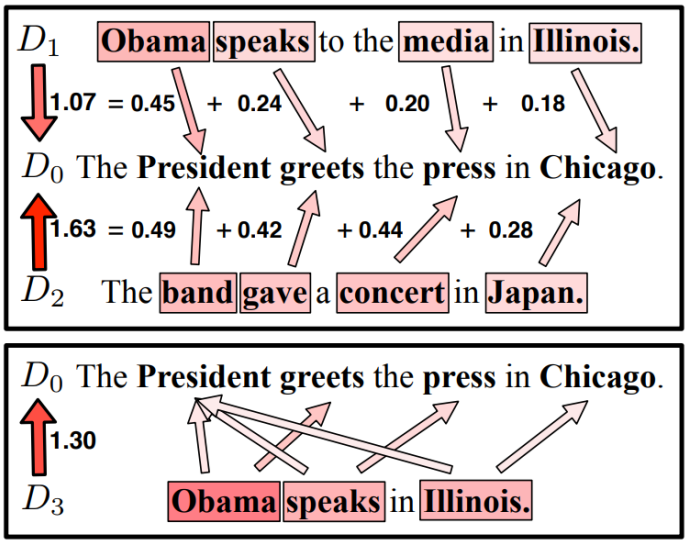


Figure 4.3: Word Mover’s Distance example. Similarities between three documents is computed and as a result, D_0 is closer to the D_1 due to less euclidean distance [30].

To retrieve the potential evidences and determine the final label of the claim using

this approach, following procedure is considered.

How it is used for evidence retrieval and claim classification?

For evidence retrieval, sentences and claims are encoded into vectors using word2vec embeddings. Next, the euclidean distance between claim and each sentence is computed, and n sentences with the lowest distance are considered as relevant to the claim. After that, these relevant evidences are provided to the claim classification unit, which assigns the label to each sentence based on the distances. Following criteria is used to determine the label for each claim:

$$label = \begin{cases} NEI & distance > 2.2 \\ REFUTES & 1 \leq distance \leq 2.2 \\ SUPPORTS & distance < 1 \end{cases} \tag{4.2}$$

This module assigns n labels to the n relevant sentences. Later on, meta-classifier is incorporated to determine whether relevant evidences are entailing, contradicting or does not contain any information related to the claim.

4.1.4 Simple LSTM Network using word2vec Embeddings

This is the deep learning approach that uses LSTM-Ns for evidence retrieval and claim classification with the help of word2vec embeddings. We have already explained the details of word2vec and the working of LSTM Networks in previous sections (2.5.1 and 2.2.1). By Simple LSTM network, we mean LSTM units are uni-directional rather than bi-directional. To feed the sentences and claims to the deep learning model, we encode the text using pre-trained word2vec embeddings. The dimensions of embeddings are the tuning parameters (such as either of 50-D, 100-D, 200-D, and 300-D is selected based on the accuracy on validation set). To solve the task of labeling the claims, we train the models for evidence retrieval as well as claim classification independently. While training the evidence retrieval, it is assumed that document fetching related to the claim is 100% correct, and during the training of claim classification, we assume that predictions of document retrieval, as

well as sentence retrieval, are fully accurate.

Figure 4.4 shows the LSTM architecture that is employed to train the sentence retrieval model on FEVER Support set (explained in section 5.1). To train both the models, first we consider claims and sentences in parallel, then embeddings are computed and provided as input to the LSTM unit. Next, resulting vectors of claims and sentences are concatenated, and fed to the fully connected layer followed by the dropout and output layer to produce the final label. In the coming section, we describe the procedure of training the evidence retrieval and claim classification components.

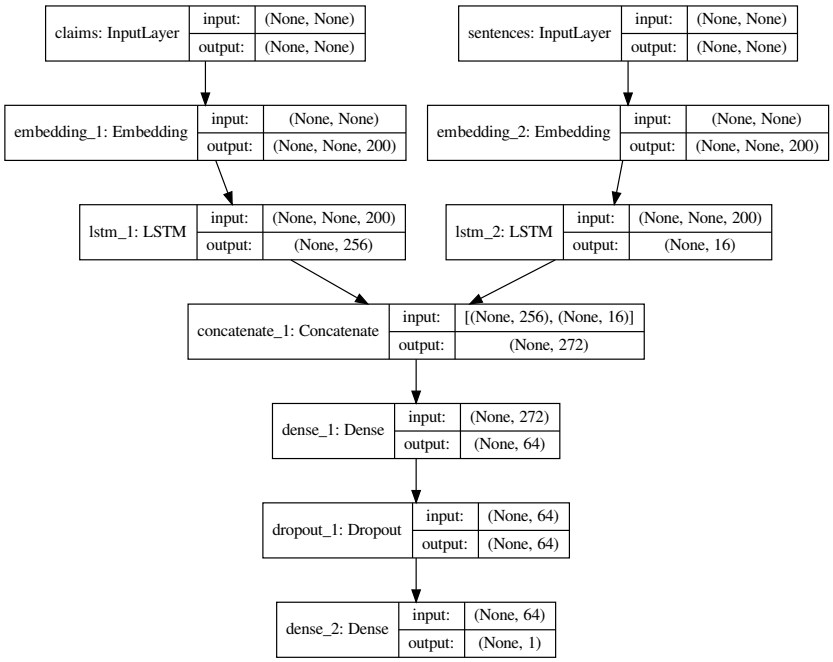


Figure 4.4: LSTM Architecture trained on FEVER Support set. Here we use embeddings of 200-dimensions. We consider separate LSTM units for claims and sentences. Later on, both the LSTM units are concatenated and fed to the Fully connected layer (Dense) followed by the dropout and final output layer.

How training of evidence retrieval model is performed?

To train the model, we treat this problem as a binary classification. Assume we

have a single claim and m sentences retrieved by related documents to the claim. We train the model with an equal number of true positives and negatives. Like, Along with the true positive sentences related to the claim, we also randomly sample true negatives from the retrieved documents. This is done so that the model is not biased towards one class. Following this procedure, we make the pairs of $< claim, sentence_i >$ where i = 1,2,..,m and assign the label as 1 (means evidence is related to the claim) to the true positives and 0 to the true negatives. As far as loss and optimizer are concerned, we use "Binary cross-entropy" and "Adam optimizer" to train the model.

How training of claim labeling model is performed?

For this model, we consider this task as the multi-class classification. This means the final output will be any of these: "SUPPORTS", "REFUTES", or "NOT ENOUGH INFO". Similar to the training of the previous model, we also consider here an equal number of true positives and negatives sentences. Using this process, we create the pairs of $< claim, sentence_i >$ and assign the label as "SUPPORTS", or "REFUTES" based on the true label to the true positive and "NOT ENOUGH INFO" to the true negative evidences. To train this classifier, we consider "categorical cross-entropy" and "Adam optimizer".

Lastly, output of claim classifier is fed to the meta-classifier, where single final label is determined based on the majority voting.

4.1.5 Simple LSTM Network using ELMo Embeddings

This is another approach that uses LSTM-Ns, however, instead of word2vec embeddings, it considers representation obtained using ELMo (explained in 2.5.2). As we discussed previously, word2vec is unable to represent the words based on the context. E.g: representation of word "Bill" is different in the sentence "Bill Gates is the founder of the Microsoft", and "I am paying the coffee bill". But it assigns similar representation in both the contexts to the word "Bill". In this approach, we use the pre-trained ELMo embeddings proposed in [44]. In a nutshell, these embeddings are obtained by using Bi-LSTM-Ns, that can handle out-of-vocabulary words and encode words based on the context of the sentence. Similar to the previous approach, the first step is to represent the tokens in the form of vectors. To accomplish this, we

use AllenNLP[1] python library. Next, we fed the embeddings of the claims and the evidences in parallel to the LSTM-Ns units. After that, outputs of both the LSTM-Ns units are merged and fed to the fully connected layers followed by the final output layer. Like other approaches, we train the sentence retrieval as well as claim classifier individually and combine the results while testing on the test datasets. To train the evidence retrieval model, we use "Binary cross-entropy" loss and "Adam optimizer" due to the 2-class classification task. Since claim labeling, is a 3-class problem, thus we employ "Categorical cross-entropy" and "Adam optimizer" to solve this task. In the end, we use a simple classifier to predict the final label of the claim.

Figure 4.5 shows the LSTM architecture using ELMo embeddings that are employed to train the sentence retrieval model on FEVER Support set. To train both the models, first we consider claims and sentences parallel, then embeddings are computed and provided as input to the LSTM unit. Next, we concatenate both the parallel LSTM units and feed to the dense layer followed by the dropout and output layer to determine the final label.

[1]https://allennlp.org/elmo

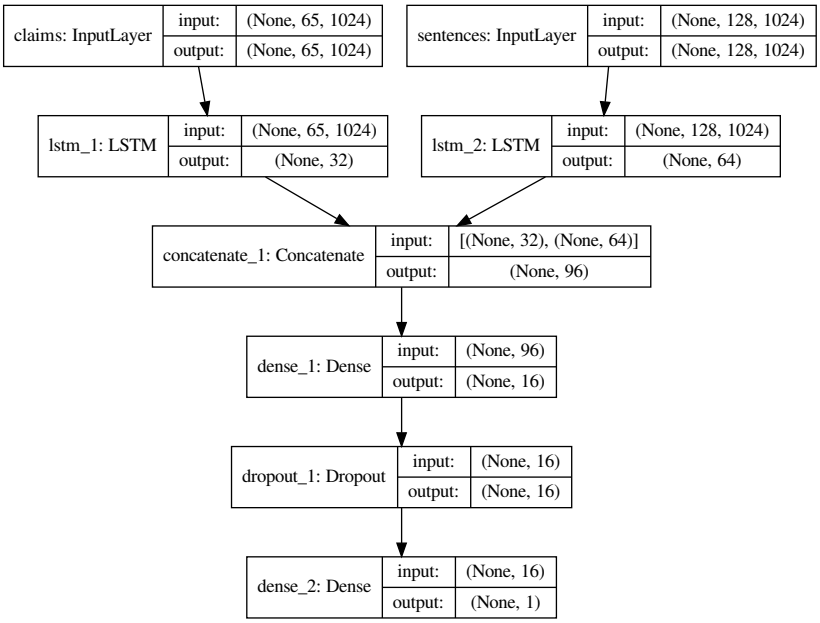


Figure 4.5: LSTM Architecture trained on FEVER Support set. Here we use embeddings of 1024-dimensions. Max length of claims and sentences is considered as 65 and 128. We selected these lengths based on the performance on the validation set. We consider separate LSTM units for claims and sentences. Later on, both the LSTM units are concatenated and fed to the Fully connected layer (Dense) followed by the dropout and final output layer.

Note: We do not explain the training process for evidence retrieval and claim labeling for this method as it is similar to the previous deep learning approach.

4.1.6 Simple LSTM Network using BERT Embeddings

Another method that we consider for sentence selection, and claim labeling is simple LSTM network using BERT embeddings. As discussed in section 2.5.3, the difference between BERT and ELMo is that BERT encodes the words that are jointly dependent on the left and right context. This is done by incorporating the Transformer network (explained in section 2.4) and eliminating the RNNs. To use it for sentence retrieval, first claims and evidences are fed to the embedding layer for transformation

in vectors form. To obtain the pre-trained BERT embeddings we use open-source implementation [2], these embeddings are based on $BERT_{Base}$ model. Due to BERT's internal architecture, we concatenate the embeddings of a claim and each sentence as shown in figure 2.20. This combined representation is fed to the LSTM units, which is followed by fully connected and the final dense layer. Figure 4.6 shows the network that is trained on FEVER support dataset. Here, we consider maximum sequence length as 256 and each word is represented with 768 dimensions. The maximum sequence length is selected based on the average number of words present in combined claim and sentence. Similar to the previous approaches, we train the two individual models for evidence selection and claim classifier. So we select "Binary cross-entropy" and "Categorical cross-entropy" losses and "Adam optimizer" to train both models.

[2]https://github.com/hanxiao/bert-as-service

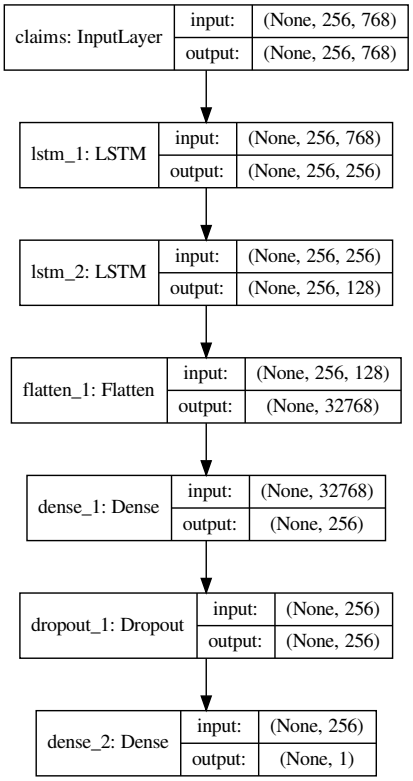


Figure 4.6: LSTM Architecture to train LSTM networks for Sentence Retrieval on FEVER Support dataset. Here we combine embeddings of a claim and each evidence. Each word is encoded using 768 dimensions, and the maximum length of each sequence is 256. "None" represents that the corresponding layer can accept any number of samples as the input.

4.2 Meta Classifier

4.2.1 Majority Vote Classifier

Majority vote classifier is a way of combining the results from the claim classifier module to determine the single label for each claim. Consider we have a single claim (C), and there are m related sentences retrieved for each claim. These m evidences are fed to the claim labeling module. This component outputs m labels for the claim.

So, to get the final label, we consider majority vote approach. For suppose, if m=5 and labels assigned by claim classifier are: [S, R, S, S, NEI]. Based on the frequency of labels, we assign final label as "Supports". However, if we have this case: [S, R, S, NEI, R], then we randomly choose either "Support", or "Refute". Following table 4.1 shows the few cases based on which classifier assigns the final label. We give preference to S/R in 2nd use case because we assume that the sentence retrieval module returns sentences related to the claim. This means, there is less probability that claim classification module outputs "NEI".

Use Cases	Final Label
[S, S, R, R, NEI]	random(S,R)
[S, R, NEI, NEI, NEI]	random(S,R)
[NEI, NEI, NEI, NEI, NEI]	NEI
[S, S, S, R, NEI]	S
[R, R, R, S, S]	R

Table 4.1: Majority Vote Classifier.

4.3 Integration with DeFactoNLP Fact Validation Pipeline

As we already discussed, fact validation is consists of document retrieval, sentence selection, and the claim classification. For this work, we are focusing on sentence selection and claim labeling modules. So, we use the document retrieval component from DeFactoNLP on FEVER original dataset. In the following section, we explain the integration of all the components to obtain the final label for each claim.

4.3.1 Document Retrieval

To retrieve the related documents, authors consider two methods: (1) TF-IDF vectors and (2) Named Entity Recognition (NER). To determine the relevant docs using TF-IDF vectors, initially claim and documents are transformed into vectors, and top k-docs are fetched based on the cosine similarity scores. On the other hand, to acquire the closely related documents to the claim, first entity is detected from the claim and after that cosine similarity between the entity and the document title is computed. Next, top n-docs are returned based on the similarity scores. In the end,

results obtained using both the methods are merged and used as the input of sentence retrieval module.

4.3.2 Sentence Retrieval

This module scans all the retrieved documents and fetches the top m relevant sentences to the claim. To do this, we compute similarity between claim (C) and each sentence (s_i) where i = [1,2,...,m] using six approaches that are explained in section 4.1. All the methods assign label to each sentence (s_i) as the "related" and "not related". Sentences that are "related" to the claim are fed to the claim labeling module. However, if there is no "related" sentence to the claim, "NEI" is assigned as the final label.

4.3.3 Claim Classification

The task of this module is to take m sentences selected by the previous component as input and assign a label to each sentence. Like evidence retrieval, here we also employ six different approaches to classify the relation between claim and each sentence. Now, this module assign m labels to m sentences. These labels are either "Support", "Refute", or "NEI". These results are supplied to the meta classifier to output the single final label.

4.3.4 Meta Classifier

The task of this module is to combine all the evidence labels and produce a single label. For each possible label ("Support", "Refute", or "NEI"), associated evidences are collected and based on the majority voting among labels, final label along with the respective evidences are considered as final output related to the claim.

5

Experiments and Evaluation

In this chapter, we present the datasets and results of all the approaches that we employed during this work. During our project, we considered two datasets: FEVER original and simple claims (which was further divided into 4 subsets). FEVER simple claims is extracted from FEVER original by Chawla et al. [46]. To evaluate the approaches metrics such as precision, recall, f1-score, label accuracy, and the FEVER score are used. In the end, we integrate and replace the best sentence retrieval and claim classification components in the DeFactoNLP project and compare the results with SOTA approaches.

5.1 Datasets

5.1.1 FEVER Original

Fact Extraction and VERification (FEVER) is the first open-source large-scale dataset for fact extraction and verification against textual sources. It consists of 185,445 claims that are verified manually from the Wikipedia and classified as: "SUPPORTS", "REFUTES", and "NOT ENOUGH INFO (NEI)". All the claims were generated by human annotators and mutated in various ways. Furthermore, authentication of each claim was performed in a different annotation process by annotators who only knew the page number rather than the sentence from which the original claim was generated. This dataset also contains the evidence pages of each claim, so to get the higher FEVER score, the system should not only classify the claim correctly but also retrieve

the evidences related to the claim accurately. Table 5.1 shows the different datasets that were introduced for FEVER shared task. FEVER-Train is used for the training of the model, while for testing FEVER-Development (Dev) set was considered. On the other hand, FEVER Blind set is used in FEVER shared task to benchmark all SOTA approaches.

Dataset	Label	No. of examples
FEVER-Train	Support	3000
	Refute	3000
	NEI	4000
FEVER-Dev	Support	6666
	Refute	6666
	NEI	6666
FEVER Blind	Support	6666
	Refute	6666
	NEI	6666

Table 5.1: FEVER original datasets.

5.1.2 FEVER Simple Claims

This dataset is created from FEVER original dataset. The main idea is to test the models for a different combination of classes such as (Support vs Refute/Not Enough Info, Refute vs Support/Not Enough, and so on). In this dataset, all the claims contain only one evidence. This means, there is no need for training the evidence retrieval module as dataset already contains the relevant sentence to the claim. Further, all the claims are represented in a subject-predicate-object form. Snapshot of this dataset is shown in figure 5.1. This dataset is split into four sets that are discussed below. Table 5.2 illustrate the number of examples in each split.

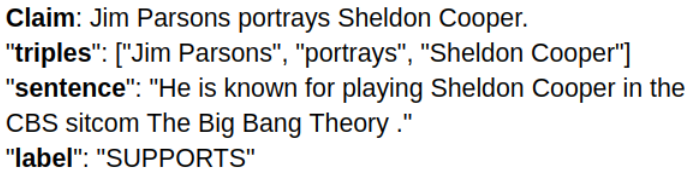


Figure 5.1: A Sample is taken from FEVER simple claims dataset. We can see that evidences related to the claim are already retrieved. So, we directly predict the label using claim classification module [46].

FEVER Support

This set contains claims having true labels as "SUPPORTS" and "Not Enough Info". Here "Not Enough Info" contain evidences that are refuting as well as containing no information related to the claim. To combat the problem of imbalanced classes, we consider the equal number of "SUPPORTS" and "Not Enough Info" examples.

FEVER Reject

This dataset considers claims whose true label is "REFUTES" and "Not Enough Info". Similarly, in this dataset, "Not Enough Info" consider evidences that are either supporting or does not contain any information to the input claim.

FEVER Binary

In this dataset, we group examples having label "SUPPORTS" and "REFUTES" in one class called as "RELATED", and "Not Enough Info" in another class called as "NOT-RELATED".

FEVER Multi-Class

In this set, we consider examples having "SUPPORTS", "REFUTES", and "Not Enough Info" labels. This is done to determine how the performance of the classifier varies with more number of classes.

5.2 Evaluation Metrics

We use five metrics to evaluate the performance of all six approaches. In case of FEVER simple claims set, we use labels to compute the precision, recall, and F1

Dataset	Label	No. of examples
FEVER Support	Support	2761
	NEI	2761
FEVER Reject	Reject	2955
	NEI	2955
FEVER Binary	Related	5608
	Non-Related	2804
FEVER Multi-Class	Support	2761
	Reject	2847
	NEI	2804

Table 5.2: FEVER simple claims datasets.

scores. However, for FEVER original set, all the metrics are calculated based on the correct evidences retrieved.

Precision

It is computed as the ratio of positive examples predicted correctly to the sum of examples predicted as positive. It is given as[1]:

$$precision = \frac{TP}{TP + FP} \tag{5.1}$$

where TP = true positive and FP = false positive

Recall

It is computed as the ratio of positive examples predicted correctly to the sum of positive examples predicted correctly and examples predicted as negative. It is given as[2]:

$$Recall = \frac{TP}{TP + FN} \tag{5.2}$$

where TP = true positive and FN = false negative

[1] `https://en.wikipedia.org/wiki/Precision_and_recall`
[2] 1

F1 Score

F1 score is computed as the harmonic mean between precision and recall. The higher the F1 score, the better the model[3]. It is given as:

$$F1 = 2.(\frac{precision * recall}{precision + recall}) \tag{5.3}$$

Label Accuracy

It is computed as the ratio of the number of samples that are predicted correctly to the total number of predicted samples by the system. It can be written as[4]:

$$accuracy = \frac{N}{M} \tag{5.4}$$

N = No. of samples predicted correctly and M = total no. of samples

FEVER Score

This score is especially proposed for FEVER shared task. For a claim and set of predicted evidences, authors [59] assign FEVER score based on two conditions:

- Determine if the predicted label of the claim is correct.
- If the predicted label is accurate, then count score is assigned, if an entire group of predicted evidences is present in the true evidences.

  Note: Authors [59] consider top 5 retrieved evidences and ignore remaining without any penalty.

5.3 Experiments

In this section, we discuss the experiments that are conducted on FEVER simple claims as well as FEVER original dataset. Later on, we also present the results using all the approaches on the aforementioned datasets.

[3]https://en.wikipedia.org/wiki/F1_score

[4]`https://developers.google.com/machine-learning/crash-course/classification/accuracy`

5.3.1 Experiments on FEVER Simple Claims dataset

As we discussed, this dataset is divided into four subsets as shown in table 5.2. During training, we split the dataset into the train, validation, and the test set. Since these datasets already contain related evidence for each claim, therefore, we only train and test the claim classification module. We use all six approaches (discussed in chapter 4) on this dataset. To train the deep learning model, we use Keras [5] library and NVIDIA Tesla V100 GPU on our university cluster [6]. While, to train the classical approaches, manually thresholds were tuned on the validation set. We do not compute FEVER Score for this dataset as we already have related evidences to the claim. However, for FEVER full dataset, we compute FEVER score as the task is also focused on evidence retrieval as well as classification of the claims. In addition to this, for training, validating, and testing the classical and deep learning approaches, we divide the data in the ratio of 60%, 20%, 20% and 80%, 10%, and 10% respectively. Since deep learning approaches require more data, therefore, 80% of the data is used to train the models. In the coming section, we show the results of different approaches on FEVER simple claims dataset.

Results

Table 5.3 illustrates the performance of all the methods considered in this work, as well as the SOTA approaches. From the table, it is noticeable that deep learning approaches have better accuracy as compare to the classical (Vector space, TF-IDF, and WMD) and feature-based approaches (XGBoost and XI-FEATURE MLP). This is because classical approaches assign labels based on the thresholds rather than predicting them by using learning algorithms. Among all the methods, Simple LSTM using BERT embedding outperforms on all FEVER simple claims datasets. This shows how efficient and meaningful are the representations of texts generated using BERT. Furthermore, it can be noticed that the performance of all methods decreases when the task is to solve the multi-class (3-class) classification task. This is due to the confusion between the classes as shown in confusion matrices figures (5.2, 5.3, and 5.4). We can notice that WMD is unable to differentiate between the "NEI" class, as

[5] https://github.com/keras-team/keras
[6] https://wr0.wr.inf.h-brs.de

all the examples related to this class are misclassified as "Support" or "Refute". In overall, Simple LSTM using BERT outperforms all the approaches by 5%, 2%, and 18% on FEVER Support, Reject and Multi-class dataset. The table does not contain the results of SOTA approaches for FEVER-Binary set as previous approaches do not use it. Furthermore, we also compare the training time (in Minutes) of all the deep learning approaches on this dataset shown in table 5.4. It is noticed that the training time of Simple LSTM using BERT is way longer as compare to all other methods. This is because BERT uses Transformer network that contains more trainable parameters and is computationally expensive.

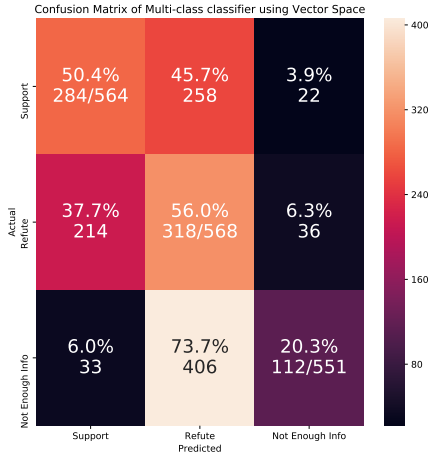
Dataset	Approach	Label Acc.	Precision	Recall	F1
Support	XGBoost [61]	0.766	0.766	0.766	0.762
	TE [23]	0.691	0.835	0.655	0.734
	XI-FEATURE MLP [46]	0.79	0.76	0.81	0.78
	SimpleLSTM [46]	0.850	0.834	0.856	0.845
	Vector Space	0.665	0.614	0.852	0.714
	TF-IDF	0.711	0.735	0.644	0.686
	WMD	0.665	0.490	1.000	0.658
	Simple LSTM (word2vec)	0.774	0.733	0.835	0.78
	Simple LSTM (ELMo)	0.835	0.801	0.876	0.837
	Simple LSTM (BERT)	**0.908**	**0.891**	**0.921**	**0.906**
Reject	XGBoost [61]	0.74	0.738	0.736	0.73
	TE [23]	0.548	0.759	0.533	0.626
	XI-FEATURE MLP [46]	0.74	0.69	0.78	0.73
	SimpleLSTM [46]	0.816	0.836	0.811	0.824
	Vector Space	0.683	0.645	0.795	0.712
	TF-IDF	0.704	0.779	0.559	0.651
	WMD	0.683	0.493	**1.000**	0.661
	Simple LSTM (word2vec)	0.796	0.768	0.819	0.793
	Simple LSTM (ELMo)	0.824	0.794	0.852	0.822
	Simple LSTM (BERT)	**0.842**	**0.882**	0.771	**0.823**
Binary	Vector Space	0.704	0.720	0.701	0.720

	TF-IDF	0.715	0.726	0.718	0.726
	WMD	0.442	0.325	0.167	0.725
	Simple LSTM (word2vec)	0.789	0.721	0.637	0.676
	Simple LSTM (ELMo)	0.815	0.712	0.761	0.844
	Simple LSTM (BERT)	**0.871**	**0.812**	**0.840**	**0.893**
Multi-class	XGBoost [61]	0.535	0.54	0.534	0.539
	TE [23]	0.418	0.372	0.622	0.465
	XI-FEATURE MLP [46]	0.59	0.61	0.62	0.61
	SimpleLSTM [46]	0.635	0.643	0.620	0.642
	Vector Space	0.424	0.504	0.424	0.414
	TF-IDF	0.494	0.527	0.495	0.470
	WMD	0.424	0.201	0.331	0.191
	Simple LSTM (word2vec)	0.606	0.661	0.533	0.590
	Simple LSTM (ELMo)	0.613	0.652	0.521	0.575
	Simple LSTM (BERT)	**0.799**	**0.806**	**0.793**	**0.800**

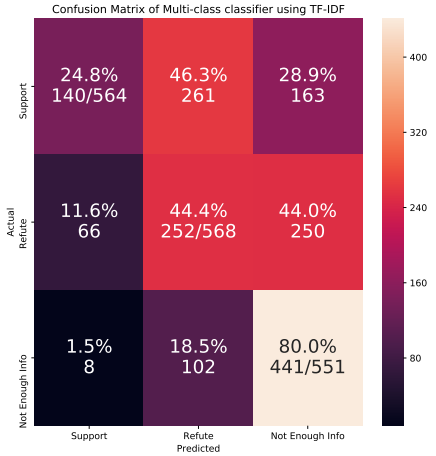
Table 5.3: Results on FEVER Simple Claims dataset. It is noticed that simple LSTM using BERT outperforms all the approaches.

	Training Time (In Minutes)			
Method	Support	Reject	Binary	Multi-class
Simple LSTM (word2vec)	3.03	4.1	5.71	5
Simple LSTM (ELMo)	5.4	5.75	7.02	7.2
Simple LSTM (BERT)	107	115	194	200

Table 5.4: Training time of Deep Learning approaches on FEVER Simple Claims dataset.

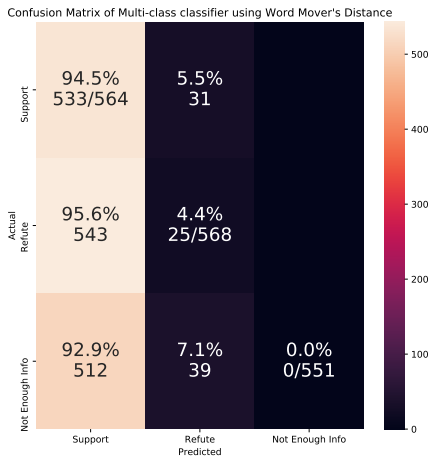


(a) Confusion Matrix on FEVER Multi-Class dataset using Vector Space

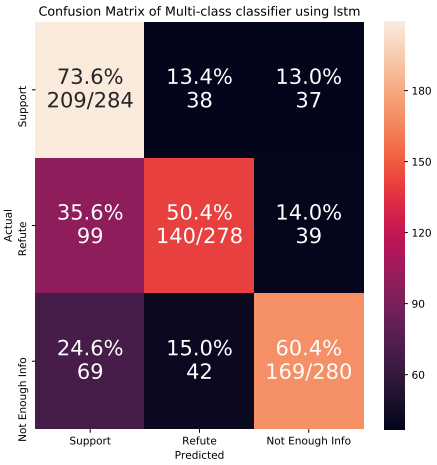


(b) Confusion Matrix on FEVER Multi-Class dataset using TF-IDF

Figure 5.2: Confusion matrices of Vector Space and TF-IDF.

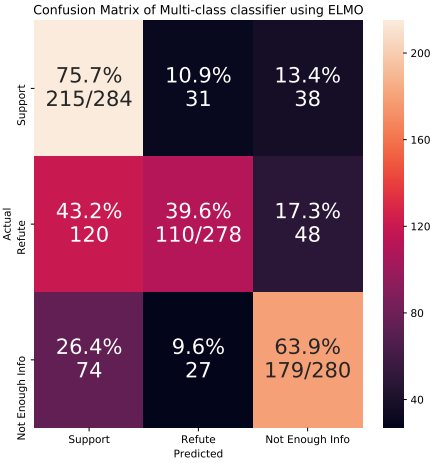


(a) Confusion Matrix on FEVER Multi-Class dataset using WMD

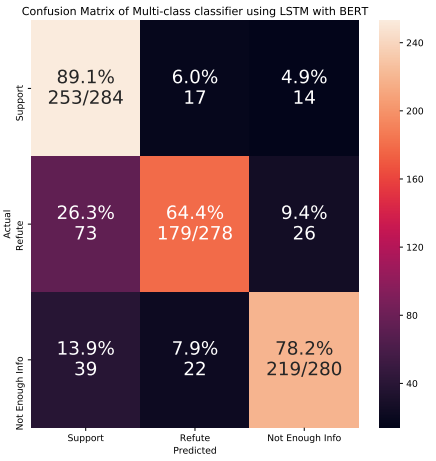


(b) Confusion Matrix on FEVER Multi-Class dataset using simple LSTM with word2vec

Figure 5.3: Confusion matrices of Word Mover's Distance (WMD) and Simple LSTM using word2vec.



(a) Confusion Matrix on FEVER Multi-Class dataset using simple LSTM with ELMo



(b) Confusion Matrix on FEVER Multi-Class dataset using simple LSTM with BERT

Figure 5.4: Confusion matrices of simple LSTM using ELMo and BERT.

5.3.2 Experiments on FEVER Original Dataset

As we have seen before, this dataset is consists of three subsets: FEVER Train, Dev., and Blind set. Train and Dev. sets are labeled so we train the sentence retrieval and claim classification models on Train set and evaluate the results on Dev. Furthermore, the dataset does not contain documents and evidences for "NEI" label, so to perform training for this class, we use document retrieval from DeFactoNLP on Train and Dev. set, to fetch closely related documents to the claim, and select random sentences from each retrieved document. To evaluate the performance of sentence retrieval as well as claim classifier individually, Dev. set is employed. Furthermore, the Blind set is also provided that is used to benchmark the approaches for FEVER shared task. It does not contain true labels of the claims. So to figure out the performance on Blind set, we used document retrieval (from DeFactoNLP), sentence retrieval, and claim classification to determine the final label of the claim. These results are submitted to FEVER CodaLab [7] to get the final scores. In addition to this,

[7] `https://competitions.codalab.org/competitions/18814`

we also conduct an ablation study, where sentence retrieval and claim classification are individually tested on Dev. set. Furthermore, to learn the parameters for the classical approaches, cut-off thresholds were manually tuned on the validation set. While, for training the deep learning approaches, we employed Keras library and GPUs (Nvidia Tesla K80 and V100) on our university cluster. We trained two models for each approach: one for sentence retrieval and second for claim classification. Both the models were trained independently and based on the assumption that previous components are 100% accurate. In the following section, we report the results of the complete pipeline as well as of evidence retrieval and claim classification modules individually on FEVER Dev. and Blind set.

Results

Tables 5.5 and 5.6 depicts the performance of all the methods on FEVER Dev. and Blind set. On FEVER Dev. set, it is observed that Simple LSTM using BERT embeddings has the best accuracy and FEVER score. While vector space is second among other methods based on label accuracy and FEVER score. Figure 5.5 shows the comparison of different approaches on FEVER Dev set. Here, we also conducted an ablation study of sentence retrieval and claim classification models. We noticed that all the approaches have better accuracy on the individual components. However, when we combine the independently trained models for sentence retrieval and claim classification, overall accuracy significantly drops. This is because each module is dependent on previous modules, so if sentence retrieval produces false positives, this affects the performance of claim classification and the overall pipeline.

Furthermore, on FEVER Blind set, we integrated document retrieval from DeFactoNLP with the sentence retrieval, and claim classification modules. As a result, we observed that all the sentence retrieval and claim classification approaches have a quite similar label accuracy, but it is clear that classical approaches, as well as simple LSTM using word2vec, have lowest evidence precision as compare to simple LSTM using ELMo and BERT. This is due to the prediction of the majority of false-positive evidences, and most of the claims are misclassified by these approaches to "NEI" class. In addition to this, overall label accuracy and FEVER score are not better as compare to the SOTA approaches, this is because every unit in the pipeline depends on the previous module, so the prediction of false-positives by any component degrades the

performance of the current unit. Moreover, it is also noticed that with the integration of document retrieval on FEVER Blind set, overall label accuracy and Evidence F1 score drops by 8% and approximately 20% respectively as compared to the Dev. set. This is due to the additional noise introduced by the document retrieval.

Table 5.7 illustrates the comparison of DeFactoNLP with the SOTA methods. UNC-NLP [39] has better label accuracy and FEVER score as compared to other approaches. Furthermore, it is observed that, after integration of WMD method with DeFactoNLP, its FEVER score is better as compared to the University of Arizona [59] obtaining FEVER score of 26.7%, though its Evidence F1 score is very low. On the other hand, integration of BERT with DeFactoNLP obtains better evidence F1 score, lower label accuracy, and FEVER score as compared to the FEVER Baseline [60] and University of Arizona.

Approach	**Label Accuracy**	**FEVER Score**	**Evidence Precision**	**Evidence Recall**	**Evidence F1**
Vector Space	0.419	0.235	0.318	0.44	0.369
TF-IDF	0.268	0.151	0.284	0.301	0.292
WMD	0.123	0.085	0.241	**0.653**	0.352
Simple LSTM (word2vec)	0.368	0.169	0.277	0.506	0.358
Simple LSTM (ELMo)	0.324	0.154	0.32	0.537	0.4
Simple LSTM (BERT)	**0.43**	**0.238**	**0.35**	0.441	**0.39**

Table 5.5: Results on FEVER Dev. set. This considers only evidence retrieval and claim classification. We do not use document retrieval, as true documents are already present in the dataset.

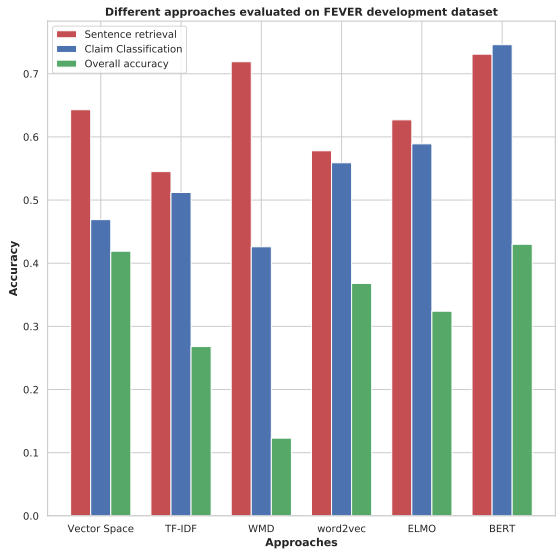


Figure 5.5: Comparison on FEVER Dev. set. Sentence retrieval and claim classification models were evaluated individually on FEVER Dev. set. Overall accuracy is obtained based on combination of sentence retrieval and claim classification trained models.

Approach	**Label Accuracy**	**FEVER Score**	**Evidence Precision**	**Evidence Recall**	**Evidence F1**
Vector Space	0.341	0.087	0.04	0.125	0.06
TF-IDF	0.345	0.159	0.098	0.112	0.105
WMD	0.339	**0.267**	0.033	0.129	0.053
Simple LSTM (word2vec)	0.332	0.072	0.048	0.155	0.073
Simple LSTM (ELMo)	0.328	0.115	**0.205**	0.150	0.173
Simple LSTM (BERT)	**0.357**	0.173	0.144	**0.383**	**0.209**

Table 5.6: Results on FEVER Blind set. This includes document retrieval from DeFactoNLP, evidence retrieval, and claim classification from this project.

Team Name	**Evidence (%)**			**Label**	**FEVER**
	Precision	**Recall**	**F1**	**Acc.**	**Score**
UNC-NLP	42.27	70.91	52.96	**68.21**	**64.21**
UCL	22.16	82.84	34.97	67.62	62.52
Athene UKP.	23.61	85.19	36.97	65.46	61.58
Papelo	**92.18**	50.02	**64.85**	61.08	57.36
SWEEPer	18.48	75.39	29.69	59.72	49.94
Columbia NLP	23.02	75.89	35.33	57.45	49.06
Ohio State Uni.	77.23	47.12	58.53	50.12	43.42
GESIS Cologne	12.09	51.69	19.60	54.15	40.77
FujiXerox	11.37	29.99	16.49	47.13	38.81
withdrawn	46.60	51.94	49.12	51.25	38.59
Uni-DuE	50.65	36.02	42.10	50.02	38.50
DeFactoNLP	51.91	36.36	42.77	51.36	38.33
withdrawn	12.90	54.58	20.87	53.97	37.13
Py.ro	21.15	49.38	29.62	43.48	36.58
SIRIUS-LTG.	19.19	70.82	30.19	48.87	36.55
withdrawn	0.00	0.01	0.00	33.45	30.20
BUPT-NLPer	45.18	35.45	39.73	45.37	29.22
withdrawn	23.75	**86.07**	37.22	33.33	28.67
withdrawn	7.69	32.11	12.41	50.80	28.40
FEVER Baseline	11.28	47.87	18.26	48.84	27.45
DeFactoNLP with WMD (ours)	3.3	12.99	5.3	33.9	26.7
withdrawn	49.01	29.66	36.95	44.89	23.76
withdrawn	26.81	12.08	16.65	57.32	22.89
withdrawn	26.33	12.2	16.68	55.42	21.71
University of Arizona	11.28	47.87	18.26	36.94	19.00
DeFactoNLP with BERT (ours)	14.4	38.3	20.9	35.7	17.3

Table 5.7: SOTA results on FEVER Blind set. DeFactoNLP with WMD and BERT considers document retrieval from DeFactoNLP, sentence retrieval, and claim classification from this work.

Ablation Study on Fever Development Set

The main aim of conducting ablation study is to investigate how good approaches can perform on individual tasks such as sentence/evidence retrieval and claim classification. To perform this experiment, we consider FEVER Dev. set, as we have ground truth of all related evidences as well as labels of the claims. Further, to determine the performance of sentence retrieval, we consider that document retrieval is 100% correct and module predicts whether a sentence is relevant or irrelevant to the claim. Similarly, to evaluate the results of claim classification individually, it is assumed that document and sentence retrieval are 100% accurate.

Tables 5.8 and 5.9 depict the performance of sentence retrieval and claim classifier. From tables, we can observe that outcome of classical methods such as TF-IDF, vector space, and word mover's distance is better on sentence retrieval as compared to claim classification in terms of label accuracy and F1 score. On the other hand, the performance of deep learning approaches on both tasks is approximately similar. In addition to this, performance of all the methods degrades for claim classification as compare to sentence retrieval, except simple LSTM using BERT embeddings. Furthermore, it can also be seen that simple LSTM using BERT has outperformed all the approaches in both the tasks.

Approach	Label Accuracy	Precision	Recall	F1 Score
Vector Space	0.643	0.268	0.643	0.328
TF-IDF	0.545	0.305	0.545	0.339
WMD	0.719	0.168	**0.719**	0.247
Simple LSTM (word2vec)	0.578	0.583	0.625	0.603
Simple LSTM (ELMo)	0.627	0.629	0.664	0.646
Simple LSTM (BERT)	**0.731**	**0.778**	0.667	**0.718**

Table 5.8: Ablation study of sentence retrieval on FEVER Dev. set.

Approach	Label Accuracy	Precision	Recall	F1 Score
Vector Space	0.469	0.534	0.469	0.471
TF-IDF	0.512	0.549	0.511	0.471
WMD	0.426	0.305	0.426	0.354
Simple LSTM (word2vec)	0.559	0.588	0.469	0.503
Simple LSTM (ELMo)	0.564	0.364	0.42	0.572
Simple LSTM (BERT)	**0.746**	**0.759**	**0.731**	**0.739**

Table 5.9: Ablation study of claim classifier on FEVER Dev. set.

5.4 Discussion

Based on the results, we can deduce that the complexity of datasets certainly affects the performance of the approaches. Like, FEVER simple claims dataset consists of simple claims along with their evidences, which do not exceed more than one sentence. It is also limited to the claim classification only. On the other hand, FEVER original dataset contains complicated claims and the evidences, the task is to fetch the related document, relevant sentences, and labeling of the claim. So each unit in the pipeline is dependent on the preceding component. Furthermore, we observed that Simple LSTM using BERT outperformed the SOTA with the accuracy of 90.8%, 84.2%, and 80% on FEVER simple claims (Support, Reject and Multi-class dataset respectively). On the other hand, the performance of all the approaches on FEVER Dev. and Blind set is not better than SOTA due to the following possible reasons:

- Most of the time, retrieving potentially related sentences to the claim is hard. This is because most of the entities present in the claims are represented by the pronouns in the evidences, leading to the coreference resolution task. Although we could have used the coreference resolution system, it may also introduce errors.

- The difference in the length of claims and the evidences, which is up to 40 to 50 and around 100 to 150 words respectively.

- The overall accuracy of the model drops significantly when individually trained models are combined due to noise introduced by previous components.

- Document retrieval is not robust, as authors in DeFactoNLP observed that even for simple claims such as *"Happiness in Slavery is a gospel song by Nine Inch Nails."*[60] which does not contain any documents in labeled dataset, it returns false documents, as a result, sentence retrieval and claim classifier also finds sentences related to the claim and assigns "Supports" label. This shows noise produced by any component propagates to the complete pipeline.

- As dataset is created from the Wikipedia without any pre-processing, thus it contains noisy information i.e.repetitive words. This may degrade the performance of the models. One example from the dataset is shown below:

  - e.g "She is known for her role as Donna Pinciotti in all eight seasons of the Fox sitcom That '70s Show (1998 – 2006) , and for her role as Alex Vause in the Netflix original comedy-drama series Orange Is the New Black (2013 – present). Orange Is the New Black Orange Is the New Black Donna Pinciotti Donna Pinciotti Fox Fox Broadcasting Company That '70s Show That '70s Show Alex Vause Alex Vause Netflix Netflix" [60]

6

Conclusions

In this theses, we evaluated and implemented the different SOTA approaches for sentence/evidence retrieval and claim classification (code available in github [1]). We introduced theoretical concepts that are required for this work. We also illustrated how recently introduced embeddings such as ELMo and BERT can be used in the domain of fact validation. To perform this work, we chose two datasets: FEVER simple claims and FEVER original set. FEVER simple claims consist of simple claims and the task is to solve the claim classification problem only. Whereas, on FEVER original set, the problem is to retrieve the relevant document, fetch related evidence and determine the label of the claims. For FEVER simple claim set, we trained a single model for claim classification for each approach. On the other hand, for FEVER original dataset, we trained separate models for evidence retrieval and claim classification for all the approaches. Then, we combined the labels for each potential evidence using majority label classifier to determine the single label for the claim.

On FEVER simple claims dataset, we observed that Simple LSTM using BERT embeddings outperformed the SOTA. In general, all the deep learning methods have beaten the classical (TF-IDF, vector space, and WMD) as well as feature-based approaches. From FEVER original set, we used FEVER Dev. to determine the performance of sentence retrieval and claim classification individually. Moreover, an ablation study was conducted based on the assumption that preceding components in the pipeline are 100% correct, such as for sentence retrieval it is assumed that document retrieval is always accurate and for claim classification, document and

[1] `https://github.com/DeFacto/EvidenceRetrieval-ClaimClassification`

sentence retrieval are 100% perfect. The results of ablation study showed that simple LSTM using BERT surpassed all the approaches on sentence retrieval and claim classification.

Apart from ablation study on FEVER Dev. set, we also tested the sentence retrieval and claim classifier combined, and observed that simple LSTM using BERT has better label accuracy and FEVER score as compared to other methods. However, it is also noticed that, when two trained models (sentence retrieval and claim classification) are combined sequentially, overall label accuracy for all the approaches drops significantly. This is because of false positives predicted by the 1^{st} model (sentence retrieval) degrades the performance of the 2^{nd} model (claim classification).

We also employed FEVER blind set where the task was to fetch relevant documents, related evidence, and determine the label of the claims. We used document retrieval components from DeFactoNLP project, evidence retrieval, and the claim classification from this work. As a result, we noticed that all the approaches that were implemented in this work have lower label accuracy and FEVER score as compare to the SOTA. This is because combining different components is always challenging, the noise produced by single component affects the remaining also, and as a consequence overall accuracy is reduced.

7

Future Directions

Several ways to improve the task of sentence retrieval and claim classification for future are discussed in this section:

New Language Models

Recently presented methods enhance the BERT, either based on overall performance or inference speed. Table 7.1 shows the comparison of BERT with recently introduced systems based on the architecture size, training time, overall performance, amount of data used for training, and method of training employed. RoBERTa [70] is introduced by Facebook that considers more data and improved training strategy. Apart from dynamic masking training procedure, it uses 10 times more data as compare to BERT. As a result, it outperforms BERT by 2-20%. XLNET [72] is another language model presented to enhance the performance of BERT, which uses more computational power as well as training data. In addition to this, unlike Masking training strategy in BERT, it uses the permutation-based language modeling, where words/tokens in the sequences are predicted in random order. DistilBERT [65] is a lighter and compressed version of BERT, that uses knowledge distillation technique in which a smaller or lighter model is trained to mimic the performance of the original BERT model. Although it uses lesser parameters as compared to the original BERT base model, its performance reduces by 5% from BERT.

	BERT	RoBERTa	DistilBERT	XLNET
Size (millions)	**Base**: 110 **Large**: 340	**Base**: 110 **Large**: 340	**Base**: 66	**Base**: ~110 **Large**: ~340
Training Time	**Base**: 8 x V100 x 12 days* **Large:** 64 TPU Chips x 4 days (or 280 x V100 x 1 day*)	**Large**: 1024 x V100 x 1 day; 4-5 times more than BERT	**Base**: 8 x V100 x 3.5 days; 4 times less than BERT	**Large**: 512 TPU Chips x 2.5 days; 5 times more than BERT
Performance	Outperforms SOTA in Oct 2018	2-20% improvement over BERT	5% degradation from BERT	2-15% improvement over BERT
Data	16 GB BERT data (Books Corpus + Wikipedia), 3.3 Billion words	160 GB (16 GB BERT data + 144 GB additional)	16 GB BERT data. 3.3 Billion words	**Base**: 16 GB BERT data **Large**: 113 GB (16 GB BERT data + 97 GB additional). 33 Billion words
Method	BERT (Bidirectional Transformer with MLM and NSP)	BERT without NSP**	BERT Distillation	Bidirectional Transformer with Permutation based modeling

Table 7.1: Comparison among BERT and recently introduced approaches. * represents estimated time on GPU. ** specifies method employs larger learning rates, mini-batches, and step sizes along with different masking process [57].

End-to-End Models

In this work, we trained the separate models for sentence retrieval as well as claim classification, and observed that overall accuracy significantly drops, when both the models are combined. The performance on this task can be improved by training the end-to-end models for all three components: document retrieval, sentence retrieval, and claim classification.

Apart from this, instead of relying on knowledge obtained from single source i.e. Wikipedia, we can also use Commonsense knowledge database[1] or knowledge graphs to determine the veracity of the claims. This can also enhance the overall performance of the fact validation systems.

[1] https://en.m.wikipedia.org/wiki/Commonsense_knowledge_(artificial_intelligence)

References

[1] Recurrent neural network and lstm. https://bit.ly/2DP6gG2. accessed: 2019-May-12.

[2] Understanding categorical cross entropy loss and binary cross-entropy loss. https://bit.ly/2Kirr82. accessed: 2019-Jun-13.

[3] Reuters dataset. https://bit.ly/2kyynm1. accessed: 2018-Oct-27.

[4] TF-IDF advantages and disadvantages. http://www.tfidf.com. accessed: 2018-Dec-20.

[5] Ajit Rajasekharan. Brief review of word embedding families. https://bit.ly/2kB079v. accessed: 2019-Jun-9.

[6] Tim Salimans Ilya Sutskever Alec Radford, Karthik Narasimhan. Improving language understanding by generative pre-training. 2018.

[7] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness. accessed: 2019-May-26.

[8] Anish Singh Walia. Types of optimization algorithms used in neural networks and ways to optimize gradient descent. https://bit.ly/2pbRVLk. accessed: 2019-Jun-20.

[9] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014. cite arxiv:1409.0473.

[10] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003. ISSN 1532-4435.

[11] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 120–128, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics. ISBN 1-932432-73-6.

[12] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. ISSN 2307-387X.

[13] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2015.

[14] Tuhin Chakrabarty, Tariq Alhindi, and Smaranda Muresan. Robust document retrieval and individual evidence modeling for fact extraction and verification. In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*, pages 127–131. Association for Computational Linguistics, 2018.

[15] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. *CoRR*, 2017.

[16] Minmin Chen, Zhixiang Xu, Kilian Q. Weinberger, and Fei Sha. Marginalized denoising autoencoders for domain adaptation. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, pages 1627–1634, USA, 2012. Omnipress. ISBN 978-1-4503-1285-1.

[17] Qian Chen, Xiao-Dan Zhu, Zhen-Hua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced lstm for natural language inference. In *ACL*, 2017.

[18] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 670–680. Association for Computational Linguistics, 2017. doi: 10.18653/v1/D17-1070.

[19] D. Pomerleau and D. Rao. Fake news challenge 2017. http://www.fakenewschallenge.org. accessed: 2018-Nov-11.

[20] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, 2018.

[22] Edward Ma. Word distance between word embeddings. https://bit.ly/2KEaaWQ. accessed: 2018-Jun-26.

[23] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. Allennlp: A deep semantic natural language processing platform. 2017.

[24] Andreas Hanselowski, Hao Zhang, Zile Li, Daniil Sorokin, Benjamin Schiller, Claudia Schulz, and Iryna Gurevych. Ukp-athene: Multi-sentence textual entailment for claim verification. *CoRR*, 2018.

[25] Simon S. Haykin. *Neural Networks and Learning Machines*. Pearson Education, Upper Saddle River, NJ, third edition, 2009.

[26] Christopher Hidey and Mona Diab. Team sweeper: Joint sentence extraction and fact checking with pointer networks. In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*, pages 150–155. Association for Computational Linguistics, 2018.

[27] Jay Alammar. Mechanics of seq2seq models with attention . https://bit.ly/2AzmocB, . accessed: 2019-Jun-6.

[28] Jay Alammar. Illustrated transformers. http://jalammar.github.io/illustrated-transformer, . accessed: 2019-Jul-22.

[29] Jiwon Jeong. The most intuitive and easiest guide for recurrent neural network. https://bit.ly/30U6RQk. accessed: 2019-May-29.

[30] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 957–966. JMLR.org, 2015.

[31] Lilian Weng . Attention attention! `https://bit.ly/2JTY9Lz`. accessed: 2019-Jul-18.

[32] Christopher Malon. Team papelo: Transformer networks at fever. In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*, pages 109–113. Association for Computational Linguistics, 2018.

[33] Marcelo Viana. Loss functions. `https://bit.ly/2yo0dnY`. accessed: 2019-Jun-13.

[34] Maxime Allard. An introduction to transformers and sequence-to-sequence learning for machine learning. `https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04`. accessed: 2019-Jun-6.

[35] Michael Nguyen. Illustrated guide to lstm's and gru's. `https://bit.ly/2Ieio3V`. accessed: 2019-May-29.

[36] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, 2013.

[37] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held Dec. 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 3111–3119, 2013.

[38] Mahesh Chandra Mukkamala and Matthias Hein. Variants of rmsprop and adagrad with logarithmic regret bounds. *CoRR*, 2017.

[39] Yixin Nie, Haonan Chen, and Mohit Bansal. Combining fact extraction and verification with neural semantic matching networks. *CoRR*, 2018.

[40] Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2249–2255. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1244.

[41] Jeff Pasternack and Dan Roth. Generalized fact-finding. In *Proceedings of the 20th International Conference Companion on World Wide Web*, WWW '11, pages 99–100, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0637-9. doi: 10.1145/1963192.1963243.

[42] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *In EMNLP*, 2014.

[43] Alessandro Perina, Nebojsa Jojic, Manuele Bicego, and Andrzej Truski. Documents as multiple overlapping windows into grids of counts. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 10–18. Curran Associates, Inc., 2013.

[44] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237. Association for Computational Linguistics, 2018. doi: 10.18653/v1/N18-1202.

[45] Matthew E. Peters, Waleed Ammar, Chandra Bhagavatula, and Russell Power. Semi-supervised sequence tagging with bidirectional language models. *CoRR*, 2017.

[46] Karthik Pujar Piyush Chawla, Diego Esteves and Jens Lehmann. Simplelstm: A deep-learning approach to claim-classification. EPIA 2019, 2019.

[47] Kashyap Popat, Subhabrata Mukherjee, Jannik Strötgen, and Gerhard Weikum. Credibility assessment of textual claims on the web. In *Proceedings of the 25th*

ACM International on Conference on Information and Knowledge Management, CIKM '16, pages 2173–2178, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4073-1. doi: 10.1145/2983323.2983661.

[48] Kashyap Popat, Subhabrata Mukherjee, Jannik Strötgen, and Gerhard Weikum. Where the truth lies: explaining the credibility of emerging claims on the web and social media. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, pages 1003–1012, Republic and Canton of Geneva, Switzerland, 2017. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4914-7. doi: 10.1145/3041021.3055133.

[49] Prasoon Goyal. Pros and cons of stochastic gradient descent. `https://bit.ly/2Yaqemv`. accessed: 2019-Jun-20.

[50] Lorien Pratt and Barbara Jennings. A survey of transfer between connectionist networks. *Connection Science*, 8(2):163–184, 1996.

[51] Aniketh Janardhan Reddy, Gil Rocha, and Diego Esteves. Defactonlp: Fact verification using entity recognition, tfidf vector comparison and decomposable attention. *CoRR*, 2018.

[52] Benjamin Riedel, Isabelle Augenstein, Georgios P. Spithourakis, and Sebastian Riedel. A simple but tough-to-beat baseline for the Fake News Challenge stance detection task. pages 1–6, 2017.

[53] S.E. Robertson, S. Walker, S. Jones, M.M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. pages 109–126, 1996.

[54] G. Salton. *The smart retrieval system-experiments in automatic document processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.

[55] N. J Sanders. Tsanders-twitter sentiment corpus. 2001.

[56] Sebastian Ruder. An overview of gradient descent optimization algorithms. `http://ruder.io/optimizing-gradient-descent`. accessed: 2019-Jun-20.

[57] Suleiman Khan. BERT, RoBERTa, DistilBERT, XLNet - which one to use? `https://bit.ly/2lJnSwf`. accessed: 2019-Sept-7.

[58] Suvro Banerjee. Unrolled recurrent neural network. https://bit.ly/2Ev55MD. accessed: 2019-May-12.

[59] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. The fact extraction and verification (fever) shared task. In *NAACL-HLT*, 2018.

[60] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. FEVER: A large-scale dataset for fact extraction and verification. In *NAACL-HLT*, 2018.

[61] Melanie Tosik, Antonio Mallia, and Kedar Gangopadhyay. Debunking fake news one feature at a time. *CoRR*, 2018.

[62] Traian Rebedea. TF-IDF. https://bit.ly/31dPmuq. accessed: 2018-Oct-27.

[63] Udeme Udofia. Basic overview of convolutional neural network. https://bit.ly/2PAyjR1. accessed: 2019-Apr-25.

[64] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, 2017.

[65] Victor Sanh. Introducing DistilBERT, a distilled version of BERT. https://medium.com/huggingface/distilbert-8cf3380435b5 . accessed: 2019-Jun-13.

[66] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2692–2700, Cambridge, MA, USA, 2015. MIT Press.

[67] Vitaly Bushaev. Adam - latest trends in deep learning optimization . https://bit.ly/2QuTmnR. accessed: 2019-Jun-24.

[68] Andreas Vlachos and Sebastian Riedel. Fact checking: task definition and dataset construction. In *LTCSS@ACL*, 2014.

[69] Xuezhi Wang, Cong Yu, Simon Baumgartner, and Flip Korn. Relevant document discovery for fact-checking articles. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, pages 525–533, Republic and Canton of Geneva, Switzerland, 2018. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-5640-4. doi: 10.1145/3184558.3188723.

[70] Naman Goyal Jingfei Du Mandar Joshi Danqi Chen Omer Levy Mike Lewis Luke Zettlemoyer Yinhan Liu, Myle Ott and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. 2019.

[71] Takuma Yoneda, Jeff Mitchell, Johannes Welbl, Pontus Stenetorp, and Sebastian Riedel. Ucl machine reading group: Four factor framework for fact finding (hexaf). In *Proceedings of the First Workshop on Fact Extraction and VERification (FEVER)*, pages 97–102. Association for Computational Linguistics, 2018.

[72] Yiming Yang Jaime G. Carbonell Ruslan Salakhutdinov Zhilin Yang, Zihang Dai and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. *CoRR*, 2019.

[73] Yukun Zhu, Ryan Kiros, Richard S. Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. *CoRR*, 2015.