Norwegian University
of Life Sciences

Master's Thesis 2021 30 ECTS
Faculty of Science and Technology

Mining Medical Academic Articles using Recurrent Neural Networks

Mohamed Radwan
Master of Science in Data Science

Acknowledgment

I would like to thank my supervisors Oliver Tomic and Kristian Hovde Liland for the great guidance, feedbacks and encouragements through this thesis.

2nd June, 2021

Mohamed Radwan

Abstract

In this thesis, we present our methods and results for mining the MedMentions data [Mohan and Li, 2019]. We propose a pipeline for combining mention classification and mention disambiguation. We will use the Long Short Term Memory (LSTM) neural network architecture for mention detection and mention classification. Also we use nearest neighbour search using embeddings in the Unified Medical Language System (UMLS) concepts in order to disambiguate mentions.

Our optimal results are achieved by combining five different models predictions using Cosine Similarity threshold. The Optimal model achieved micro F1 of 0.629 which is 1.1 micro F1 point behind State Of The Art (SOTA) study on this data by Fraser et al. [2019] that achieved 0.64.

The SOTA model by Fraser et al. [2019] is based on using BiLSTM with a concatenation of the last layer of both BioBERT [Lee et al., 2019] and BERT [Devlin et al., 2019] models that generate a combination of general and domain specific representations of the mentions in the data. Our method is based on using different BiLSTM networks with CODER [Yuan et al., 2020], SciBERT [Beltagy et al., 2019], UMLSBERT [Michalopoulos et al., 2021] and BioBERT [Lee et al., 2019] as feature encoders. We hoped that this could present a better overall features extraction given that the four pretrained models used different methodologies in their training. We balanced the obtained predictions from mention disambiguation against UMLS knowledge base with the different BiLSTM models into one prediction using Cosine Similarity threshold and plurality voting.

The first step in our pipeline is Mention Detection where we aim to extract mentions of interests from the free text. We used BiLSTM with pretrained BERT embeddings for this classification task. Second step is Mention Classification where we use the same BiLSTM architecture to predict the Semantic Types (STY) of the mentions in the text. The third step is Mention Disambiguation where we take the extracted mentions from the first step and disambiguate them against the UMLS knowledge base hoping to extract the nearest neighbour of the query mention that share the correct STY. Forth step is combining the Mention Classifications from second step and Mention Disambiguation from the third step to boost the results.

The reason for not achieving higher results than the SOTA model is attributed to the poor results that Mention Detection model achieved which made the nearest neighbour search prone to error. We believe there is a room for development in our implementation of nearest neighbour search method that could be able to further boost the performance.

Contents

1 Introduction **1**
1.1 Background . . . 1
1.2 Problem Statement . . . 2
1.3 Structure of thesis . . . 2

2 Theory **3**
2.1 Artificial Neural networks (ANNs) . . . 3
2.1.1 Activation functions . . . 6
2.1.2 Loss functions . . . 8
2.1.3 Optimizers . . . 10
2.1.4 Backpropgation of errors . . . 16
2.2 Deep Neural Networks . . . 18
2.3 Embeddings . . . 23
2.4 Attentions . . . 25
2.4.1 Scaled dot-product attention . . . 27
2.4.2 Multi-Head attention . . . 27
2.4.3 Transformer architecture . . . 28
2.4.4 Bidirectional Encoder Representations from Transformers 28

3 Data and Materials **33**
3.1 Raw Data . . . 33
3.1.1 Formatting Raw Data . . . 34
3.1.2 Baseline . . . 35

4 Methods **37**
4.1 Resources 37
4.2 Evaluation Metrics 38
4.3 Proposed Workflow 38
4.3.1 Mention Detection 39
4.3.2 Mention Classification 43
4.3.3 Mention Disambiguation 44
4.3.4 Ensemble of Predictions 45

5 Results **47**
5.1 Mention Detection 47
5.2 Mention Classification 48
5.2.1 Mention Classification using BiLSTM with CODER as feature encoder 48
5.2.2 Mention Classification using BiLSTM with SciBERT as feature encoder 50
5.2.3 Mention Classification using BiLSTM with UMLSBERT as feature encoder 52
5.2.4 Mention Classification using BiLSTM with BioBERT as feature encoder 52
5.3 Mention Disambiguation 55
5.4 Ensembles 55
5.5 Summary 57

6 Discussion **61**
6.1 Mention Detection 61
6.2 Mention Classification 63
6.3 Mention Disambiguation 63
6.4 Comparison with other studies 64

7 Conclusions and Future Work **65**

Bibliography **i**

A **vii**

List of Tables

3.1 Basic data statistics 34
3.2 Train, test and validation subset of the data 34
3.3 Example of the built data 35

4.1 Used Software 37
4.2 Strict measures versus token-level measure 39
4.3 Entity detection labels 41
4.4 Mention Detection Network 43

5.1 Results achieved by the different BiLSTM models, nearest neighbour search in UMLS knowledge base and the ensemble model on test data 58
5.2 Run times in minutes for of different steps of the pipeline 59

6.1 Results comparison between the achieved metrics from Medlinker mention detection. 62
6.2 esults comparison between the achieved metrics from different studies. Results in bold show the highest achieved for each metric. . . 64

A.1 Strict evaluation metrics of the BiLSTM with CODER as feature encoder. vii
A.2 Strict evaluation metrics of the BiLSTM with SciBERT as feature encoder. viii
A.3 Strict evaluation metrics of the BiLSTM with UMLSBERT as feature encoder. ix
A.4 Strict evaluation metrics of the BiLSTM with BioBERT as feature encoder. x
A.5 Strict evaluation of nearest neighbour search. xi
A.6 Strict evaluation of the plurality voting ensemble model (BiLSTM models + Nearest Neighbour). xii
A.7 Strict evaluation of the Cosine Similarity threshold ensemble model (BiLSTM models + Nearest Neighbour). xiii

List of Figures

2.1 One hidden layer perceptron with d hidden nodes, m inputs and t outputs 4
2.2 Logistic and hyperbolic tangent activation functions. 7
2.3 Logistic and ReLU activation functions and their derivations. Notice that derivative of the Logistic function approaches zero as the activation increases while the derivative of ReLU is constant as the net input is more than zero. 9
2.4 ReLU and Leaky ReLU activation functions. 10
2.5 Weight updates through gradient descent steps until the minimum of the loss function $J(\boldsymbol{w})$ is reached. 11
2.6 The local minimum shows significant low value but it is not be truly global minimum. A right choice of learning rate (yellow arrows) lead to the global minimum while red arrows learning rate lead to sub-optimal local minimum 12
2.7 Loss function curve using SGD algorithm with respect to number of batches. Notice the oscillations in the curve 13
2.8 Contour lines depict a quadratic loss function. Yellow line shows the trajectory of the optimizer to the minimum of the contour for SGD with out momentum at left figure and with momentum at right figure. The momentum reduces the oscillations of SGD algorithm 14
2.9 Single hidden layer RNN. The units h represent the the hidden units at different time steps while x and o represent the items in the input sequence and the output at different time steps respectively. Notice that the weights associated with the recurrent edge is W_{hh} . 18
2.10 Different types of recurrences 19
2.11 Different types of RNNs 20
2.12 Vanishing and exploding gradient 21
2.13 LSTM cell architecture. Forget gate (f) determines how much information to delete from the memory. Input gate (i) determines how much information $(\tilde{C})$ to store. Output gate (o) determines what to output. The yellow circles represent element-wise operation (product, sum) 24

2.14 Transformer Architecture 26
2.15 Embeddings of the BERT input are sum of the token embeddings, segment embeddings, and positional embeddings 29

4.1 Used Workflow for the optimal model. STY represents Semantic Types 40

5.1 Mention detection training and validation metrics curves for the BiLSTM model with CODER as feature encoder for three KFolds 49
5.2 Mention classification training and validation metrics curves for the BiLSTM model with CODER as feature encoder for three KFolds 51
5.3 Mention classification training and validation metrics curves for the BiLSTM model with SciBERT as feature encoder for three KFolds. 53
5.4 Mention classification training and validation metrics curves for the BiLSTM model with UMLSBERT as feature encoder for three KFolds. 54
5.5 Mention classification training and validation metrics curves for the BiLSTM model with BioBERT as feature encoder for three KFolds 56
5.6 Evaluation metrics vs different Cosine Similarity thresholds on the portion of the train data 57

Chapter 1

Introduction

1.1 Background

Data exists in different format and structures. In terms of structuring, data is categorized into structured and unstructured. Structured data is highly organized with clearly defined data types and is easily understood by machines. On the other hand, unstructured data (i.e. text data and images) is the type of data that does not follow an organized format which makes the analysis of it more challenging.

Text mining is an Artificial Intelligence (AI) method and is used to extract structured meaning from this unstructured text data. Text Mining is mentioned for the first time in Feldman et al. [1998]. According to Hotho et al. [2005], Text mining involves three perspectives: information extraction, data mining, and Knowledge Discovery in Databases. In this report, we focus on Information extraction perspective. Information extraction is viewed as the process where we know in advance what kind of features we want to extract from text. According to Hotho et al. [2005], "The task of information extraction naturally decomposes into a series of processing steps, typically including tokenization, sentence segmentation, part-of-speech tagging, and the identification of named entities".

Named Entity Recognition is a critical step in many applications such as chatbot systems, where information extraction and question answering (QA) are central. Extracting entities from text is helpful in identification of the key elements in a text such as the names of persons, organisations and geographical entities. For a large amount of data, extracting those entities aims to detect the most important information in the text. In other words, extracting entities helps in reducing the text into fewer features.

1.2 Problem Statement

The data used in this thesis is the newly released and challenging MedMentions [Mohan and Li, 2019]. The data is manually annotated resource for the recognition of biomedical concepts and is mapped to the Unified Medical Language System (UMLS). The task is to develop models that are able to extract and classify the entities from the data to their semantic types. Taking the example sentence "In fact, both the resistive and elastic components of the work of breathing increase due to airway obstruction and chest wall and lung stiffening, respectively.", we aim in this thesis to build a model that can classify the span "chest wall" as "biologic structure". In this thesis, we use Recurrent Neural Networks along with attention based models to extract and classify mentions from the MedMentions data.

1.3 Structure of thesis

In Section 2, we explain the theoretical details behind the methods we use in this thesis. In Section 3, we explain details about the data and the materials we use in this thesis. In Section 4, we explain our workflow in details and further develop several ideas to make Section 2 more concrete. In Section 5, the numerical results and observations are explained in details supported by figures. In Section 6, we use our findings and combine our observed results with explanations. In Section 7, we conclude our work and compare our results with the SOTA model results and provide possible directions for future improvements.

Chapter 2

Theory

Text Mining employs machine learning to automate the analysis of the text data. Machine learning is the process where machines learn to identify patterns in data in order to make predictions. There are three types of machine learning algorithms; supervised, unsupervised and reinforcement learning algorithms. Supervised learning algorithms are used when the data has a ground truth labels and these labels are used in calculation of cost function that is needed to be minimized to achieve the best model characteristics. Classification and regression tasks are typically a supervised learning. Unsupervised learning is the second major family of machine leaning algorithms where the labels are not used in the analysis such as clustering of similar data samples or detection of anomalies in the data. The third family is Reinforcement Learning where the machine learns by taking actions that maximize the rewards and minimize the penalties. In this thesis, we will focus on supervised learning.

2.1 Artificial Neural networks (ANNs)

Artificial Neural networks (ANNs) are machine learning algorithms that are inspired by how the biological neural networks in the brain work. Multilayer perceptron (MLP) is a specific kind of ANNs where the network layers are fully connected. An example of MLP is shown in Figure 2.1.

The first step of training the MLP is forward propagation of input features through the network to calculate the output. Second step is to calculate the errors between the predictions and the ground truth labels using loss function that is described in section 2.1.2 on page 8. Third step is to propagate this error to find it's derivative with respect to each weight in the network. This process is repeated until the loss function is reduced to minimum.

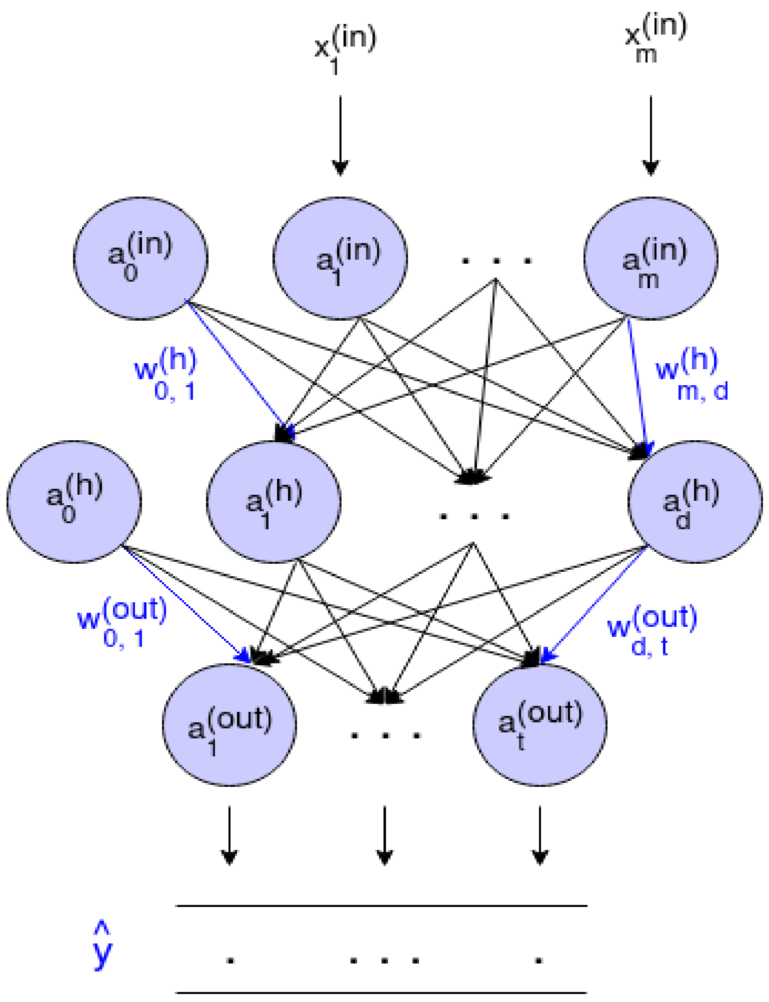


Figure 2.1: *One hidden layer perceptron with d hidden nodes, m inputs and t outputs*

The activation $a_1^{(h)}$ of the hidden layer can be calculated as follows:

$$a_1^{(h)} = \phi(z_1^{(h)}) \tag{2.1}$$

where $z_1^{(h)}$ is known as net input and is calculated as follows:

$$z_1^{(h)} = a_0^{(in)} w_{0,1}^h + a_1^{(in)} w_{1,1}^h + \cdots + a_m^{(in)} w_{m,1}^h \tag{2.2}$$

where $w_{0,1}^h$, $w_{1,1}^h$ and $w_{m,1}^h$ are the weights that connect the input units $a_0^{(in)}$, $a_1^{(in)}$ and $a_m^{(in)}$ to the hidden unit $a_1^{(h)}$.

Activation functions are explained in further details in the section 2.1.1 on the next page. The activations of the input features of the sample $x^{(in)}$ can be vectorized as:

$$\boldsymbol{a}^{(in)} = \begin{bmatrix} a_0^{(in)} \\ a_1^{(in)} \\ \vdots \\ a_m^{(in)} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1^{(in)} \\ \vdots \\ x_m^{(in)} \end{bmatrix} \tag{2.3}$$

The dimension of $\boldsymbol{a}^{(in)}$ is $1 \times (m+1)$ where $m+1$ is the number of features plus the bias unit. The net inputs of the hidden layer is shown as:

$$\boldsymbol{z}^{(h)} = \boldsymbol{a}^{(in)}\boldsymbol{W}^{(h)} \tag{2.4}$$

$$\boldsymbol{a}^{(h)} = \phi(\boldsymbol{z}^{(h)}) \tag{2.5}$$

The dimension of $\boldsymbol{W}^{(h)}$ is $(m+1) \times d$ where d is the number of units in the hidden layer. Using matrix multiplication, the dimension of $\boldsymbol{z}^{(h)}$ is $1 \times (d+1)$. For all n data samples, the net input is:

$$\boldsymbol{Z}^{(h)} = \boldsymbol{A}^{(in)}\boldsymbol{W}^{(h)} \tag{2.6}$$

and the activation function for all n data samples is:

$$\boldsymbol{A}^{(h)} = \phi(\boldsymbol{Z}^{(h)}) \tag{2.7}$$

The dimension of $\boldsymbol{A}^{(in)}$ is $n \times (m+1)$ and the dimension of $\boldsymbol{Z}^{(h)}$ is $n \times (d+1)$. The dimension of $\boldsymbol{A}^{(h)}$ is $n \times (d+1)$. Similarly, activations of output layer is calculated using the following equations.

$$\boldsymbol{Z}^{(out)} = \boldsymbol{A}^{(h)}\boldsymbol{W}^{(out)} \tag{2.8}$$

$$\boldsymbol{A}^{(out)} = \phi(\boldsymbol{Z}^{(out)}) \tag{2.9}$$

The dimension of $\boldsymbol{W}^{(out)}$ is $(d+1) \times t$. We obtain the matrix $\boldsymbol{Z}^{(out)}$ with dimension $n \times t$ and t is the number of output units.

2.1.1 Activation functions

Activation functions are used in ANNs to introduce non-linearity into the network that enables the network to learn the complex patterns in the data. Some activation functions are used in the hidden layers like ReLU while other activation functions are usually used in the output layer. The output layer can have activation functions such as Sigmoid for binary classification, Softmax for multilclass classification or no activation in case of regression.

Logistic function

Logistic function is a special case of a Sigmoid function. The equation of the Logistic function is shown in Equation 2.10. Net input could theoretically have values from negative to positive infinity. According to Goodfellow et al. [2016], "One way to solve this problem is to use the logistic sigmoid function to squash the output of the linear function into the interval $(0, 1)$ and interpret that value as a probability". Here, logistic function maps the range of the net input into values between 0 and 1 that represent probabilities.

$$\phi_{logistic}(z) = \frac{1}{1 + e^{-z}} \tag{2.10}$$

Hyperbolic Tangent (Tanh) function

Tanh function, as shown in equation 2.11, is similar to the above mentioned Logistic function but it returns a value between -1 and 1. According to Raschka and Mirjalili [2019], "The Tanh function is just a rescaled version of the logistic sigmoid function". Figure 2.2 on the facing page shows the difference between Logistic and Tanh function.

$$\phi_{tanh}(z) = \frac{sinh(z)}{cosh(z)} = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{2.11}$$

According to Goodfellow et al. [2016], using Sigmoid output units is to ensure that there is always a strong gradient. The mathematical reasoning behind usage of Sigmoid activation is explained in further details in chapter 6 in Goodfellow et al. [2016]. The point is to represent the probability distribution over a binary variable (equation 2.12).

$$\hat{y} = P(y = 1|\boldsymbol{x}) \tag{2.12}$$

A sigmoid output unit is given as

$$\hat{y} = \phi_{Sigmoid}(z) \tag{2.13}$$

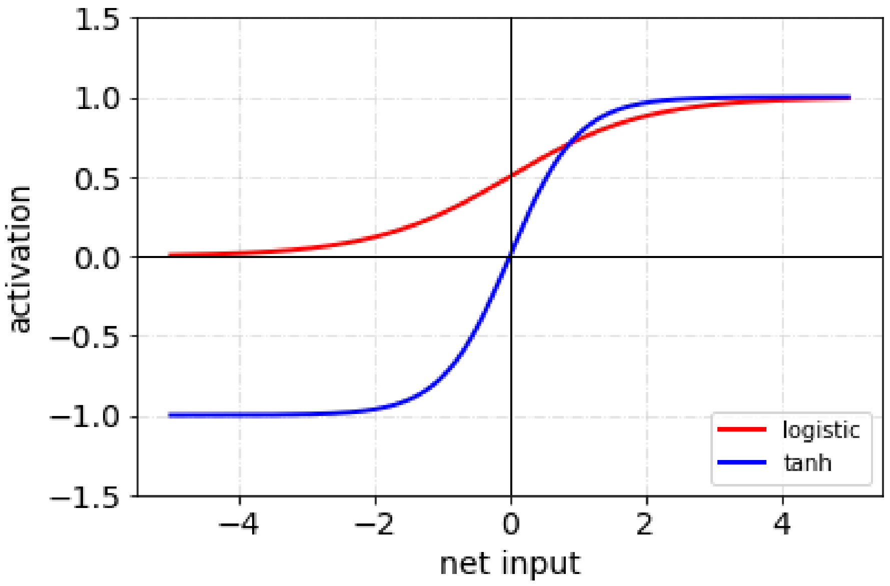


Figure 2.2: *Logistic and hyperbolic tangent activation functions.*

Softmax activation function

While Logistic function is used for binary classifications, Softmax function is a generalization of Logistic function that is used for multiclass classifications. Softmax is usually in the output layer to give probability distribution over C number of classes. Consider an input vector with C classes, Softmax value is the exponent of each input divided by the sum of the exponents of all the inputs in the vector (Equation 2.14).

$$\phi_{Softmax}(z_i) = \frac{e^{z_i}}{\sum\limits_{c=1}^{C} e^{z_c}} \text{ for } i = 1 \ldots C \tag{2.14}$$

Equation 2.15 provides predictions $\hat{y}$ using Softmax which is simply a generalization of Equation 2.12.

$$\hat{\boldsymbol{y}}_i = P(y = i|\boldsymbol{x}) \tag{2.15}$$

Rectified Linear Unit (ReLU) activation function

ReLU activation function is a ramp function that linearly outputs the input directly if it is positive, otherwise, it outputs zero as shown in equation 2.16.

$$\phi_{ReLU}(z) = max(0, z) \tag{2.16}$$

In practice, ReLU activation function have shown to train better than sigmoid activation functions. That is because ReLU can handle vanishing gradient problem. Vanishing gradient is an interesting and a challenging problem in training neural networks. Taking the Logistic function, the derivative of activation with respect to the net input vanishes as net input increases in magnitude (positive or negative) as shown in Figure 2.3 on the facing page. This makes the weights updates to become very slow. On the other hand, ReLU activation has a constant derivative of 1 when the net input is greater than 1.

Leaky ReLU

Leaky ReLU is a variant of ReLU. While ReLU outputs zero for all net inputs that are zero or negative, Leaky ReLU has a small negative slope (e.g. 0.01) as shown in Equation 2.17. Figure 2.4 on page 10 shows the difference between ReLU and Leaky ReLU.

$$\phi_{LeakyReLU}(z) = max(0.01 \times z, z) \tag{2.17}$$

2.1.2 Loss functions

For classification tasks, Softmax and Sigmoid activations, as explained in Section 2.1.1 on page 6, are used in the output layer to convert the net input z into a representation of probability. Without having Sigmoid or Softmax activations in the output layer, the model will only compute the logits. The main concept behind loss function is to calculate the errors between the predictions and the ground truth which is minimized using gradient optimization. There are various loss functions to be used depending on the task at hand.

Binary Cross Entropy is used for binary classifications. For one sample, Binary Cross Entropy is given by Equation 2.18 as:

$$BCE = -(y_s \log(p_s) + (1 - y_s) \log(1 - p_s)) \tag{2.18}$$

where y_s is the ground truth label binary value (0 or 1) and p_s is predicted probability of that class in sample s.

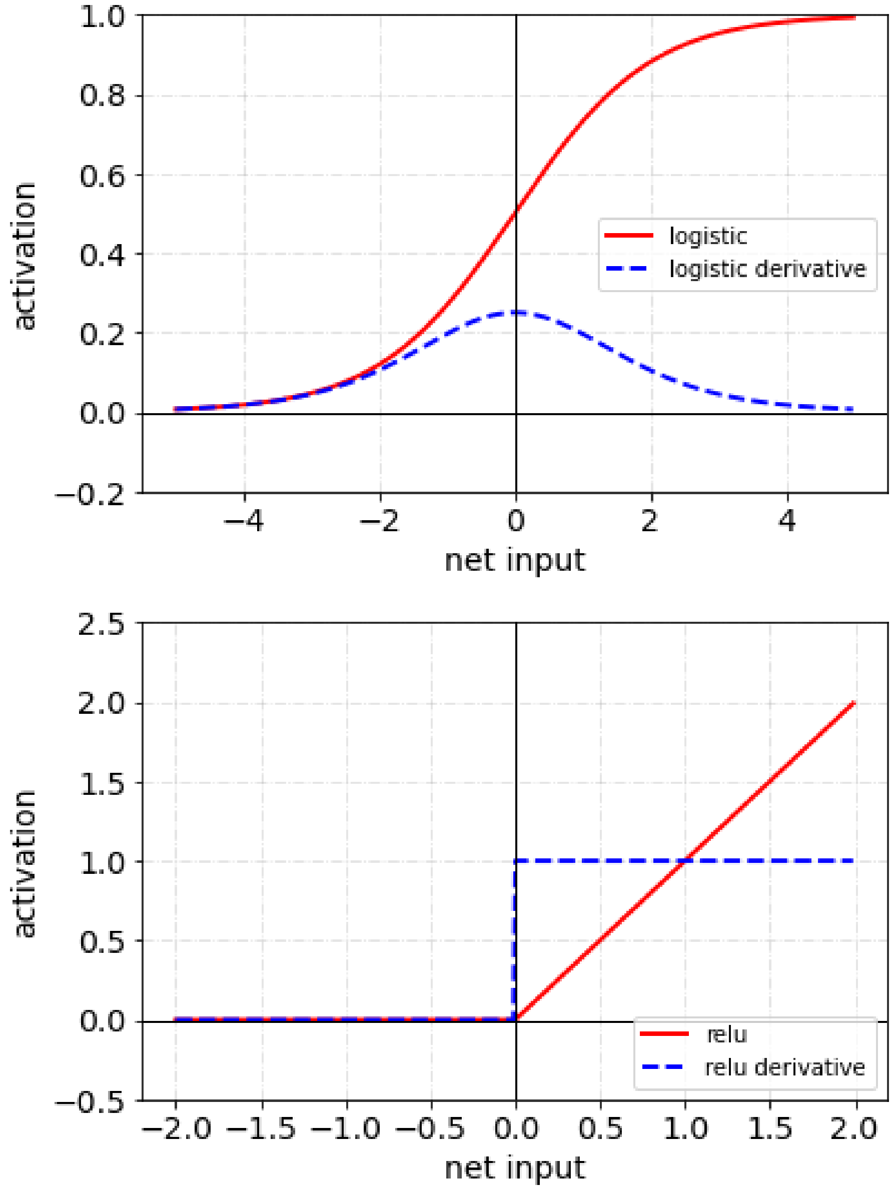


Figure 2.3: *Logistic and ReLU activation functions and their derivations. Notice that derivative of the Logistic function approaches zero as the activation increases while the derivative of ReLU is constant as the net input is more than zero.*

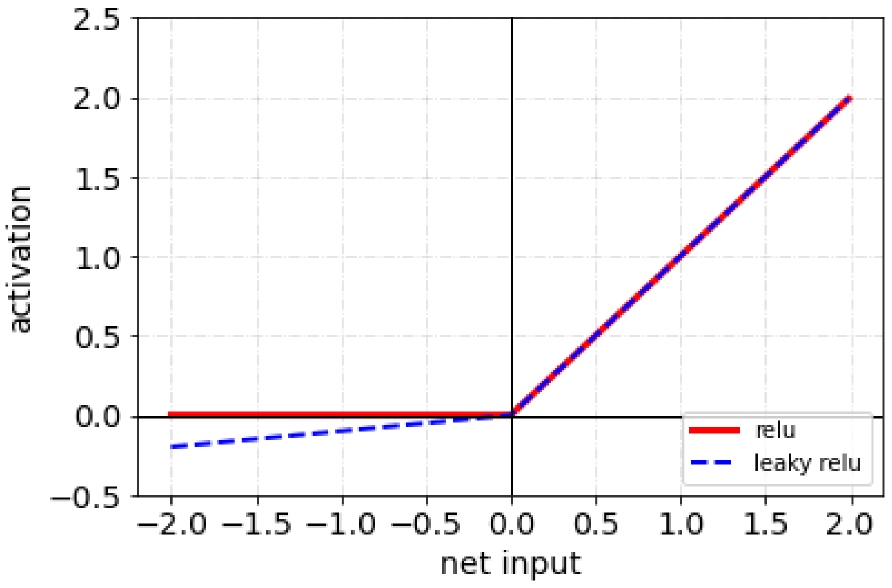


Figure 2.4: *ReLU and Leaky ReLU activation functions.*

For multiclass classifications, Categorical Cross Entropy loss is used which is given by Equation 2.19 for C classes. Categorical Cross Entropy loss is the negative log of the Softmax output (Equation 2.14) for the true label y_s and is given for number of classes C as:

$$CCE = -\sum_{c=1}^{C} y_{s,c} \log(p_{s,c}) \tag{2.19}$$

Here, $p_{s,c}$ is the predicted probability of the class c or Softmax output probability for the sample s and class c. In other words, Categorical Cross Entropy is the sum of the separate loss for each class.

In this thesis, we will only use Categorical Cross Entropy loss function for training our models.

2.1.3 Optimizers

In order to obtain the weights that minimize the above mentioned loss function, one needs an optimization algorithm called gradient descent[1]. Figure 2.5 shows a

[1]Pseudocodes of the explained algorithms are shown in details in chapter 8 in Goodfellow et al. [2016]

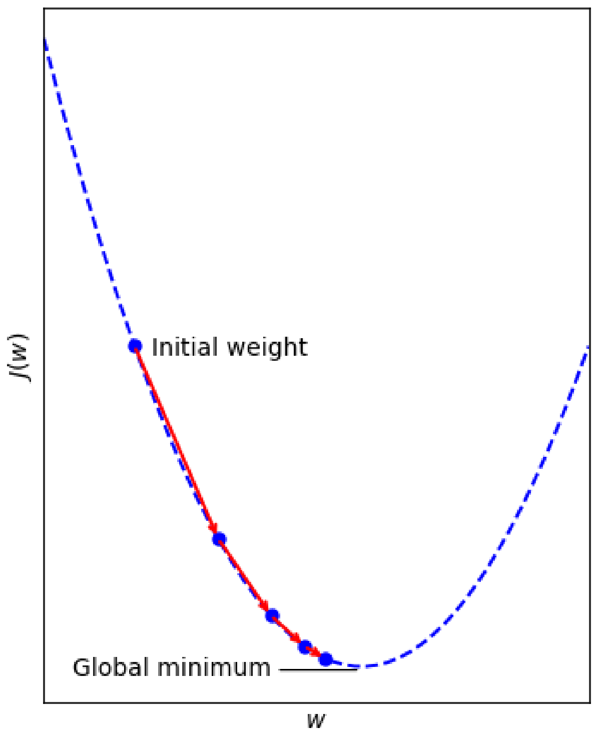


Figure 2.5: *Weight updates through gradient descent steps until the minimum of the loss function $J(\boldsymbol{w})$ is reached.*

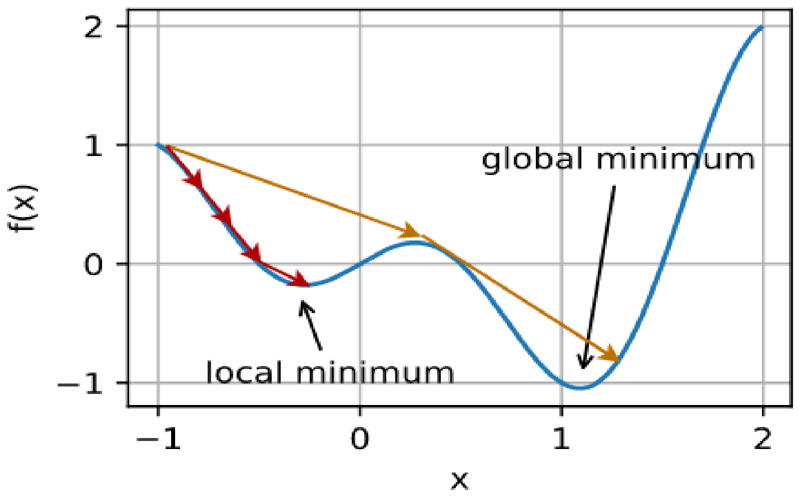


Figure 2.6: *The local minimum shows significant low value but it is not be truly global minimum. A right choice of learning rate (yellow arrows) lead to the global minimum while red arrows learning rate lead to sub-optimal local minimum. (Modified after CC by Zhang et al. [2020])*

graphical overview of how the optimizer works in general. The weights are updated in each step as:

$$\boldsymbol{w_{t+1}} := \boldsymbol{w_t} - \Delta \boldsymbol{w_t} \tag{2.20}$$

where $\boldsymbol{w_{t+1}}$ and $\boldsymbol{w_t}$ are the old and updated weights respectively. The $\Delta \boldsymbol{w_t}$ is given as:

$$\Delta \boldsymbol{w_t} = \eta \nabla J(\boldsymbol{w_t}) \tag{2.21}$$

where $\nabla J(\boldsymbol{w_t})$ is the gradient of the loss function and η is the learning rate. A large learning rate would lead to divergence from the global minimum and a small learning rate would lead to slow convergence and might get stuck in a non-optimal local minimum as explaind in Figure 2.6. In practice, it requires a lot of experimentation to find the learning rate that leads to the best solution.

There are three variants of gradient descent optimizer: batch gradient, Stochastic Gradient Descent (SGD) and mini batch gradient. These variants differ in how much data is included to compute the gradient. Batch gradient descent uses the entire dataset to compute one update which is impractical in terms of memory and speed specially when dealing with large datasets. On the other hand, SGD performs

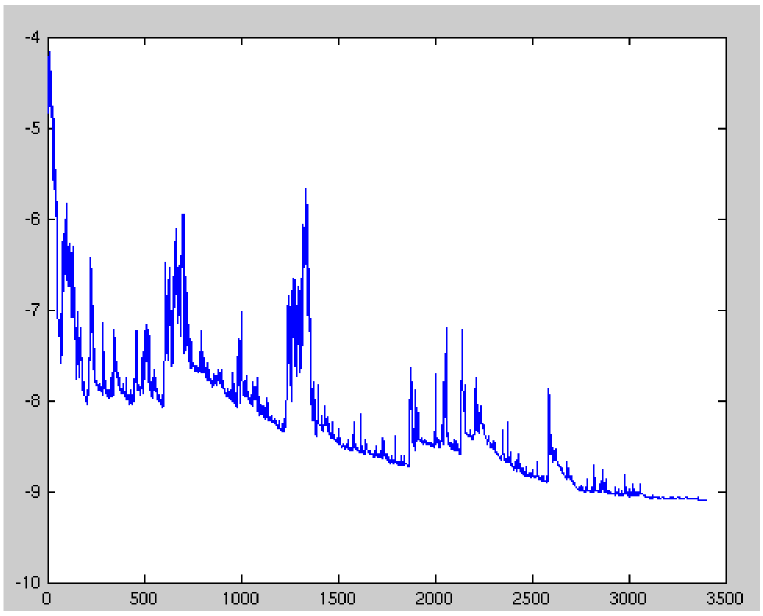


Figure 2.7: *Loss function curve using SGD algorithm with respect to number of batches. Notice the oscillations in the curve. (CC by Wikipedia [2020b])*

the updates for every training sample of the dataset. SGD is much faster than batch gradient descent but it can highly fluctuate until it reaches global minimum. Minibatch gradient descent performs the updates using a batch of several training samples at a time which reduces the fluctuations that accompany with SGD and enables for more stable solution.

Momentum

Information from previous updates can be used to accumulate momentum [Polyak, 1964] which is used to accelerate the gradient descent. For example, if the loss has been decreased in a particular direction, an exponentially decaying moving average of past gradients is accumulated using the momentum term that continues to move in that direction even if the loss increases again. The momentum helps to keep moving in the direction that decreases the loss. This allows to reduce the fluctuations of the gradient descent, as shown in figure 2.8, by adding an additional term to equation 2.21 as in the following:

$$\Delta \boldsymbol{w}_{t+1} = \alpha \Delta \boldsymbol{w}_t + \eta \nabla J(\boldsymbol{w}_t) \tag{2.22}$$

where α is the momentum factor (typically 0.9).

However, using only the accumulated momentum is still unsatisfactory as the opti-

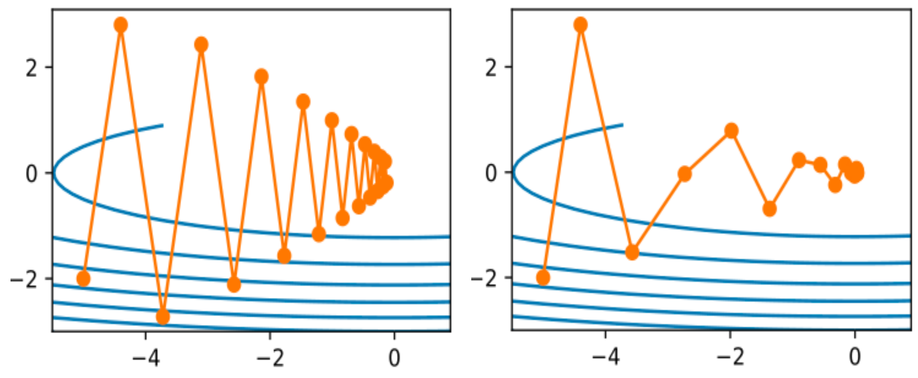


Figure 2.8: *Contour lines depict a quadratic loss function. Yellow line shows the trajectory of the optimizer to the minimum of the contour for SGD with out momentum at left figure and with momentum at right figure. The momentum reduces the oscillations of SGD algorithm (CC by Zhang et al. [2020])*

mizer moves blindly following the slope. A better version of momentum is needed that can slow down the optimizer before the loss increases again.

Nesterov Momentum

Sutskever et al. [2013] introduced a variant of the previously mentioned momentum in section 2.1.3 following Nesterov's accelerated gradient by Nesterov [1983]. Equation 2.22 shows that a momentum term is added in order to accumulate past gradients in the calculations. Here, the calculation of weight updates $\alpha\Delta\boldsymbol{w_t}$ gives an approximation of the next position of the weights $\boldsymbol{w_{t+1}}$. We look ahead by calculating the gradient with respect to the approximate future position of the parameters. Equation 2.22 is updated as:

$$\Delta\boldsymbol{w_{t+1}} = \alpha\Delta\boldsymbol{w_t} - \eta\nabla J(\boldsymbol{w_t} - \alpha\Delta\boldsymbol{w_t}) \tag{2.23}$$

Nesterov Momentum can be seen as a correction to the original momentum [Polyak, 1964]. Nesterov Momentum first makes a jump in the direction of the previous accumulated gradient and measures the gradient. Then, it makes a correction to prevent the optimizer from going too fast. Until now, the momentum performs an update for all parameters at once using the same learning rate η.

Adagrad

Adagrad [Duchi et al., 2011] adapts the learning rate based on the variation rate of the parameters. The parameters with the largest partial derivative of the loss have a rapid decrease in their learning rate, while parameters with small partial derivatives have a relatively small decrease in their learning rate. In order to do this, Equation can be rewritten as:

$$\boldsymbol{w_{t+1,p}} = \boldsymbol{w_{t,p}} - \frac{\eta}{G_{t,pp} + \epsilon} \dot{\nabla} J(\boldsymbol{w_{t,p}}) \tag{2.24}$$

where $\Delta \boldsymbol{w_{t+1,p}}$ is the new update for the parameter p. Adagrad modifies the learning rate η at each time for every parameter w_p based on the past gradients that have been computed for that parameter. $\boldsymbol{G_{t,pp}}$ is a diagonal matrix where each diagonal element pp is the sum of the squares of the gradients with respect to the parameter w_p up to time step t and ϵ is a small smoothing term to avoid division by zero.

RMSProp

The main weakness of Adagrad is its accumulation of the squared gradients in the denominator which causes the learning rate to diminish monotonically. The RMSProp is an unpublished algorithm proposed by Geoffrey Hinton and Swersky [2012] as a modification to Adagrad to fix this weakness. Instead of storing previous squared gradients, RMSProp divides the learning rate by an exponentially decaying average of past squared gradients. Exponentially decaying average are used to discard history from the extreme past. The average $E(\nabla J(\boldsymbol{w_t})^2))$ at time step t is given as:

$$E(J(\boldsymbol{w_t})^2) = \gamma E(\nabla J(\boldsymbol{w_{t-1}})^2) + (1-\gamma)\nabla J(\boldsymbol{w_t})^2 \tag{2.25}$$

where γ is similar to the momentum term (typically 0.9). Equation .2.24 can be rewritten as:

$$\boldsymbol{w_{t+1}} = \boldsymbol{w_t} - \frac{\eta}{E(\nabla J(\boldsymbol{w_t})^2) + \epsilon} \dot{\nabla} J(\boldsymbol{w_t}) \tag{2.26}$$

Adam

Adam [Kingma and Ba, 2017] is another adaptive learning rate optimization algorithm. Adam can be seen as a combination of RMSProp and momentum. Adam keeps exponentially decaying moving average of both past squared gradients (sm_t) and past gradients (m_t). The exponentially decaying average of past gradients is given as:

$$\boldsymbol{m_t} = \beta_1 \boldsymbol{m_{t-1}} + (1-\beta_1)\nabla J(\boldsymbol{w_t}) \tag{2.27}$$

and the exponentially decaying average of past squared gradients is given as:

$$\boldsymbol{sm_t} = \beta_2 \boldsymbol{sm_{t-1}} + (1 - \beta_2)\nabla J(\boldsymbol{w_t})^2 \tag{2.28}$$

$\boldsymbol{sm_t}$ and $\boldsymbol{m_t}$ are initialized as vectors of zeros, the authors Kingma and Ba [2017] observe that the vectors are biased towards zero, especially during the initial time steps. This can be corrected by:

$$\hat{\boldsymbol{m_t}} = \frac{\boldsymbol{m_t}}{1 - \beta_1^t} \tag{2.29}$$

$$\hat{\boldsymbol{sm_t}} = \frac{\boldsymbol{sm_t}}{1 - \beta_2^t} \tag{2.30}$$

The authors Kingma and Ba [2017] propose default values of 0.9 for β_1 and 0.999 for β_2. Similar to RMSProp, the weights are updated as:

$$\boldsymbol{w_{t+1}} = \boldsymbol{w_t} - \frac{\eta}{\sqrt{\hat{sm}_t} + \epsilon}\hat{m}_t \tag{2.31}$$

2.1.4 Backpropgation of errors

Backpropagtion [Rumelhart et al., 1986] is a widely used algorithm to train neural networks. From forward propagation, we can get the output layer activation by rewriting Equation 2.4 and Equation 2.5 as:

$$\boldsymbol{z}^{(out)} = \boldsymbol{a}^{(h)}\boldsymbol{W}^{(out)} \tag{2.32}$$

$$\boldsymbol{a}^{(out)} = \phi(\boldsymbol{z}^{(out)}) \tag{2.33}$$

where $\boldsymbol{z}^{(out)}$, $\boldsymbol{a}^{(out)}$ and $\boldsymbol{W}^{(out)}$ are the net input, activation, weights of the output layer while $\boldsymbol{a}^{(h)}$ is the activation of the hidden layer. Similarly, forward propagation of the hidden layer is given as:

$$\boldsymbol{z}^{(h)} = \boldsymbol{a}^{(in)}\boldsymbol{W}^{(h)} \tag{2.34}$$

$$\boldsymbol{a}^{(h)} = \phi(\boldsymbol{z}^{(h)}) \tag{2.35}$$

Backpropagation of multi-layer perception with one hidden layer can be summarized in the following steps:

Calculation of error term for output layer: this is the straightforward difference between the true label the activation of the output layer and this error is given as:

$$\boldsymbol{\delta^{(out)}} = \boldsymbol{a}^{(out)} - \boldsymbol{y} \tag{2.36}$$

where $\boldsymbol{y}$ is the true labels vector.

Calculation of error term for hidden layer: this can be given as:

$$\delta^{h} = \boldsymbol{\delta^{out}}(\boldsymbol{W}^{(out)})^{T} \odot \frac{\partial \phi(z^{(h)})}{\partial z^{(h)}} \tag{2.37}$$

where $\frac{\partial \phi(\boldsymbol{z^{(h)}})}{\partial \boldsymbol{z^{(h)}}}$ is the derivative of the Sigmoid activation [2] and it can be given as:

$$\frac{\partial \phi(\boldsymbol{z^{(h)}})}{\partial \boldsymbol{z^{(h)}}} = \boldsymbol{a^{(h)}} \odot (1 - \boldsymbol{a^{(h)}}) \tag{2.38}$$

where $\odot$ represent element-wise multiplication. So, the error of hidden layer is:

$$\boldsymbol{\delta^{h}} = \boldsymbol{\delta^{out}}(\boldsymbol{W^{(out)}})^{T} \odot (\boldsymbol{a^{(h)}} \odot (1 - \boldsymbol{a^{(h)}})) \tag{2.39}$$

Derivation of the loss function: after we obtain the error terms for output and hidden layers, the derivative of the loss function with respect to the parameters are given for sample i and node j as:

$$\frac{\partial J(\boldsymbol{W})}{\partial w_{i,j}^{(out)}} = a_{j}^{(h)} \delta_{i}^{(out)} \tag{2.40}$$

$$\frac{\partial J(\boldsymbol{W})}{\partial w_{i,j}^{(h)}} = a_{j}^{(in)} \delta_{i}^{(h)} \tag{2.41}$$

and for all the data samples as:

$$\boldsymbol{\Delta^{(out)}} = (\boldsymbol{A}^{(h)})^{T} \boldsymbol{\delta^{(out)}} \tag{2.42}$$

$$\boldsymbol{\Delta^{(h)}} = (\boldsymbol{A}^{(in)})^{T} \boldsymbol{\delta^{(h)}} \tag{2.43}$$

[2] derivative of Sigmoid activation is explained in details in chapter 12 in Raschka and Mirjalili [2019]

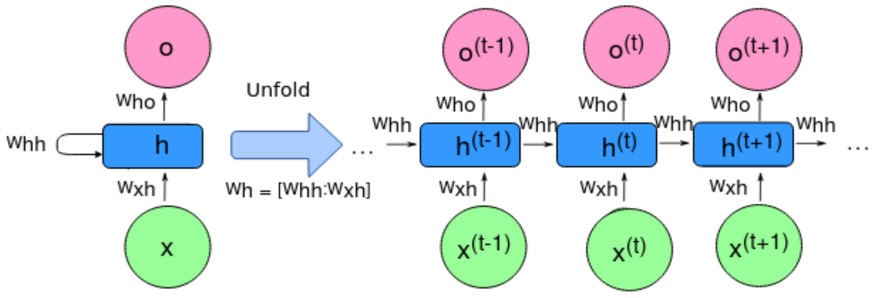


Figure 2.9: *Single hidden layer RNN. The units h represent the the hidden units at different time steps while x and o represent the items in the input sequence and the output at different time steps respectively. Notice that the weights associated with the recurrent edge is W_{hh} (Modified after CC by Wikipedia [2020a])*

Finally after we compute the gradients, we can update the weights in the general form as:

$$\boldsymbol{W}^{(l)} := \boldsymbol{W}^{(l)} - \eta \boldsymbol{\Delta}^{(l)} \tag{2.44}$$

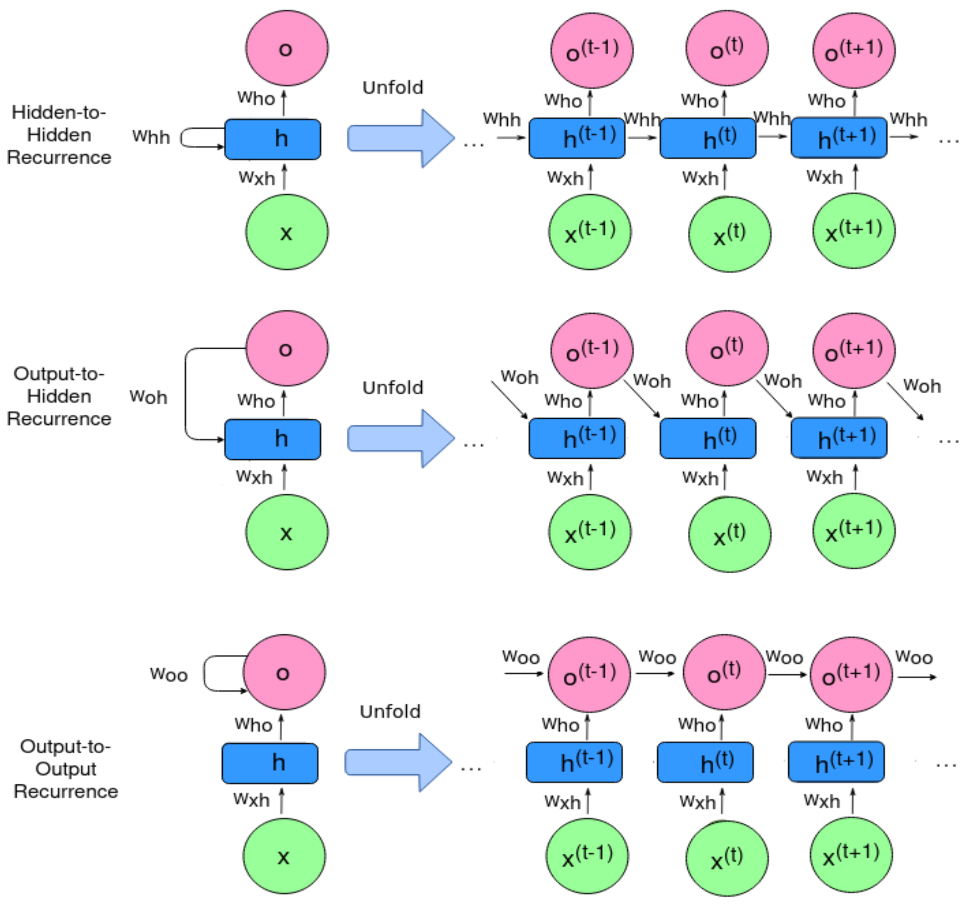
where l is the layer.

2.2 Deep Neural Networks

Deep Neural Networks (DNNs) are types of ANNs which have more than one hidden layers. There are various architectures of neural networks. The main architecture that is used to process text and sequence data is the Recurrent Neural Network or RNNs (Rumelhart et al. [1986]).

RNNs are suitable to process text data because it can leverage the order of the items in the sequence. In other words, RNNs can be seen as having memory to the items in the sequence. In RNNs, the hidden layer, as shown in Figure 2.9, receives its input from both the input layer of the current time step $x^{(t)}$ and the hidden layer from the previous time step $h^{(t-1)}$. The flow of information in adjacent time steps in the hidden layer allows the network to have a memory of past events. This flow of information is usually displayed as a loop, also known as a recurrent edge which can be unfolded as explained in Figure 2.9.

RNNs can be categorized based on the type of recurrence connection. The recurrence can be within the hidden units or within the outputs units or from the previous output unit to the current hidden unit. Hidden-to-hidden recurrence is when the recurrence connections is from the previous hidden unit to the current hidden unit as shown in Figure 2.10. The weights associated with hidden-to-hidden is denoted



Hidden-to-Hidden Recurrence
Whh
Who
Wxh
o
h
x
Unfold
o(t-1)
o(t)
o(t+1)
h(t-1)
h(t)
h(t+1)
x(t-1)
x(t)
x(t+1)
Output-to-Hidden Recurrence
Woh
Output-to-Output Recurrence
Woo

__Figure 2.10:__ Different types of recurrences (Modified after CC by Wikipedia [2020a])

as $\boldsymbol{W_{hh}}$. Output-to-hidden recurrence is when the recurrence connections is from the previous output unit to the current hidden unit. The weights associated with such recurrence is $\boldsymbol{W_{oh}}$. Output-to-output recurrence is when the recurrence connections is from the previous output unit to the current output unit. The weights associated with such recurrence is $\boldsymbol{W_{oo}}$.

There are different types of RNNs. The choice of suitable architecture depends on the task that the network is used for (figure 2.11).

- Many-to-many architecture: is used when the input and the output are sequences. This can be synchronised or asynchronised. An example of the synchronised architecture is video frame by frame classification while the asynchronised architecture is text translation where the input has to be entirely input before translation.

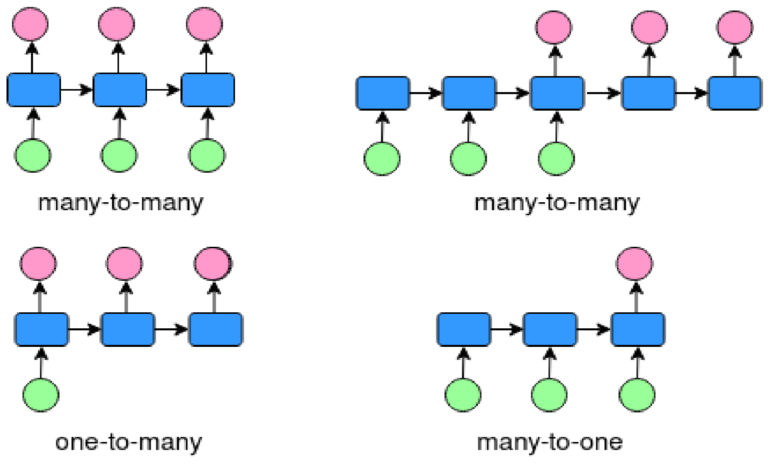


Figure 2.11: *Different types of RNNs*

- One-to-many architecture: is used when the input is a scalar and the output is a sequence. An example of the applications that use this architecture is image captioning.
- Many-to-one architecture: is used when the input is a sequence and the output is scalar. An example of applications of this architecture is sentiment analysis such as classification of a sequence whether it is a positive or negative sentiment.

Forward Propagating RNN

In figure 2.9, there are three weight matrices. The matrix $\boldsymbol{W_{xh}}$ is the weights between the input $\boldsymbol{x}^{(t)}$ and the hidden layer $\boldsymbol{h}^{(t)}$. $\boldsymbol{W_{hh}}$ is the weights matrix associated with the recurrent edge. The matrix $\boldsymbol{W^{ho}}$ is the weights between the hidden layer and output layer. Using those weights, the net input $\boldsymbol{z}_{\boldsymbol{h}}^{(t)}$ is calculated using Equation 2.45 as:

$$\boldsymbol{z}_{\boldsymbol{h}}^{(t)} = \boldsymbol{W_{xh}}\boldsymbol{x}^{(t)} + \boldsymbol{W_{hh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b_h} \tag{2.45}$$

where $\boldsymbol{b_h}$ is the bias vector for the hidden units. Taking the net input, the activations of the hidden units at time step t is given as:

$$\boldsymbol{h}^{(t)} = \phi_h\left(\boldsymbol{z}_{\boldsymbol{h}}^{(t)}\right) = \phi_h\left(\boldsymbol{W_{xh}}\boldsymbol{x}^{(t)} + \boldsymbol{W_{hh}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b_h}\right) \tag{2.46}$$

$$\boldsymbol{h}^{(t)} = \phi_h\left([\boldsymbol{W_{xh}}; \boldsymbol{W_{hh}}]\begin{bmatrix}\boldsymbol{x}^{(t)} \\ \boldsymbol{h}^{(t-1)}\end{bmatrix} + \boldsymbol{b_h}\right) \tag{2.47}$$

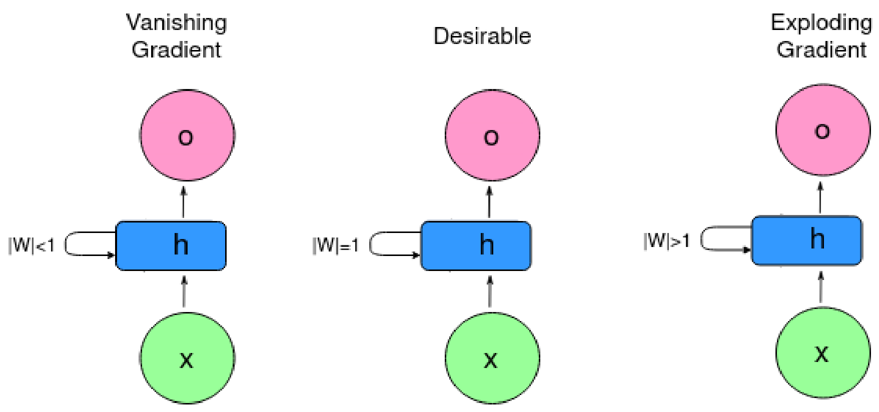


Figure 2.12: *Vanishing and exploding gradient*

and the activation of the output units is given as:

$$\boldsymbol{o}^{(t)} = \phi_o\big(\boldsymbol{W_{ho}}\boldsymbol{h}^{(t)} + \boldsymbol{b_o}\big) \tag{2.48}$$

Backward Propagating RNNs

Essentially in RNNs, the loss function doesn't only depend on the neurons that participated in the calculation of the output but also on contribution of these neurons far back in time. So, errors are to be backpropagated all the way back through time to these neurons. The full derivation of the backpropagation through time is explained in details in Werbos [1990]. The loss function is dependent on the hidden units at all time steps $(1 : t)$ and is given as in Equation 2.49 as:

$$L = \sum_{t=1}^{T} L^{(t)} \tag{2.49}$$

The derivative of the loss function will be:

$$\frac{\partial L^{(t)}}{\partial \boldsymbol{W_{hh}}} = \frac{\partial L^{(t)}}{\partial \boldsymbol{o_t}} \times \frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{h_t}} \times \left(\sum_{k=1}^{t} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(k)}} \times \frac{\partial \boldsymbol{h}^{(k)}}{\partial \boldsymbol{W_{hh}}} \right) \tag{2.50}$$

where k refers to the different time steps.

$$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{h}^{(i-1)}} \tag{2.51}$$

In Equation 2.51, the term $\frac{\partial \boldsymbol{h}^{(i)}}{\partial \boldsymbol{h}^{(i-1)}}$ is multiplied $t-k$ times (Figure 2.12). This means multiplication of the weight $\boldsymbol{W_{hh}}$ by itself $t-k$ times which would result in vanishing gradient if $\boldsymbol{W_{hh}}$ is less than 1 and exploding if $\boldsymbol{W_{hh}}$ is greater than 1 (Pascanu et al. [2013]). Several techniques are used to tackle the vanishing and exploding gradient problem such as Long Short-Term Memory, gradient clipping and truncated backpropagation through time (TBPTT). Using gradient clipping, a threshold for the gradients is specified and values that exceed that threshold value are cut off. TBPTT limits the number of time steps that the signal can backpropagate after each forward pass to the most recent time steps. In this thesis, we will only use LSTM that will be discussed in the next section in details.

Long Short Term Memory

Long Short Term Memory or LSTM is a special architecture of RNN and was introduced by Hochreiter and Schmidhuber [1997] and further developed by Gers et al. [1999]. LSTM is used to capture long term dependencies using gates to delete and add information from earlier states. These gates are shown in Figure 2.13 and they are as follows:

Forget gate : is a sigmoid function, as expressed in Equation 2.52, that takes in the output of the previous layer and the current layer input. It yields an output value between 0 and 1 where 1 means to “keep” and 0 means “delete”.

$$\boldsymbol{f_t} = \sigma(\boldsymbol{W_{xf}}\boldsymbol{x}^{(t)} + \boldsymbol{W_{hf}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b_f}) \tag{2.52}$$

where $\boldsymbol{W_{xf}}$ and $\boldsymbol{b_f}$ are the weights and bias between the input data at time t and the forget gate. $\boldsymbol{W_{hf}}$ is the weights matrix between the hidden unit and the forget gate. $\boldsymbol{h}^{(t-1)}$ is the hidden layer at time $t-1$. $\boldsymbol{x}^{(t)}$ is the input data at time step t.

Input gate and candidate creation has two parts. The first part, as expressed in Equation 2.53, defines the input gate and is a sigmoid that decides what will be updated.

$$\boldsymbol{i_t} = \sigma(\boldsymbol{W_{xi}}\boldsymbol{x}^{(t)} + \boldsymbol{W_{hi}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b_i}) \tag{2.53}$$

where $\boldsymbol{W_{xi}}$ and $\boldsymbol{b_i}$ are the weights and bias between the input data at time t and the input gate. $\boldsymbol{W_{hi}}$ is the weights matrix between the hidden unit and the input gate.

The second part, as expressed in equation 2.54, is a $tanh$ activation layer that outputs the candidate values that will be added to the new state.

$$\tilde{\boldsymbol{C_t}} = \tanh\left(\boldsymbol{W_{xc}}\boldsymbol{x}^{(t)} + \boldsymbol{W_{hc}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b_c}\right) \tag{2.54}$$

where $\boldsymbol{W_{xc}}$ and $\boldsymbol{b_c}$ are the weights and bias between the input data at time t and the candidate value. $\boldsymbol{W_{hc}}$ is the weights matrix between the hidden unit and the candidate value. This is followed by combination of the three equations 2.52, 2.53 and 2.54 from the input and forget gates to produce the cell state $\boldsymbol{C^{(t)}}$ at time t as expressed in equation 2.55

$$\boldsymbol{C^{(t)}} = \boldsymbol{C^{(t-1)}} \odot \boldsymbol{f_t} \oplus (\boldsymbol{i_t} \odot \tilde{C}_t) \tag{2.55}$$

Output gate contains two steps. The first step is a sigmoid activation as expressed in equation 2.56.

$$\boldsymbol{o_t} = \sigma(\boldsymbol{W_{xo}}\boldsymbol{x^{(t)}} + \boldsymbol{W_{ho}}\boldsymbol{h^{(t-1)}} + \boldsymbol{b_o}) \tag{2.56}$$

where $\boldsymbol{W_{xo}}$ and $\boldsymbol{b_o}$ are the weights and bias between the input data at time t and the output gate. $\boldsymbol{W_{ho}}$ are the weights between the hidden unit and the output gate. The second step, as expressed in equation 2.57, is a $tanh$ of the output of the second gate multiplied by the output of the first step in the output gate. The weights used in those equations are the cell memory.

$$\boldsymbol{h^{(t)}} = \boldsymbol{o_t} \odot \tanh{(\boldsymbol{C^{(t)}})} \tag{2.57}$$

An extension of LSTM was developed by Schuster and Paliwal [1997] by having bidirectional LSTMs or BiLSTM. In BiLSTM, one is responsible for the forward states (from start to end) and the other is responsible for backward states (reverse direction). This network can improve the model performance taking into account the dependence of previous sequence units on the future sequence units. Generally, LSTM can be seen as a way to encode the input sequence into hidden states. LSTM does not give importance or attention to some of the input sequence that are more relevant to the context compared to other words while modeling.

2.3 Embeddings

In order to extract features from words, words have to be encoded. One way to encode words is by one-hot-encoding [Harris and Harris, 1990], but this method will produce a sparse matrix that that has implications in training: the feature learning process will suffer from the curse of dimensionality [Raschka and Mirjalili, 2019]. A more compact way is to convert words into vectors through word embedding by using floats vector instead of sparse matrix. Representing words as vectors is usually what is used to capture the relative meaning of words though their vectors

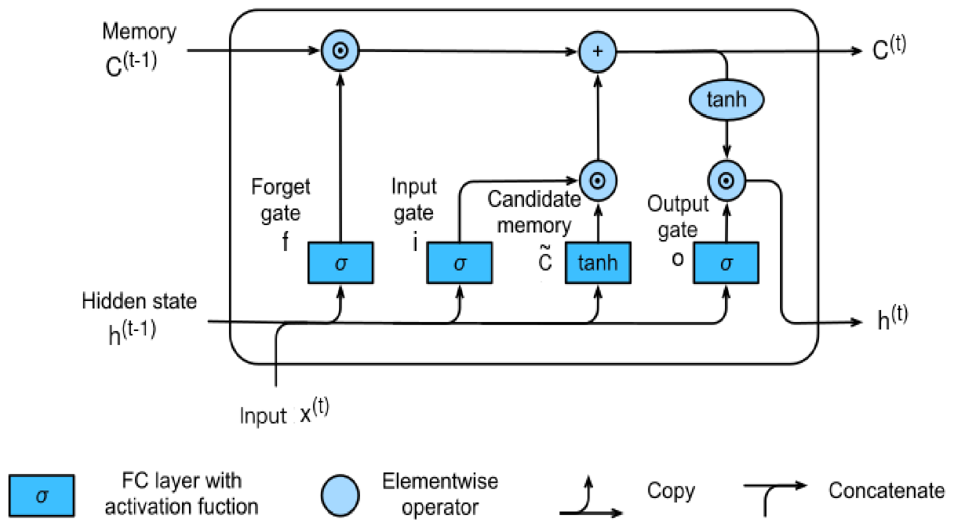


Figure 2.13: *LSTM cell architecture. Forget gate (f) determines how much information to delete from the memory. Input gate (i) determines how much information $(\tilde{C})$ to store. Output gate (o) determines what to output. The yellow circles represent element-wise operation (product, sum). Modified after CC by Zhang et al. [2020]*

representation. This means that words with similar meaning have similar representations. By clustering the words using the cosine similarity[3], one can find a relative meaning of the words. This embeddings matrix is used as input layer for the neural network. Several algorithms have been developed in this area like Word2Vec [Mikolov et al., 2013] and GloVe [Pennington et al., 2014].

Word2Vec uses a neural network to learn these representations. The intuition behind Word2Vec is based on the assumption that the word is related to its surrounding text. So, a certain word can be trained by a classifier with goal to predict the context of the that word. Word2Vec has two approaches: Continuous Bag-Of-Words (CBOW) and Skip-Gram (SG). CBOW approach predicts the word based on the context while SG approach predicts the context based on the word. For SG, Negative sampling method usually is used such as Skip-gram Negative Sampling (SGNS) where pairs of negative and positive samples are built and the objective is to maximize the predictions of the pairs that appear together and minimize the predictions of pairs that do not appear together. For example, if we take one word and use the above mentioned one-to-many network to predict the context from the

[3]Cosine similarity is a similarity measure between two vectors or the cosine of the angle between the two vectors. For vectors A and B, cosine similarity is given as $\frac{A \cdot B}{||A|| \cdot ||B||}$ where $||A||$ and $||B||$ are normalization of the vectors A and B

word. Once the training is complete, there will be an updated set of weights which represent the embedding of that word. One important observation in Word2Vec is that it does not take into account the frequency of co-occurrence of words.

On the other hand, GloVe takes these co-occurrence or frequencies into account which can provide more information. The GloVe algorithm is trained to aggregate word to word co-occurrence statistics in a corpus. It uses matrix factorization technique form linear algebra to measure term frequency to represent the co-occurrence matrix. Given a corpus that have n words, the co-occurrence matrix will be an $n \times n$ matrix where the matrix is populated with how many times word have co-occurred with other words where the vector of the word can be inferred from the context information. The authors Pennington et al. [2014], proposed to learn the ratios of these co-occurrence probabilities. Taking two words, the dot product of two vectors equals the log of number of times the two words will occur near each other. The authors Pennington et al. [2014] explained an example: if the $P(solid|ice)$ is large and $P(solid|steam)$ is small, the ratio of $P(solid|ice)/P(solid|steam)$ is large. The objective of the model, given a certain word, is to maximize the probability of a context to word occurrence. GloVe also makes use of CBOW and Skip-Gram similar to Word2Vec.

In terms of context representation, a lot of research has been done in this area in order to provide a better representation of the word within context. Better representation of the context allows for better feature extraction by the neural network and better accuracy. One way is using Embeddings from Language Models (ELMo) which was developed by Peters et al. [2018]. Another example of the recent advancement in this area is the Bidirectional Encoder Representations from Transformers (BERT) by Devlin et al. [2019]. In this thesis, ELMo is beyond our scope. Instead we will focus our attention on the BERT architecture which will be discussed in details in section 2.4.4. Before going deeper into the BERT architecture, it is more beneficial to discuss the concept of attention which is the main building block for the BERT architecture .

2.4 Attentions

The architecture was developed by Vaswani et al. [2017] which is known as "Transformer" and it has shown to outperform traditional gated RNNs. Current State Of The Art NLP systems usually incorporate LSTMs with attentions. The original attention idea was first developed by Bahdanau et al. [2016] and further improved by Vaswani et al. [2017]. Vaswani et al. [2017] introduced the Transformer as an encoder-decoder architecture. The encoder layers process the inputs to generate encodings where each encoder layer passes it's output to next layer. Those encodings contain attention information about which parts of the inputs are relevant to each other. The decoder layers take all those encodings and generate an output sequence.

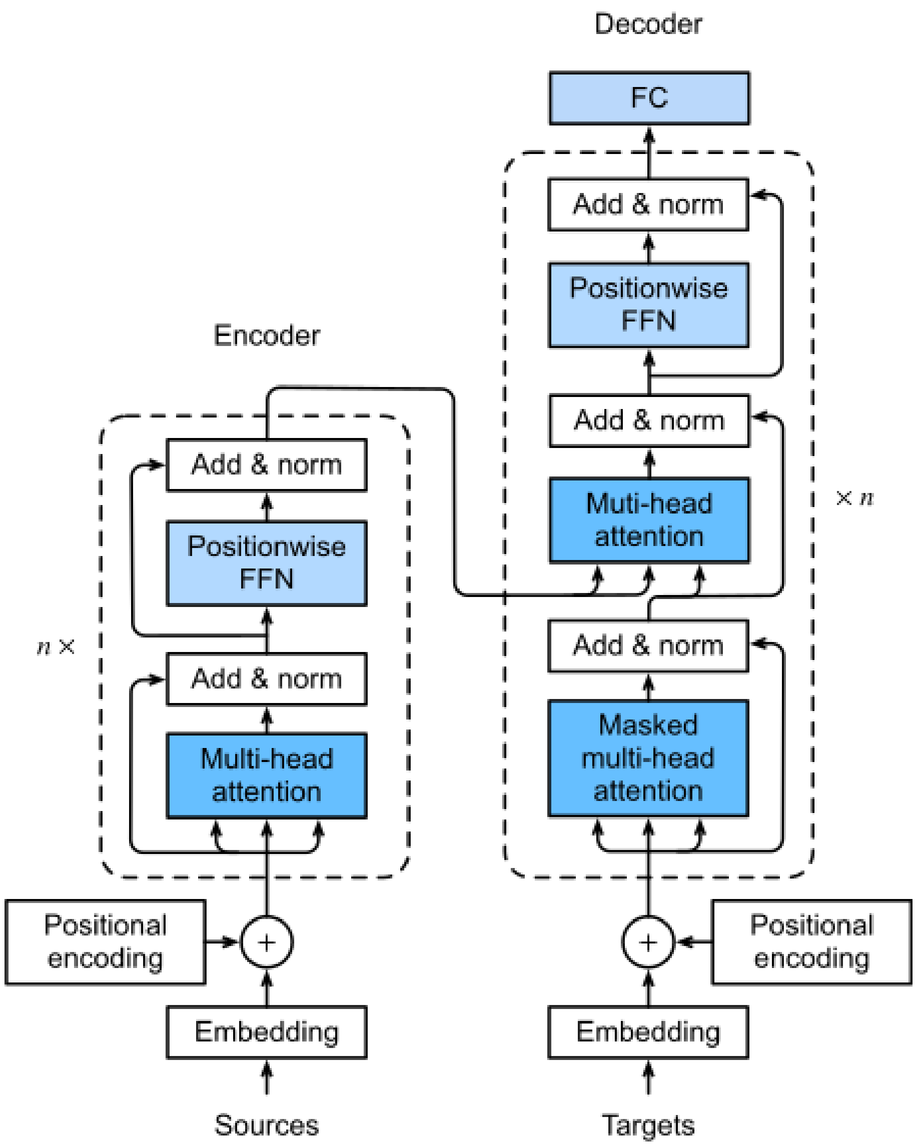


Figure 2.14: *Transformer Architecture. It consists of encoder and decoder. The encoder starts with embedding of input token in addition to the position encoding of the token in the sequence before feeding into self multi-head attention function. In the decoder, the output embedding along with the positional encoding go through a self multi-head attention function and further cross multi-head attention to quantify how relevant the input to itself and to the output. Source: CC by Zhang et al. [2020]*

2.4.1 Scaled dot-product attention

Attention weights are calculated between every word simultaneously. The attention unit produces embeddings for every token in the context that contain information not only about the token itself, but also a combination of other relevant words weighted by the attention weights.

For each attention unit, the transformer learns three weight matrices; Query weights $\boldsymbol{W_Q}$, key weights $\boldsymbol{W_K}$ and value weights $\boldsymbol{W_V}$. For each token i, the word embedding $\boldsymbol{x_i}$ is multiplied with each of the three weight matrices to produce a query vector $\boldsymbol{q_i} = \boldsymbol{x_i W_Q}$, a key vector $\boldsymbol{k_i} = \boldsymbol{x_i W_K}$, and a value vector $\boldsymbol{v_i} = \boldsymbol{x_i W_V}$. The attention weight a_{ij} from token i to token j is the dot product between $\boldsymbol{q_i}$ and $\boldsymbol{k_j}$ divided by the square root of the dimension d_k of the key vector followed by application of a softmax function to obtain the weights on the values. For large values of d_k, the dot products grow large in magnitude, pushing the softmax function to have small gradients. So, dot products are scaled by $\sqrt{d_k}$ to stabilize gradients as shown in Equation 2.58. The attention function computations are done using matrix multiplication as all the queries, keys and values are packed together in matrices $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$ respectively.

$$Atten(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = softmax(\frac{\boldsymbol{QK^T}}{\sqrt{d_k}})\boldsymbol{V} \tag{2.58}$$

2.4.2 Multi-Head attention

For every word, there will be a number h of different attention matrices or attention heads ($Atten(\boldsymbol{QW^Q}, \boldsymbol{KW^K}, \boldsymbol{VW^V})$). In order to reduce the total computational cost of performing the attention functions, Multi-Head attention is used to combine several different attention mechanisms to be performed in parallel. These attention heads are concatenated and multiplied with a single weight matrix to get a single attention head that will capture the information from all the attention heads. The multiple outputs for the multi-head attention layer are concatenated to pass into the feed-forward neural network layers. For h number of attention heads, the multi-head attention is:

$$MultiHead(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = concat(head_1, head_2...head_h)W^O \tag{2.59}$$

where the head or single head of the h heads is:

$$head_i = Atten(\boldsymbol{QW_i^Q}, \boldsymbol{KW_i^K}, \boldsymbol{VW_i^V}) \tag{2.60}$$

2.4.3 Transformer architecture

In the transformer architecture, the input (sources) and output (targets) sequence are added with positional encoding before being fed into the encoder and the decoder. The positional information is necessary for the transformer to make use of the order of the sequence.

Encoder

The encoder is a stack of multiple identical layers (denoted as n in Fig. 2.14). According to Vaswani et al. [2017], there are 6 identical layers in the network. Each layer has two sublayers: the first is a multi-head self-attention and the second is a position-wise feed-forward network. In the self-attention sublayer, queries, keys, and values are all from the the outputs of the previous encoder layer. Each sublayer adopts a residual connection, similar to ResNet neural architecture design by He et al. [2015], and a layer normalization [Ba et al., 2016]. As a result, the transformer encoder outputs a vector representation for each position of the input sequence. According to Vaswani et al. [2017], the dimension of the output is 512.

Layer normalization [Ba et al., 2016] normalizes the activations along the feature direction instead of the mini-batch direction as in batch normalization [Ioffe and Szegedy, 2015]. Layer normalization is more suitable in sequence data task whose inputs are often with different length. Layer Normalization normalizes each feature to zero mean and unit variance.

Decoder

The decoder is also a stack of multiple identical layers (6 identical layers) with residual connections and layer normalizations sublayers. Besides the two sublayers described in the encoder, the decoder has an extra sublayer, known as the encoder-decoder attention, between the encoder and the decoder. In the encoder-decoder attention, queries are from the outputs of the previous decoder layer, and the keys and values are from the encoder outputs. In the decoder self-attention, queries, keys, and values are all from the the outputs of the previous decoder layer. However, each position in the decoder is allowed to only attend to all positions in the decoder up to that position. This masked attention preserves the auto-regressive property, ensuring that the prediction only depends on those output tokens that have been generated. The last decoder is followed by a final linear transformation and Softmax layer to produce the output probabilities over the vocabulary.

2.4.4 Bidirectional Encoder Representations from Transformers

Bidirectional Encoder Representations from Transformers or BERT [Devlin et al., 2019] makes use of the above mentioned transformer architecture. As explained

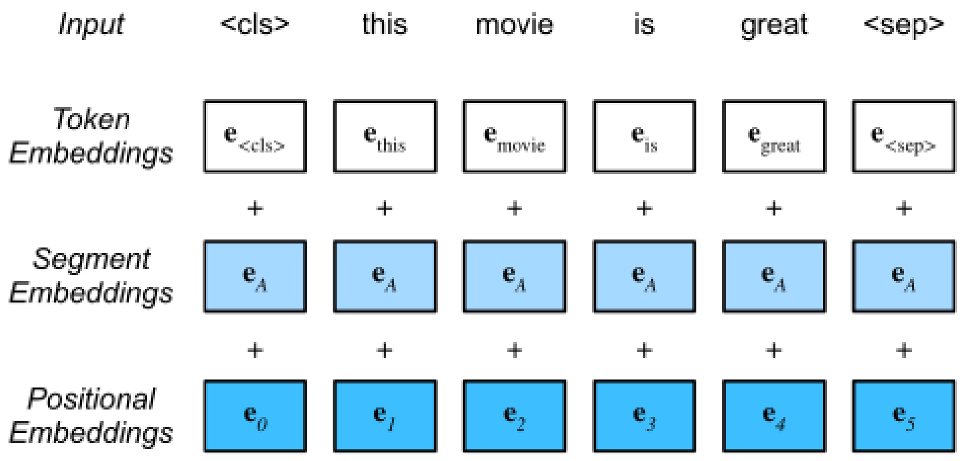


Figure 2.15: *Embeddings of the BERT input are sum of the token embeddings, segment embeddings, and positional embeddings. Source: Zhang et al. [2020]*

in 2.4, transformer model consists of encoder and decoder. BERT is a pre-trained language model, so, only the encoder is needed here. BERT is designed as a bidirectional model that is trained on the BookCorpus dataset [Zhu et al., 2015] and Wikipedia. The network effectively captures information from both right and left context of the token.

BERT has two main architecture; BERT-BASE and BERT-LARGE. BERT-BASE consists of 12 encoder layers while BERT-LARGE consists of 24 encoder layers compared to 6 layers in the original architecture by Vaswani et al. [2017] that was discussed earlier in section 2.4.3. Both BERT-BASE and BERT-LARGE also have larger feedforward-networks with 768 and 1024 hidden units, respectively, and 12 and 16 attention heads respectively. BERT-BASE contains 110M parameters while BERT-LARGE has 340M parameters. This model takes [CLS] token as input first, then it is followed by a sequence of words as input. Here [CLS] is a classification token. Then, it passes the input to the above layers. Each encoder layer applies self-attention, passes the result through a feedforward network after then it hands off to the next encoder. The model outputs hidden states with size 768 for BERT-BASE compared to 512 in the original architecture by Vaswani et al. [2017] that was discussed earlier in section 2.4.3.

In order to represent an input sequence (see Fig .2.15), input embeddings is a combination of three different embeddings. The first is position embedding which is used to express the position of words in a sentence to capture the order of the sequence. The second is segment embedding which is used by the model to distinguish between the sentences. The third embedding is the token embeddings for the specific token from the word piece token vocabulary.

In terms of tokenization, all BERT models support a huge list of vocabulary depending on the data that the model is pretrained on. The BERT tokenizer is used to convert each word into a unique number. In practice, BERT uses wordpiece tokenizer [Wu et al., 2016] which breaks the word into word pieces and those word pieces are tokenized accordingly. For example, ”I-Macroaggregated Albumin” is broken to (“i”, “-”, “mac”, “##roa”, “##gg”, “##re”, “##gated”, “album” and “##in”) using UMLSBERT pretrained tokenizer by [Michalopoulos et al., 2021]. In this context, BERT provides two ways of tokenization; BERT cased and BERT uncased. In BERT cased, the text remains the same with no changes as it is tokenized while in BERT uncased, the text is lowercased before wordpiece tokenization.

Input representations are used to compute the loss function for pretraining BERT. The loss function is a linear combination of both the loss functions for the two following training steps; masked language modeling and next sentence prediction.

Masked Language Modeling (MLM)

Instead of trying to predict the next word in the sequence, we replace the word with [MASK] and MLM predicts the missing word from within the sequence itself. The model is trained in such a way that it should be able to predict the missing word. 15% of tokens will be selected and masked randomly. However, the masked words were not always replaced by the masked tokens [MASK]. So, the researchers proposed the following technique; 80% of the time the words were replaced with the masked token [MASK]. 10% of the time the words were replaced with another random words and ask the model to predict the correct word. 10% of the time the words were left unchanged. This technique help to add noise that encourages BERT to be less biased towards the masked word.

Next Sentence Prediction

This training step aims to model the logical relationship between sentence pairs based on the assumption that the random sentence will be disconnected from the first sentence. BERT considers next sentence prediction as a binary classification task in the pretraining. When generating sentence pairs for pretraining, 50% of the time they are consecutive sentences while for the other 50% of the time the second sentence is randomly sampled from the corpus.

BERT for feature extraction

BERT can be used for token classification or Named Entity Recognition (NER). In NER, the system receives a text sequence and is required to extract and classify the various types of entities in the text. Using BERT, a NER model can be trained by feeding the output encodings of each token into a classification layer that predicts the NER label. In this setting, BERT is used to create contextualized word embeddings that are fed to your existing model such as LSTM model in order to be used for NER tasks. According to Devlin et al. [2019], concatenation of the encodings of the last four layers gives the best results.

Biological BERT

Variety of BERT models are trained on domain specific biomedical texts such as BioBERT [Lee et al., 2019], SciBERT [Beltagy et al., 2019], UMLSBERT [Michalopoulos et al., 2021] and CODER [Yuan et al., 2020].

BioBERT: was developed by [Lee et al., 2019] and it has the same architecture as BERT-BASE and BERT-LARGE and is initialized using pre-trained weights from BERT [Devlin et al., 2019]. Earlier version of BioBERT was based on BERT-BASE and BERT vocabulary while Latest version of BioBERT is based on BERT-LARGE with a new costumed 30K vocabulary. The authors Lee et al. [2019] released several versions of this model based on how many training steps are used in the pretraining. As explained earlier, the BERT model is trained on BookCorpus dataset [Zhu et al., 2015] and Wikipedia. BioBERT is trained further on the biomedical corpora; PubMed abstracts [4] and PMC full-text articles [5]. The authors Lee et al. [2019] showed that BioBERT outperformed BERT on several biomedical named entity recognition benchmark datasets such as NCBI disease corpus [Dogan et al., 2014] and BC5CDR [Li et al., 2016].

SciBERT: is another interesting pretrained model that was developed by Beltagy et al. [2019]. SciBERT uses a randomly 1.14M papers from Semantic Scholar corpus [Lo et al., 2020] and with vocabulary size of 30K subwords. 82% of the chosen papers are from the biomedical domain. Four versions of SciBERT were released based on the BERT-BASE version with and without casing and whether the model used the BERT weights of BERT as initial weights or was trained from scratch with a customized scientific vocabulary.

UMLSBERT: was developed by [Michalopoulos et al., 2021]. UMLSBERT updates the MLM procedure from the original BERT model [Wu et al., 2016] to consider the associations between the words specified in the UMLS Metathesaurus. This is done by introducing a semantic type embeddings that add further encodings to the input text. In other words, UMLSBERT adds a new embedding besides the token, segment and positional embeddings that are explained earlier in Section 2.4.4 and Fig .2.15. The aim behind this method is to learn the semantic grouping of the input words. The authors

[4]https://pubmed.ncbi.nlm.nih.gov/

[5]https://www.ncbi.nlm.nih.gov/pmc/

Michalopoulos et al. [2021] gave an example that if the word "lungs" is replaced with [MASK], BERT model will predict "lungs" while UMLSBERT will predict "lungs" and "pulmonary" as the two words belong to the same Concept Unique Identifer (CUI) in the UMLS Metathesaurus. UMLSBERT uses 33792 additional parameters that represent the number of unique UMLS semantic types; 44 multiplied by transformer's hidden dimension 768. The authors Michalopoulos et al. [2021] made changes to the loss function to adopt this multi-label words from the same concept instead of single-label word in the original BERT [Wu et al., 2016].

CODER: is short for **co**ntrastive learning on knowle**d**ge graphs for cross-lingual m**e**dical term **r**eprensentation [Yuan et al., 2020]. CODER is another BERT-based model which uses UMLS in the pre-training with aims to increase similarity between words from the same CUI.

CODER does this using multi-similarity loss [Wang et al., 2020]. Given anchor, positive and negative term from UMLS where positive belongs to the same semantic group as the anchor while negative does not belong to that semantic group. CODER learns term representations by maximizing similarity between the anchor and the positive term and reducing the similarity between the anchor and the negative term. The multi-similarity loss is given as:

$$\begin{aligned} MS = \frac{1}{2k}\sum_{i=1}^{2k} & \frac{(\log(1+\sum_{j\in P_i} exp(-\alpha(S_{ij}-\lambda)))}{\alpha} \\ & + \frac{\log(1+\sum_{j\in N_i} exp(-\beta(S_{ij}-\lambda))}{\beta}) \end{aligned} \tag{2.61}$$

where α, β, λ are hyper-parameters. P_i and N_i are positive and negative classes for the anchor term i. S_{ij} is the cosine similarity between the terms i and j. The first term in Equation 2.61 is to maximize the similarity between the anchor and the positive term while the second term is to ensure the negative term to have as low as possible similarity with the anchor.

According to the authors of CODER [Yuan et al., 2020], the model achieved superior results on the medical term normalization[6] datasets such as Cadec [Karimi et al., 2015] and PsyTar [Zolnoori et al., 2019].

[6]Normalization is mapping between medical terms to standardized medical vocabularies

Chapter 3

Data and Materials

The data used in this thesis is the MedMentions data by Mohan and Li [2019]. The data contains $4,392$ abstracts from medical articles with mentions or entities that are linked to the Unified Medical Language System (UMLS). The main objective of releasing this data is to encourage research in named entity recognition and entity linking.

3.1 Raw Data

The MedMentions corpus was annotated manually using the text data processing tool GATE[1] (version 8.2). All mentions are mapped to the Unified Medical Language Systems (UMLS) metathesaurus[2] (2017AA release) that was developed by Bodenreider [2004]. The 2017AA release of UMLS metathesaurus contains approximately 3.2 million concepts. Each mention from the dataset has a Concept Unique Identifier (CUI). In the UMLS metathesaurus, each CUI maps to a certain concept and its aliases. Each concept is also linked to one or more Semantic Types (STY). The metathesaurus contains 127 Semantic Types. Each Semantic Type or STY also has a Type Unique Identifier or TUI in short. For example, in the article with the id "25847295", the mention "apoptosis" is mapped to the CUI "C0162638" in UMLS which represents "Apoptotic Process" and also is mapped to the TUI "T043" which represents the STY "Cell Function".

According to Mohan and Li [2019], the precision of the annotations is estimated to be 97.3% by two biologists who didn't participate in the annotation task. The authors Mohan and Li [2019] provided a subset of the full data that is known as "ST21pv" with only 21 Semantic Types. The difference between the full data and

[1]https://gate.ac.uk/

[2]https://www.nlm.nih.gov/research/umls

"ST21pv" subset is shown in table 3.1 with basic statistics about the data. The data is subdivided into train, validation and test sets according to the article ID. The table 3.2 show the basic statistics about the train test split. A very important observation in the split is that not all concepts in the test data are covered in the train nor validation data which introduces the main challenge to work with this data.

Table 3.1: *Basic data statistics*

	Full	**ST21pv**
Documents	4392	4392
Mentions (Entities)	352496	203282
Unique Concepts	34724	25419
Tokens	1176058	1176058
Proportion of Mentions	49.3%	31.2%
Unique Semantic Types	127	21

Table 3.2: *Train, test and validation subset of the data (Source: Mohan and Li [2019])*

	Train	**Val**	**Test**
documents	2635	878	879
Mentions	12224	40884	40157
Unique Concepts	18520	8643	8457
Number of concepts overlap with train	–	4984	4867
Proportion of concepts overlap with train	–	57.7%	57.5%
Number of concepts overlap with train and val	–	–	5217
Proportion of concepts overlap with train and val	–	–	61.7%

3.1.1 Formatting Raw Data

In order to avoid errors in formatting the raw data for this thesis, we used the formatted data that has been already built by Nejadgholi et al. [2020]. The data was annotated by Nejadgholi et al. [2020] according to IOB2 annotation scheme which is short for Inside, Outside and Beginning of mentions. For the named entity chunks, annotation "O" represents token outside chunk. The annotation "B" is the beginning of a chunk while the annotation "I" is inside the chunk. Also, the annotation "B" is given for a single chunk entity. An example sentence of formatted data is shown in table 3.3.

Table 3.3: *Example of the built data (Source: Nejadgholi et al. [2020])*

Word	**Ground Truth Label**
Novel	O
insights	O
into	O
the	O
molecular	O
mechanism	O
of	O
sperm-egg	B-biologic-function
fusion	I-biologic-function
via	O
IZUMO1	B-chemical

3.1.2 Baseline

The authors Mohan and Li [2019] provided a baseline for the subset ST21pv to develop CUI linking systems. But our target is to do STY linking. So, the baseline for our study is the current State Of The Art (SOTA) model and was developed by Fraser et al. [2019] for STY linking. The baseline model metrics are **0.640**, **0.630** and **0.650** for F1, Precision and Recall metrics respectively.

Chapter 4

Methods

All the methods and results are described in details in a series of notebooks in the GitHub repository `https://github.com/mhmdrdwn/thesis`.

4.1 Resources

We use the Kaggle kernel[1] for our computations. The Kaggle kernel provides Tesla P100 GPU with 16 GB of VRAM, two cores of Intel Xeon CPU and TPU v3-8. The used software are shown in the table 4.1 with the version of each software.

Table 4.1: *Used Software*

Software	**Version**
Python	3.7.9
Numpy	1.19.5
Scikit-Learn	0.24.1
Tensorflow	2.4.1
Transformers	4.2.2
Sentence Transformers	1.1.0
Seqeval	1.2.2
Faiss	1.5.3

[1]`https://www.kaggle.com/`

4.2 Evaluation Metrics

We record F1, Precision, Recall scores [Powers, 2008] to evaluate and optimize the models. Precision measures the number of positive class predictions (TP) that actually belong to the positive class:

$$P = \frac{TP}{TP + FP} \tag{4.1}$$

On the other hand, Recall is the number of positive class predictions of all positive samples:

$$R = \frac{TP}{TP + FN} \tag{4.2}$$

F1 combines both Precision and Recall in a balanced measure as:

$$F1 = \frac{2 \times P \times R}{P + R} \tag{4.3}$$

We use both the strict and token-level measures of F1, Precision and Recall. The strict measure is widely used for Named Entity Recognition tasks and that is what is required to be measured following the MedMentions paper [Mohan and Li, 2019]. In terms of strict measures, We use Seqeval package [Nakayama, 2018] that only quantifies True Positive (TP) if the whole entity is predicted correctly. This means that partially predicted entities do not count as TP as explained by the example in Table. 4.2. In Table 4.2, prediction 1 is counted as one TP with strict while prediction 2 is two TP because the word “fusion” is mistakenly predicted as “O” in prediction 1. On the other hand, prediction 1 is counted as two TP while prediction 2 is counted as three TP in in token-level measure. F1, Precision and Recall scores can be measured as arithmetic mean of all the measured F1, Precision and Recall scores for each class or macro measure where each class is given equal weight (macro measure). Weighted average F1, Precision and Recall scores can also be measured as arithmetic mean of all the measured F1, Precision and Recall scores for each class weighted by the number of support of that class. Micro-averaged F1, Precision and Recall are calculated using the total TP by looking at all the samples together. For example, micro Precision counts the total number of TP and FP in the entire data before calculating the measure. For macro Precision, TP and FP are counted and the measure is calculated for each class before taking arithmetic mean. If the data is balanced, macro and micro measures will show identical results.

4.3 Proposed Workflow

The methodology of this thesis involves two stages: Mention classification and Entity Linking (EL). In mention classification, we use Four BiLSTMs with CODER

Table 4.2: *Strict measures versus token-level measure*

Word	**Ground Truth**	**Prediction 1**	**Prediction 2**
Novel	O	O	O
insights	O	O	O
into	O	O	O
the	O	O	O
molecular	O	O	O
mechanism	O	O	O
of	O	O	O
sperm-egg	B-biologic-function	B-biologic-function	B-biologic-function
fusion	I-biologic-function	O	I-biologic-function
via	O	O	O
IZUMO1	B-chemical	B-chemical	B-chemical

[Yuan et al., 2020], SciBERT [Beltagy et al., 2019], UMLSBERT [Michalopoulos et al., 2021] and BioBERT [Lee et al., 2019] as the feature encoder. Entity Linking (EL) or Mention Linking[2] involves two phases: Mention Detection and Mention Disambiguation. Mention Detection is the process of recognition the mentions of interest from the free text while Mention Disambiguation is the process of searching for an exact or similar concept in the knowledge base. The knowledge base we used in this thesis is the UMLS metathesaurus (see 3.1). A schematic workflow of the methods is shown in Figure 4.1.

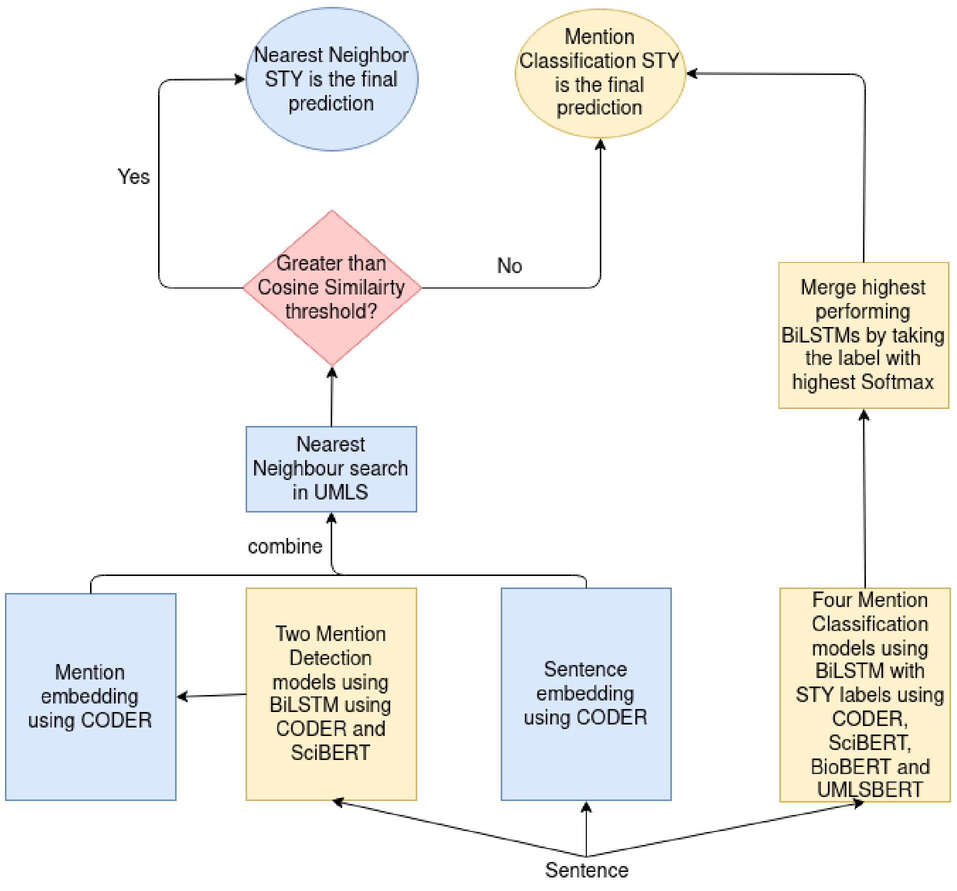
4.3.1 Mention Detection

In this step, we formulate the problem as classification task where we want to tell whether that the token is an a part of entity or not. In order to do that, we use the original labels of the data to build new labels as shown in Table 4.3 where all the labels are transformed into three labels ("B-Entity", "I-Entity", "O").

Features Encoding

We used the CODER [Yuan et al., 2020], SciBERT [Beltagy et al., 2019] models for features encoding during the modeling process. In order to use the CODER model, we add token [CLS] and [SEP] at the start and end of each sentence respectively. This is the way that the model has been trained, and it expects the input data to be in this way in order to know where is the start and end of each sentence.

[2]We use the word "Mention" to represent "Entity" as it is more appropriate for the task at hand.



Nearest Neighbor STY is the final prediction
Mention Classification STY is the final prediction
Yes
Greater than Cosine Similairty threshold?
No
Merge highest performing BiLSTMs by taking the label with highest Softmax
Nearest Neighbour search in UMLS
combine
Mention embedding using CODER
Two Mention Detection models using BiLSTM using CODER and SciBERT
Sentence embedding using CODER
Four Mention Classification models using BiLSTM with STY labels using CODER, SciBERT, BioBERT and UMLSBERT
Sentence

Figure 4.1: *Used Workflow for the optimal model. STY represents Semantic Types*

The special tokens [CLS] and [SEP] are labeled with the label "O". In order to encode the tokens, we use the CODER and SciBERT pretrained tokenizers. BERT tokenizer encode words is by breaking the word into several sub-words or word pieces (Wu et al. [2016]) and assign a unique number for each sub-word. The main challenge in this task is to preserve the labels against the sub-words. Following the article (Wu et al. [2016]), we use only the label of the first word piece. We assign the label "O" for subsequent word pieces. In the same time, we make a mask for those subsequent word pieces to avoid training on them.

Another argument to preserve the labels of the word pieces is to consider the word pieces as parts of the IOB2 annotation. In other words, we consider the first word piece as "B-Entity" and the rest subsequent word pieces as "I-Entity". In practice, we notice no difference in performance between assigning "O" or "I-Entity" to the subsequent word pieces. Another less efficient way that can be used to preserve the labels is to repeat the label for each word piece. However, this could be problematic in measuring the performance of the models as one "B-Entity" of the original word

Table 4.3: *Entity detection labels*

Word	Original Label	Detection Label
[CLS]	O	O
Novel	O	O
insights	O	O
into	O	O
the	O	O
molecular	O	O
mechanism	O	O
of	O	O
sperm-egg	B-biologic-function	B-Entity
fusion	I-biologic-function	I-Entity
via	O	O
IZUMO1	B-chemical	B-Entity
[SEP]	O	O

is broken into several "B-Entity" that increases the number of TP. Taking only the label of the first word piece is helpful in the calculation of evaluation metrics since we do not increase the number of support labels for each class which means that the evaluation metrics will be more realistic.

Padding

Taking into account that the different BERT models have different vocabularies, they will breaks words in different manners. Hence, the maximum length of sequences will be different for each BERT model. The maximum length of all sentences depends on how the used BERT tokenizes the words. In CODER, maximum length is 302 while maximum length using SciBERT is 293. We add padding to each of the sentences with zeros at the end in order to make all sentences of the same length. We build attention masks in order for the model to avoid training on the padded values. This is done by making an array of the same size of the sentences array where the padded values correspond to zeros and the other non padded values correspond to ones. The label we assign for the padded values are the "O" label.

Labels Encoding

We build a Python dictionary for the labels and convert each of the labels to the corresponding numbers. The dictionary has three labels. We convert those labels to a one-hot-encoded (Harris and Harris [1990]) matrix where the label index is removed and a binary value, either 0 or 1, is inserted instead. This means that the labels vector for each training sequence is converted to $s \times 3$ matrix where s is the length of the sequence and 3 is number of labels.

Building The Network

We use the Keras functional API to build the network. The network is shown in table 4.4. The data inputs are the input IDs and the masks which are the sentences and the padding masks after applying tokenization and padding which have the length 302. The CODER model outputs the 13 hidden states and each hidden state can be considered as an embeddings matrix of the batch sentences. Devlin et al. [2019] recommended using concatenation of last four layer of the 13 layers. In our implementations, we take the average of the last four hidden states. This is done in the addition and scaling steps and the results of those step is an embedding vector of the size 768. Those embeddings are then feed into 128 BiLSTM units followed by another 128 LSTM unit before feeding into a Time Distributed layer with 128 units. The output layer is a Dense layer which is a regular fully connected neural network layer with the number of labels and softmax activation. The number of labels is set to three. The loss function is categorical cross entropy. We freeze the BERT model pretrained parameters. This leads to only $1,132,547$ trainable parameters.

In Table 4.4, the LSTM layer where we set it to return sequence which means that it is generating a sequence of outputs. This means that the output of LSTM contain sequential information in it. This sequence is fed to a Time Distributed layer which is a Dense layer[3] used to process each of these sequential outputs one at a time by applying the same Dense on the sequential outputs by slicing the sequence in the time dimension and iterating through it.

Cross Validation

The validation split is set to 10000 samples and the model is evaluated with different validation sets three times. This is done manually by slicing the dataset where the first validation set is the first 10000 samples and the rest of samples are training set. The training and validation token-level F1, precision and recall are measured during the training and are shown in order to optimize the hyperparameters.

Hyperparameter Optimization

Because of the large size of model and the data, we use manual grid search of hyperparameters until we find the best combination. We pick the learning rates from logarithmic scales before we expand the search into a finer scale. The learning rates

[3]Fully Connected Layer

Table 4.4: *Mention Detection Network*

	Output Shape	**Parameters**	**Connected to**
Input IDs	302	0	–
Input Mask	302	0	–
CODER	13	109,482,240	Input IDs Input Mask
Addition	768	0	CODER[10] CODER[11] CODER[12] CODER[13]
Scaling	768	0	Addition
BiLSTM	256	918,528	Scaling
LSTM	128	197,120	BiLSTM
Time Distributed	128	16,512	LSTM
Output Layer	3	387	Time Distributed
Total Parameters Trainable Non-trainable	– – –	110,614,787 1,132,547 109,482,240	– – –

we use in the grid search are $(0.0001, 0.0005, 0.0008, 0.001, 0.002, 0.005, 0.01)$. We use different combination of number of LSTM units $(64, 128, 256)$, batch sizes $(32, 64)$ and optimizers (Adam, RMSProp).

Decoding Sequence IDs

In practice, it is not possible to recover the exact original sentences by decoding the sequence IDs. Although, we can decode a version that are as close as possible to the original sentences. we use the Transformers package to recover the word pieces from the IDs by detokenizing them using the the used BERT tokenizer. Then, we take those word pieces that contain ## in the beginning and merge them with previous word pieces in the same sentence. The label of the merged word is considered as the predicted and ground truth label of the first word piece.

4.3.2 Mention Classification

Similarly to section 4.3.1, we use the same model in the table 4.4 with changing the labels and number of labels. The number of labels in this model is 43. This is 21 STY labels transformed to IOB2 annotation scheme plus the "O" label. Similarly, this model is trained under TPU environment and we record the token-level F1, Precision and Recall and duration of the training.

In this task, we use the CODER [Yuan et al., 2020], SciBERT [Beltagy et al., 2019], UMLSBERT [Michalopoulos et al., 2021] and BioBERT [Lee et al., 2019] pretrained models for feature encoding.

4.3.3 Mention Disambiguation

In this step, we aim to boost the performance of the BiLSTM models in section 4.3.2 by using the UMLS knowledge base to search for nearest neighbours for the detected mentions that are extracted using the methods in section 4.3.1. This step has two substeps: Candidate Generation and Candidate Ranking. Candidate Generation aims to reduce the size of the search space for each mention and Candidate Ranking aims to get the nearest neighbours from each set of candidates for each mention. Candidate Generation can be done surface string similarity. However, we notice that searching for similar medical terms in knowledge base is difficult using surface string similarity. For example, using simstring (Okazaki and Tsujii [2010]) similarity algorithm, following Loureiro and Jorge [2020] and Kaewphan et al. [2018], to search the top 100 approximate nearest neighbours for the mention "Lifting Fatigue Failure Tool" does not include candidates that belong to the correct Semantic Type (STY) label. So, we skip the candidate generation step. Another reason for us to skip the Candidate Generation step is that it aims to reduce the space of concepts aliases to save computation cost of generating embeddings and nearest neighbour search. Searching the top 100 approximate nearest neighbours for each extracted mentions yields approximately two millions unique aliases which are not big reduction from the original 2.5 millions aliases in the knowledge base. Furthermore, we use the sentence transformers package [Reimers and Gurevych, 2019] to generate the embeddings of all the aliases which takes approximately 30 minutes to save all embeddings of all the knowledge base aliases under GPU environment.

Nearest Neighbour

In this step, we use the Sentence Transformers package [Reimers and Gurevych, 2019] to extract embeddings of all the aliases in the UMLS knowledge base. Also, we extract embeddings of extracted mentions, or query mentions, that we predict from section 4.3.1 and their sentences (or context). We sum the embeddings of the context and the query mention using weighted embeddings sum (0.7 of mentions embeddigs + 0.3 of the context embeddings). For the sentence transformers, we use the CODER as encoder. We normalize the extracted embeddings for both the query mentions and knowledge base aliases. We use the Faiss package [Johnson et al., 2017] to search top nearest neighbours using the cosine similarity measure. We aim from this step to extract the neighbours that share the correct STY class with the query mentions. We use the UMLS knowledge base STY of the nearest neighbour as the predicted label of the query mention.

The embedding vectors of all the aliases in the UMLS knowledge base are stored as Numpy arrays on disk and loaded in batches into the Faiss search index. Each vector is identified by an integer using the Faiss package. The similarity search is done usually using Euclidean distance (L2) or dot product. If the vectors are normalized, the dot product equals cosine similarity. In our search, we set number of expected labels to be 21 which represents number of the STY classes.

4.3.4 Ensemble of Predictions

Loureiro and Jorge [2020] and Mohan et al. [2021] used a Cosine Similarity threshold to make a balance between the BiLSTM and the nearest neighbour search. This is based on the idea that if nearest neighbour with high Cosine Similarity is more accurate in prediction than BiLSTM. In our methods, we use Cosine Similarity threshold and plurality voting.

Before applying Cosine Similarity threshold, we use the two highest performed BiLSTM models and take the label which has the highest Softmax probability. In Cosine Similarity threshold, we try several choices from 0.89 to 0.97 on the portion of the train data (first 3000 samples) and we take the optimal choice and apply it on the test data.

In plurality voting, we use the five model predictions which are the retrieved nearest neighbour STY for each detected mention as explained in Section 4.3.3 and four BiLSTM models with different encoders in Section 4.3.2. We take the most common STY predictions of the five models as the final prediction.

Chapter 5

Results

This chapter will review our observations and results but the analysis will be covered in chapter 6. The results of the Mention Detection will be shown in section 5.1. The results of the Mention Classification will be covered in Section 5.2 on the following page. The results of the Mention Disambiguation task will be shown in Section 5.3 on page 55. The results of combined model will be shown in Section 5.4 on page 55. A final summarized comparison between our models is shown in Section 5.5 on page 57. Classification reports for all the models in this chapter are shown in Appendix A.

5.1 Mention Detection

In this task, we predict 3 labels which are “B-Entity”, “I-Entity” and “O”. In the training and validation curves, we exclude the “O” tags by using mask in the metrics calculations. This means that we only measure the token-level evaluation metrics of “B-Entity” and “I-Entity”.

Token-level F1, Precision, Recall metrics for the BiLSTM model using CODER as feature encoder are shown in Figure 5.1. From these figures, we notice that all the validation data usually shows relatively better F1, Precision and Recall scores than training data at the first 12 epochs of training. For example, validation data at the second epoch shows higher F1, precision and recall scores by approximate margins of 0.03, 0.04 and 0.05 than the scores of the training data respectively. We notice that validation data measured metrics stop increasing after epoch number 12 while training data measured metrics continue to increase slightly. The optimal number of epochs to train the full model on the entire train data is 12. We notice that training curves highly overlap even with the different training dataset and the initial randomness of the weights while the validation curves change slightly for each KFolds. Using 30 epochs, Each KFold takes approximately 20 minutes. This means that training time is approximately 60 minutes for the three KFolds under TPU environments[1].

In terms of strict measures, we predict only one label which is “Entity” and which is “O”. Hence, the recorded strict micro, macro and weighted scores are identical. Using strict measures on test data, the achieved F1, Precision and Recall scores are 0.703, 0.711 and 0.695 respectively. We record these strict metrics on a small portion of train data (first 3000 samples of the train data). We do not measure the strict metrics for the model on the entire train data because merging train data word pieces in an expensive that takes approximately four and half hours for each model. Using the strict measures on the portion of the train data, the achieved F1, Precision and Recall scores are 0.764, 0.757, 0.770 respectively.

The recorded time for training the model over the full data (training and validation data) is approximately 50 seconds for each epoch under TPU environment. This means that the entire training process on the full training data takes approximately 11 minutes for the 12 epochs. We noticed a considerable implication of using word pieces in the training is the duration needed to decode the original words from those word pieces. The duration required to merge all word pieces in the test data is approximately 80 minutes while the time required to merge the word pieces in the portion of the train data is approximately 27 minutes.

Similarly, We record the metrics of the BiLSTM with SciBERT as feature encoder. We notice similar results to what is achieved using CODER as feature encoder. The strict F1, Precision and Recall for using the BiLSTM with SciBERT as feature encoder on the test data are 0.701, 0.694 and 0.708 respectively. By combining both BiLSTM models using the label with highest Softmax probability, we record 0.713, 0.717 and 0.709 for F1, Precision and Recall respectively.

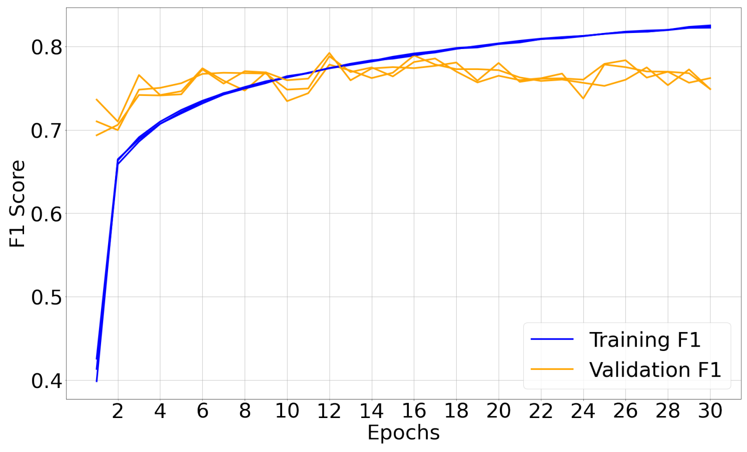
5.2 Mention Classification

In this task, we predict 43 labels. These labels contain the 21 Semantic Types (STYs) that are annotated according to the IOB2 scheme ($21 * 2$ plus the “O” tag).

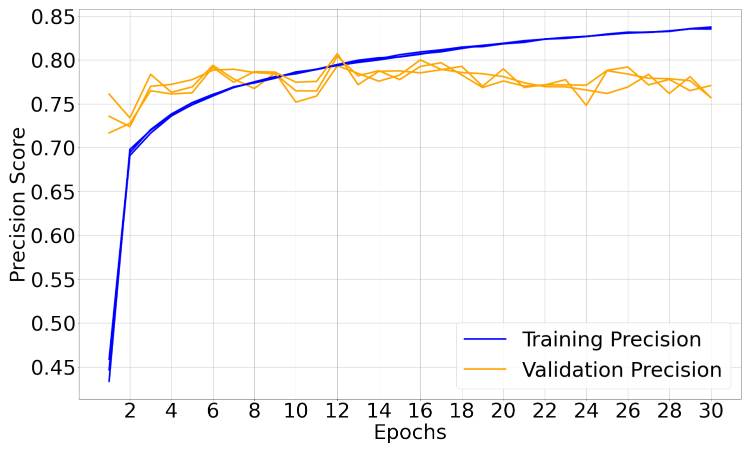
5.2.1 Mention Classification using BiLSTM with CODER as feature encoder

The training and validation curves for F1, Precision, Recall metrics are as shown in the Figure 5.2. From the figures, we notice that all the validation data usually shows relatively better F1, Precision and Recall scores than training data across the first 16, 10, 18 epochs of training respectively. For example, validation data

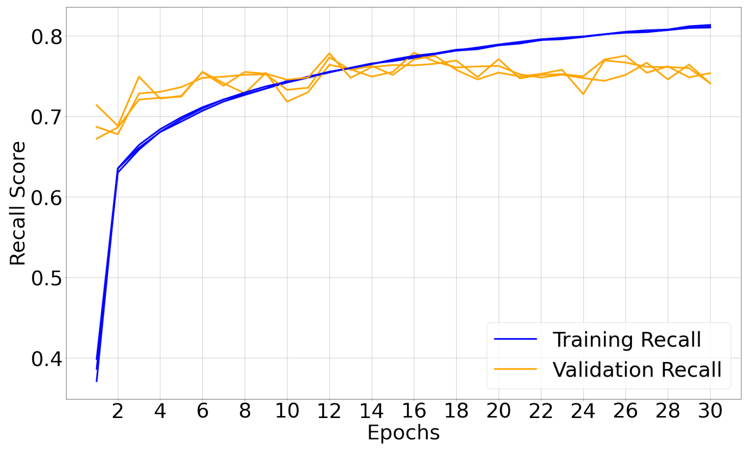
[1]The recorded duration might change according to the availability of the TPU resources.



***(a)** F1*



***(b)** Precision*



***(c)** Recall*

***Figure 5.1:** Mention detection training and validation metrics curves for the BiLSTM model with CODER as feature encoder for three KFolds*

at the second epoch shows higher F1, Precision and Recall scores by approximate margins of 0.15, 0.15 and 0.20 than the scores of the training data. We notice that validation data measured metrics stop changing after epoch number 12 while training data measured metrics continue to increase. The maximum recorded token-level F1, Precision and Recall measures are 0.63, 0.68, 0.61 respectively. We use the optimal number of epochs as 28 where the model show highest F1 measure to train the full model on the entire train data. We notice that training curves highly overlap even with the different training set and the initial randomness of the weights while the validation curves change very slightly for each KFolds. Using 30 epochs, the three KFolds training time is approximately 61 minutes under TPU environments.

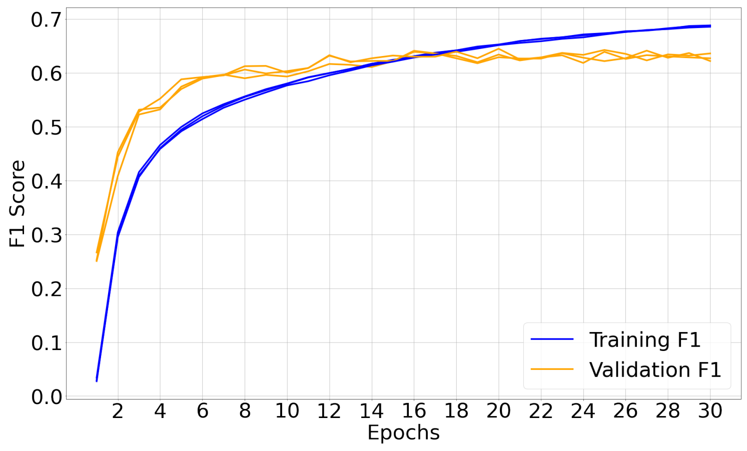
Using test data, the recorded strict micro, macro and weighted F1 are 0.610, 0.563 and 0.605 respectively. The recorded micro, macro and weighted precision scores are 0.605, 0.567 and 0.599 respectively. The recorded micro, macro and weighted recall scores are 0.615, 0.569 and 0.615 respectively. In general, weighted and micro metrics show close values while macro metrics are lower by a margins of approximately 0.047, 0.038 and 0.046 for F1, Precision and Recall scores respectively. Additionally, we record the evaluation metrics of a portion the train data. The overall recorded micro, macro, weighted F1 using the portion of the train data are 0.739, 0.736 and 0.738 respectively. The overall recorded micro, macro, weighted precision are 0.727, 0.730 and 0.727 respectively. Finally, the overall recorded micro, macro, weighted recall are 0.751, 0.747 and 0.751 respectively.

The recorded time for training the model over the full data (training and validation data) is 50 seconds for each epoch under TPU environment. This means that the entire training process on the full data for 28 epochs takes approximately 23 minutes.

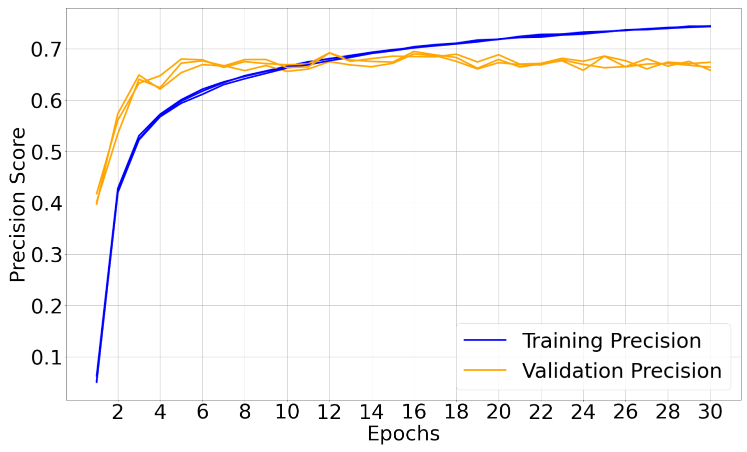
Similar to Mention Detection, the duration required to merge all word pieces in the test data is approximately 80 minutes while the time required to merge the word pieces in the portion of the train data is approximately 27 minutes.

5.2.2 Mention Classification using BiLSTM with SciBERT as feature encoder

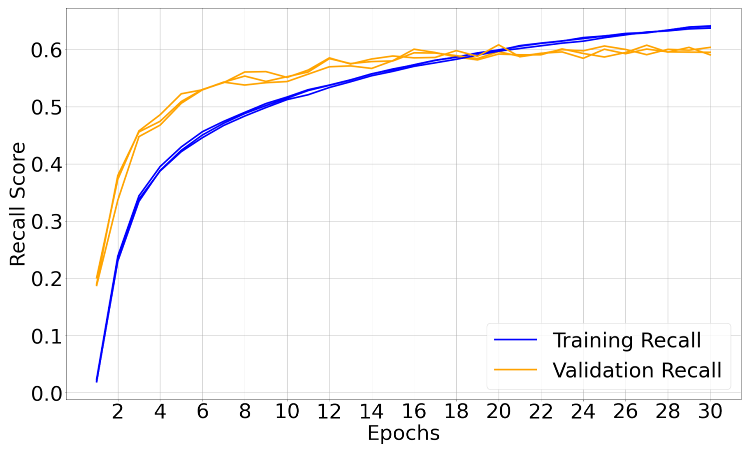
Figure 5.3 shows the training and validation curves for F1, Precision, Recall metrics. Similar to our observations in Section 5.2.1, we notice that all the validation data usually shows relatively better F1, Precision and Recall scores than training data across the first 17, 10, 20 epochs of training respectively before the training metrics increase. We notice that validation data measured metrics are not changing after epoch number 14 while training data measured metrics continue to increase. The maximum recorded F1, Precision and Recall measures are approximately 0.61, 0.68, 0.58 respectively. We use optimal number of epochs as 28 to train the full



***(a)** F1*



***(b)** Precision*



***(c)** Recall*

***Figure 5.2:** Mention classification training and validation metrics curves for the BiLSTM model with CODER as feature encoder for three KFolds*

model on the entire train data. The training curves highly overlap even with the different training set across the KFolds and the initial randomness of the weights and the validation curves are showing similar results. In comparison to the measured metrics for BiLSTM with CODER as feature encoder, using CODER has slightly better performance than using SciBERT.

Using test data, the recorded strict micro, macro and weighted F1 are 0.599, 0.555 and 0.595 respectively. The recorded micro, macro and weighted precision scores are 0.589, 0.556 and 0.584 respectively. The recorded micro, macro and weighted recall scores are 0.609, 0.564 and 0.609 respectively. Additionally, the overall recorded micro, macro, weighted F1 using the portion of the train data are 0.709, 0.697 and 0.707 respectively. The overall recorded micro, macro, weighted precision are 0.696, 0.694 and 0.696 respectively. Finally, the overall recorded micro, macro, weighted recall are 0.722, 0.706 and 0.722 respectively.

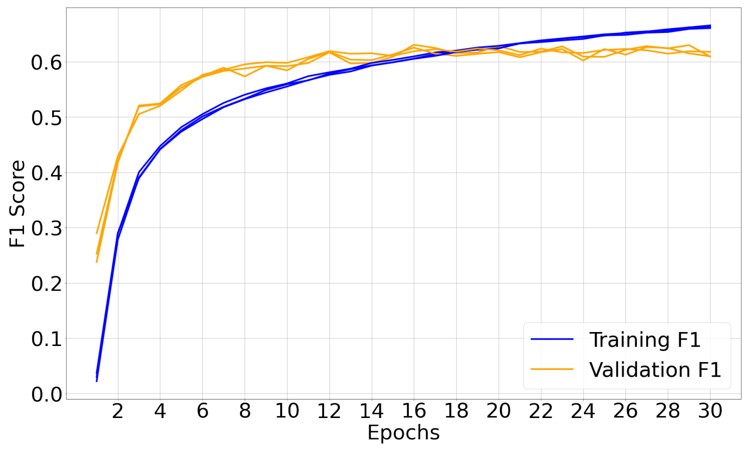
5.2.3 Mention Classification using BiLSTM with UMLSBERT as feature encoder

In general, the model measured metrics show worse performance than using CODER or SciBERT as feature encoder. We notice that the validation F1, Precision and Recall stop increasing after the epoch 16 at approximately 0.54, 0.60, and 0.48 respectively. Figure 5.4 shows the training and validation curves for F1, Precision, Recall metrics. Similar to our observations in Section 5.2.1, we notice that all the validation data usually shows relatively better F1, Precision and Recall scores than training data for the first 20, 18, 24 epochs. We use the optimal number of epochs as 28 to train the full model on the entire train data. Similar to previous observations in Section 5.2.1, the training curves highly overlap while validation metrics change slightly for the different KFolds.

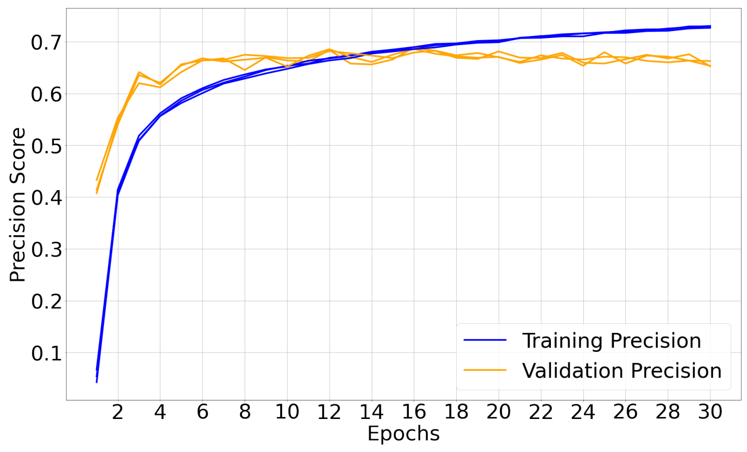
Using test data, the recorded strict micro, macro and weighted F1 are 0.525, 0.473 and 0.519 respectively. The recorded micro, macro and weighted precision scores are 0.534, 0.487 and 0.530 respectively. The recorded micro, macro and weighted recall scores are 0.516, 0.473 and 0.516 respectively. Using the portion of the train data, the overall recorded micro, macro, weighted F1 are 0.638, 0.600 and 0.635 respectively. The overall recorded micro, macro, weighted precision are 0.650, 0.617 and 0.651 respectively. The overall recorded micro, macro, weighted recall are 0.626, 0.594 and 0.626 respectively.

5.2.4 Mention Classification using BiLSTM with BioBERT as feature encoder

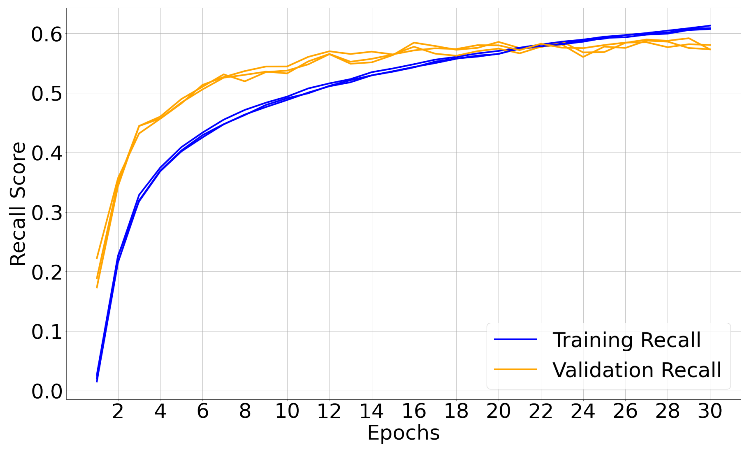
The measured metrics show sub-optimal performance than using CODER or SciBERT as feature encoder. We notice that the validation F1, Precision and Recall stop increasing after the epoch 16 at approximately 0.58, 0.63, and 0.54 respec-



***(a)** F1*

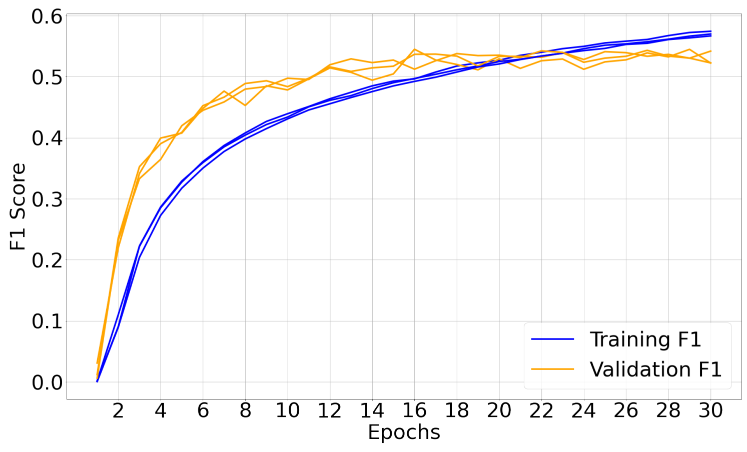


***(b)** Precision*

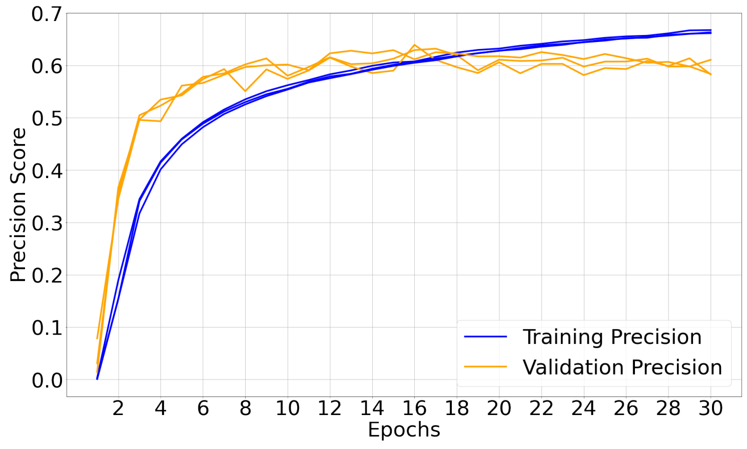


***(c)** Recall*

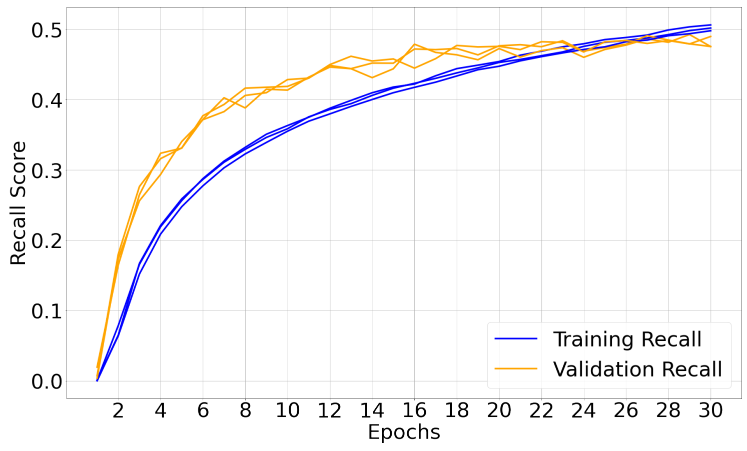
***Figure 5.3:** Mention classification training and validation metrics curves for the BiLSTM model with SciBERT as feature encoder for three KFolds.*



***(a)** F1*



***(b)** Precision*



***(c)** Recall*

***Figure 5.4:** Mention classification training and validation metrics curves for the BiLSTM model with UMLSBERT as feature encoder for three KFolds.*

tively. Figure 5.5 shows the training and validation curves for F1, Precision, Recall metrics. We notice that all the validation data usually shows relatively better F1, Precision and Recall scores than training data for the first 16, 12, 18 epochs. The training curves highly overlap while validation metrics change slightly for the different KFolds.

Using test data, the recorded strict micro, macro and weighted F1 are 0.563, 0.506 and 0.557 respectively. The recorded micro, macro and weighted precision scores are 0.558, 0.511 and 0.551 respectively. The recorded micro, macro and weighted recall scores are 0.568, 0.511 and 0.568 respectively. Using the portion of the train data, the overall recorded micro, macro, weighted F1 are 0.700, 0.685 and 0.698 respectively. The overall recorded micro, macro, weighted precision are 0.691, 0.682 and 0.693 respectively. Finally, the overall recorded micro, macro, weighted recall are 0.710, 0.699 and 0.710 respectively.

5.3 Mention Disambiguation

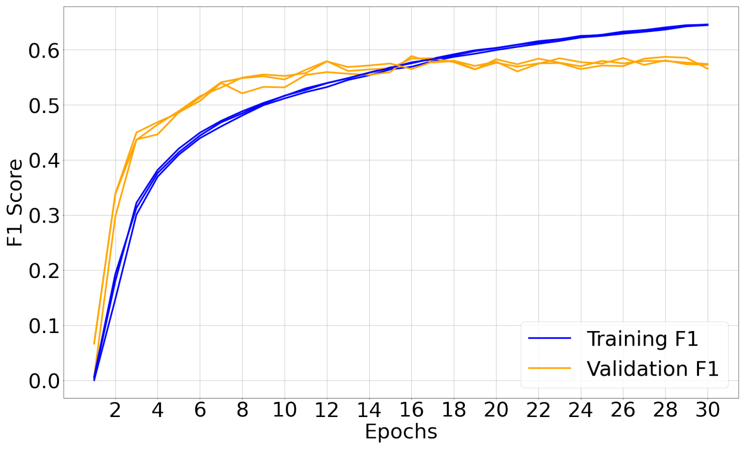
Searching for nearest neighbour for each recognized mention in UMLS knowledge base provides poor results compared to the BiLSTM models results in Section 5.2. Using test data, the recorded strict micro, macro and weighted F1 are 0.482, 0.463 and 0.485 respectively. The recorded micro, macro and weighted precision scores are 0.475, 0.473 and 0.485 respectively. The recorded micro, macro and weighted recall scores are 0.489, 0.467 and 0.489 respectively.

Similarly, we report the evaluation metrics on the same portion of train data (first 3000 samples). The achieved micro, macro, weighted F1 are 0.534, 0.508 and 0.538 respectively. The overall recorded micro, macro, weighted precision are 0.525, 0.507 and 0.537 respectively. Finally, The overall recorded micro, macro, weighted recall are 0.543, 0.515 and 0.543 respectively.

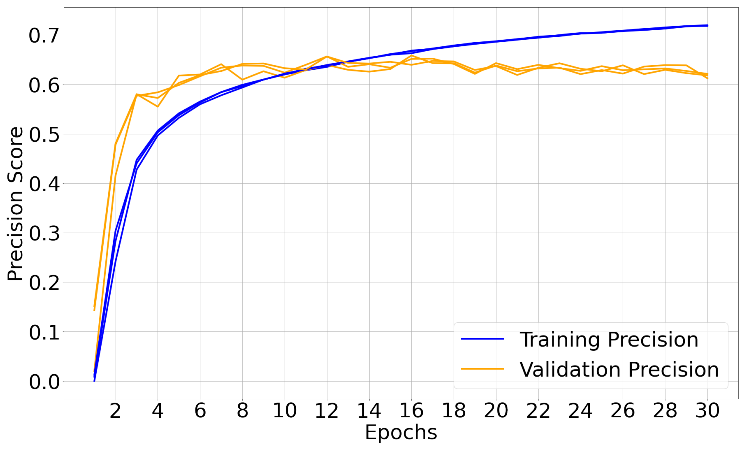
5.4 Ensembles

Figure 5.6 shows the choices of Cosine Similarity thresholds vs the recorded evaluation metrics on the protion of the train data. In Figure 5.6, we notice that optimal threshold is 0.948. By using cosine similarity threshold of 0.948, the achieved micro F1, Precision and Recall scores are 0.629, 0.628 and 0.631 respectively.

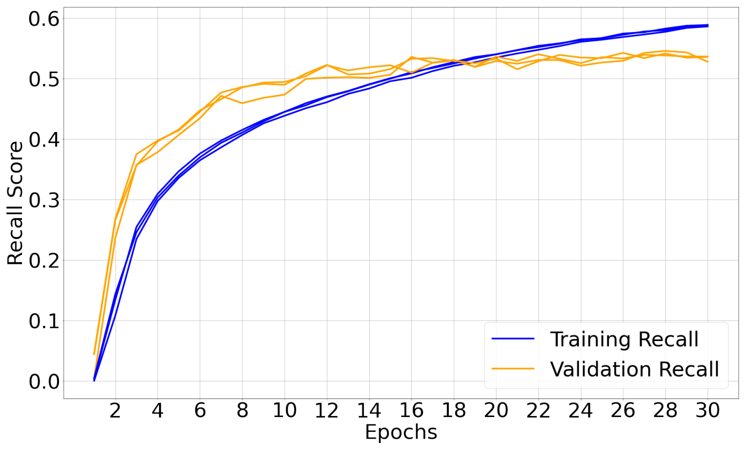
Using plurality voting of the five models, we record 0.626, 0.628 and 0.624 for micro strict F1, Precision and Recall scores respectively.



***(a)** F1*



***(b)** Precision*



***(c)** Recall*

***Figure 5.5:** Mention classification training and validation metrics curves for the BiLSTM model with BioBERT as feature encoder for three KFolds*

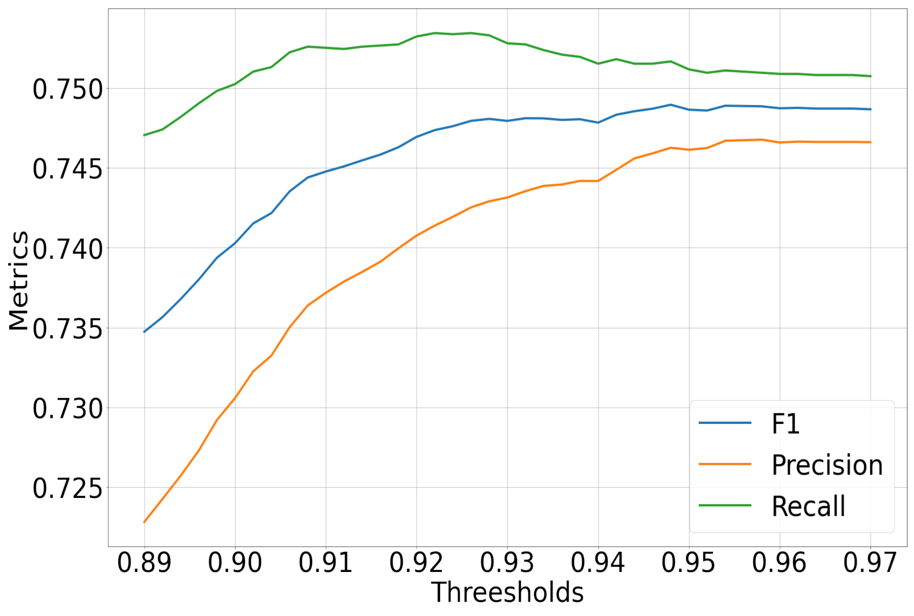


Figure 5.6: *Evaluation metrics vs different Cosine Similarity thresholds on the portion of the train data.*

5.5 Summary

The following table 5.1 summarizes the achieved results from the all the models we use in this study. Using Cosine Similarity threshold, we notice improvement in the overall metrics of the ensemble model over the best performing BiLSTM model models from 0.610, 0.605 and 0.615 to 0.629, 0.628 and 0.631 for micro F1, Precision and Recall scores respectively.

Table 5.2 shows the running time for each step in the pipeline. The most time consuming task is merging the word pieces which takes approxiamtely 103 minutes for merging the entire test data and the portion of the train data.

Table 5.1: *Results achieved by the different BiLSTM models, nearest neighbour search in UMLS knowledge base and the ensemble model on test data*

		Micro	Macro	Weighted
BiLSTM + CODER	F1	0.610	0.563	0.605
	Precision	0.605	0.567	0.599
	recall	0.615	0.569	0.615
BiLSTM + SciBERT	F1	0.599	0.555	0.595
	Precision	0.589	0.556	0.584
	recall	0.609	0.564	0.609
BiLSTM + UMLSBERT	F1	0.525	0.473	0.519
	Precision	0.534	0.487	0.530
	recall	0.516	0.473	0.516
BiLSTM + BioBERT	F1	0.563	0.506	0.557
	Precision	0.558	0.511	0.551
	recall	0.568	0.511	0.568
Nearest Neighbour	F1	0.481	0.462	0.484
	precision	0.474	0.472	0.485
	recall	0.488	0.465	0.488
Plurality Voting	F1	0.626	0.586	0.620
	precision	0.628	0.606	0.622
	recall	0.624	0.580	0.624
Similarity Threshold	F1	0.629	0.593	0.624
	precision	0.628	0.606	0.622
	recall	0.631	0.592	0.631

Table 5.2: *Run times in minutes for of different steps of the pipeline*

		Accelerator	Run Time
Mention Detection	Cross Validation	TPU	60
	Optimal Model	TPU	11
	Merge Word Pieces	CPU	103
Mention Classification	Cross Validation	TPU	60
	Optimal Model	TPU	23
	Merge Word Pieces	CPU	103
Build Embeddings		GPU	30
Nearest Neighbour		GPU	60

Chapter 6

Discussion

For all the measured metrics, we noticed that validation metrics usually saturate at a certain value. This has been noticed in the SOTA results by Fraser et al. [2019]. According to the authors of the MedMentions [Mohan and Li, 2019, Mohan et al., 2021], “we attribute the low performance of SOTA models on MedMentions to its low resource nature”. This means that the training data has low coverage of the concepts as explained earlier in Section 3.1.

We noticed that the recorded strict metrics for the portion of the train data is relatively higher than the recorded strict metrics for test data. The reason for that is the small size of this portion which is not a real representation of the entire train data. However, we still use it as we try different implementation of ensembles in order to have methods that show a generalization on train and test data.

6.1 Mention Detection

We reported in Section 5.1 from the curves that validation data shows better evaluation metrics than training data at the first epochs of training for the two BiLSTM models. We have two possible explanation for that. First is the small size of the validation data which is only 10000 samples of the train data which might has effect on the number of mis-classifications. Second reason is that the validation data are less challenging to be predicted correctly compared to the train data. In other words, in the first epochs the model already captured the patterns that made it predicted the validation data correctly. The validation data measured metrics stopped improving after epoch 13 while train data measured metrics continued to improve as the model is overfitting the train data.

We noticed that training curves for the three different KFold as almost identical which gave an indication of the robustness of the model and the ability to reproduce the results easily. Models from the different KFolds show difference between validation curves in the first few epochs but they overlapped later as the model continue in training.

The overall measured strict metrics for F1, Precision and Recall are 0.703, 0.711 and 0.695 respectively. This result is identical to the highest achieved results that was done by Loureiro and Jorge [2020] for mention detection which used SciBERT Beltagy et al. [2019] as a feature encoder. Our models achieved comparable results and the last combined model achieved higher results than Loureiro and Jorge [2020] mention detection system by very small margins of 0.007 and 0.017 for F1 and Precision respectively. Since the labels are updated in the combined model, the mentions need to re-merge according to the new labels which is time consuming task given that the marginal increases in the metrics are not large. So, we still use that BiLSTM + CODER mention for the Mention Disambiguation. However, we believe this is still poor results and has implications on the nearest neighbour search. This is because of the propagation of error through the pipeline as we detect not accurate mentions that makes the nearest neighbour search more prone to errors.

Table 6.1: *Results comparison between the achieved metrics from Medlinker mention detection by Loureiro and Jorge [2020]. Metrics in bold show the highest achieved in each metric.*

	Micro F1	Micro Precision	Micro Recall
Exact match [Loureiro and Jorge, 2020]	0.401	0.513	0.330
BiLSTM + NCBIBERT [Loureiro and Jorge, 2020]	0.694	0.694	0.694
BiLSTM + SciBERT [Loureiro and Jorge, 2020]	0.706	0.694	**0.718**
BiLSTM + BioBERT [Loureiro and Jorge, 2020]	0.702	0.700	0.704
Our model BiLSTM + SciBERT	0.701	0.694	0.708
Our model BiLSTM + CODER	0.703	0.711	0.695
Our Ensemble Model	**0.713**	**0.717**	0.709

We noticed that difference in performances between using SciBERT [Beltagy et al., 2019] and CODER [Yuan et al., 2020] as feature encoder is very small. This observation is similar to what have been reported by Loureiro and Jorge [2020] where the difference between the metrics of the various encoders is very small as shown in Table 6.1. For example, the recorded strict F1 using BioBERT [Lee et al., 2019] and SciBERT [Beltagy et al., 2019] are 0.694 and 0.706 respectively.

6.2 Mention Classification

Similar to Mention Detection, we reported in Section 5.2 from the curves that validation data shows better evaluation metrics than training data at the first 12 epochs of training for the different models that is attributed to the small size for the validation data compared to the train data.

We noticed that BiLSTM with CODER as feature encoder achieved the optimal results compared to the other models. This means that CODER provides relatively better representation for the concepts in the data which support the claims of Yuan et al. [2020] that CODER could provide better concept embeddings than BioBERT [Lee et al., 2019] and SciBERT [Beltagy et al., 2019]. However, this is subject to our implementation of the methods. This means that slight changes in the implementation of the methods might show that other models such as BioBERT [Lee et al., 2019] and SciBERT [Beltagy et al., 2019] provide better representation. Loureiro and Jorge [2020] showed that SciBERT [Beltagy et al., 2019] is the optimal model for feature encoding while [Fraser et al., 2019] showed that a concatenation of the last layers of both BERT [Devlin et al., 2019] and BioBERT [Lee et al., 2019] provide the optimal model for feature encoding.

6.3 Mention Disambiguation

We noticed relatively poor results for Mention Disambiguation using nearest neighbour search. We compared our results with the MedLinker system [Loureiro and Jorge, 2020] nearest neighbour results and we noticed a significant difference. Our measured strict F1 on test data is 0.482 while the MedLinker achieved 0.588. Our measured strict Precision on test data is 0.463 while the MedLinker achieved 0.531. Finally, our measured strict Recall on test data is 0.485 while the MedLinker achieved 0.659. We found these observations interesting taking into account that we recorded similar results to Loureiro and Jorge [2020] for mention detection which means there are still a room for development for our implementation of nearest neighbour search.

The poor results of nearest neighbour search might be attributed to two reasons. First reason is related to the complexity of data itself in which the used method of combining the sentence and mention embeddings was not helpful in the search. Second reason is related to the entity detection itself as it showed F1 score 0.703 which affected the nearest neighbour search as the errors propagated through the pipeline. One possible solution is by using other pretrained models such as SciBERT and BioBERT which could be able to achieve higher results.

6.4 Comparison with other studies

We noticed that combining nearest neighbour search predictions with the different BiLSTM models predictions has increased the overall measured metrics. This increase has been recorded in train and test data which means that our implementation of the ensemble is robust. The table 6.2 summarizes the achieved metrics for our model vs several other models from related studies. The State Of The Art (SOTA) is achieved by Fraser et al. [2019]. Our model is still behind the SOTA model by a F1 margin of 0.011. that is attributed of having lower Recall that the other studies.

Table 6.2: *Results comparison between the achieved metrics from different studies. Results in bold show the highest achieved for each metric. Note that exact match, QuickUMLS and ScispaCy measured metrics are reported by Loureiro and Jorge [2020].*

	Micro F1	Micro Precision	Micro Recall
Exact match [Loureiro and Jorge, 2020]	0.387	0.490	0.32
QuickUMLS [Soldaini, 2016]	0.156	0.145	0.169
ScispaCy [Neumann et al., 2019]	0.154	0.101	0.317
Fraser et al. [2019]	**0.640**	0.630	**0.650**
MedLinker [Loureiro and Jorge, 2020]	0.634	**0.631**	0.637
Our Optimal Model	0.629	0.628	0.631

Chapter 7

Conclusions and Future Work

Despite that Named Entity Recognition and Entity Linking are relatively easy tasks in general domains and are considered by many researchers as a solved task, the medical domain text data still provides a challenging and interesting task to solve. The challenges come from the ambiguity of the medical terms which can not be easily disambiguated using traditional methods such as string surface similarity. Our method implementations and observations lead us to a conclusion about the complexity of the task and give us hints about what could be possible directions we can follow for future work to build more accurate models.

The optimal model in this thesis achieved micro F1 of 0.629 which is 1.1 micro F1 point behind the SOTA model by Fraser et al. [2019] that achieved 0.64. The current SOTA provides a challenging baseline to outperform. We tried several methods to achieve higher results than the baseline. We leveraged different pre-trained models for our methods. We hope by our work to provide a good starting point for other researchers to further develop the methods.

We noticed that combination of different BiLSTM with various BERT models as encoder with nearest neighbour search helped to boost the performance of the highest performed BiLSTM model. Further work and development of the implemented models are still needed. Several directions can be taken in order to improve the achieved results such as including other BERT models in the combined model.

There is still room for development in our Mention Detection system that could be able to achieve higher results than our recorded micro F1 of 0.703. With higher predictions results from the Mention Detection, the nearest neighbour could be helpful to improve the results of Mention Disambiguation.

Further development of the Mention Disambiguation system could also be helpful in boosting the overall performance. Since we noticed that combination of various

BERT models as feature encoder for BiLSTM was helpful in boosting the performance, a combination of various BERT models could also be helpful in the nearest neighbour search.

Until now, we use our methods to predict the STY of the mentions in the data which are only 21 Semantic Types (STY). It will be more challenging to predict Concept Unique Identifiers (CUI) of the entities as we have many CUI labels covered in the data as explained earlier in Section 3.1.

Bibliography

J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization, 2016.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2016.

I. Beltagy, K. Lo, and A. Cohan. Scibert: Pretrained language model for scientific text. In *EMNLP*, 2019.

O. Bodenreider. The unified medical language system (umls): integrating biomedical terminology. *Nucleic acids research*, 32 Database issue:D267–70, 2004.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

R. I. Dogan, R. Leaman, and Z. Lu. Ncbi disease corpus: A resource for disease name recognition and concept normalization. *Journal of biomedical informatics*, 47:1–10, 2014.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61): 2121–2159, 2011. URL `http://jmlr.org/papers/v12/duchi11a.html`.

R. Feldman, M. Fresko, Y. Kinar, Y. Lindell, O. Liphstat, M. Rajman, J. Schler, and O. Zamir. Text mining at the term level. In *PKDD*, 1998.

K. C. Fraser, I. Nejadgholi, B. D. Bruijn, M. Li, A. LaPlante, and K. Z. E. Abidine. Extracting umls concepts from medical text using general and domain-specific deep learning models, 2019.

N. S. Geoffrey Hinton and K. Swersky. Neural networks for machine learning, 2012. URL `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`. [Online; accessed 14-May-2021].

F. Gers, urgen Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm learning to forget: Continual prediction with lstm. 1999.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

D. Harris and S. Harris. igital design and computer architecture (2nd ed.). *Proceedings of the IEEE*, 78(10):129, 1990.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

A. Hotho, A. Nürnberger, and G. Paass. A brief survey of text mining. *LDV Forum*, 20:19–62, 2005.

S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

S. Kaewphan, K. Hakala, N. Miekka, T. Salakoski, and F. Ginter. Wide-scope biomedical named entity recognition and normalization with crfs, fuzzy matching and character level modeling. *Database: The Journal of Biological Databases and Curation*, 2018, 2018.

S. Karimi, A. Metke-Jimenez, M. Kemp, and C. Wang. Cadec: A corpus of adverse drug event annotations. *Journal of biomedical informatics*, 55:73–81, 2015.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2017.

J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C. H. So, and J. Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 09 2019. ISSN 1367-4803. doi: 10.1093/bioinformatics/btz682. URL `https://doi.org/10.1093/bioinformatics/btz682`.

J. Li, Y. Sun, R. J. Johnson, D. Sciaky, C.-H. Wei, R. Leaman, A. P. Davis, C. J. Mattingly, T. C. Wiegers, and Z. Lu. BioCreative V CDR task corpus: a resource for chemical disease relation extraction. *Database*, 2016, 05 2016. ISSN 1758-0463. doi: 10.1093/database/baw068. URL `https://doi.org/10.1093/database/baw068`. baw068.

K. Lo, L. L. Wang, M. Neumann, R. Kinney, and D. Weld. S2ORC: The semantic scholar open research corpus. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4969–4983, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.447. URL `https://www.aclweb.org/anthology/2020.acl-main.447`.

D. Loureiro and A. M. Jorge. Medlinker: Medical entity linking with neural representations and dictionary matching. In J. M. Jose, E. Yilmaz, J. Magalhães,

P. Castells, N. Ferro, M. J. Silva, and F. Martins, editors, *Advances in Information Retrieval*, pages 230–237, Cham, 2020. Springer International Publishing. ISBN 978-3-030-45442-5.

G. Michalopoulos, Y. Wang, H. Kaka, H. Chen, and A. Wong, 2021.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In Y. Bengio and Y. LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL `http://arxiv.org/abs/1301.3781`.

S. Mohan and D. Li. Medmentions: A large biomedical corpus annotated with UMLS concepts. *CoRR*, abs/1902.09476, 2019. URL `http://arxiv.org/abs/1902.09476`.

S. Mohan, R. Angell, N. Monath, and A. McCallum. Low resource recognition and linking of biomedical concepts from a large ontology, 2021.

H. Nakayama. seqeval: A python framework for sequence labeling evaluation, 2018. URL `https://github.com/chakki-works/seqeval`. Software available from https://github.com/chakki-works/seqeval.

I. Nejadgholi, K. C. Fraser, and B. D. Bruijn. Extensive error analysis and a learning-based evaluation of medical entity recognition systems to approximate user experience, 2020.

Y. Nesterov. A method for solving the convex programming problem with convergence rate o(1/k^2). *Proceedings of the USSR Academy of Sciences*, $269 : 543 - -547, 1983$.

M. Neumann, D. King, I. Beltagy, and W. Ammar. ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing. In *Proceedings of the 18th BioNLP Workshop and Shared Task*, pages 319–327, Florence, Italy, Aug. 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-5034. URL `https://www.aclweb.org/anthology/W19-5034`.

N. Okazaki and J. Tsujii. Simple and efficient algorithm for approximate dictionary matching. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 851–859, Beijing, China, August 2010. URL `http://www.aclweb.org/anthology/C10-1096`.

R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks, 2013.

J. Pennington, R. Socher, and C. Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, Oct. 2014.

Association for Computational Linguistics. doi: 10.3115/v1/D14-1162. URL https://www.aclweb.org/anthology/D14-1162.

M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations, 2018.

B. Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964. ISSN 0041-5553. doi: https://doi.org/10.1016/0041-5553(64)90137-5. URL https://www.sciencedirect.com/science/article/pii/0041555364901375.

D. Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness correlation. 2008.

S. Raschka and V. Mirjalili. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition*. Packt Publishing, 2019. ISBN 9781789958294. URL https://books.google.no/books?id=sKXIDwAAQBAJ.

N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986. doi: 10.1038/323533a0. URL http://www.nature.com/articles/323533a0.

M. Schuster and K. Paliwal. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.*, 45:2673–2681, 1997.

L. Soldaini. Quickumls: a fast, unsupervised approach for medical concept extraction. 2016.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2017.

X. Wang, X. Han, W. Huang, D. Dong, and M. R. Scott. Multi-similarity loss with general pair weighting for deep metric learning, 2020.

P. Werbos. Backpropagation through time: What it does and how to do it. 1990.

Wikipedia. File:recurrent neural network unfold.svg — wikimedia commons, the free media repository, 2020a. URL https://commons.wikimedia.

org/w/index.php?title=File:Recurrent_neural_network_unfold.svg&oldid=447577937. [Online; accessed 15-May-2021].

Wikipedia. File:stogra.png — wikimedia commons, the free media repository, 2020b. URL https://commons.wikimedia.org/w/index.php?title=File:Stogra.png&oldid=511318085. [Online; accessed 15-May-2021].

Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, Łukasz Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean. Google's neural machine translation system: Bridging the gap between human and machine translation, 2016.

Z. Yuan, Z. Zhao, and S. Yu. Coder: Knowledge infused cross-lingual medical term embedding for term normalization. *arXiv preprint arXiv:2011.02947*, 2020.

A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into Deep Learning*. 2020. https://d2l.ai.

Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books, 2015.

M. Zolnoori, K. W. Fung, T. B. Patrick, P. Fontelo, H. Kharrazi, A. Faiola, N. D. Shah, Y. S. Shirley Wu, C. E. Eldredge, J. Luo, M. Conway, J. Zhu, S. K. Park, K. Xu, and H. Moayyed. The psytar dataset: From patients generated narratives to a corpus of adverse drug events and effectiveness of psychiatric medications. *Data in Brief*, 24:103838, 2019. ISSN 2352-3409. doi: https://doi.org/10.1016/j.dib.2019.103838. URL https://www.sciencedirect.com/science/article/pii/S2352340919301891.

Appendix A

Table A.1: *Strict evaluation metrics of the BiLSTM with CODER as feature encoder.*

STY	Precision	Recall	F1	Support
anatomical-structure	0.575	0.572	0.574	3768
bacterium	0.664	0.670	0.667	449
biologic-function	0.635	0.667	0.651	8102
biomedical-discipline	0.418	0.337	0.373	196
body-substance	0.603	0.679	0.639	212
body-system	0.569	0.326	0.414	89
chemical	0.712	0.774	0.741	7396
clinical-attribute	0.543	0.588	0.565	323
eukaryote	0.699	0.728	0.713	1748
finding	0.474	0.330	0.389	3209
food	0.487	0.481	0.484	322
health-care-activity	0.572	0.609	0.590	4784
injury-or-poisoning	0.559	0.694	0.576	352
intellectual-product	0.493	0.429	0.459	2364
medical-device	0.472	0.310	0.374	355
organization	0.544	0.565	0.555	383
population-group	0.674	0.673	0.674	1263
professional-group	0.576	0.739	0.647	360
research-activity	0.573	0.666	0.616	1847
spatial-concept	0.466	0.505	0.485	2406
virus	0.589	0.709	0.644	172

Table A.2: *Strict evaluation metrics of the BiLSTM with SciBERT as feature encoder.*

STY	Precision	Recall	F1	Support
anatomical-structure	0.536	0.573	0.554	3768
bacterium	0.602	0.670	0.634	449
biologic-function	0.620	0.662	0.641	8102
biomedical-discipline	0.526	0.362	0.429	196
body-substance	0.667	0.642	0.654	212
body-system	0.517	0.337	0.408	89
chemical	0.712	0.756	0.733	7396
clinical-attribute	0.465	0.588	0.519	323
eukaryote	0.660	0.722	0.690	1748
finding	0.463	0.320	0.379	3209
food	0.465	0.416	0.439	322
health-care-activity	0.554	0.593	0.573	4784
injury-or-poisoning	0.565	0.520	0.541	352
intellectual-product	0.468	0.453	0.461	2364
medical-device	0.482	0.341	0.399	355
organization	0.526	0.529	0.527	383
population-group	0.670	0.715	0.692	1263
professional-group	0.580	0.672	0.623	360
research-activity	0.554	0.685	0.612	1847
spatial-concept	0.467	0.497	0.481	2406
virus	0.587	0.785	0.672	172

Table A.3: *Strict evaluation metrics of the BiLSTM with UMLSBERT as feature encoder.*

STY	Precision	Recall	F1	Support
anatomical-structure	0.500	0.447	0.472	3768
bacterium	0.496	0.470	0.483	449
biologic-function	0.560	0.583	0.571	8102
biomedical-discipline	0.370	0.224	0.279	196
body-substance	0.548	0.566	0.557	212
body-system	0.386	0.247	0.301	89
chemical	0.611	0.657	0.633	7396
clinical-attribute	0.592	0.489	0.536	323
eukaryote	0.565	0.564	0.564	1748
finding	0.478	0.256	0.334	3209
food	0.445	0.329	0.379	322
health-care-activity	0.514	0.518	0.516	4784
injury-or-poisoning	0.520	0.483	0.501	352
intellectual-product	0.404	0.353	0.377	2364
medical-device	0.229	0.273	0.249	355
organization	0.393	0.495	0.438	383
population-group	0.694	0.597	0.642	1263
professional-group	0.436	0.661	0.525	360
research-activity	0.521	0.619	0.566	1847
spatial-concept	0.454	0.386	0.417	2406
virus	0.512	0.715	0.597	172

Table A.4: *Strict evaluation metrics of the BiLSTM with BioBERT as feature encoder.*

STY	Precision	Recall	F1	Support
anatomical-structure	0.528	0.483	0.505	3768
bacterium	0.538	0.530	0.534	449
biologic-function	0.596	0.618	0.607	8102
biomedical-discipline	0.339	0.286	0.310	196
body-substance	0.577	0.618	0.597	212
body-system	0.431	0.281	0.340	89
chemical	0.642	0.740	0.687	7396
clinical-attribute	0.517	0.511	0.514	323
eukaryote	0.616	0.680	0.647	1748
finding	0.455	0.308	0.367	3209
food	0.513	0.360	0.423	322
health-care-activity	0.526	0.544	0.535	4784
injury-or-poisoning	0.531	0.514	0.522	352
intellectual-product	0.443	0.420	0.431	2364
medical-device	0.385	0.217	0.277	355
organization	0.437	0.518	0.474	383
population-group	0.614	0.683	0.646	1263
professional-group	0.523	0.683	0.593	360
research-activity	0.551	0.661	0.601	1847
spatial-concept	0.446	0.455	0.450	2406
virus	0.589	0.622	0.565	172

Table A.5: *Strict evaluation of nearest neighbour search.*

STY	Precision	Recall	F1	Support
anatomical-structure	0.464	0.437	0.450	3768
bacterium	0.741	0.548	0.630	449
biologic-function	0.597	0.546	0.570	8102
biomedical-discipline	0.271	0.383	0.317	196
body-substance	0.483	0.604	0.537	212
body-system	0.568	0.236	0.333	89
chemical	0.547	0.576	0.561	7396
clinical-attribute	0.239	0.285	0.260	323
eukaryote	0.456	0.593	0.515	1748
finding	0.287	0.390	0.330	3209
food	0.526	0.410	0.461	322
health-care-activity	0.418	0.428	0.461	4784
injury-or-poisoning	0.519	0.428	0.424	352
intellectual-product	0.311	0.324	0.317	2364
medical-device	0.299	0.338	0.317	355
organization	0.490	0.466	0.478	383
population-group	0.703	0.590	0.642	1263
professional-group	0.523	0.672	0.588	360
research-activity	0.493	0.534	0.513	1847
spatial-concept	0.428	0.399	0.413	2406
virus	0.572	0.576	0.574	172

Table A.6: *Strict evaluation of the plurality voting ensemble model (BiLSTM models + Nearest Neighbour).*

STY	Precision	Recall	F1	Support
anatomical-structure	0.597	0.572	0.585	3768
bacterium	0.696	0.677	0.686	449
biologic-function	0.650	0.679	0.665	8102
biomedical-discipline	0.514	0.383	0.439	196
body-substance	0.650	0.693	0.671	212
body-system	0.689	0.348	0.463	89
chemical	0.715	0.796	0.753	7396
clinical-attribute	0.625	0.594	0.610	323
eukaryote	0.697	0.743	0.719	1748
finding	0.531	0.332	0.409	3209
food	0.542	0.438	0.485	322
health-care-activity	0.589	0.606	0.597	4784
injury-or-poisoning	0.591	0.554	0.572	352
intellectual-product	0.509	0.437	0.470	2364
medical-device	0.494	0.335	0.399	456
organization	0.560	0.573	0.567	383
population-group	0.732	0.705	0.718	1263
professional-group	0.578	0.728	0.645	360
research-activity	0.590	0.679	0.631	1847
spatial-concept	0.528	0.506	0.517	2406
virus	0.641	0.808	0.715	172

Table A.7: *Strict evaluation of the Cosine Similarity threshold ensemble model (BiLSTM models + Nearest Neighbour).*

STY	Precision	Recall	F1	Support
anatomical-structure	0.603	0.578	0.590	3768
bacterium	0.709	0.679	0.694	449
biologic-function	0.657	0.685	0.671	8102
biomedical-discipline	0.477	0.423	0.449	196
body-substance	0.642	0.703	0.671	212
body-system	0.689	0.348	0.463	89
chemical	0.714	0.796	0.753	7396
clinical-attribute	0.645	0.613	0.629	323
eukaryote	0.696	0.749	0.721	1748
finding	0.531	0.332	0.409	3209
food	0.563	0.472	0.514	322
health-care-activity	0.585	0.606	0.597	4784
injury-or-poisoning	0.602	0.568	0.585	352
intellectual-product	0.501	0.445	0.471	2364
medical-device	0.485	0.369	0.419	456
organization	0.561	0.581	0.571	383
population-group	0.736	0.713	0.724	1263
professional-group	0.583	0.742	0.653	360
research-activity	0.589	0.682	0.632	1847
spatial-concept	0.527	0.517	0.522	2406
virus	0.641	0.820	0.719	172