Mining Text Data Using Recurrent Neural Networks

Data Science Report, NMBU, Spring 2021

Mohamed Radwan
mohamed.radwan@nmbu.no

Abstract

In this report, we will apply named entity recognition on the Groningen Meaning Bank (GMB) corpus [Bos et al. 2017]. We will use the Long Short Term Memory (LSTM) neural network architecture for Named Entity Recognition (NER). Application of LSTM on this dataset gave a non-strict F1 score of 0.9633 on test data and strict F1 score of 0.793. By customizing the embedding layer in the network through adding pretrained weights into the embedding, the non-strict F1 score improved slightly and reached 0.9695 and the strict F1 score is 0.810. The increase in strict F1 score is around 0.017 and it gives us an impression that GloVe has better representation of the words in the data. This means that the embedding model GloVe is slightly more accurate than the vectors that are being trained within the used model.

1 Introduction

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 don't follow an organized format. This makes the analysis of text data 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 in 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".

Text mining employs machine learning to automate the analysis of the text data. Recurrent Neural Networks (RNNs) are specific neural network architecture that is particularly suited for sequence and text data [Rumelhart et al. 1986].

Named Entity Recognition is a critical step in many applications such as question answering (QA), information extraction and building chatbot systems. 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. In this report, we aim to use RNNs to extract entities from text. In Section 2, we will explain the theoretical details behind the the methods. In Section 4, we explain the workflow we use in this study. 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.

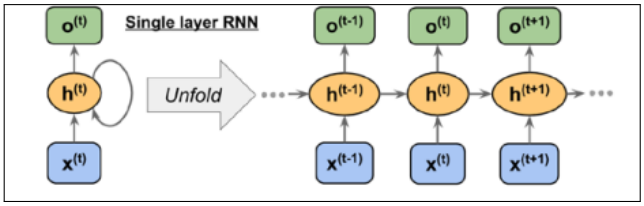

Figure 1: Single hidden layer RNN. The units $(h^{(t-1)}, h^{(t-1)}, h^{(t+1)} \dots)$ represent the the hidden units at different time steps while $(x^{(t-1)}, x^{(t-1)}, x^{(t+1)} \dots)$ represent the items in the input sequence at different time steps. Source: [Raschka and Mirjalili 2019]

2 Theory

Recurrent neural networks or RNNs [Rumelhart et al. 1986] are suitable to process text data because they can leverage the order of the items in the sequence. In other words, RNNs can be seen that it has memory to the items in the sequence. RNNs is multiple copies of a hidden units (h^{t-1}, h^{t}, h^{t+1}...*etc.*), as shown in Figure 1, and each copy is passing information to its successor. This way of passing information along is what makes RNNs powerful in capturing the relationship between the elements of sequences. In RNNs, the hidden layer, as shown in Figure 1, 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 (figure 1). The hidden unit $h^{(t)}$ takes input data item $x^{(t)}$ in addition to the output of previous time step $h^{(t-1)}$

There are several types of RNNs based on the shapes of inputs and outputs as shown in figure 2. Many-to-one RNN is used when the output is a label such as sentiment analysis. One-to-many RNN are used to extract sequences from one input. Image captioning is an example of a one-to-many architecture where the input image is a single non-sequential data point and output is generated text [Vinyals et al. 2015]. Many-to-many architecture is used when both the input and output are sequences. Many-to-many architecture can be subdivided based on whether the input and output are synchronized. An example of a synchronized many-to-many task is video frame-by-frame image classification. An example of a delayed or asynchronized many-to-many modeling task would be machine

translation where the input sentence must be entirely processed before its translation is produced (See Figure 2).

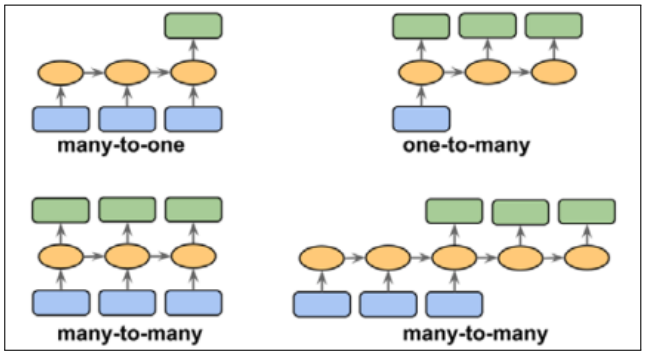

Figure 2: Main architectures of RNNs. Source:[Raschka and Mirjalili 2019]

2.1 Vanishing and Exploding Gradient

An interesting and challenging problem with training RNNs on long sequences is what is known as vanishing and exploding gradient [Pascanu et al. 2013]. This problem occurs when the absolute value of the recurrent connection weight becomes less than 1 or greater than 1 as shown in figure 3.

Essentially, the loss function does not only depends on the neurons that participated in the calculation of the output but also on the contribution of these neurons back in time. So, backpropgation of errors has to be done all the way back through time to these neurons. The full derivation of the backpropagation through time is explained in [Werbos 1990]. The loss function is dependent on the hidden units at all time steps $(1 : t)$. The derivative of the loss function will be:

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

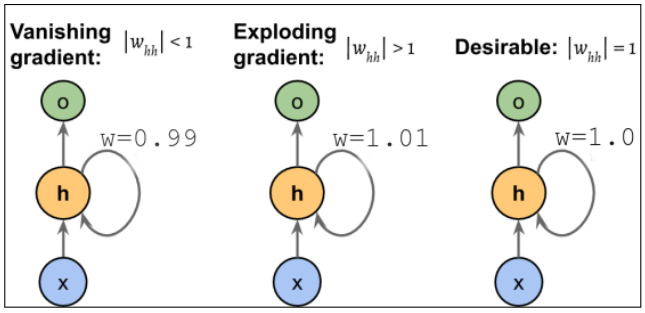

Figure 3: Vanishing and exploding gradient. W_{hh} is the recurrent connection weight. Source: [Raschka and Mirjalili 2019]

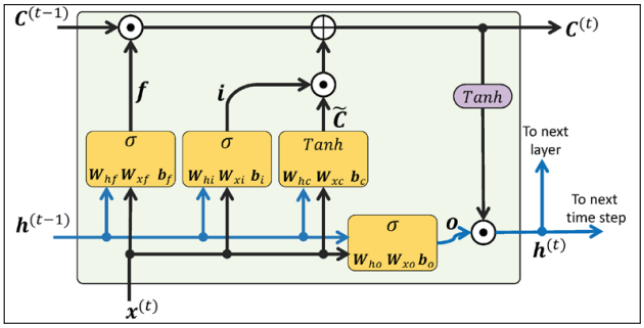

Figure 4: LSTM cell architecture. The symbol $\odot$ is an elementwise product while the symbol $\oplus$ is an elementwise summation. Forget gate f decides what information to delete from the memory. Input gate i decides what information $\tilde{C}$ to store. Output gate o decides what to output. Source: [Raschka and Mirjalili 2019]

where k refers to the different time steps and o_t refers to the activations of the output unit. The derivative $\frac{\partial h^{(t)}}{\partial h^{(k)}}$ is a product of adjacent time steps as shown in Eq. 2.

$$\frac{\partial h^{(t)}}{\partial h^{(k)}} = \prod_{i=k+1}^{t} \frac{\partial h^{(i)}}{\partial h^{(i-1)}} \tag{2}$$

In Eq. 2, the term is multiplied $t - k$ times. This means multiplication of the weight W_{hh} by itself $t - k$ times which would result in vanishing gradient if W_{hh} is less than 1 and vanishing exploding if W_{hh} is greater than 1. One possible solution to tackle this problem is by using the Long Short-Term Memory network.

2.2 Long Short Term Memory

Several techniques are used to tackle the vanishing and exploding gradient problem. One of them is the Long Short-Term Memory or LSTM. LSTM is RNN architecture and was introduced by [Hochreiter and Schmidhuber 1997] and further developed by [Gers et al. 1999] and is used to capture long-term dependencies. LSTM uses gates to delete and add information from earlier states (Fig. 4). These gates are as follows:

Forget gate is a sigmoid function that takes in the output of the previous layer and the current layer input. The forget gate dtermines which information to keep and which information to suppress. The forget gate is shown in Eq. 3.

$$f_t = \sigma(W_{xf} x^{(t)} + W_{hf} h^{(t-1)} + b_f) \tag{3}$$

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

Input gate has two branches. The first branch is a sigmoid that determines what will be updated:

$$i_t = \sigma(W_{xi}x^{(t)} + W_{hi}h^{(t-1)} + b_i) \quad (4)$$

Here, W_{xi} and b_i are the weights and bias between the input data at time t and the input gate. W_{hi} is the weights matrix between the hidden unit and the input gate.
The second branch is a hyperbolic tangent activation layer that outputs the candidate values that will be added to the new state:

$$\tilde{C}_t = \tanh(W_{xc}x^{(t)} + W_{hc}h^{(t-1)} + b_c) \quad (5)$$

where W_{xc} and b_c is the weights matrix and bias between the input data at time t and the candidate value. W_{hc} is the weights between the hidden unit and the candidate value. This is followed by combination of the three Eq. 3, Eq. 4 and Eq. 5 from the input and forget gates to produce the cell state $C^{(t)}$ at time t as expressed as in Eq. 6. Eq. 6 is simply an element-wise summation of what to forget from previous cell state $C^{(t-1)}$ and what to update using the candidate $\tilde{C}_t$.

$$C^{(t)} = (C^{(t-1)} \odot f_t) \oplus (i_t \odot \tilde{C}_t) \quad (6)$$

Output gate contains two steps. The first step is a logistic sigmoid layer as expressed as:

$$o_t = \sigma(W_{xo}x^{(t)} + W_{ho}h^{(t-1)} + b_o) \quad (7)$$

where W_{xo} and b_o are the weights and bias between the input data at time t and the output gate. W_{ho} is the weights between the hidden unit and the output gate. The second step, as expressed in Eq. 8, combines o_t with the internal states to obtain the output of the cell at time t. It is a hyperbolic tangent 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.

$$h^{(t)} = o_t \odot \tanh(C^{(t)}) \quad (8)$$

An extension of LSTM was developed by Schuster and Paliwal [1997] by having two LSTMs (Bidirectional LSTM or Bi-LSTM). 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.

2.3 Word Embedding

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 large sparse matrix that is problematic in training: The feature learning process will suffer from curse of dimensionality [Raschka and Mirjalili 2019]. A better way is to convert words into vectors through word embedding. Word embedding is a fundamental step for any Natural Language Processing task. Representing words as vectors, as shown in Fig. 5, is usually what is used to capture the relative meaning of words though their vectors representation. This means that words with similar meaning have similar representations. By clustering the words using the cosine similarity function, one can find a relative meaning of the words as shown in Fig. 5. Cosine similarity is a similarity measure between two vectors or the cosine of the angle between the two vectors. 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 the words that surround it. So, a certain word can be trained by a log-linear 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 word. Once the training is complete, there will be updated set of weights which is the embedding of that word. One important observation in Word2Vec is that Word2Vec does not take into account the frequency of co-occurance of words.
On the other hand, GloVe takes these co-occurance or frequencies into account which provide more information. 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, having n words, the co-occurrence matrix will be $n \times n$ matrix where the matrix is populated with how many times word has co-occurred with other words where the vector of the word can be inferred from the context information. The authors of the GloVe , [Pennington et al. 2014], propose to learn the ratios of these co-occurence 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 of [Pennington et al. 2014] explained an example: if the $P(solid|ice)$ is large and $P(solid|steam)$ is small. Therefore, 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.

3 Data

The data used in this study is Groningen Meaning Bank (GMB) data that was originally developed at the University of Groningen by [Bos et al. 2017; Walia, A. 2019]. The data contains documents authored by Voice of America (VOA) with documents from the MASC dataset ([Ide et al. 2008]).
According to [Bensal 2021], "This dataset is not considered a gold standard. This means that the data set is built using automatic tagging software, followed by human raters updating subsets of the data. However, this is a very large and rich data set. This data has a lot of useful annotations that make it quite suitable for training models".

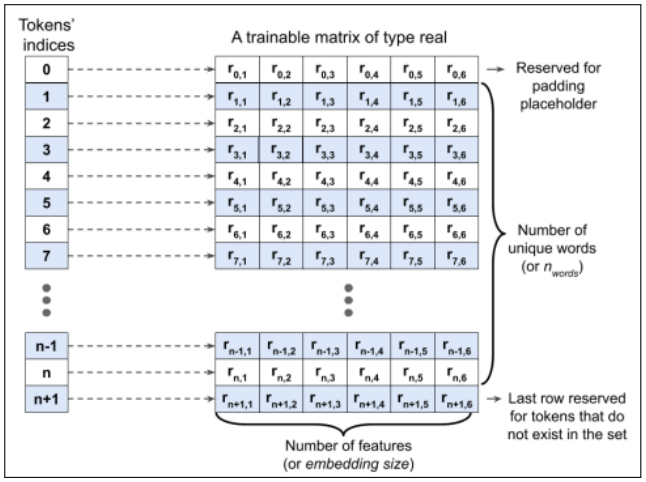

Figure 5: General way of conversion of tokens to vectors where the embeddings can be learned through neural network. Each token is converted to vector where the distance between words is related to semantic similarity. Source: [Raschka and Mirjalili 2019]

Table 1: Text Entities in the data with their representation before the application of the IOB annotation scheme.

Entity	Representation
geo	Geographical Entity
org	Organization
per	Person
gpe	Geopolitical Entity
tim	Time indicator
art	Artifact
nat	Natural Phenomenon
eve	Event

The data contains 47959 sentences and each word in the sentences is annotated as a certain entity. The vocabulary size of the data is 31817 which represent the number of unique words in the corpus. A sample sentence is shown in Table. 2. The data were annotated as entities according to the IOB2 annotation scheme (short for Inside, Outside and Beginning of entities). For the named entity chunks, annotation "O" represents tokens outside chunk, the annotation "B" is the beginning of a chunk while "I" marks the inside of the chunk. Also, the annotation "B" is given for a single chunk entity. The data contain 17 unique entities as shown in Table. 1. The entire data is summarized in the Table. 3.

4 Methods

We use Python Programming Language v3.7.9 and Tensorflow keras API v2.4.1 to build the neural networks that are used in this report. Other relevant packages version are shown in Table. 4. The work is saved in the notebook named `modeling.jpynb` with the hash `20e3736d` in the GitLab repository https://gitlab.com/mhmdrdwn/datascience_report. The models are trained under Tesla P100 GPU with 13 GB of RAM and 16 GB of GPU RAM.

Table 2: Part of sentence 4 with labels. The sentence is "U.N. relief coordinator Jan Egeland said Sunday, U.S., Indonesian and Australian military ..."

Word	Label	Word	Label
U.N.	B-geo	,	O
relief	O	U.S.	B-geo
coordinator	O	,	O
Jan	B-per	Indonesian	B-gpe
Egeland	I-per	and	O
said	O	Australian	B-gpe
Sunday	B-tim	military	O

Table 3: Data Summary

Number of unique Tokens	31817
Number of sentences (samples)	47959
Number of all words in data	1048575
Maximum length of sentence	104
Number of unique entities	17

Table 4: Used Softwares

Software	Version
Tensorflow	2.4.1
Numpy	1.19.5
Pandas	1.2.2
Seqeval	1.2.2
ScikitLearn	0.24.1
Seaborn	0.11.1
Matplotlib	3.3.3

4.1 Tokenization

We started by building the vocabulary of the unique words in the sentences and assign an index to each word. We use the Keras tokenizer for this process. The way the tokenizer works is that it assigns 1 to the most common word in the text and then assigns 2 to the second most common word until all the unique words in the text are encoded. Similarly, we build label dictionary to map each of the 17 unique tags to a unique number. Then we convert those unique numbers to a one-hot-encoded matrix where the class 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 $s \times 17$ matrix where s is the length of the sequence and 17 is number of labels.

4.2 Padding

The lengths of each sentence in the data is different. The longest sentence has 104 tokens (see Table. 3). In order to have all the sentences in the same length, we performed padding by adding zeros

Table 5: Bidirectional LSTM network. The parameters of the embedding is product of (300×31817 plus 300 biases where 31817 is number of unique tokens).

Layer	Output Shape	Parameters
Input Layer	104	0
Default Embedding	(104, 300)	9545400
Bidirectional LSTM	(104, 128)	186880
LSTM	(104, 64)	49408
Dense	(104, 18)	1170
Parameters		9782858

at the end of the each sentence until all sentences are of length 104.

4.3 Word Embedding

We used two models with different methods of embedding. The first model used the Keras default embedding layer where the word representation will be learned in the network. We set the dimension of embedding to 300. We notice that the embedding dimension has slight effect on measured metrics. This means that the vector dimension for each word is 300 and dimension of the embedding sentence is $(104, 300)$. The second method was by using GloVe pretrained model to extract the embedding vectors of the words and then inserting these weights as initial weights for the embedding layer in the model. We use the version of GloVe that has 840 billion unique words from Takumi, O. [2019]. The embedding vector dimension of words in GloVe is 300.

4.4 Building the model

We split the data into training and testing data where the training data represent 0.8 and testing data represent 0.2 (testing data contains 9592 sequences). The used neural network architectures are as shown in tables 5 and 6. Both the models start with embedding the features followed by hidden 16 bidirectional LSTM units. The output layer (Dense layer in Table. 5) is a regular fully connected neural network layer with the number of labels. The number of labels is set to 18 which represent the number of labels plus the padding zero. The loss function is categorical cross entropy (Eq. 9). The validation data (0.1 of the train data) is chosen randomly within the Keras model to monitor the accuracy.

$$CCE = -\sum_{i=1}^{C} t^i \log(p_i) \tag{9}$$

In Eq. 9, t_i and p_i are the ground truth and the activation of the class i. This loss is calculated for each of the C classes.
The number of trainable parameters for Bidirectional LSTM and Bidirectional LSTM with GloVe are 513906 and 9586570 respectively. The significant increase in the number of trainable parameters is because the GloVe algorithm gives a vector of dimension 300 for each word. The model produces a probability using Softmax activation function in the output layer.

Table 6: Bidirectional LSTM network with Glove embedding

Layer	Output Shape	Parameters
Input Layer	104	0
Glove Embedding	(104, 300)	9545400
Bidirectional LSTM	(104, 128)	186880
LSTM	(104, 64)	49408
Dense	(104, 18)	1170
Parameters		9782858

Table 7: Strict Measures. Prediction 1 is counted as one TP while Prediction 2 is zero TP.

	coordinator	Jan	Egeland	said
True	O	B-Per	I-Per	O
Prediction 1	O	B-Per	I-Per	O
Prediction 2	O	B-Per	O	O

4.5 Evaluation metrics

We recorded Precision, Recall, F1 score [Powers 2008] and confusion matrix to evaluate the models and to perform error analysis. Precision measures the number of positive class predictions (TP) that actually belong to the positive class:

$$P = \frac{TP}{TP + FP} \tag{10}$$

On the other hand, Recall is the number of positive class predictions made out of all positive classes:

$$R = \frac{TP}{TP + FN} \tag{11}$$

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

$$F1 = \frac{2 \times P \times R}{P + R} \tag{12}$$

Since we add padding zeros at the end of the sequences that will affect the measured metrics (increases metrics), we exclude the padding zeros from the calculations of all these metrics using a mask in our implementation of the metrics.
For further evaluation of the models, we use the package Seqeval (see Table. 4) to quantify F1, Precision and Recall. Compared to the above mentioned metrics, Seqeval uses strict metrics that only quantify True Positive (TP) if the whole entity is predicted correctly [Nakayama 2018]. This means that partially predicted entities do not count as TP as shown in Table. 7. Precision, Recall and F1 scores can be measured as per sentence (micro measure) and over the entire corpus (macro measure). The weighted measure is calculated for each label and then their average is weighted by the label support.

Table 8: Hyperparameters optimization choices

	Hyperparameters Search	Optimal Choice
Learning rates	(0.00001, 0.0001, 0.001, 0.005, 0.008, 0.01, 0.02, 0.05, 0.1)	0.01
Optimizers	(Adam, RMSprop)	Adam
Number of Units	(32 , 64, 128, 256)	64
Dropout	(0.2, 0.3, 0.4, 0.5)	0.5

Table 9: Results achieved by Bidirectional LSTM (BiLSTM) and Bidirectional LSTM with Glove embedding (BiLSTM-GloVe).

		Train	Test
BiLSTM	Recall	0.9634	0.9556
	Precision	0.9731	0.9634
	F1	0.9682	0.9595
BiLSTM-GloVe	Recall	0.9658	0.9598
	Precision	0.9745	0.9669
	F1	0.9702	0.9633

4.6 Hyperparameters Optimization

We experiment manually using different combination of hyperparameters while monitoring the validation set evaluation metrics. We choose learning rate from a logarithmic scale (i.e., 0.00001 ,0.0001, 0.001, 0.01, 0.1). Then, we focus search further on learning rates with-in the range (0.005, 0.05) which shows best results. The optimal choice is using Adam optimizer with learning rate 0.01 which shows the highest validation scores. Regularization dropouts of 0.5 were used in both network. This means that half of the neurons are randomly muted to reduce the complexity of the network and the overfitting according to [Srivastava et al. 2014]. A full list of experimented hyperparameters are shown in Table. 8. Other hyperparameters such as the number of units have minimal effect on the measured metrics of the validation set.

5 Results

Table 9 shows the comparison between the results of the two models. The resulting scores show that the two implemented model are generalizing on the validation and the testing data. Evaluating the models on the validation and testing data shows very similar values. The recorded training time for both the implemented models is approximately 205 seconds per epoch. This means that the total duration of training each of the models is approximately 34 minutes.

5.1 Bidirectional LSTM

After training for 10 epochs, bidirectional LSTM achieved F1 score of 0.9682 on training data while the scored F1 scores on test data 0.9595 respectively. From Fig. 6, It is noticed that the validation F1 score remains almost constant around 0.9600 after epoch number 5 while the training score continued to increase slightly until it reaches 0.9783 at epoch number 10.

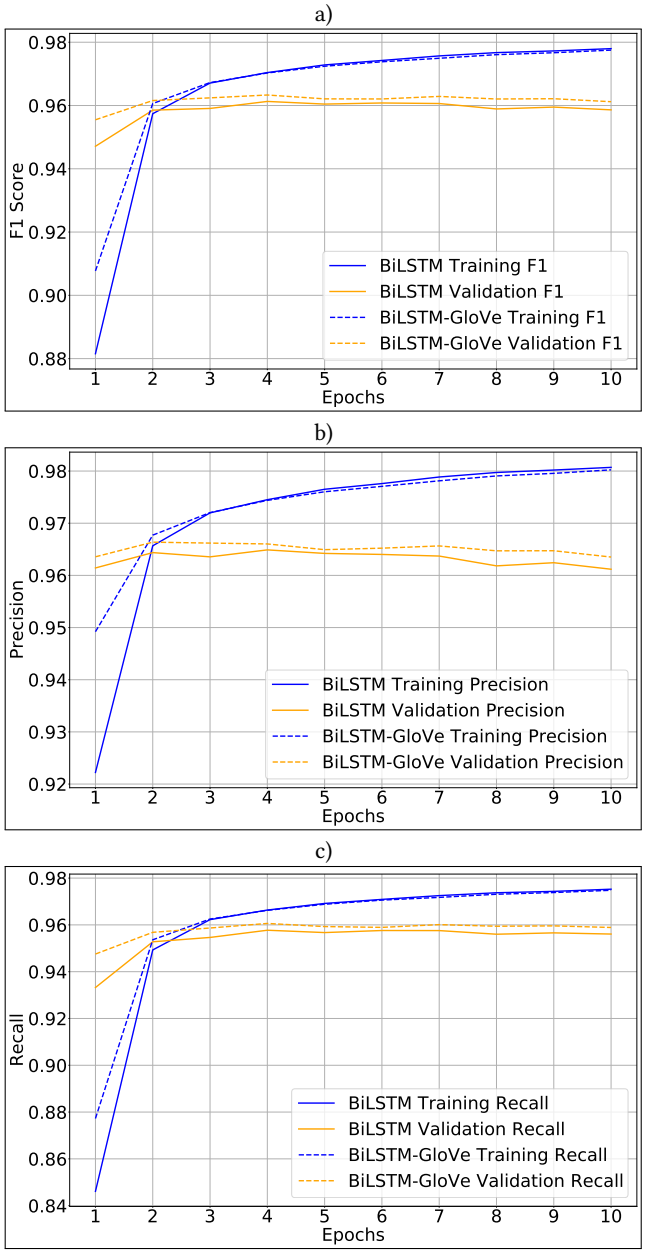

Figure 6: Training and validation a) F1, b) Precision and c) Recall score using the two models.

Fig. 6 also shows the precision score achieved by bidirectional LSTM which starts at 0.9631 and ends at 0.9605 on validation data. Despite the change of precision score on training data from 0.8315 to 0.9811 after 10 epochs, validation precision score remains almost constant. The precision score of the testing data is 0.9609.
Fig. 6 shows the recall score of bidirectional LSTM model. The recall score is 0.9634, 0.9543 and 0.9556 on training and testing data

Table 10: Strict evaluation of Bidirectional LSTM network. Results might vary very slightly because of the randomness of the model weights initialization.

	Precision	Recall	F1	Support
ART	0.000	0.000	0.000	94
EVE	0.812	0.186	0.302	70
GEO	0.827	0.854	0.841	7558
GPE	0.957	0.932	0.944	3142
NAT	0.429	0.075	0.128	40
ORG	0.666	0.546	0.600	4151
PER	0.751	0.696	0.722	3400
TIM	0.861	0.825	0.842	4077
Micro	0.815	0.772	0.793	22532
Macro	0.663	0.514	0.547	22532
Weighted	0.806	0.772	0.787	22532

respectively. We notice from Fig. 6 that validation recall starts at 0.9485 and end 0.9587 after 10 epochs.
Heatmap in Fig. 7 shows the confusion matrix between the ground truth labels and the predicted labels. The confusion matrix indicates the label 'O' or 'other' is accompanied with most of the errors in the prediction (false positive and false negative).

To further evaluate the model for testing, we measure the metrics in Table. 10. The recorded metrics using the package seqeval (see Section 4.5) for the prediction of the test. The recorded micro and macro F1 are 0.793 and 0.547 respectively. From the table, we notice that the label 'ART' has the lowest measured metrics while the label 'GPE' has the highest measured metrics (see Table. 1 for representation of entities). It is important to notice that the recorded values in Table. 10 is different from the recorded values from Table. 9 because we use strict measures here.

5.2 Bidirectional LSTM with GloVe embedding

From Table. 9, Bidirectional LSTM with GloVe embedding achieved F1 score of 0.9702 on training data while the scored F1 scores using test data is 0.9633. It is noticed from Fig. 6 that the model converges slightly faster than the bidirectional LSTM. The validation F1 score starts with 0.9564 and ends with with slight improvement at 0.9609 after epoch number 10 while the training score continued to increase slightly from 0.8272 until it reaches 0.9782 at epoch number 10.
Fig. 6 shows the precision score achieved by model which starts at 0.9644 and ends at 0.9632 on validation data. Despite the change of precision score on training data from 0.8996 till 0.9809, validation precision score remains almost constant. The precision score of the test data is 0.9669.
Fig. 6 shows the recall score of bidirectional LSTM model. The recall scores are 0.9755 and 0.9587 and 0.9581 on training and validation data. The recall of the test data is 0.9598.
From the heatmap in the Fig. 7, we noticed that the label "O" label or "other", is also accompanied with most of the errors in the prediction similar to the predictions of bidirectional LSTM model. We noticed that several classes such as "I-tim", "I-per" and "I-gpe" and

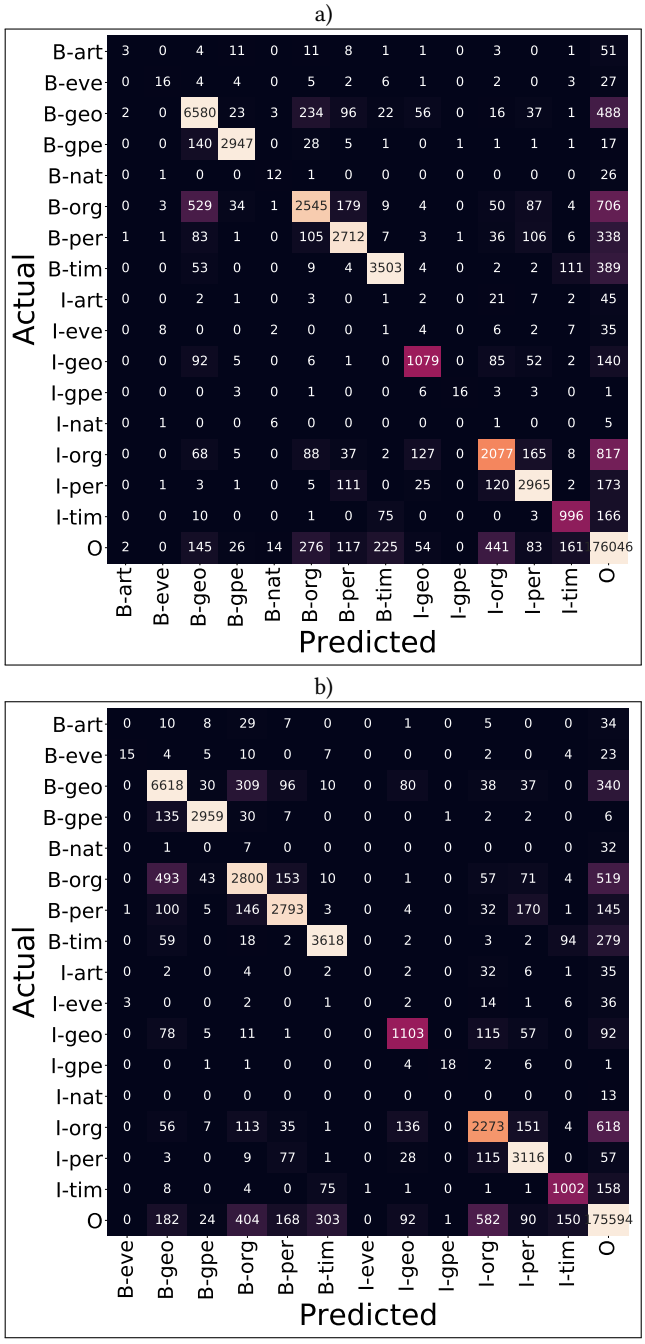

Figure 7: Confusion matrix for the ground truth labels vs the predicted labels using a) Bidirectional LSTM and b) Bidirectional LSTM with GloVe. Notice the increase in number of TP for using the bidirectional LSTM with GloVe over the Bidirectional LSTM model.

Table 11: Part of a random sample from test sentences with prediction and actual labels. The sentence is "The report calls on president bush and congress to urge chinese officials..."

Word	Actual	Prediction
the	O	O
report	O	O
calls	O	O
on	O	O
president	B-per	B-per
bush	I-per	I-per
and	O	O
congress	B-org	B-org
to	O	O
urge	O	O
chinese	B-gpe	B-gpe
officials	O	O

Table 12: Strict evaluation of Bidirectional LSTM with GloVe network. Results might vary very slightly because of the randomness of the model weights initialization

	Precision	Recall	F1	Support
ART	0.000	0.000	0.000	94
EVE	0.714	0.214	0.330	70
GEO	0.840	0.869	0.854	7558
GPE	0.954	0.940	0.947	3142
NAT	0.308	0.300	0.304	40
ORG	0.668	0.586	0.624	4151
PER	0.754	0.749	0.752	3400
TIM	0.883	0.851	0.867	4077
Micro	0.821	0.799	0.810	22532
Macro	0.640	0.564	0.585	22532
Weighted	0.814	0.799	0.806	22532

several other labels were predicted more correct with using bidirectional LSTM with GloVe than using birectional LSTM. However, The number of correct predictions of "O" decrease when using the birectional LSTM with GloVe.

The Table. 9 shows the the recorded metrics of prediction of the test data using strict measures. The recorded micro and macro F1 are 0.810 and 0.585 respectively. From the table, we notice that the label 'ART' is never predicted correctly using the strict framework. But in general, the micro F1 is of using Bi-LSTM with GloVe is higher than the one with using Bi-LSTM by margin of 0.02. Similaly, Bi-LSTM with GloVe recall is higher than the recall of Bi-LSTM by a margin of 0.025 .

6 Discussion

From the observed results, we notice that bidirectional LSTM with GloVe shows slightly better performance than the simple bidirectional LSTM. This observation is supported using strict and non-strict measures. Strict metrics evaluation of the BiLSTm with GloVe shows improvement by a margin of 0.02 on the micro F1 score, around 0.025 on the micro recall score and around 0.005 on the precision score. These observations indicate that the GloVe model gives a better representation of the words than the simple default embedding layer in the model.

We notice that the increase in the metrics in validation curve is small. One possible reason for this observation is that the network already captures the patterns in the features while training just after the first epoch.

The two models are facing difficulty to predict the "EVE", "ART" and "NAT" correctly and showed poor prediction results. For example, the precision of both the models for "EVE" tag is high relative to the recall because the increase in number of FN. This means that several entities has been predicted as "EVE" in a wrong way.

In this report, we do not make explicit comparison with other related work. There are two reason for that: first reason, it is difficult to compare results given the variance in the used methods and the use of evaluation metrics. For example, Entity Disambiguation systems usually uses a different evaluation metrics that tell about the entities that are found and retrieved correctly from a knowledge base such as Wikipedia within N number of nearest neighbours. Second reason is that the data, to the best of our knowledge, was not pre-splitted strictly into train and test set. So, there are no fixed test set to compare the results. In general, several studies used this data by implementing more advanced models such as the NorNE system that was developed by Jørgensen et al. [2020] that achieved state of the art results on this data. Other models are achieving F1 score of 0.93 by following the Bidirectional Encoder Representations from Transformers architecture (BERT) by Devlin et al. [2018] which is beyond the scope of this report.

The data is relatively easy to use for named entity recognition and easy to achieve a relatively better results using simple LSTM networks than applying named entity recognition in other specialized domains. The reason for is that majority of the entities are of the type "B-entity", or unigram for simplicity, where each one of those is considered as one TP in the metrics calculations. Another reason is that the words can be represented easily using GloVe or Word2Vec given that the words are commonly written in articles and general and have less ambiguity. Using more sophisticated pre-trained models such as BERT or ELMo [Peters et al. 2018] for encoding these words with context embeddings can help even more in boosting our results.

7 Conclusion

Two LSTM architectures are used to extract entities from the Groningen Meaning Bank (GMB) corpus [Bos et al. 2017]. The data in general provides a relatively easy task to solve given that it is from general domain where a model like GloVe can excel. GloVe provides a better representation of the words in the data which is helpful in training the model. Recently, several recent models are built using BERT models which provide an accurate way to encode the word features through combination of word pieces and these models can achieve state of the art results which could be the next step to further develop the results. Also, Further directions and development of the methods are expected to improve the measured metrics on this data by using feature engineering.

8 Acknowledgement

We want to express our gratitude to Prof. Hans Plesser for the guidance and providing detailed feedbacks, encouragement and the help through this study.

References

A. Bensal. 2021. *Advanced Natural Language Processing with TensorFlow 2: Build effective real-world NLP applications using NER, RNNs, seq2seq models, Transformers, and more.* Packt Publishing.

Johan Bos, Valerio Basile, Kilian Evang, Noortje Venhuizen, and Johannes Bjerva. 2017. The Groningen Meaning Bank. In *Handbook of Linguistic Annotation*, Nancy Ide and James Pustejovsky (Eds.). Vol. 2. Springer, 463–496.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *CoRR* abs/1810.04805 (2018). arXiv:1810.04805 http://arxiv.org/abs/1810.04805

Ronen Feldman, M. Fresko, Y. Kinar, Yehuda Lindell, Orly Liphstat, M. Rajman, Jonathan Schler, and Oren Zamir. 1998. Text Mining at the Term Level. In *PKDD*.

F. Gers, urgen Schmidhuber, and F. Cummins. 1999. Learning to Forget: Continual Prediction with Lstm Learning to Forget: Continual Prediction with Lstm.

David Harris and Sarah Harris. 1990. Digital design and computer architecture (2nd ed.). *Proc. IEEE* 78, 10 (1990), 129.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

A. Hotho, A. Nürnberger, and G. Paass. 2005. A Brief Survey of Text Mining. *LDV Forum* 20 (2005), 19–62.

Nancy Ide, Collin F. Baker, C. Fellbaum, C. Fillmore, and Rebecca J. Passonneau. 2008. MASC: the Manually Annotated Sub-Corpus of American English. In *LREC*.

Fredrik Jørgensen, Tobias Aasmoe, Anne-Stine Ruud Husevaag, Lilja Ovrelid, and Erik Velldal. 2020. NorNE: Annotating Named Entities for Norwegian. *ArXiv* abs/1911.12146 (2020).

Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1301.3781

Hiroki Nakayama. 2018. seqeval: A Python framework for sequence labeling evaluation. https://github.com/chakki-works/seqeval Software available from https://github.com/chakki-works/seqeval.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. 2013. On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*. 1310–1318.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1532–1543. https://doi.org/10.3115/v1/D14-1162

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *CoRR* abs/1802.05365 (2018). arXiv:1802.05365 http://arxiv.org/abs/1802.05365

D. Powers. 2008. Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness Correlation.

S. Raschka and V. Mirjalili. 2019. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2, 3rd Edition.* Packt Publishing.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986. Learning Representations by Back-propagating Errors. *Nature* 323, 6088 (1986), 533–536. https://doi.org/10.1038/323533a0

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

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15, 56 (2014), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

Takumi, O. 2019. *GloVe 840B.* https://www.kaggle.com/takuok/glove840b300dtxt [Online; accessed 09-05-2021].

Oriol Vinyals, A. Toshev, S. Bengio, and D. Erhan. 2015. Show and tell: A neural image caption generator. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), 3156–3164.

Walia, A. 2019. *Annotated GMB Corpus.* https://www.kaggle.com/abhinavwalia95/entity-annotated-corpus/ [Online; accessed 12-03-2021].

P. J. Werbos. 1990. Backpropagation through time: what it does and how to do it. *Proc. IEEE* 78, 10 (1990), 1550–1560. https://doi.org/10.1109/5.58337