CE/CZ4045 Assignment Part 2

Lim Jun Hong*
Nanyang Technological University
LIMJ0209@e.ntu.edu.sg

Tammy Lim Lee Xin*
Nanyang Technological University
TLIM045@e.ntu.edu.sg

Lee Han Wei*
Nanyang Technological University
B160017@e.ntu.edu.sg

Pang Yu Shao*
Nanyang Technological University
C170134@e.ntu.edu.sg

1 INTRODUCTION

In this assignment, we implement two different Neural Network models using `Pytorch` to perform NLP tasks such as **Text Prediction/ Generation** as well as **Named Entity Recognition (NER)**

2 DESIGNING A FEED-FORWARD NEURAL NETWORK FOR TEXT GENERATION

In this section, a simple Feed-Forward Neural Network (FFN) architecture will be designed and implemented for the purpose of language modelling (i.e., to predict the next word given a sequence of words.)

2.1 Dataset

The dataset to be used for this task is the **wikitext-2** dataset, which is a collection of text from Wikipedia.

2.1.1 Data Pre-processing. Various pre-processing is done to the dataset before it is used to train the model, such as:

Case-folding (i.e., reducing all alphabet characters to lower case): This is done as same words with different characters in differing cases are treated as separate words. For instance, `"The"` and `"the"` would be treated as 2 completely different words and would be mapped to different Word IDs, which may result in completely different embeddings.

Removal of headings: Upon inspection of the dataset, many lines containing section headings can be identified. An example of a heading is given below:

```
= = = Influence on Japanese literature = = =
```

Therefore, such lines shall be discarded to prevent our model from learning and predicting the header tokens `"="`.

2.2 Implementing the Baseline FFN

Refer to Figure 1 for the architecture of the proposed FFN.

2.2.1 Input Layer. As the model is to be trained on 8-grams (i.e., a sequence of 8 words), the input to the FFN will be the first 7 words (in word IDs format). The input embedding layer will then convert the word IDs to their respective word vectors. These word vectors will be transposed and concatenated and fed to the hidden layer.

2.2.2 Hidden Layer. The hidden layer of the network is a simple dense layer of **200 neurons** in the beginning and will be tuned for further gains in subsequent sections. The activation function used for the hidden layer is the **tanh** function before the outputs are fed to the output layer.

2.2.3 Output Layer. The output layer of the network is a **Softmax** layer which gives the probability of the word ID representing the predicted word following the sequence of 7 preceding words that are fed into the input layer.

2.2.4 Training Set-up and Hyperparameters. For the baseline model, the following set-up and hyperparameters are used:

- Learning Rate: **0.0001** (With annealing factor of **0.25**)
- Optimiser: **Adam**
- Hidden Layers: **1**
- Neurons in Hidden Layers: **200**
- Dropout in each layer: **20%**
- Training Epochs: **50**
- Loss function: **Cross-Entropy**

2.3 Training Results of the Baseline FFN

After training the FFN for 50 epochs with the set-up described in 3.2.1, the "best" results are obtained at the epoch with the lowest validation loss:

Epoch: 20	Loss (Cross-Entropy)	Perplexity	Prediction Accuracy
Training Set	4.86	128.79	26.4%
Validation Set	5.40	221.29	20.6%

Table 1: Best Baseline FFN Training Results

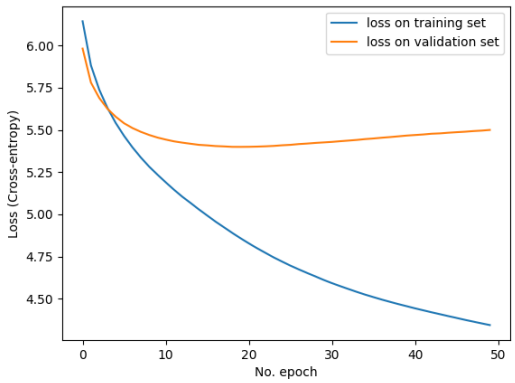

Figure 2: Loss of Baseline Model Over 50 Epochs

*All authors contributed equally to this project.

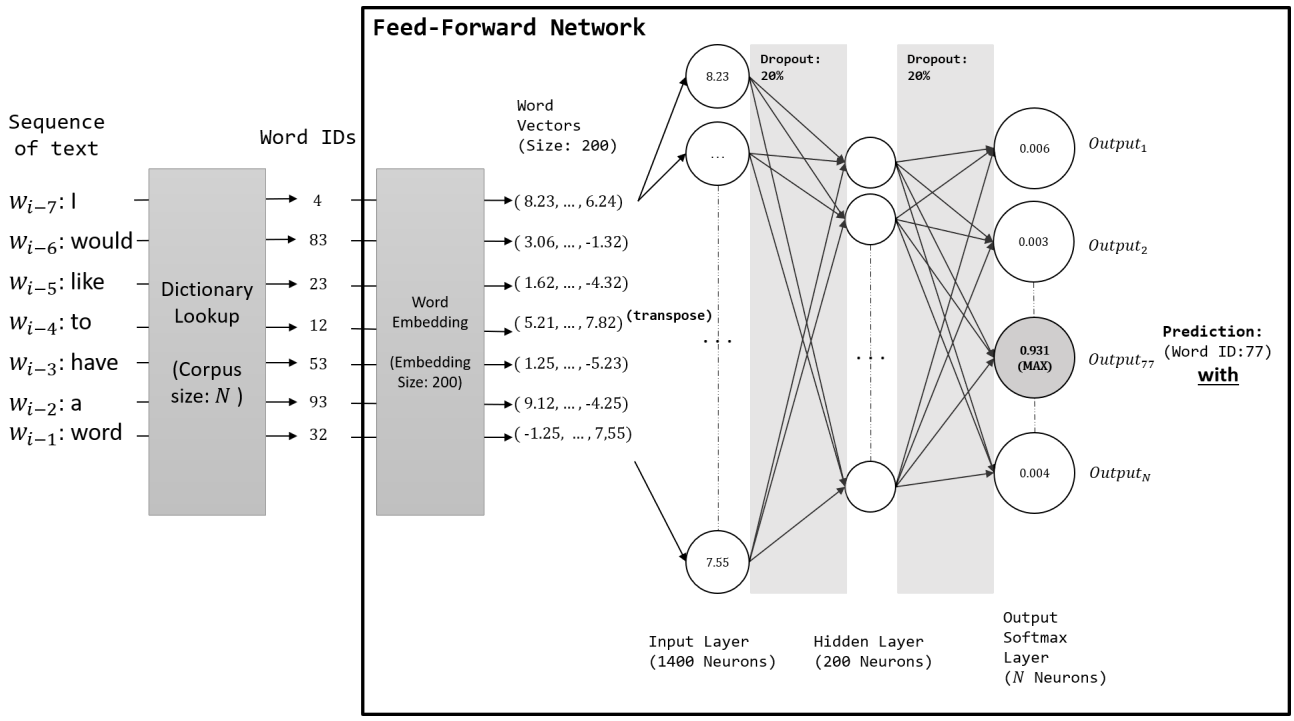

Figure 1: Architecture of FFN

2.3.1 Analysis. From the obtained results, it can be seen that from Figure 2 there is heavy over-fitting of the model from about epoch 20 onwards. The performance of the model in terms of prediction accuracy on the validation set is only at **20.6%** and a perplexity score of **221.29** at epoch 20.

In the following sections, we will aim to improve on this Baseline FFN's accuracy by experimenting with various strategies such as **Regularisation** as well as **Hyperparameter tuning** to arrive at a model with the best performance.

2.4 Overcoming Over-fitting

Over-fitting occurs when a model learns the training data too closely, and as a result, fails to **generalise** to unseen data. This is seen in Figure 2 where the model's loss on training data is shown to continue decreasing while the model's loss on the validation data is shown to be increasing. This means that while the model would perform very well on inputs from the training examples (i.e., having a low loss on the training data), it would have poor performance on unseen input data.

2.4.1 Early Stopping. **Early stopping**, as its name implies, is a technique used to halt training of the model prematurely when over-fitting is detected. While it does not ensure that over-fitting is reduced and the model would have a better performance, it will ensure that the model would not be over-trained. This would therefore save time that would be otherwise spent on training the model. For the training of the model in Section 2.3, Early Stopping would stop the training shortly after Epoch 20 as the validation loss did not improve after that.

For subsequent models, Early Stopping will be implemented with a **patience** of 5 Epochs. This means that if the **validation loss** does not improve for 5 consecutive Epochs, the training would be halted prematurely.

2.4.2 L2 Regularisation. **L2 Regularisation** is one of the strategies that can be used to prevent over-fitting of a model. One of the causes of over-fitting is when there are very high weight values learnt by the model which causes a the model to learn one feature very closely/strongly which may only be specific to the training data.

Therefore, L2 Regularisation aims to prevent over-fitting of the model by penalizing very large weights that are learned by the model. This is done by summing all the squared weight values and then multiplying them by a factor before adding it to the computed error value. This results in the following loss function:

$$Loss = Error + \beta \sum_{i=1}^{W} {W_i}^2$$

Where Error is the Cross-Entropy loss and β is the regularising factor.

A L2 regularisation factor of $\mathbf{10^{-6}}$ is applied and implemented in Pytorch as a **weight decay** parameter to the optimiser. As a result, the training took longer to converge. The training was done for **100** epochs instead of 50 to allow the model more time to converge.

During the training process, the model did not detect over-fitting and hence did not terminate early. At the end of the 100 epochs, the model had considerably better performance on the validation set as compared to the initial baseline model.

Model	(Metrics evaluated on **Validation** Set) Best Perplexity	Best Loss	Best Prediction Accuracy
Baseline	221.29	5.40	20.6%
With L2 Reg	183.53	5.21	21.5%
ΔPerplexity Score	**-37.76**		

Table 2: Performance of model with L2 Regularisation

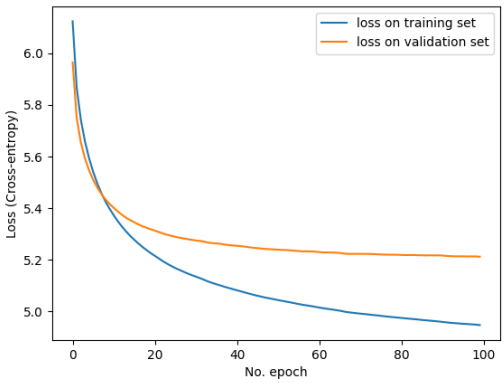

Figure 3: Loss of Model with L2 Over 100 Epochs

2.5 Hyperparameter Tuning

With the over-fitting measures in place, this sub-section aims to further improve on the model's performance by tuning the hyperparameters of the model. Specifically, the effects of tuning the **Embedding Size** and **Hidden Layer Size** of the network will be evaluated and values of hyperparameters yielding the best performing model will be chosen.

2.5.1 Methodology. A simple **Grid Search** is performed on the two hyperparameters. The Embedding Size and Hidden Layer Size are also kept to the same to each other as it is required to explore the effects of **Parameter Sharing** in a later section. The following values are used for the Grid Search:

$$Size \in \{200, 300, 400, 500, 600, 700, 800\}$$

2.5.2 Results. The grid search was performed and the following results are obtained:

Emb Size & nhid Size	200	300	**400**	500	600	700	800
Validation Perplexity	183.53	179.75	**179.45**	180.40	181.76	182.99	184.38

Table 3: Results of Grid Search

From Table 3, it can be seen that that an Embedding Size and Hidden Layer Size of **400** yields the best performance in terms of validation perplexity, therefore it is selected for the model.

2.5.3 Further Refinements and Chosen Model. As the embedding and hidden layer dimensions have increased, the increased dimensions might lead to more over-fitting of the model. Therefore, the L2 regularisation term is increased from $\mathbf{10^{-6}}$ to $\mathbf{2 * 10^{-6}}$.

The final specifications of the model are as follows:

- Learning Rate: **0.0001** (With annealing factor of **0.25**)
- Optimiser: **Adam**
- Embedding Size: **400**
- Hidden Layers: **1**
- Neurons in Hidden Layers: **400**
- Dropout in each layer: **20%**
- L2 Regularisation Term: $\mathbf{2 * 10^{-6}}$

The model was trained for 95 Epochs, where it early-stopped when the validation loss ceased to decrease. The following results are obtained and compared with the baseline model:

Model	(Metrics evaluated on **Validation** Set) Best Perplexity	Best Loss	Best Prediction Accuracy
Baseline	221.29	5.40	20.6%
Final Model	177.06	5.18	21.8%
ΔPerplexity Score	**-44.23**		

Table 4: Performance of Model w/ Hyperparameter Tuning

2.6 Parameter Sharing between Embedding and Output Softmax Layer

In this section, the effects of Parameter Sharing (or **Weight Tying**) is explored.

In previous studies, it is shown that sharing the parameters of the embeddings to the output softmax layer, would allow the output layer to have a more informed distribution than vanilla one-hot layers which would lead to improved learning. Also, the number of parameters to be learned by the model would also be reduced [1], which could result in faster learning times per epoch.

2.6.1 Results. With the model chosen in Section 2.5.3, Parameter Sharing was implemented in Pytorch by simply assigning the encoder's weights to the decoder's weights

```
self.decoder.weight = self.encoder.weight
```

The model was then trained until it early-stopped and the following observations are made:

- The training reaches minimum validation loss at Epoch **85**, instead of **95** without weight tying.
- The time of training per Epoch is decreased from **67s** to **55s**

The following performance metrics are obtained:

Model	(Metrics evaluated on **Validation** Set) Best Perplexity	Best Loss	Best Prediction Accuracy
Without Weight Tying	177.06	5.18	21.8%
With Weight Tying	174.29	5.16	21.7%
ΔPerplexity Score	**-2.77**		

Table 5: Performance of Model w/ Parameter Sharing

2.6.2 Analysis. With the observations made in Section 2.6.1 and Table 5, it can be concluded that Parameter Sharing is indeed beneficial for language modelling. With Parameter Sharing, there are improvements in both having a final lower perplexity which indicates improved learning as well as having a faster training time of the model due to the decrease in the number of learnable parameters.

2.7 Generating Words with Trained FNN Model

To adapt `generate.py` from Pytorch's example code to generate text with the implemented `FNNModel`, there are two main steps that needs to be done.

- Adapt the input to be 7 word IDs instead of 1 single word ID

```
Before:
input = torch.randint(ntokens, (1, 1), dtype=
    ↪ torch.long).to(device)
After:
input = torch.randint(ntokens, (7, 1), dtype=
    ↪ torch.long).to(device)
```

- With the generated output, append it to the end of the input tensor and remove the first word ID from the tensor.

```
word_idx = torch.multinomial(word_weights, 1)[0]
word_tensor = torch.Tensor([[word_idx]]).long().
    ↪ to(device)
# Remove the first element using slicing
input = torch.cat([input[1:], word_tensor], 0)
word = corpus.dictionary.idx2word[word_idx]
```

2.7.1 Results. With the `generate.py` implemented and working with the `FNNModel`, it is then used to generate **200 words**. Refer to Appendix A for the full text generated.

2.7.2 Analysis. Upon inspection of the words generated by the model, one can see that it is mostly incoherent. However, there are some instances where the model seemed to perform well. E.g.:

```
, but failed to gain the general election
```

In the example shown above, the model was able to predict the word **"election"** given the preceding 7 words and this is a relatively good prediction as general election is a proper noun and there is no obvious grammatical mistakes made.

Therefore, the model is still able to predict the next word given the context window relatively well.

2.8 Additional: Improving FFN Performance with Attention

Instead of connecting the embedding outputs directly to the hidden layer, we try to improve the model's performance by introducing **attention**. An attention mechanism provides an attention distribution (i.e., which parts of the input is being focused on) which can then be multiplied to the original embedding inputs so that input words which are focused would have their word vectors enhanced, while words which are not focused by the attention mechanism would have their word vectors attenuated.

2.8.1 Implementation. The "**Multi-Head Attention**" mechanism proposed by Vaswani et al. [5] was implemented as the attention mechanism in our model.

In a single attention layer (i.e., Single-Head), the same word embeddings are passed to 3 separate linear layers (i.e., **self-attention**) to compute the query, key and value representations. The query and key matrices are then multiplied to obtain a vector of **attention scores**, which is then scaled and passed through a softmax function so that the scores would become an **attention distribution** where all scores would sum to 1. Finally, the attention distribution is multiplied with the value vector to yield the embeddings representation with attention information applied to it.

"**Multi-Head Attention**" is essentially having multiple attention units compute such attention representations in parallel before concatenating the outputs to form a single representation. This would result in an representation which is similar to an ensemble of all the attention units. In our implementation, we used $h = 8$ parallel attention layers (heads). Figure 4 below shows the architecture of the FFN with the Multi-Head Attention implemented between the embedding layer and the hidden tanh layer.

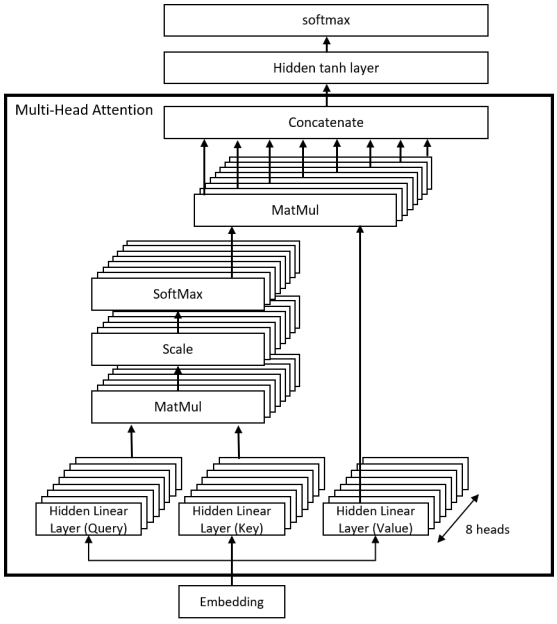

Figure 4: Architecture of the FFN with Multi-Head Attention

2.8.2 Results. The model was trained until it early-stopped and the following results are obtained:

Model	(Metrics evaluated on **Validation** Set) Best Perplexity	Best Loss	Best Prediction Accuracy
Baseline	221.29	5.40	20.6%
After Hyperparameter Optimisation (**HO**)	177.06	5.18	21.8%
HO + weight tying (**WT**)	174.29	5.16	21.7%
HO + **WT** + **Multi-Head Attention**	**167.14**	**5.12**	**22.0%**

Table 6: Comparison of Performance Between Models

2.8.3 Analysis and Conclusion. From Table 6, it can be seen that introducing an attention mechanism to the encoding portion of the model significantly improves the performance of the model. In our implementation, it was able to further reduce the perplexity score of the model by **7.15** and increase the prediction accuracy by **.3%**.

3 IMPLEMENTING A CNN MODEL FOR NAMED ENTITY RECOGNITION

In this section, we implement a CNN-based model for Named Entity Recognition (NER) with reference to the **Bi-directional LSTM-CNNs-CRF** architecture proposed by Ma and Hovy [2]. We will replace the Bi-directional LSTM layer with a CNN layer and assess the performance of the model. We will then increase the number of CNN layers and explore the relation between the number of CNN layers to performance of the model and choose the best model from the results.

3.1 Dataset

The dataset to be used for the NER task is the **CoNLL-2003** dataset, which is an annotated dataset of English data taken from the Reuters Corpus built upon Reuters news stories from August 1996 to August 1997 [4]

3.2 Implementation of CNN Architecture

The architecture of the model is based off the Bi-directional LSTM-CNNs-CRF model implemented by Anirudh and Peddamail [1].

The Bi-directional LSTM layer which performs word-level encoding is replaced with a CNN layer with a kernel size of (3, 125). The convolution operation essentially applies 400 different filter/kernel to 3 consecutive word embeddings with padding applied to corner cases, resulting in 400 output values for a single word. The output of this CNN layer of shape (N, 400) where N is the number of words in the input sentence is then fed to the CRF layer. Figure 5 depicts the architecture which is implemented.

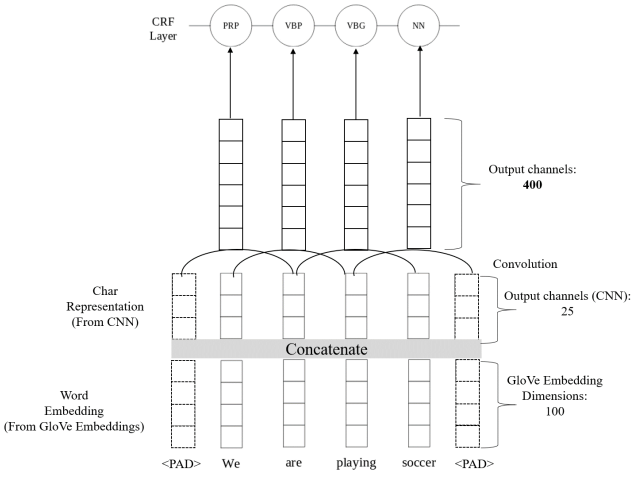

Figure 5: Proposed CNN Architecture

[1] https://github.com/jayavardhanr/End-to-end-Sequence-Labeling-via-Bi-directional-LSTM-CNNs-CRF-Tutorial

3.2.1 Training Set-up and Hyperparameters. For the baseline model, the following set-up and hyperparameters are used:

- Learning Rate: **0.0001** (with learning rate decay)
- Optimiser: **SGD**
- CNN Layers: **1**
- Dropout in fully connected layers: **50%**
- Training Epochs: **50** with early-stopping of **5** epochs patience
- Loss function: **Cross-Entropy**

3.2.2 Evaluation Metric. The performance of the model will be evaluated based on its **F-1 score** on the **Validation Set**

3.3 Performance of Baseline CNN Model

After training the model for 20 epochs, the training was terminated early as the validation F-1 score did not increase for 5 consecutive epochs. At the end of training, the model had a F-Score of **87.51%** on the **Validation Set**.

The change of training and validation F-1 scores of the model with respect to the epoch can be seen in Figure 6.

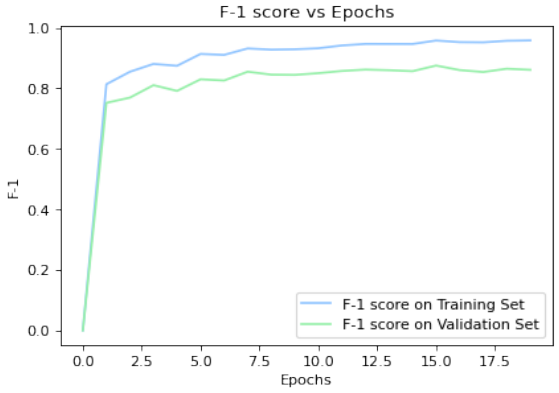

Figure 6: F-1 Scores of Baseline Model vs Epochs

3.4 Effect on Performance with Additional CNN Layers

With the baseline model implemented, the effects of adding more CNN layers on the performance of the model was then explored.

3.4.1 Implementation. The output of the CNN layer is reshaped such that it only has a single channel and a 2D shape of (N, 400). Similar to what has been done in Section 3.2, a kernel size of (3, 400) is used for the convolution with padding done for the corner words with a output of channel of 400 to generate an output with the same shape as the input. This is then repeated for each additional layer that is required.

3.4.2 Experimental Results. The following number of CNN layers are used in the experiments conducted:

$$Layers \in \{1, 2, 3, 4, 5, 6\}$$

The same training parameters (e.g., learning rate, optimiser, etc.) are used for all models and the F-1 scores of the best performing (in terms of Validation F-1 Score) is recorded in Table 7 below.

Number of CNN Layers	Best Performing Epoch	F-1 (Training)	F-1 (Validation)	F-1 (Test)
1	15	95.79%	87.51%	81.35%
2	**37**	**99.28%**	**92.72%**	**87.08%**
3	16	97.88%	92.36%	87.39%
4	20	97.78%	92.52%	86.72%
5	20	96.77%	91.82%	86.63%
6	28	95.90%	91.49%	85.99%

Table 7: Comparison of Model Performance with Respect to Number of CNN Layers

By plotting the graph of the Validation F-1 score against epochs for the 6 models, the following figure is obtained:

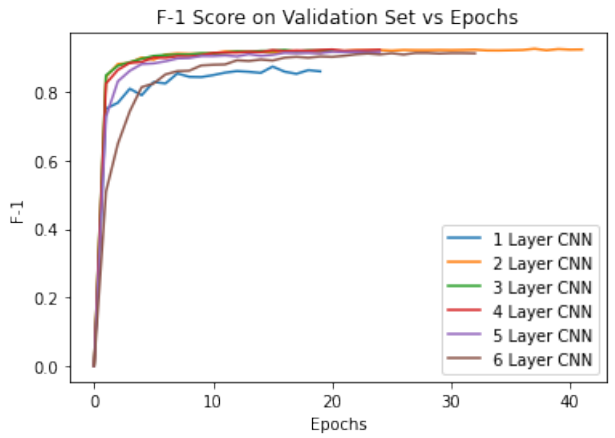

Figure 7: Validation F-1 Score versus Epochs

The validation F-1 scores for the 6 models is also visualised, resulting in the following figure:

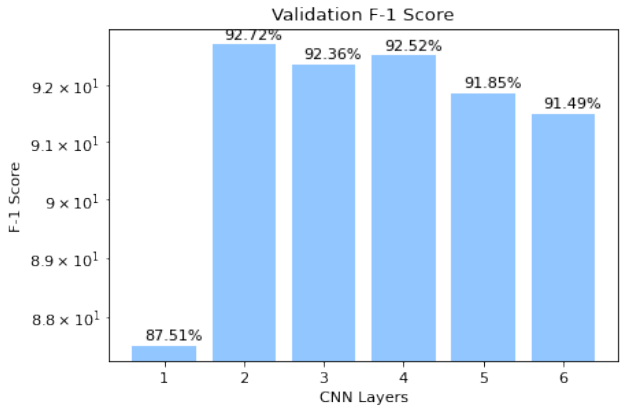

Figure 8: Validation F-1 Score against Number of CNN Layers

3.4.3 Analysis. From Figures 7 and 8, it can be seen that the F-1 score of the model is higher when the number of CNN layers is increased from 1, however the difference between the F-1 scores for the models with 2 convolution layers to 6 convolution layers do not differ much.

As the number of convolution layers is increased, it can be observed that for 6 layers of convolution, the model took longer to converge as compared to 1 to 5 layers of convolution

It is also worth noting that while the model with 2 convolution layers have the best performance, it also stopped training at a later epoch as compared to the other models. Therefore, the performance of the other modelss might be increased if the early-stopping patience of 5 epochs is increased. Which allows the model more time to train and improve on its validation F-1 score.

3.4.4 Chosen Model. From Table 7 and Figure 8, the model with **2 convolution layers** is chosen as it has the highest F-1 score amongst all the other candidate models.

3.5 Recognizing Named Entities with Trained CNN Model

With the chosen model from Section 3.4.4, we now try to perform NER tasks with it. We use the following text in our experiments:

```
George was born in Scotland and raised in England
// Expected named entities:
// George - Person
// Scotland - Location
// England - Location

George is an intern at Microsoft
// Expected named entities:
// George - Person
// Microsoft - Organisation
```

After feeding the trained model with the text, the following results were obtained:

```
Prediction:
word : tag
George : PER
was : NA
born : NA
in : NA
Scotland : LOC
and : NA
raised : NA
in : NA
England : LOC


George : PER
is : NA
an : NA
intern : NA
at : NA
Microsoft : ORG
```

From the above results, it can be seen that the model was able to recognise all entities and classify them correctly.

3.5.1 Testing Model with Out-Of-Vocabulary Entities. We now try to test the model to perform NER on entities that are Out-Of-Vocabulary (OOV) (i.e., not encountered in the dataset). From section 2.1, we see that the corpus the model was trained on is a collection of Reuters news stories from the year 1996 to 1997. Therefore, one such example of an entity that can be used would be the organisation Facebook, which was only launched later in the year 2004[3]. The following text is thus used for the test:

```
George is an intern at Facebook
```

The following results are obtained with the above text:

```
Prediction:
word : tag
George : PER
is : NA
an : NA
intern : NA
at : NA
Facebook : LOC
```

From the results above, it can be seen that Facebook has been misclassified as a Location instead of an Organisation, which confirms the initial hypothesis that the model would not perform as well when it comes to identifying and classifying Named Entities which are not in the training set.

One interesting observation is that Facebook has been tagged as a "Location" instead, this indicated that the model might have been able to learn the relations with neighbouring words (i.e., a location should directly follow the preposition "at").

3.5.2 Testing Learned Spatial Relationships. Building upon the hypothesis that the model was able to learn the spatial relationships for the recognition of named entities (i.e., unseen named entities can be identified by sentence structure), we modify the previous test text to the following:

```
George goes to Facebook for work
```

By changing the sentence structure, the identification of Facebook as an organisation is now dependant on the surrounding words.

The following results are obtained with the above text:

```
Prediction:
word : tag
George : PER
goes : NA
to : NA
Facebook : ORG
for : NA
work : NA
```

Therefore, we conclude that the model was indeed able to learn and model spatial relations between words to correctly identify unseen named entities with varying success.

4 CONCLUSION

In this assignment, we have successfully implemented two deep learning models for NLP tasks.

For the first task of implementing a FNN for the purpose of text prediction/generation, we successfully changed the implementation of the model from an RNN based model in the example code to a simple FFN architecture, as well as successfully modifying the data loading process to ensure that data is loaded in a window fashion to train the model correctly. Various regularisation techniques and hyperparameter tuning was also applied to arrive at a final model yielding the best performance. Thereafter, the effects of parameter sharing on model's training performance was also explored. Finally, we have also successfully implemented an attention mechanism for use with the FFN model and have shown that the addition of attention mechanism to the implementation has significant performance improvements to the vanilla FFN model.

For the second task of implementing a CNN architecture for NER task, we have successfully adapted the original Bi-directional LSTM-CNNs-CRF architecture to use a CNN word-level encoder instead of the original Bi-directional LSTM word-level encoding layer. Next, we added more CNN layers to the word-level encoder and explored the effects of additional layers on the performance of the model. Lastly, we have tested the model's ability to recognise entities which are OOV, and determined that the model was able to learn spatial relations between the input words as it was able to tag OOV Named Entities correctly based on neighbouring prepositions.

REFERENCES

[1] Hakan Inan, Khashayar Khosravi, and Richard Socher. 2016. Tying Word Vectors and Word Classifiers: A Loss Framework for Language Modeling. arXiv:1611.01462 [cs.LG]

[2] Xuezhe Ma and Eduard Hovy. 2016. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. arXiv:1603.01354 [cs.LG]

[3] Lily Rothman. 2015. Facebook History: TheFacebook.com Launched February 4, 2004. https://time.com/3686124/happy-birthday-facebook/

[4] Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. *CoRR* cs.CL/0306050 (2003). http://arxiv.org/abs/cs/0306050

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc., 5998–6008. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

A GENERATED WORDS FROM FNNMODEL

```
. the song had already been viewed from moving pesca
    ↪ and nebaioth to the basis to be returned as a
wild @-@ humanities humiliating his <unk> . extracted
    ↪  pitching this was revised not pervasive on
    ↪ the yorke with <unk> threadlike
. the en route , and the music transit consists of
    ↪ richard <unk> . <eos> the only sole
    ↪ recruiting elementary
adorned temporarily english england , but failed to
    ↪ gain the general election lieutenant vietnam .
    ↪  congress had not been seated
without any water or from the civil war , describing
    ↪ him as a " <unk> , but the chinese 's
headshrinkers to head , like blaine of khandoba took
    ↪ the focus of a weak marriage to traditional
    ↪ christmas featuring advice
and her love songs they are words , and live in areas
    ↪  and more disastrous . they are a hard
watery to plays field and against the streets , but
    ↪ soon ophelia , where he was atkin to free .
" the general bromwich service for english conquests .
    ↪  on april , he was placed shortly after his
    ↪ string norwalk
, he shot to the commandos upon the series . the
    ↪ leibniz deal uses block millais received a
    ↪ number of
```