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

For 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 layer will then convert the word IDs to its word vectors. These word vectors will be transposed and "concatenated" and fed as inputs 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 test, 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 2.2.4, 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

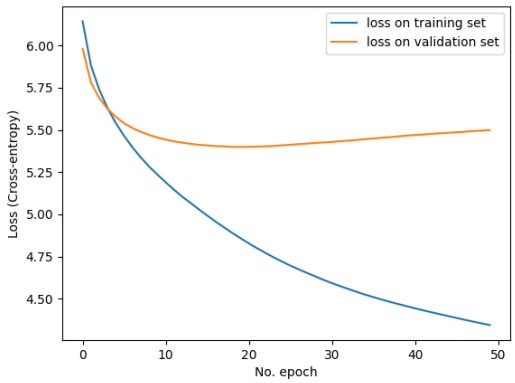

Figure 2: Loss of Baseline Model Over 50 Epochs

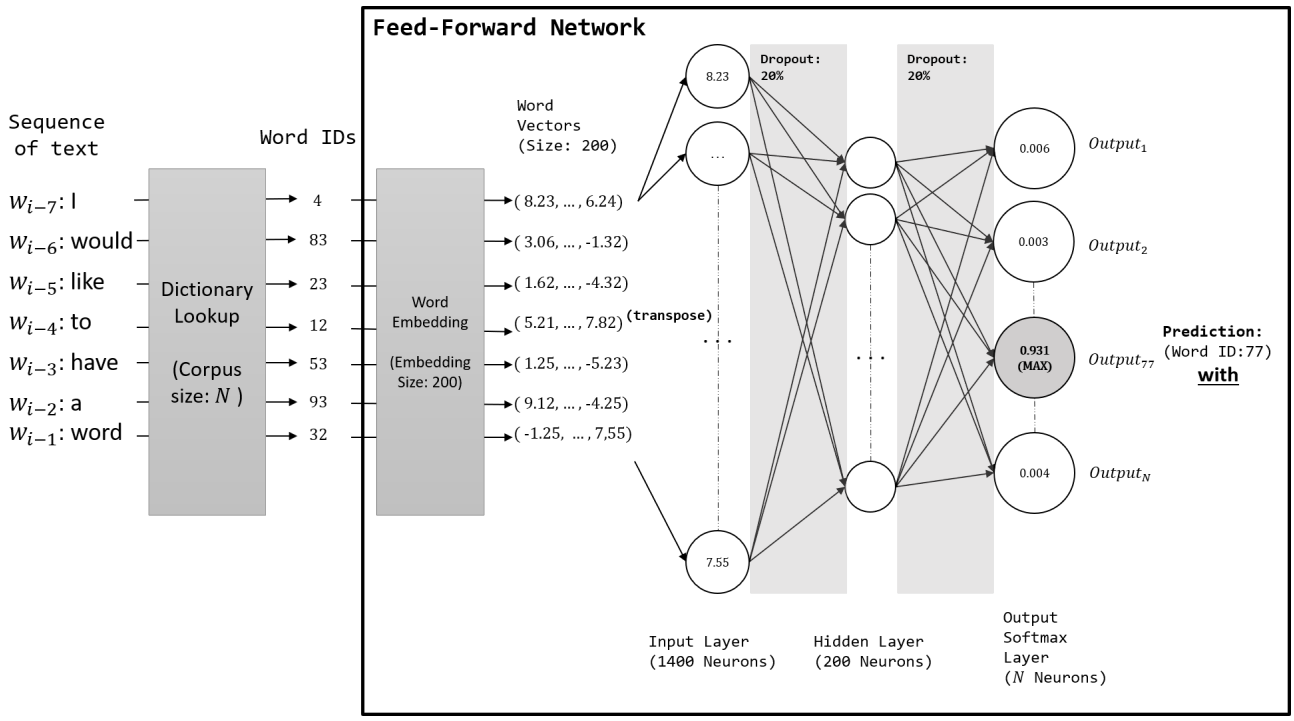

Figure 1: Architecture of FFN

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 clearly 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 when predicting inputs from the training set, it would not perform well for the prediction of 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 overfitting 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:

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

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

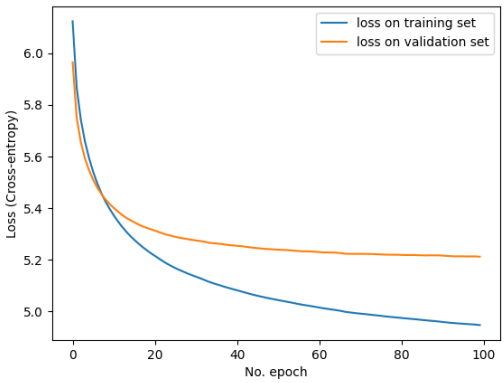

Figure 3: Loss of Model with L2 Over 100 Epochs

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 hyperparameters yielding the best performing 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 **??**, 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 is 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.

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]

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
```