Attn: Shafiq Joty (Asst. Prof)

CE/CZ4045 Natural Language Processing

We hereby declare that the attached group assignment has been researched, undertaken, completed and submitted as a collective effort by the group members listed below. We have honored the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work. We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work.

Name	Signature / Date
Lee Kai Shern U1820793J	[illegible] 26-Nov-20
Tan Zarn Yao U1820414C	[illegible] 26-Nov-20
Wang Wee Jia U1820983E	[illegible] 26-Nov-20
Yew Wei Chee U1820962G	[illegible] 26-Nov-20

Important note:

Name must **EXACTLY MATCH** the one printed on your Matriculation Card. Any mismatch leads to **THREE (3)** marks deduction.

Introduction

In this assignment, the application of deep learning model using PyTorch is demonstrated to implement word language model and name entity recognition (NER) tagging. In the first section, we will use a feed forward neural network to implement an 8-gram. Next, we will implement a word level encoder using convolutional neural network (CNN), the encoder will then be used in a NER model.

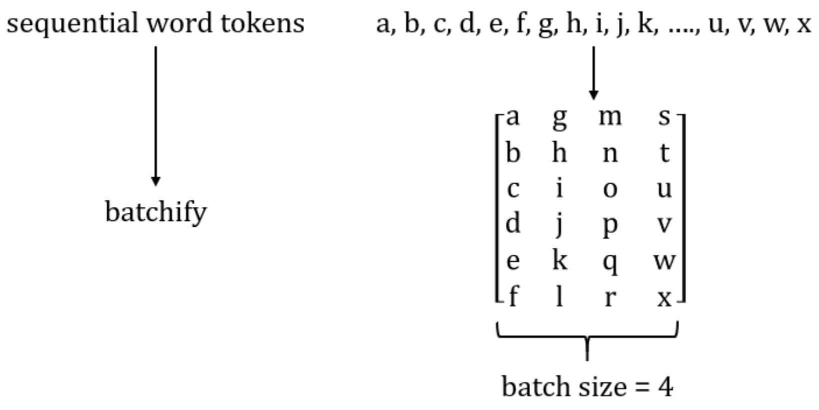
1. Word Language Model

Language model is used to predict the probability of preceding word based on currently available words in the sequence. Neural networks have gain trend in development of language model as it has surpassed the statistical language models in their effectiveness.

1.1. Data Preprocessing

In this assignment, we will be using the wikitext-2 dataset *[1]*. The dataset contains 3 files: train.txt, valid.txt and test.txt. Each of these files contains word tokens in sequence which are already split by spaces. When loading the word tokens from the text files, at the end of each line, we will add an <eos> token representing the end of line. Then we convert those word tokens into sequence of word ids.

Next, we will need to split the dataset into batches. Starting from the sequential data, a "batchify" function is implemented such that it arranges the dataset into columns. The number of columns is equal to the batch size. Each of these columns is treated independently by the model.



The problem with batchify function is that the dependence of the first word token each column with the last word token in the previous column cannot be learned. For example, the dependence of token 'g' on 'f' cannot be learned. However, the benefit of using such batching method is that it allows more efficient batch processing.

To implement an 8-gram, we will choose the ith row to the $(i+6)$th row as the input data and the $(i+7)$th row as the target, where i is the training step at a particular epoch.

1.2. Feed Forward Neural Network (FNN)

FNN is the first neural language model developed *[2]* and it has similar architecture as figure below.

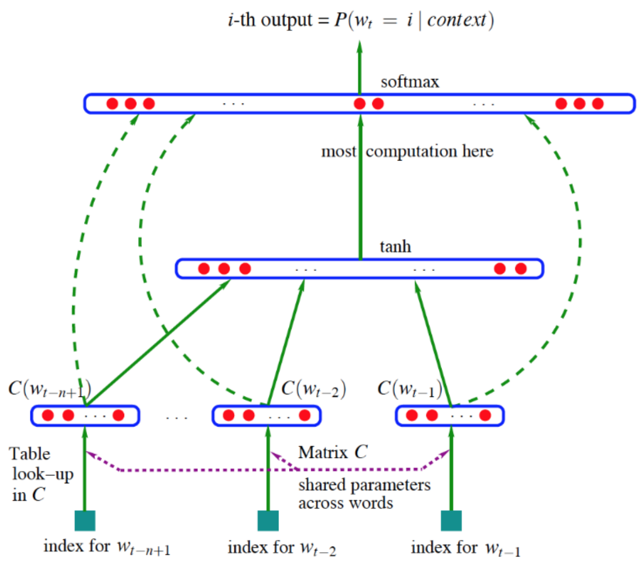


Image from FNN Architecture

The model will learn the distributed representation of each word (embedding matrix C) and the probability function of a word sequence as a function of their distributed representations. Each word in the sequence has an embedding of 200 and will be fed into the network. The hidden layer is *tanh* activation of 200 neurons and output layer is *Softmax* layer.

Else, the predictor can be expressed as:

$$p = softmax(b + U \tanh(d + Hx))$$

Where x is the concatenation of the input word feature vectors or $\mathrm{x} = \mathrm{concat}[\mathrm{C}(\mathrm{w}_0), \mathrm{C}(\mathrm{w}_1) \ldots \mathrm{C}(\mathrm{w}_\mathrm{n})]$, U is output layer weight matrix and H is hidden layer weight matrix.

1.3. Experiment and Result

Firstly, the FNNModel is trained with various SGD variants including Adam, RMSProp and SGD with Momentum. The loss function used is negative log likelihood (multiclass cross-entropy), as expressed in the equation below where $x_i \ldots x_{i+6}$ is the input sequence, $y_i = x_{i+7}$ is the correct predicted word token and n is the batch size. The probability is given by the *Softmax* activation output of the final layer of the FNN.

$$NLL = \frac{1}{n} \sum_{i=1}^{n} - \log p(y_i | x_i \ldots x_{i+6})$$

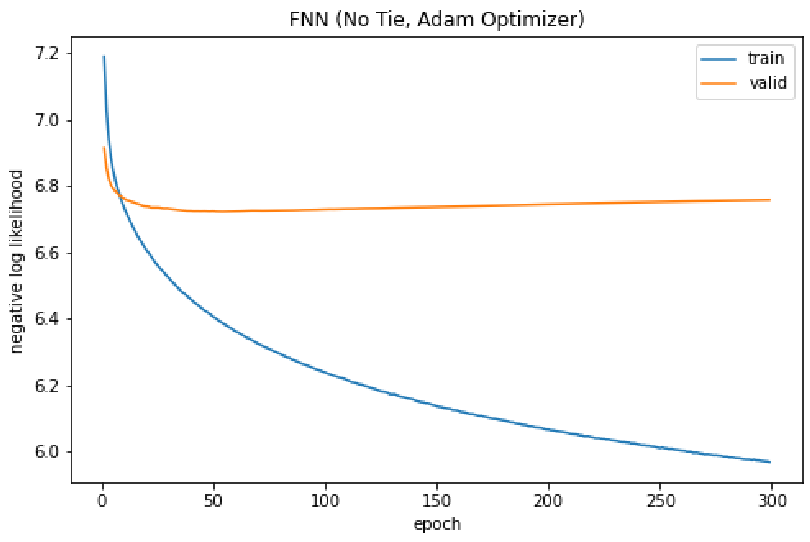
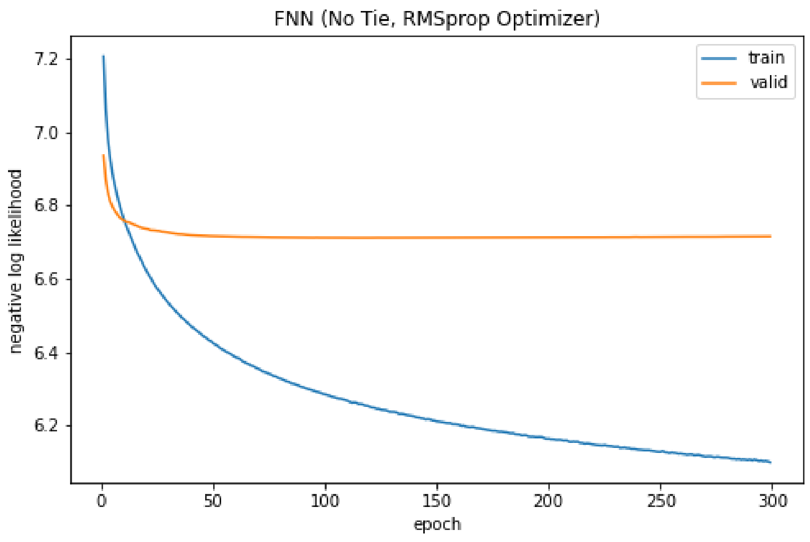
The best model is then selected based on the perplexity score on the valid set. Perplexity can be calculated using the formula below:

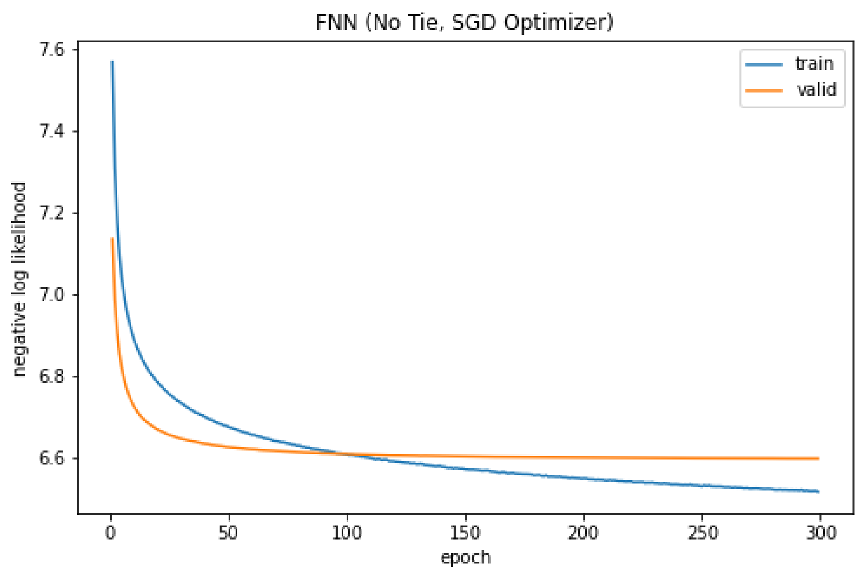
$$\text{Perplexity} = e^{\text{NLL}}$$

Here, we implemented 2 kinds of FNN:

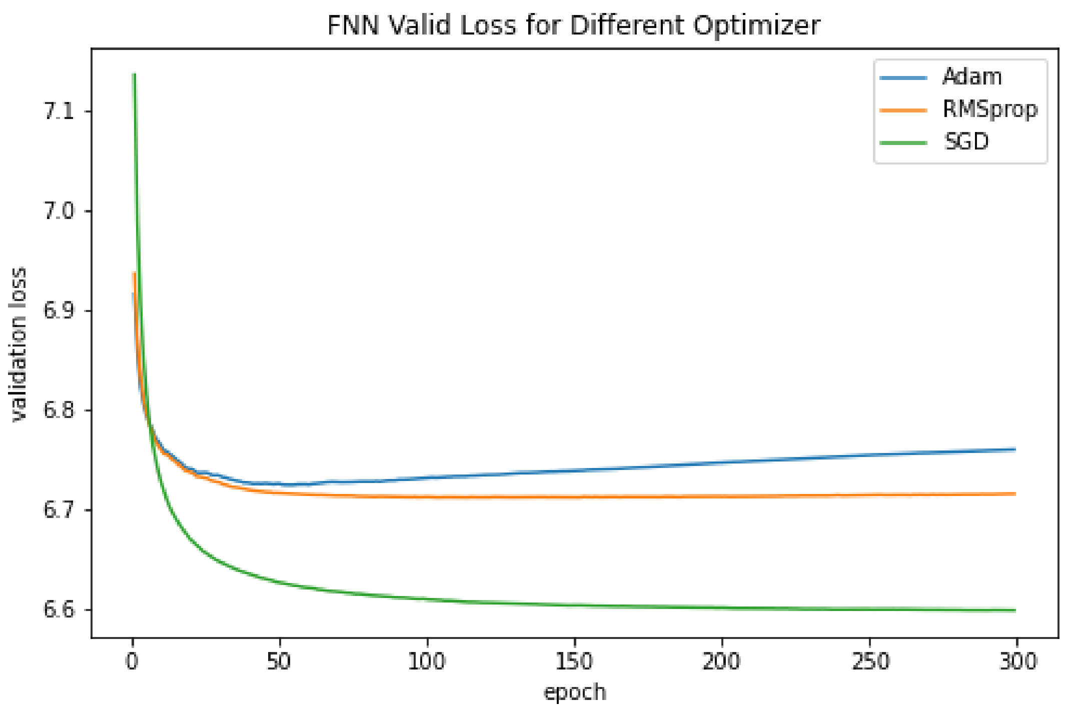
- **No tie**: the output layer weight matrix is independent of the embedding look-up matrix.
- **Tie**: the output layer weight matrix is shared with the embedding look-up matrix; they are both the same.

Firstly, we train the FNN with no tie. The negative log likelihood against epoch is shown below:

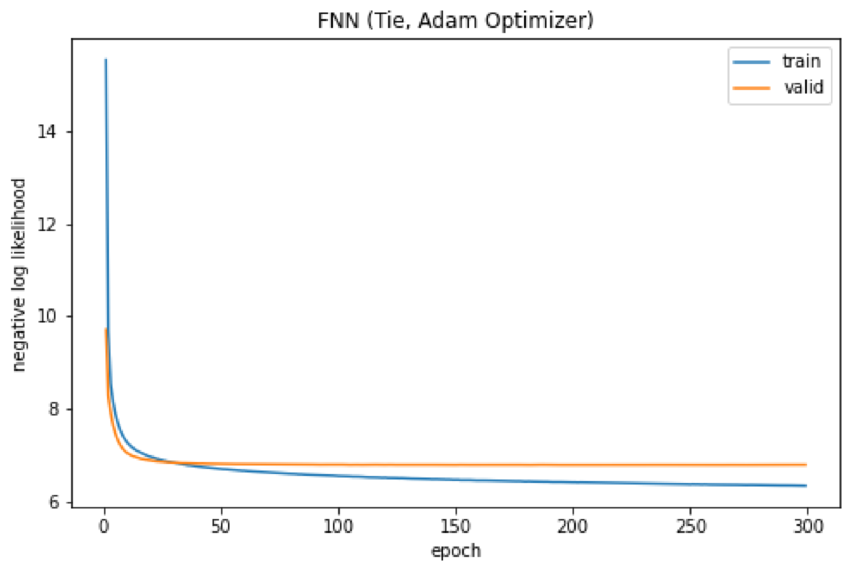
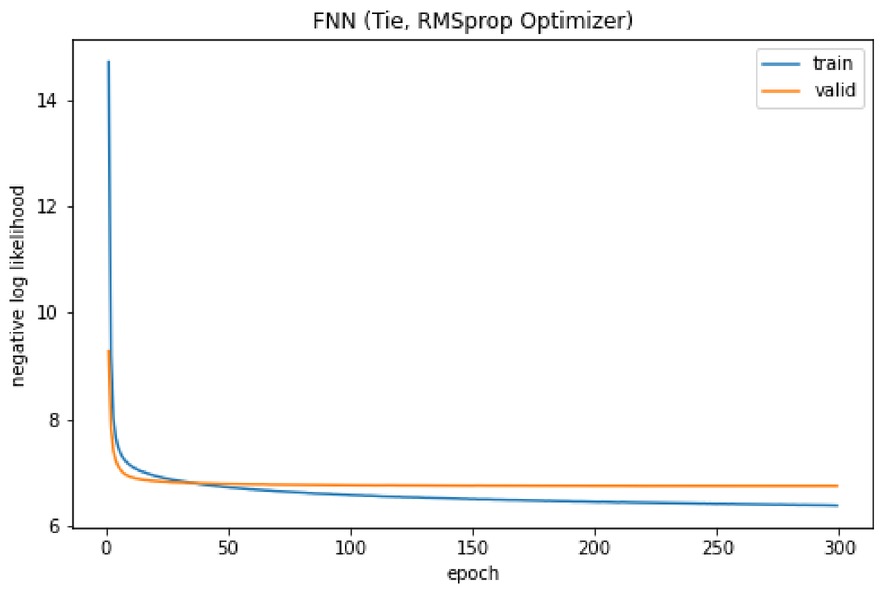
Optimizer	No Tie
Adam	
RMSProp	

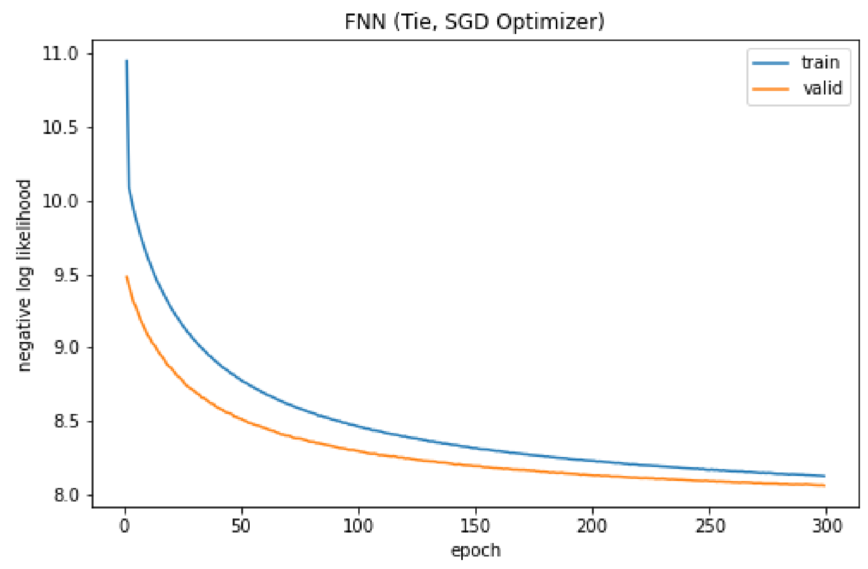
Optimizer	No Tie
SGD	

Comparing the valid negative log likelihood for model trained using the different optimizers, we can observe that SGD with Momentum is the best as it manages to achieve the lowest negative log likelihood, which also means it has the lowest perplexity.

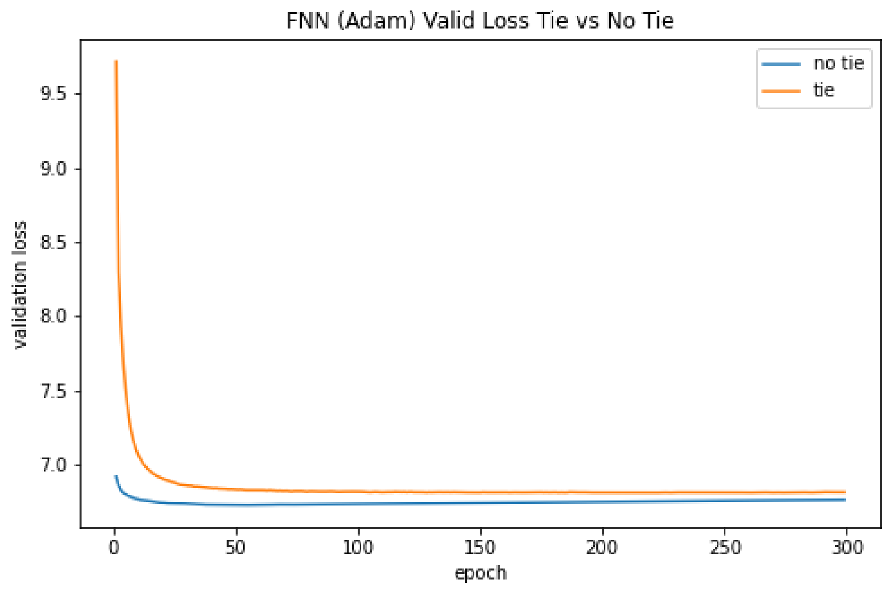


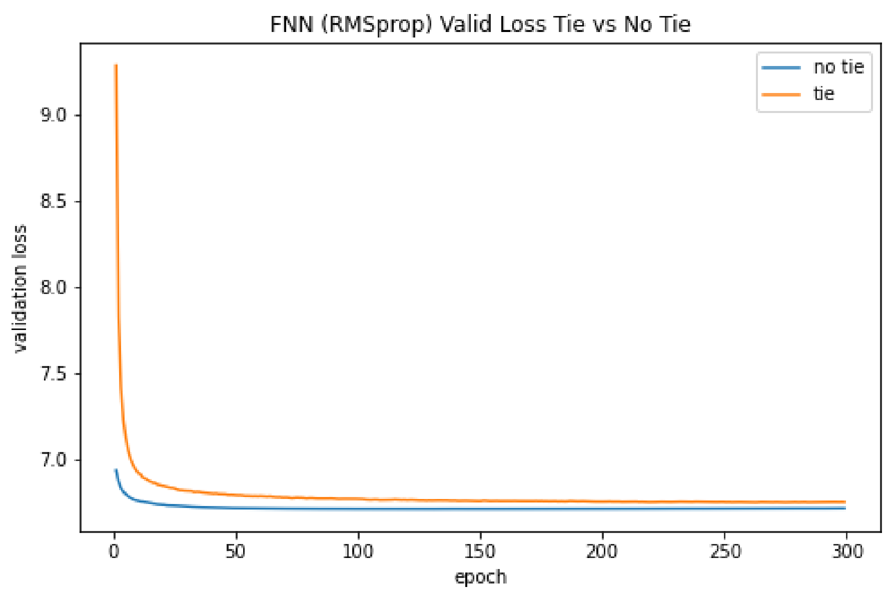
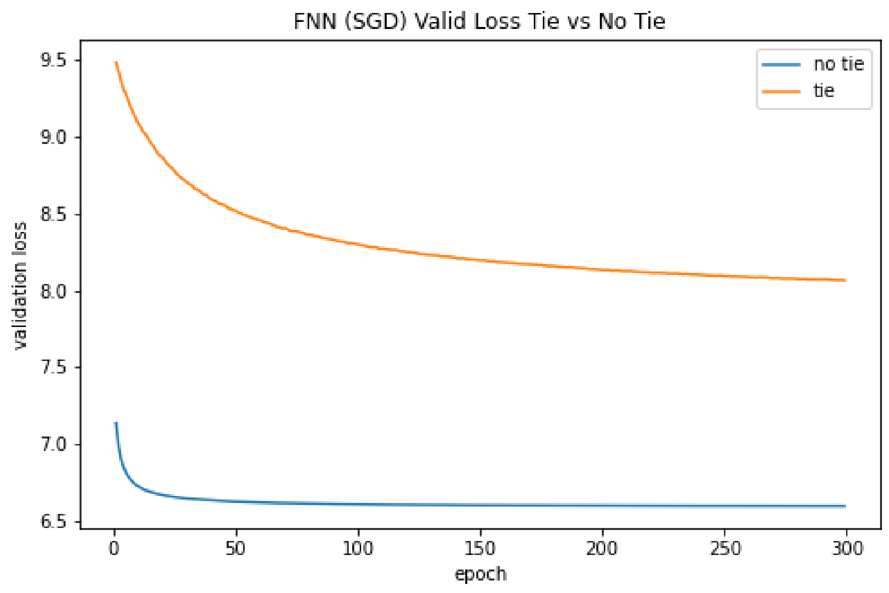
Next, we train the FNN model with tie. The negative log likelihood against epoch is shown below:

Optimizer	Tie
Adam	
RMSProp	

Optimizer	**Tie**
SGD	

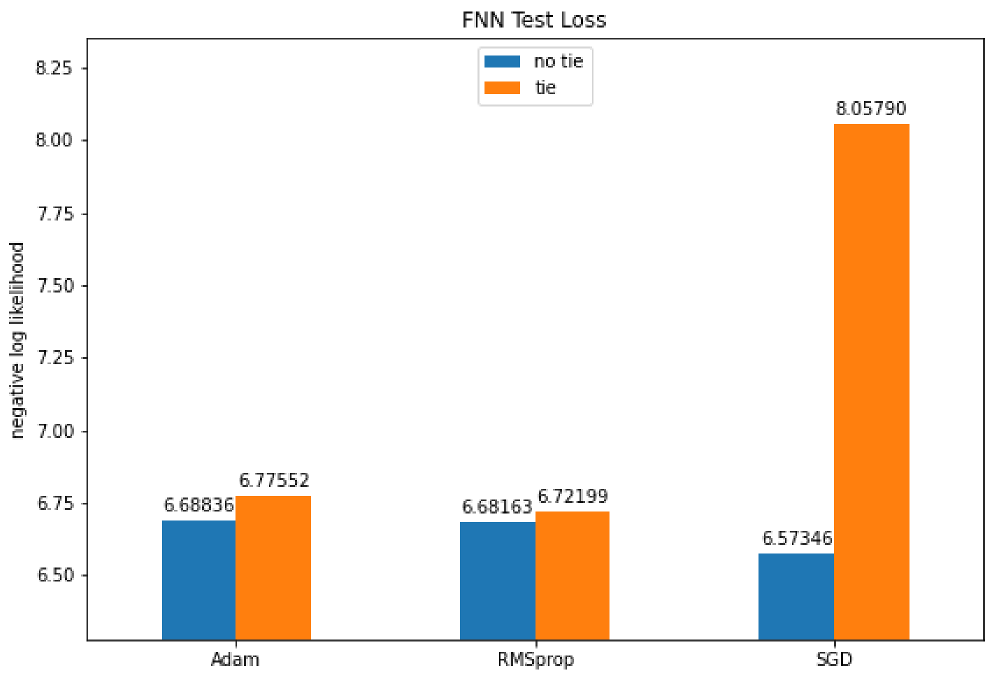
We can see that after sharing the embedding look-up matrix and the output layer weight matrix, the negative log likelihood becomes higher, indicating a worse performance model.

Optimizer	**Comparison**
Adam	

Optimizer	Comparison
RMSProp	
SGD	

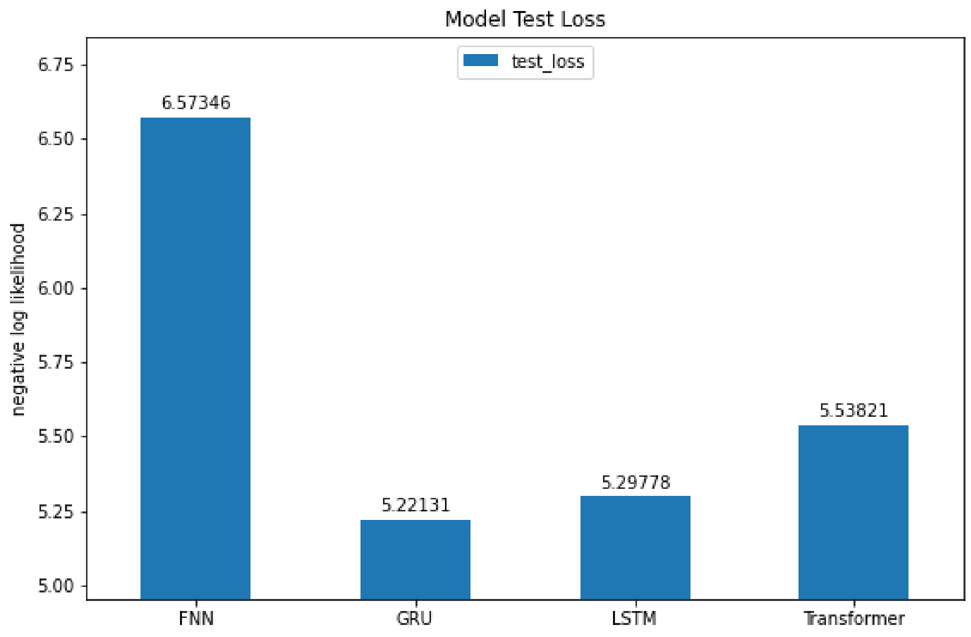
The test negative log likelihood and perplexity are shown below:

Optimizer	Perplexity (No Tie)	Perplexity (Tie)
Adam	803.00	876.13
RMSProp	797.62	830.47
SGD	**715.84**	3158.65

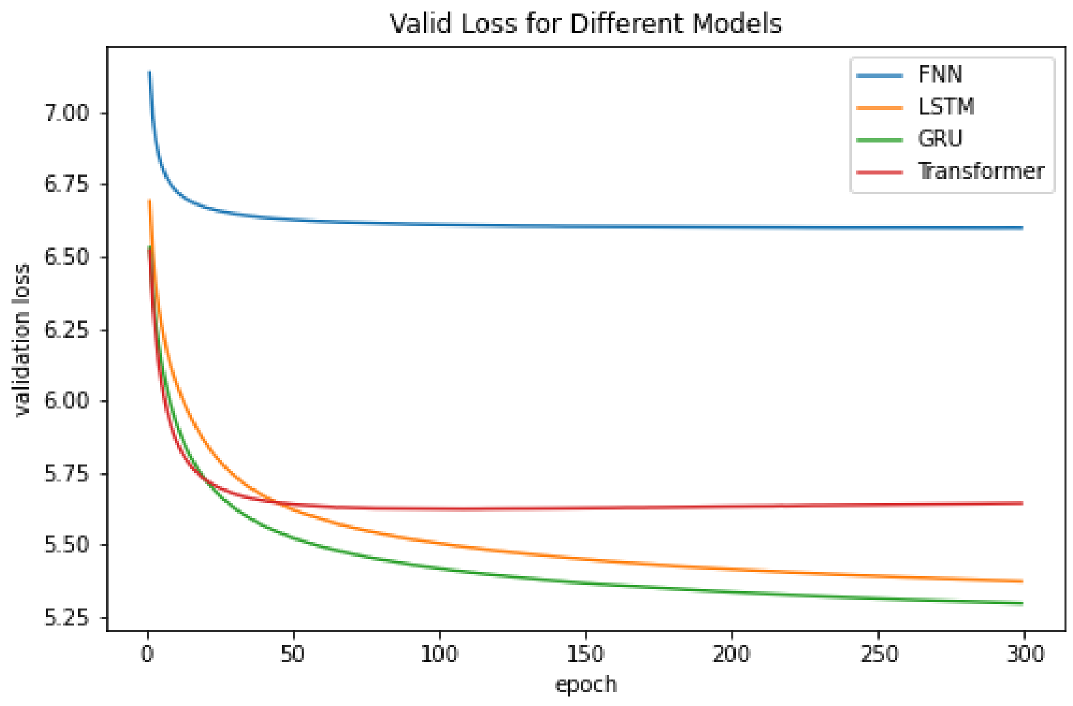


Bar chart showing the NLL of different optimizer

Besides, we also trained LSTM, GRU and Transformer based model and compared it with the performance of FNN model. The test and validation negative log likelihood are shown below. We can see that all of these models performed better than the FNN model. However, the training time is much longer than using FNN model.



Bar chart showing the NLL of different models



Graph showing the validation loss of different model

Model	Training Time per Epoch (s)
FNN	3.89836
GRU	17.61515
LSTM	17.94491
Transformer	21.37633

The Transformer model does not perform as good as expected. This might be caused by chunking of text of fixed length causes context fragmentation as the splitting of text is done without respecting the sentence or other semantic boundary.

1.4. Text Generation

For text generation using FNN, we must first receive an input of 7-word tokens, then convert them into word ids. Firstly, we will use those 7-word ids as the input to the FNN, then will take the output word id with the highest probability as the next word. At subsequent steps, we take the 2nd to the 7th input word ids together with the predicted word id at the previous time step as the input. This process is continued until a particular length of text is generated or until <eos> token is the output.

	input		predicted next word
Step 1:	[a, b, c, d, e, f, g]	$\longrightarrow$	m
Step 2:	[b, c, d, e, f, g, m]	$\longrightarrow$	h
Step 3:	[c, d, e, f, g, m, h]	$\longrightarrow$	o

The sample text generated using the input string “A family of 4 children lives at” until the first <eos> token is met for various models are shown below:

Model	**Generated Text**
FNN	A family of 4 children lives at between and Rossini Sovetskaya many was in in Villa . <eos>
GRU	A family of 4 children lives at Industrial in Kristiansand grams . in a City the Army, Like 2013 authority on southwest against and <unk> . . the of the White of <eos>
LSTM	A family of 4 children lives at Industrial in 24 league . in a February the season , Like 2013 budget 1929 first against and book . . , improved the White of <eos>
Transformer	A family of 4 children lives at the end of the Harwich Force in the season . <eos>

2. Name Entity Recognition (NER)

Name Entity Recognition (NER) is a subtask of information extraction that seeks to locate and classify named entities mentioned in unstructured text into pre-defined categories such as person names, organizations, locations and time.

2.1. Data Preprocessing

The English data from CoNLL 2003 *[3]* shared task is used in this assignment. The dataset contains four different types of named entities: PERSON, LOCATION, ORGANIZATION and MISC and uses the BIO tagging scheme. We will apply some basic preprocessing steps to generate tag mapping, word mapping and character mapping.

First, all the digits in the words are replaced by 0. This is because the information present in numerical digits does not help in predicting the entity. By replacing the digits with 0, the model can concentrate on more important alphabets.

Next, we convert the tag scheme of the dataset from BIO to BIOES. After we have updated the tag scheme, we now have a list of sentences which are words along with their modified tags. Now, we want to map these individual words, tags and characters in each word, to unique numerical ID's so that each unique word, character and tag in the vocabulary is represented by a particular integer ID. These indices for word, tags and characters help us employ matrix operations inside the neural network architecture, which are considerably faster. We found that there are 17493 unique words, 75 unique characters and 19 unique named entity tags in the dataset.

We then create a function that returns a list of dictionaries from the dataset (one dictionary per sentence). Each of the dictionary returned contains: 1. list of all words in the sentence; 2. list of word index for all words in the sentence; 3. list of lists, containing character id of each character for words in the sentence; and 4. list of tag for each word in the sentence.

Finally, we load the pre-trained word embeddings from GloVe, which contains 100-dimension vectors trained on the (Wikipedia 2014 + Gigaword 5) corpus containing 6 billion words.

2.2. Model Architecture

The original model implements a word-level encoder with a bi-directional LSTM network *[4]*. The LSTM model has 2 layers:

1. The forward layer takes in a sequence of word vectors and generates a new vector based on what it has seen so far in the forward direction (starting from the start word up until current word) this vector can be thought of as a summary of all the words it has seen.

2. The backwards layer does the same but in opposite direction, i.e., from the end of the sentence to the current word.

The forward vector and the backwards vector at current word concatenate to generate a unified representation.

The figure below shows the architecture of the bi-directional LSTM model in detail.

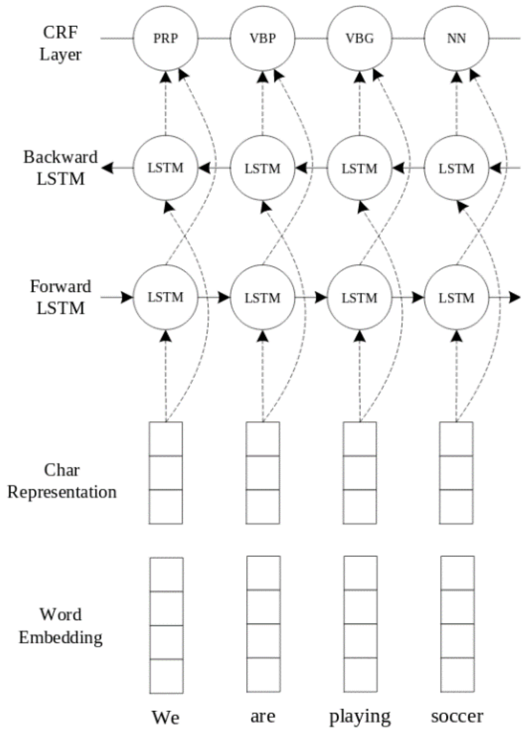
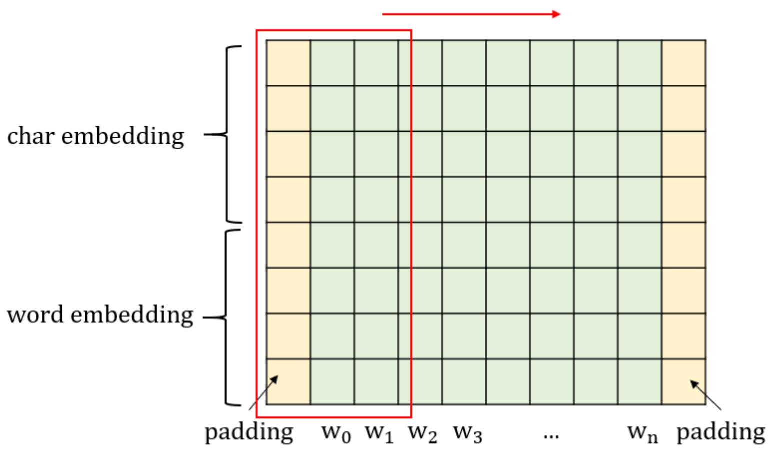


Image from Github End-to-End Sequence Labelling

In this assignment, we replace the LSTM-based word-level encoder with a Convolutional Neural Network (CNN) layer and compare their performances.

The CNN is required to have the same output dimension as the LSTM-based word-level encoder. Therefore, the number of output channels for CNN will be 2 times the hidden dimension of LSTM layer because the LSTM is bidirectional. The kernel size used is 3xD where D is the sum of the dimension of word embeddings and the dimension of character embeddings. We add 1 padding to both side of the sequence so that the output length of CNN will be the same as the LSTM. We used leaky ReLU with a negative slope of 0.5 as the output activation function for CNN layers.



2.3. Experiment and Result

First, we run the original model (LSTM-based word-level encoder) and plot its F1 score against number of epochs.

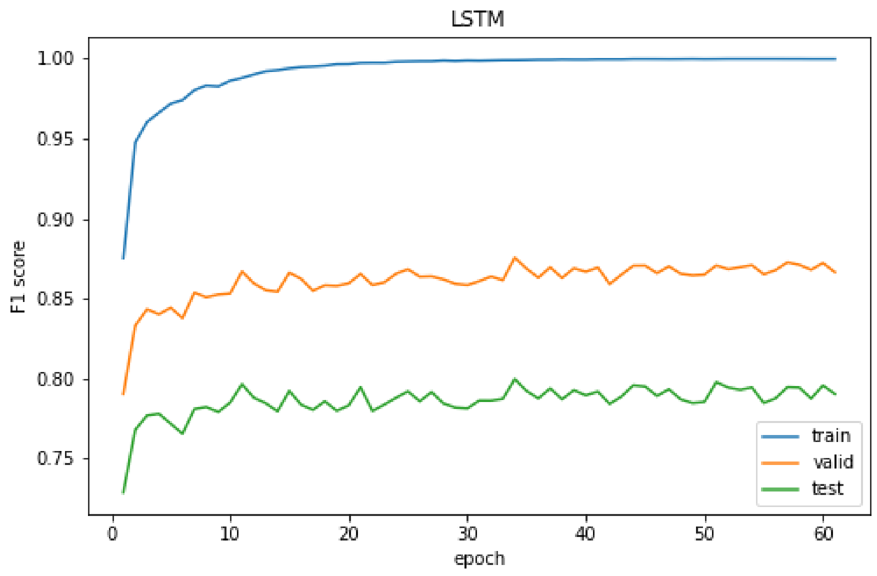


Figure showing train, valid, test F1 score of word-level encoding with LSTM

Next, we replace the LSTM-based word-level encoder with a single CNN layer.

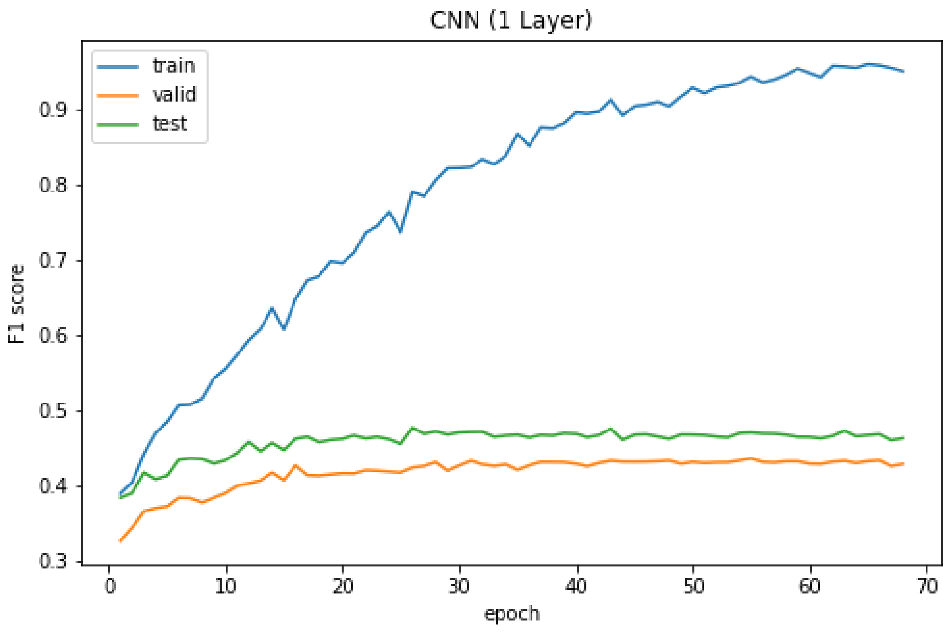


Figure showing train, valid, test F1 score of word-level encoding with single CNN layer

We gradually increase the number of CNN layers and plot their F1 score against number of epochs.

<u>2 CNN layers</u>

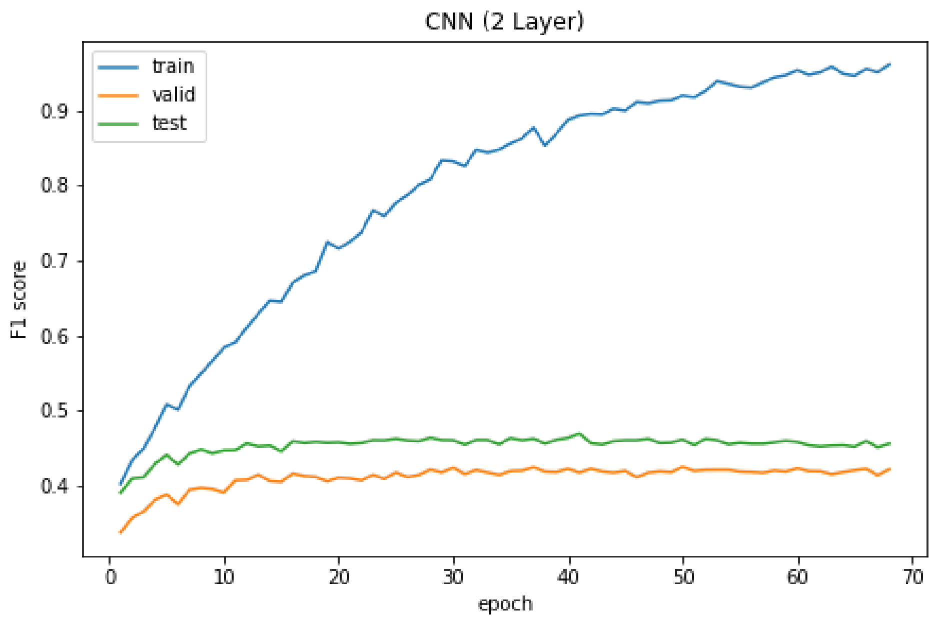


Figure showing train, valid, test F1 score of word-level encoding with 2 CNN layers

<u>3 CNN layers</u>

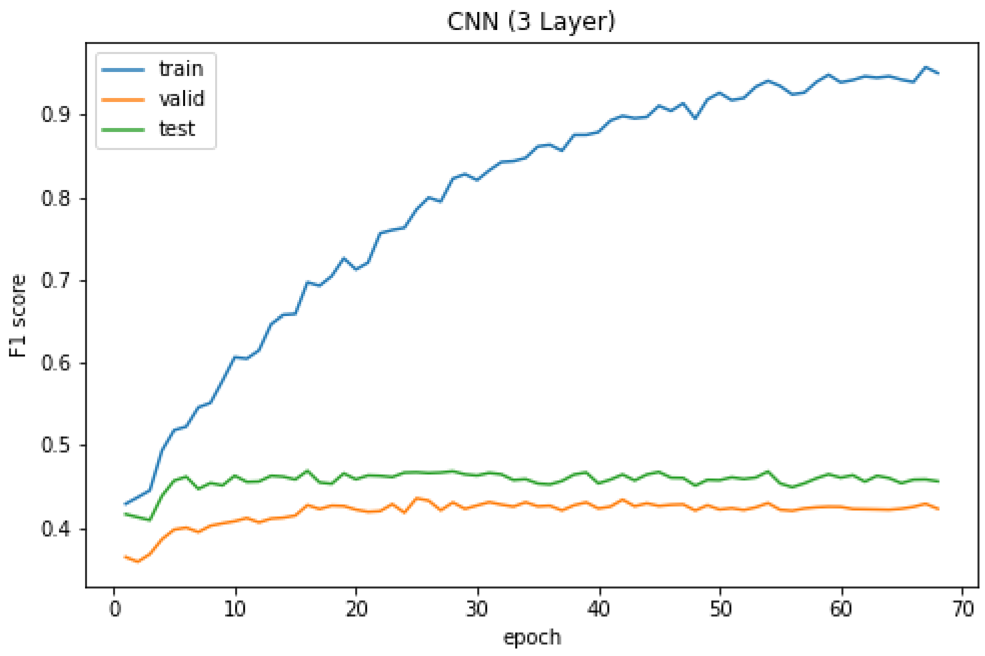


Figure showing train, valid, test F1 score of word-level encoding with 3 CNN layers

4 CNN layers

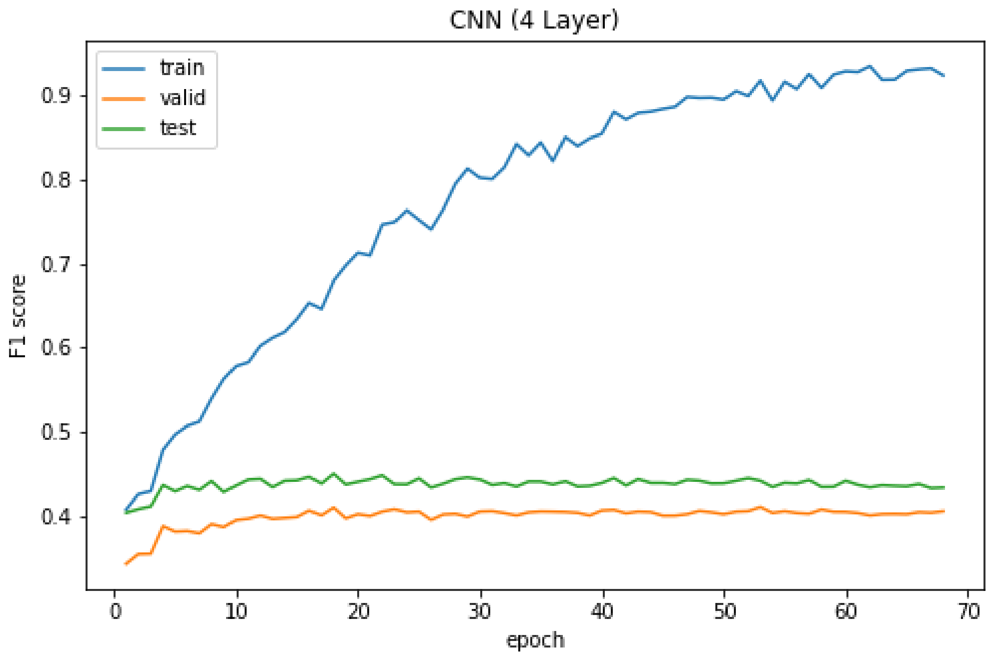


Figure showing train, valid, test F1 score of word-level encoding with 4 CNN layers

5 CNN layers

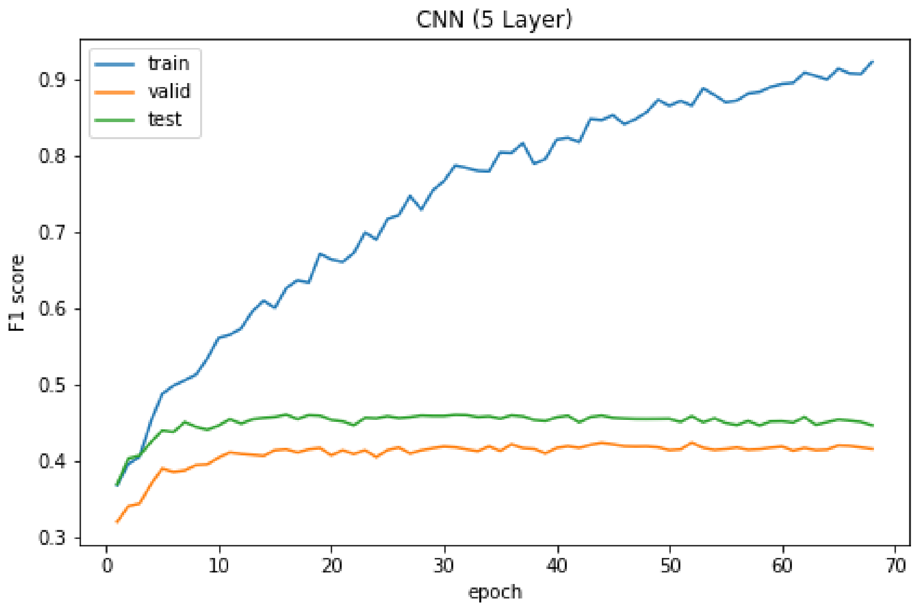


Figure showing train, valid, test F1 score of word-level encoding with 5 CNN layers

We compare the F1 scores of all the models with CNN layers for both validation set and test set.

Validation Set

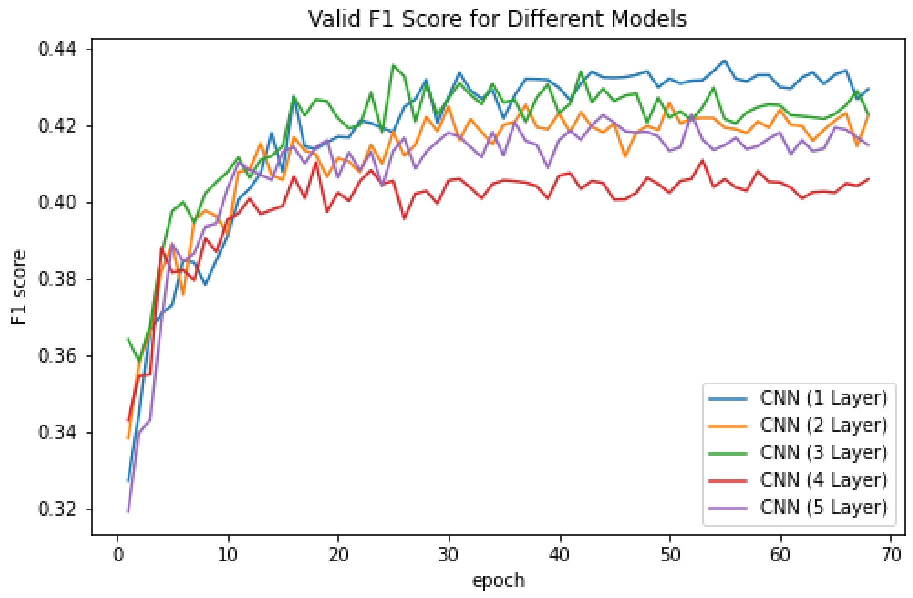


Figure showing comparison of valid F1 score between models with different number of CNN layers

Test Set

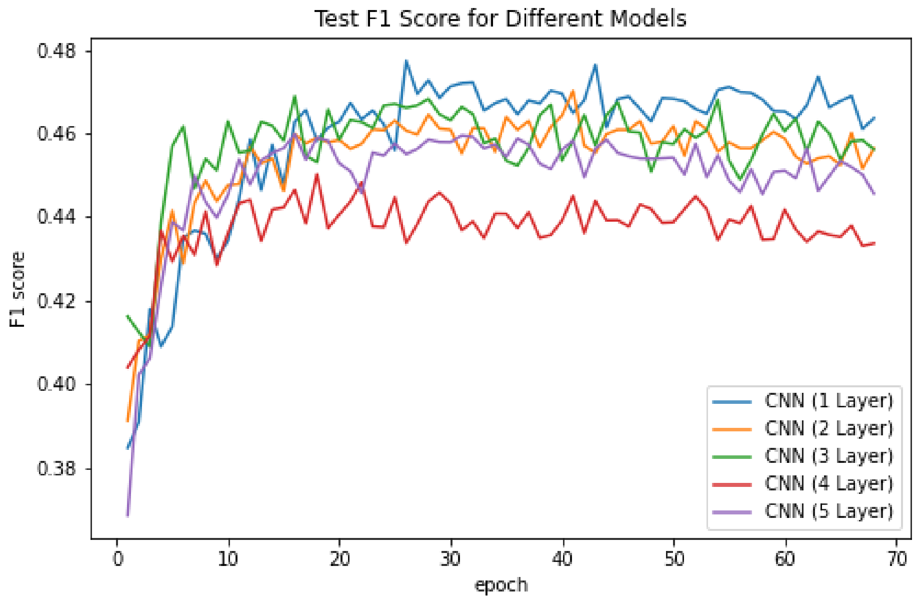
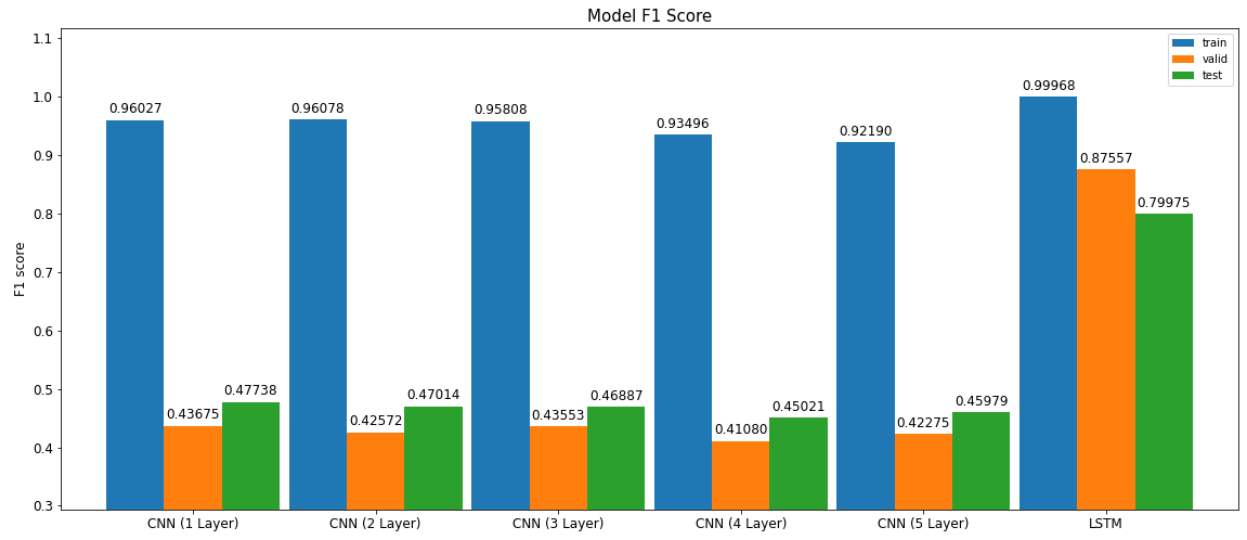


Figure showing comparison of test F1 score between models with different number of CNN layers

As shown in the two graphs above, we observe that the model with a single CNN layer outperforms other models with more CNN layers. This might be due to overfitting after adding more layers. Adding more layers of CNN can help to extract more high-level features up to only a certain extent, beyond that the model will start to overfit especially on small datasets. In our dataset, we only have 14041 sentences for training, which is a very small dataset. Therefore, 1 layer of CNN is sufficient for generalization in this case.

Finally, we compare the maximum F1 score of each model on train set, validation set and test set.



Bar chart displaying F1 score of LSTM and CNN of different number of layers

We can observe from the bar chart above that the original model which implements a word-level encoder with a bi-directional LSTM network outperforms all other models which uses CNN layer instead in all 3 splits of dataset.

This might be due to the small window size of CNN kernel used which is insufficient to capture long term dependencies between words within a sequence. In our case, the kernel size used is 3xD which only manages to capture the dependency of the center word with the context words which are 1 unit away. For multi-layer CNNs, the receptive field is larger for higher level layers, which may also capture dependencies within a longer sequence. However, the asymmetry processing of input data using CNN may be unsuitable in this case.

Conclusion

Using deep learning neural network model to implement n-gram is much more efficient than using statistical methods. This is because we do not need to store the counts for all n-grams of the corpus, instead, we only need to store the model parameters which is relatively much smaller. Also, we do not need to manually handle the sparsity problems of n-gram ourselves, the neural network model has already been trained to handle these issues.

Besides, NER tagging using deep learning is also much simpler as we do not need to specify the rules ourselves which requires a lot of human effort and knowledge. There exist multiple potential directions for future development. The model can be further improved by implementing multi-task learning approaches to ensemble more useful and correlated information. For example, we can jointly train a neural network model with both the POS and NER tags to improve the intermediate representations learned in our network *[5]*.

References

[1] Pytorch, “pytorch/examples,” GitHub. [Online]. Available: https://github.com/pytorch/examples/tree/master/word_language_model.

[2] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. J. Mach. Learn. Res., 3:1137–1155, March 2003.

[3] Xuezhe Ma and Eduard Hovy. 2016. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: LongPapers). Association for Computational Linguistics, Berlin, Germany (https://arxiv.org/pdf/1603.01354.pdf)

[4] Jayavardhanr, “jayavardhanr/End-to-end-Sequence-Labeling-via-Bi-directional-LSTM-CNNs-CRF-Tutorial,” GitHub, 16-Jul-2018. [Online]. Available: https://github.com/jayavardhanr/End-to-end-Sequence-Labeling-via-Bi-directional-LSTM-CNNs-CRF-Tutorial/blob/master/Named_Entity_Recognition-LSTM-CNN-CRF-Tutorial.ipynb.

[5] Cai, X., Dong, S. & Hu, J. A deep learning model incorporating part of speech and self-matching attention for named entity recognition of Chinese electronic medical records. BMC Med Inform Decis Mak 19, 65 (2019). https://doi.org/10.1186/s12911-019-0762-7

Contribution

Name/Section	1	2
Lee Kai Shern	25%	25%
Tan Zarn Yao	25%	25%
Wang Wee Jia	25%	25%
Yew Wei Chee	25%	25%