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
Ng Joshua Jeremiah	Joshua / 30 Nov 2020
Mark Chua De Wen	Mark / 30 Nov 2020
Koh En Rong	En Rong/ 30 Nov 2020
Khairul Amiruf Bin Ahmad Mohri	Khairul / 30 Nov 2020
Ignatius Lok Zhengrong	Ignatius / 30 Nov 2020

Important note:

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

Contribution:
Question 1 - Joshua, Mark & En Rong
Question 2 - Khairul & Ignatius

Question 1: Word-based Language Model using Forward Neural Network (FNN)

1. FNN Model

Our FNN model takes in 7 input words and encodes them into word embeddings, represented by w_1-w_7 and matrix C_1-C_7 in figure 1, respectively. The word embeddings are then fed into a hidden layer, with an activation function of tanh. The result is fed into an output layer, followed up by a softmax function to calculate the output word, the 8th word. The output word is compared with a target word using negative log likelihood loss function and the loss is back propagated through the FNN. The weights are adjusted based on the optimization algorithm chosen and it's parameters, such as the learning rate. Lastly, the working window is shifted from the 1-7 word input to the 2-8 word input, 8th word being the predicted word, and the process is repeated until the training ends.

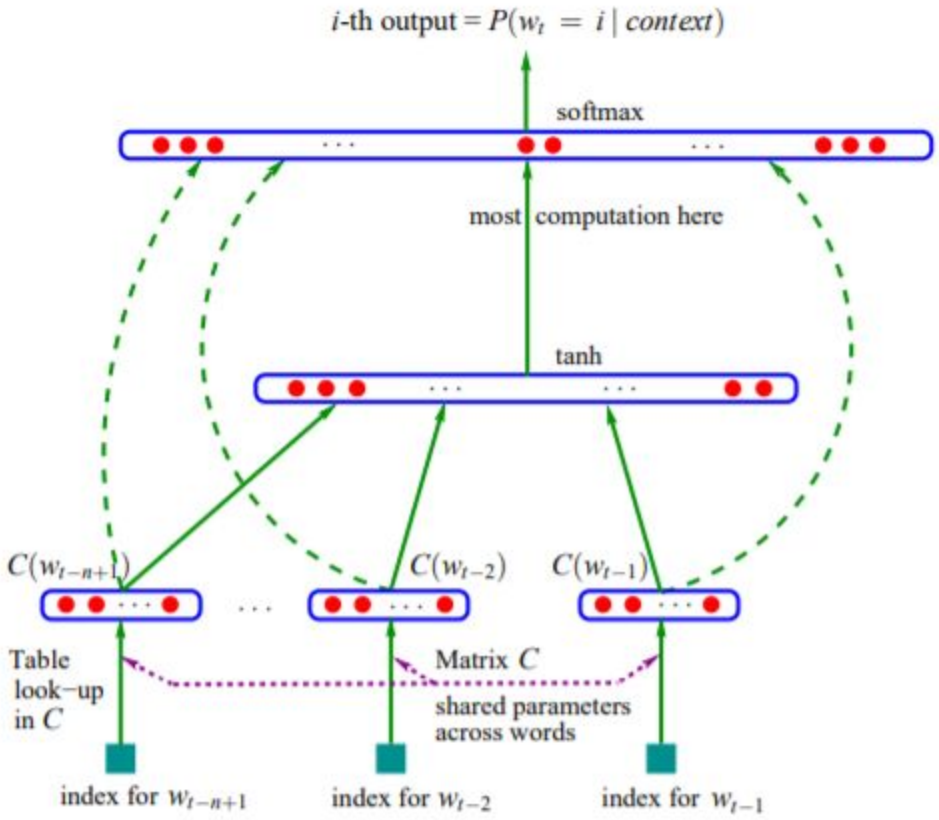


Figure 1: A Neural Probabilistic Language Model by (Yoshua et al. 1142)

1.1 Corpus

The corpus takes in all our data, training, valid and test data. From these, it performs tokenization and forms a dictionary, adding every unique word that it finds. The corpus is later used to find the number of tokens in our dictionary, which is essential for calculation of loss.

1.2 Batching

The benchmark training method used is mini-batch gradient descent, where the weights are updated after every batch. For our experiments, our sequence length is 7 and our batch size is 100, hence the resultant matrix for each batch is 100x7, with a total of 2983 batches. Each batch is independent from another, allowing for more efficient batch processing. This process is called batchify, and it is done for our training data, valid data and test data. Batches can be retrieved using the get_batch function, which extracts the batch corresponding to the index provided. The get_batch function acts as the working window, with each batching having 7 words and producing the corresponding target points as well.

1.3 Training

During training, data is put through the model batch by batch, calculating the loss based on negative log likelihood and backpropagating the loss through the model. This is also known as batch gradient descent. For FNN, gradient clipping is not required as it does not face the issue of exploding gradients, unlike RNNs and LSTMs. The perplexity can be estimated using an exponential function of the loss. For each batch, the time taken, loss and perplexity is recorded for analysis later on. This is done until all data has been put through the model, forming 1 epoch. Multiple epochs can be run for a more accurate model with lower perplexity.

1.4 Evaluation

The model is 1st set into evaluation mode which disables dropouts. The evaluation is started by passing the validation data through the trained model with the intended 7 words as data and the next 8th target word. The total loss is then calculated using the parameter of the length of data and the loss function used. It then returns the loss value of the validation data by dividing the total loss accumulated over the length of the validation data. The gradient calculation is disabled for the whole evaluation process.

1.5 Generation of text

To generate 1000 words, a random word that is determined by the random seed is used. The model is set to evaluation mode and the random word is passed through to get the weight of the word. This weight will then be used to get the highest probability of the next word. The next word will then be passed through the model to get its weight and predict the next highest probability word. Each word will be encoded in UTF-8 to ensure the words are output into a text file. This process will repeat until 1000 words are generated and saved into the file. [Refer to generated text in Appendix A]

2. Improvements

2.1 Optimization Algorithms

The given optimization method is batch gradient descent. However, the pytorch library provides many other optimization algorithms that we will be testing. These include:

SGD

- The simplest optimization algorithm that uses an iterative method for optimizing object function with suitable smoothness properties, Stochastic Gradient Descent chooses a single random observation and uses that gradient at each observation to converge to a smaller value. It updates the parameter for each training sample one by one, and with these frequent updates, it is generally more computationally expensive and might result in noisy gradients, which may cause the error rate to jump instead of decreasing.

ASGD

- Averaged Stochastic Gradient Descent is a recursive algorithm that accelerates stochastic approximation by averaging. Gradient Descent may get close to the optimal but may not really converge so ASGD does regular SGD but takes the mean as the final solution.

Adadelta

- Adadelta is a more robust extension of adagrad that adapts learning rates based on a moving window of gradient updates, instead of accumulating all past gradients.

Adagrad

- A modified version of SGD, AdaGrad uses a per-parameter learning rate that improves performance for problems with sparse gradients. Depending on the parameters, the updates/learning rate may vary. This helps in speeding up the learning for sparse datasets and eliminates the need for manual adjustment of the learning rate. However this will accumulate the squared gradients in the denominator and will result in the algorithm unable to acquire additional knowledge.

Adamax

- A variant of Adam based on the infinity norm, Adamax is a stable algorithm that helps solve the problem of Adam algorithm's variants becoming numerically unstable for sparse parameter updates.

Rprop

- Short for Resilient Backpropagation, Rprop is a batch-update algorithm that is used for full batch optimization. This iterative gradient-based optimization method with adaptive individual step sizes does not require adjusting hyperparameters of the learning algorithm. This fast and accurate algorithm implementation is however not that straightforward as it's variants require not only info about current and past gradients but also the error of current and past training batches.

RMSprop

- Also known as Root Mean Square Propagation, RMSprop is an unpublished yet widely-known variant of SGD for mini-batch learning. It solves the problem of Rprop where if the gradient of successive mini-batches vary by too large, an amount is mitigated by using a moving average of squared gradient for each weight. By dividing the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight, it shows good adaptation of learning rate in different applications.

Adam

- Adam is an adaptive learning rate method that can be seen as a combination of RMSprop and SGD with momentum. It uses a squared gradient to scale the learning rate like RMSprop and moving average of the gradient instead of the gradient itself in SGD.

AdamW

- AdamW is an improved version of Adam where the weight decay is done only after controlling the parameter-wise step size. The weight decay is proportionate to the weight itself and does not affect the moving average that much. This model generalises better and has better training loss.

2.2 Scheduler Methods

Aside from optimizers, the pytorch library also provides schedulers to adjust the learning rate based on the number of epochs. This can be useful when the perplexity stagnates across epochs and a smaller learning rate is required for further improvement. Some of the scheduler methods used in our experiment are as follows:

LambdaLR

- LambdaLR changes the learning rate based on a given function, to be triggered every epoch. Being able to provide any function as we please allows for flexibility.

MultiplicativeLR

- MultiplicativeLR multiplies the training rate with a given constant, gamma, every epoch.

StepLR

- Step LR multiplies the training rate by a given constant, gamma, every N epochs.

MultistepLR

- MultistepLR is similar to StepLR, except that you can set specific milestones that the training rate gets altered.

ExponentialLR

- ExponentialLR performs an exponential calculation LR^k on the training rate, where k is specified as a parameter of the scheduler method.

CosineAnnealingLR

- CosineAnnealingLR sets the learning rate every epoch using the cosine annealing formula:

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 + \cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right), \quad T_{cur} \neq (2k+1)T_{max};$$
$$\eta_{t+1} = \eta_t + \frac{1}{2}(\eta_{max} - \eta_{min})\left(1 - \cos\left(\frac{1}{T_{max}}\pi\right)\right), \quad T_{cur} = (2k+1)T_{max}.$$

ReduceLROnPlateau

- ReduceLROnPlateau reduces learning rate when a metric has stopped improving for a set number of epochs (patience).

CyclicLR

- CyclicLR cycles the learning rate between 2 preset boundaries, incrementing by a set amount every epoch.

CosineAnnealingWarmRestarts

- CosineAnnealingWarmRestarts is similar to CosineAnnealingLR, except that it restarts to the default learning rate every N epochs.

2.3 Weight Tying

Weight tying is a process that reduces the total number of parameters in language models. Language models consist of an embedding layer and are represented by vectors that are near each other in cosine distance. This is similar to the softmax matrix in which every word is also represented by vectors. So weight tying allows the reduction of parameters by sharing the soft and embedding matrices which is often done in language models nowaday.

3. Results and Analysis

3.1 FNN vs LSTM vs Transformer

The FNN model outperforms both the LSTM and Transformer models in both the speed/epoch and the test perplexity as in table 1. Despite the difference in perplexity, the model is still unable to form a proper sentence.

Table 1: Comparison of speed and perplexity between FNN, LSTM and Transformer models

Model Type	Speed/epoch	Test Ppl
FNN	31.63	4.37
LSTM	34.51	140.7
Transformer	46.49	221.34

3.2 Optimization Algorithms

The given optimization algorithm is the batch gradient descent, which will be used as a benchmark. We have experimented on some optimizers provided in the pytorch library to compare against the benchmark. It is difficult to judge the optimizers fairly based on their default parameters as the parameters are not optimized for our dataset. That aside, we can observe from table 2 that the fastest optimizer is batch gradient descent and the slowest is Rprop. The

optimizer with the best test perplexity is Adamax while the optimizer with the worst test perplexity is ASGD. Once again, the parameters of the optimizers have yet to be optimized and more experiments can be carried out to find out the optimal parameters for each optimization algorithm.

Table 2: Comparison of speed and perplexity between various optimizer algorithms

Optimizer	Speed/epoch	Test Ppl
Batch GD	31.63	4.37
SGD	34.55	5.4
ASGD	33.65	7.39
Adadelta	40.51	4.07
Adagrad	35.45	4.14
Adamax	39.34	4.04
Rprop	57.98	4.88
RMSprop	36.05	4.41
Adam	38.31	4.79
AdamW	38.93	4.7

3.3 Scheduler Methods

The optimizer chosen for scheduler testing is SGD. The default scheduler used, labelled ‘none’ in table 3 is a multiplicativeLR manually coded, is provided. It divides the training rate by a factor of 4 when there are no improvements in perplexity across epochs. As a result, it shares the same speed/epoch and test perplexity as the multiplicativeLR function in the pytorch library. There is minor differences between the different scheduler methods, with the speed/epoch ranging from 34-35 seconds and they test perplexity ranging from 5.4-6.8. This might be due to 2 reasons: 1) Our parameters used are defaulted from the pytorch website and have not been customized to fit our dataset; 2) We are using only epoch 6 and it has not reached plateau, the point where learning rate decay is necessary.

Table 3: Comparison of speed and perplexity between various scheduler methods

Scheduler	Speed/epoch	Test Ppl
None(SGD)	34.55	5.4
LambdaLR	34.59	5.55
MulitplicativeLR	34.55	5.55
StepLR	34.58	6.1
MultiStepLR	34.55	6.01
ExponentialLR	34.03	6.8
CosineAnnealingLR	34.54	6.1
ReduceLROnPlateau	34.55	5.4
CyclicLR	34.53	5.4
CosineAnnealingWarmRestarts	34.56	5.96

Although the results do not display the pros and cons of each scheduler, we can observe how each of them work by seeing how their learning rates change during training, across epoch 1-6 as shown in table 4. Most of them reflect our understanding from part 2.2, based on the parameters that I have set.

These are some observations from each of the learning rate scheduling methods:

- LambdaLR - function is LR*0.95 every epoch.
- MultiplicativeLR - factor to multiply set to 0.95 every epoch.
- StepLR - step by 0.5 every 2 epochs.
- MultiStepLR - step by 0.5 at milestones epoch 2,5.
- ExponentialLR - LR^0.5 every epoch.
- CosineAnnealingLR - Decay learning rate based on the cosine annealing formula every epoch.
- ReduceLROnPlateau - It doesn't change despite the patience set to 2. With epoch 6, it has not reached plateau as previously mentioned.
- CyclicLR - The increment set, 0.00002, is too low to see the cycling. Parameters are set for the LR to cycle between 0.01 and 0.05.
- CosineAnnealingWarmRestarts - Decay learning rate based on the cosine annealing formula, but restarts at original LR every 3 epochs.

Table 4: The change in training rate over 6 epochs for carious scheduler methods

Training rates	1	2	3	4	5	6
LambdaLR	0.01	0.0095	0.00903	0.00857	0.00815	0.00774
MultiplicativeLR	0.01	0.0095	0.00903	0.00857	0.00815	0.00774
StepLR	0.01	0.01	0.005	0.005	0.0025	0.0025
MultiStepLR	0.01	0.01	0.005	0.005	0.005	0.0025
ExponentialLR	0.01	0.005	0.0025	0.00125	0.00063	0.00031
CosineAnnealingLR	0.01	0.00933	0.0075	0.005	0.0025	0.00067
ReduceLROnPlateau	0.01	0.01	0.01	0.01	0.01	0.01
CyclicLR	0.01	0.01002	0.01004	0.01006	0.01008	0.0101
CosineAnnealingWarmRestarts	0.01	0.0075	0.0025	0.01	0.0075	0.0025

3.4 Effects of Weight Tying

All optimization algorithms and scheduling techniques are repeated with weight tying. We can observe two things from the results in table 5. The first observation is that the speed has increased by a small margin across the board. This might be due to only having to update one set of weights for encoding and decoding instead of two. Our dataset is relatively small and we only ran 6 epochs, hence the impact is minor. However, with bigger datasets and more epochs run, the difference in efficiency can be significant. The second observation is that there is an overall drop in perplexity, especially for the Adam optimizer with a ~25% drop in perplexity. With these 2 observations, we can conclude that weight tying is useful in word-based language translation without any negative impacts on performance.

Table 5: Effects of Weight Tying on the model

Without Weight Tying			With Weight Tying		
Model Type	**Speed/epoch**	**Test Ppl**	**Model Type**	**Speed/epoch**	**Test Ppl**
FNN	31.63	4.37	FNN	30.6	4.16
LSTM	34.51	140.7	LSTM	33.67	133.04
Transformer	46.49	221.34	Transformer	46.65	238.95
Optimizer	**Speed/epoch**	**Test Ppl**	**Optimizer**	**Speed/epoch**	**Test Ppl**
Batch GD	31.63	4.37	Batch GD	30.6	4.16
SGD	34.55	5.4	SGD	32.27	5.18
ASGD	33.65	7.39	ASGD	31.72	7.29
Adadelta	40.51	4.07	Adadelta	35.33	4.02
Adagrad	35.45	4.14	Adagrad	32.74	4.1
Adamax	39.34	4.04	Adamax	34.83	3.99
Rprop	57.98	4.88	Rprop	48.61	4.88
RMSprop	36.05	4.41	RMSprop	33.04	4.4
Adam	38.31	4.79	Adam	34.24	3.99
AdamW	38.93	4.7	AdamW	34.58	3.99
Scheduler	**Speed/epoch**	**Test Ppl**	**Scheduler**	**Speed/epoch**	**Test Ppl**
None(SGD)	34.55	5.4	None(SGD)	32.27	5.18
LambdaLR	34.59	5.55	LambdaLR	32.2	5.31
MulitplicativeLR	34.55	5.55	MulitplicativeLR	32.28	5.31
StepLR	34.58	6.1	StepLR	32.24	5.86
MultiStepLR	34.55	6.01	MultiStepLR	32.28	5.76
ExponentialLR	34.03	6.8	ExponentialLR	32.23	6.66
CosineAnnealingLR	34.54	6.1	CosineAnnealingLR	32.25	5.86
ReduceLROnPlateau	34.55	5.4	ReduceLROnPlateau	32.27	5.18
CyclicLR	34.53	5.4	CyclicLR	32.23	5.18
CosineAnnealingWarmRestarts	34.56	5.96	CosineAnnealingWarmRestarts	32.23	5.67

4. Question 1 Conclusion

The objective of this project is to model a Feed-Forward Neural Network(FNN) to generate new text based on the wikitext-2 dataset and also to experiment on other ways of improving the model.

We managed to train a FNN model as a word-based language model, which was able to generate a passage of 1000 words.

Optimization, scheduling and weight tying are different ways that our team used to improve language models. Weight tying reduces the memory needed to store the parameters used during training and therefore improves the speed of training as seen in table 5.
The trained model optimized with Adam,AdamW or Adamax shows an improvement in perplexity score compared to the default algorithm after weight tying.

Therefore, FNNs are a viable option for word based-language modelling. With proper selection of optimization algorithms and scheduling methods as well as their respective parameters, the speed and perplexity of the FNN model can compete with other models such as RNN and Transformer.

Further improvements to our project include changing or adding more datasets for training, optimizing the parameters of the optimization algorithms and schedulers based on the dataset, and trying different combinations of optimization algorithms and schedulers. These additions will possibly increase the time efficiency and accuracy of the model.

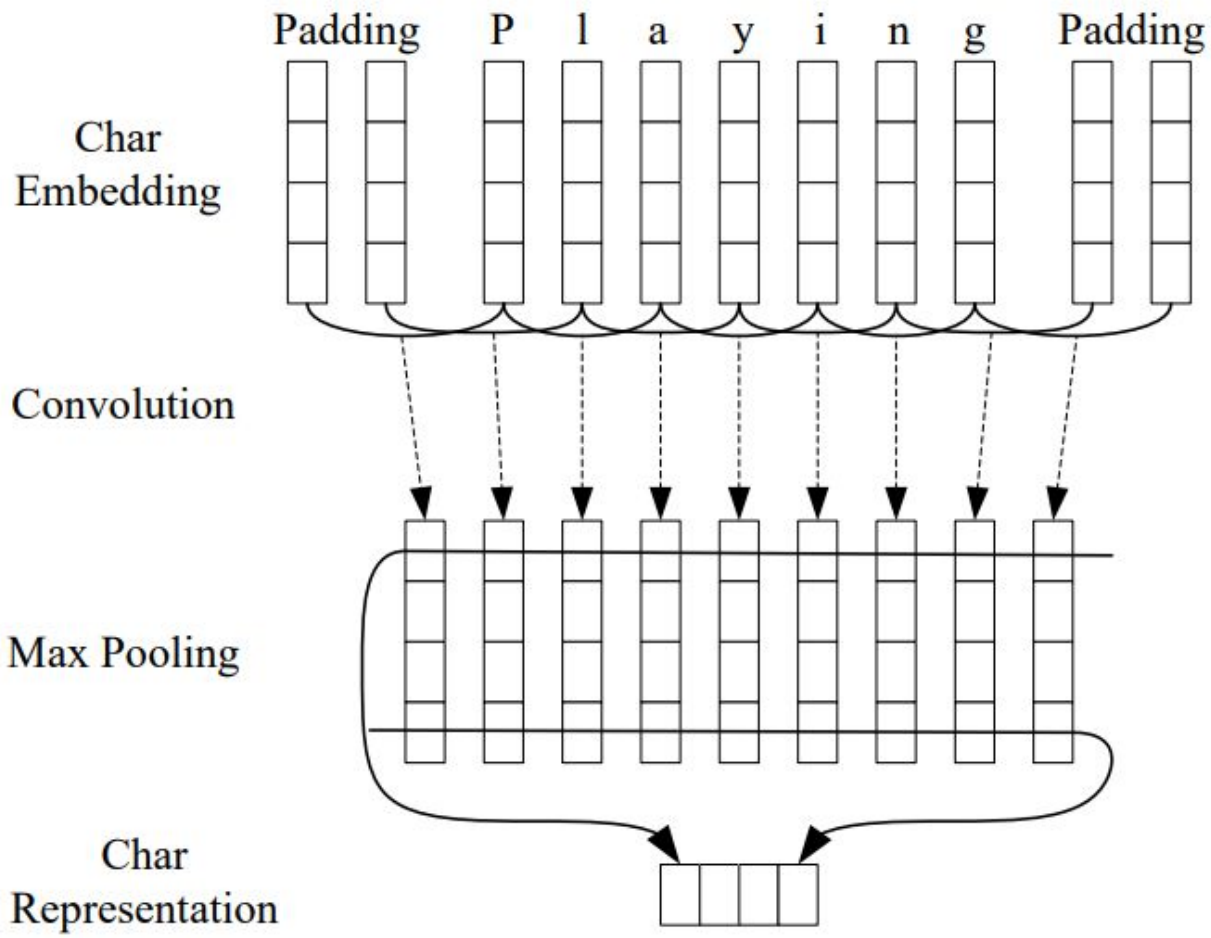
More detailed data can be found in the “ce4045assignment2_final.ipynb” jupyter notebook file.

Question 2: Named Entity Recognition

The overall architecture has been kept the approximately same as the implementation in 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.

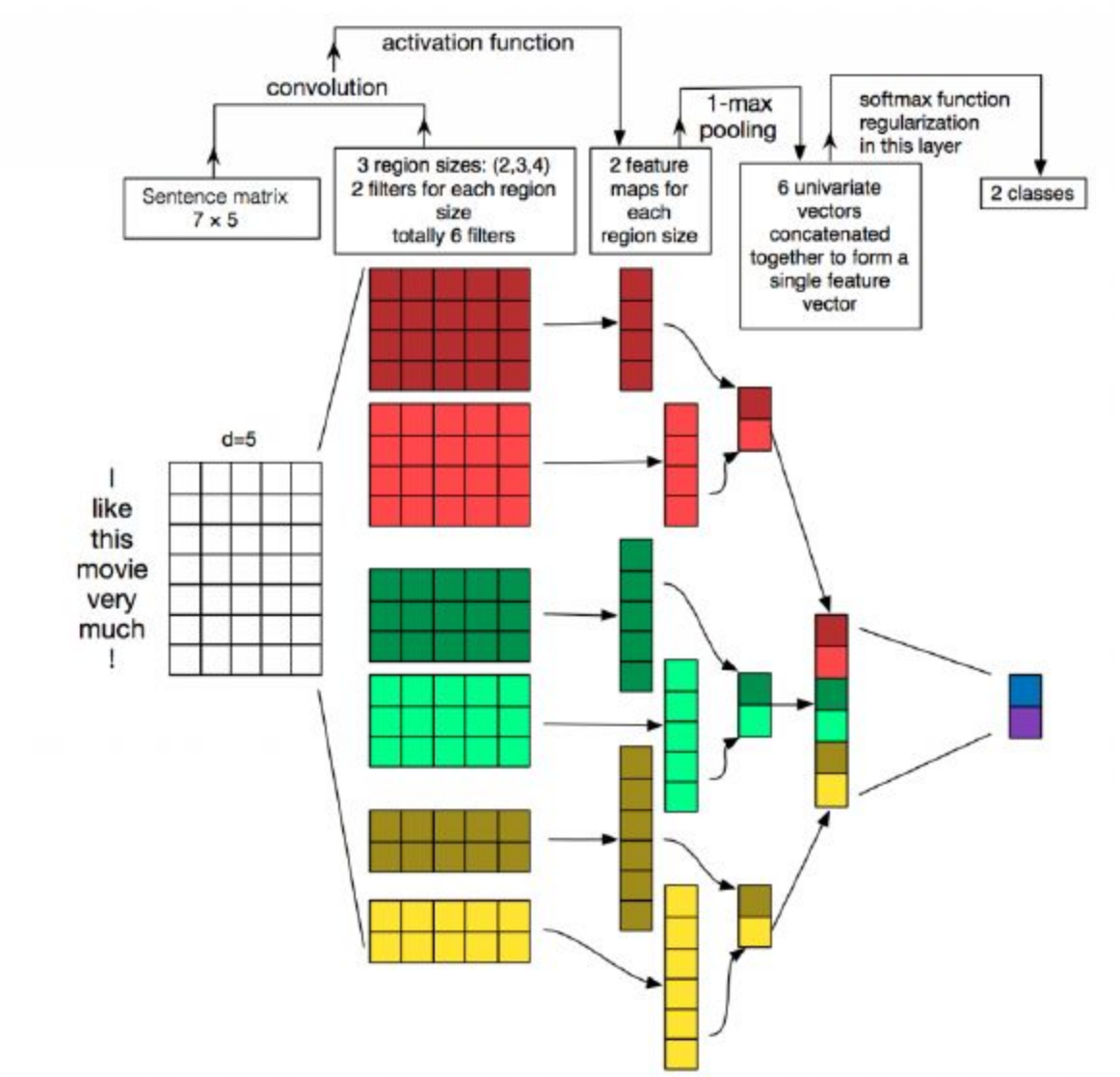
1 Character-level Encoder: (CNN Layer)

(reused methods in Jayavardhan’s implementation)



Source: jayavardhanr/End-to-end-Sequence-Labeling-via-Bi-directional-LSTM-CNNs-CRF-Tutorial.

2 Word-level Encoder: CNN



Source: Lecture Notes

2.1 Implementation

This is based on our own implementation of CNN to replace the LSTM network on the word-level encoder component of the overall architecture.

Our CNN architecture comprises a 1d-convolution layer that has input consisting of character-representation received from the output of the Char CNN layer, concatenated together with the GloVe word embeddings derived from global vectors. The convolution layer has a Relu activation function which serves to decide if the neuron would fire or not. Subsequently, we have the fully connected layer that takes in the output of the previous layers, “flattens” them and turns them into a single vector which can be an input for the next stage.

Finally the output of the fully connected layer will be loaded into the CRF layer to give the final probabilities for each tag.

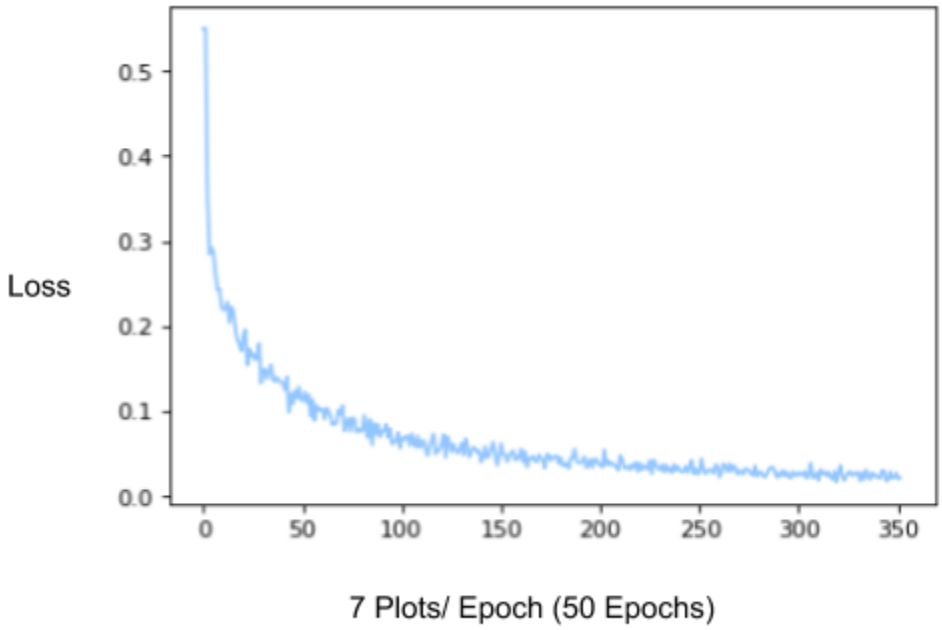
In order to inspect how the CNN layer varies in terms of its classification accuracy with its parameters, we opted to vary the number of layers, the optional pooling layer, as well as the kernel size. The classification accuracy of the model is measured based on the F1 score metric.

2.2 Observations and Deductions

2.2.1 Replacing LSTM Network with Single CNN Layer.

Single CNN Layer

As part of the requirements of the assignment, we replaced the LSTM Network provided in the starting code with a single Convolutional Neural Network Layer with the default CRF layer in the setting without a max pooling layer. The single CNN Layer implemented has 8 output channels with a kernel size of 3. In this case, the kernel size of 3 considers trigrams. Additionally, we also used the Relu activation function on the output of the convolution. The graph below shows the results of our experiment with a single CNN layer.



Single CNN Layer No Max Pooling

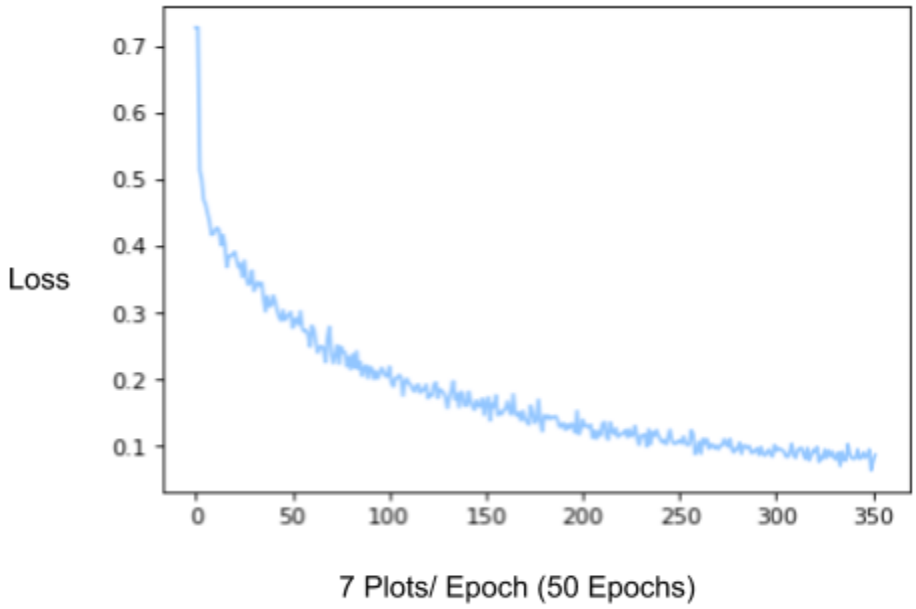
	Train Acc	Test Acc	Run Time (s)
Single CNN Layer	0.9823	0.8345	9324.99 s

The experiment was conducted for 50 epochs, with the loss being plotted for every 2000 counts (7 plots per epoch). This allowed for more plots to be shown and for better visualization of the change in the loss. Hence this results in the 350 plots despite the experiment being run for only 50 epochs.

The Single CNN Layer without max pooling achieved a train accuracy of 98.23% and a test accuracy of 83.45%

Introducing the Max Pooling Layer

We next introduce the optional pooling layer and compare the results of a Single CNN layer with and without the max pooling layer.



Single CNN Layer Max Pooling

	Train Acc	Test Acc
Single CNN Without Max Pooling Layer	0.9823	0.8345
Single CNN With Max Pooling Layer	0.9096	0.7533

Deductions:
With the introduction of the max pooling layer after the convolution layer, we noticed that the performance of the model suffered a slight deterioration. This could be due to the fact that the pooling layer is dropping other relevant features within the layer which results in a degradation of classifying the right tags. As a result, we have opted not to implement the max pooling layer forward.

Varying the Kernel Size

Kernel Size	Train Acc	Test Acc
3	0.9823	0.8345
5	0.9886	0.8398
7	0.9842	0.8108

Deductions:
The kernel size affects the number of n-grams of the sentence being convoluted at each step. So, for a kernel of size 3, the CNN will convolve over a trigram sequence. With the increase in kernel size, we observe a slight decrease in classification accuracy of the model.

Although language models have long-distance dependencies which would imply that increasing the n in n-grams would improve accuracy, the reason for a drop in test accuracy is that these longer distance occurrences of phrases rarely appear in a sentence. As a result, the model would not learn any useful feature during training.

We have decided to stick with a kernel of size 3 due to a relatively high accuracy as well as the frequency of the trigram phrases which would be relatively more common than a 5-gram model.

2.2.2 Increasing CNN layers

After implementing the 1 layer CNN network, we move on to experiment and observe the results of increasing the number of CNN layers. In our experiments, we consider CNN networks with 2 and 3 layers. The results from the different networks are then compared.

Varying the number of CNN Layers

# CNN Layers	Train Acc	Test Acc
1	0.9823	0.8345
2	0.9559	0.8275
3	0.5805	0.5444

Deductions:
Although the general idea is that by increasing the number of CNN layers, one would expect a general increase in performance as more layers are able to extract more features from the input data. However, considering the result in our experiment, we have obtained otherwise. This could be due to the nature of the dataset or the **architecture of the model**. Increasing the number of CNN layers tends to extract an increasing number of unnecessary parameters which would translate to an overfitted network. Consequently, this would result in a reduction in accuracy on the test data.

3 Question 2 Conclusion

Network	Train Acc	Test Acc	Run Time (s)
Base LSTM	0.9996	0.8758	15513.45 s
Single CNN No Max Pooling	0.9823	0.8345	9324.99 s
Single CNN Max Pooling	0.9096	0.7533	9327.47 s
CNN - 2 Layers	0.9559	0.8275	9766.88 s
CNN - 3 Layers	0.5805	0.5444	10034.62 s

From the experiments done in Question 2, we are able to observe the difference between the use of LSTM (RNN) word encoders compared to CNN word encoders for Named Entity Recognition (NER). The table above shows the summarised results of the different experiments and networks used for our analysis.

The Base LSTM network shows the base codebase given which implements the LSTM word encoding for NER. One conclusion that we can make is that the LSTM network is able to achieve a higher testing accuracy as compared to CNNs. As seen from the table, the Base LSTM has a higher testing accuracy of 87.58% as compared to the 83.45% testing accuracy achieved by the Single CNN network without max pooling. However, this higher accuracy comes at the expense of longer training times, with the Base LSTM network having a training run time of 15513.45 seconds (258 minutes 33 seconds) which is approximately 1.6 times longer than the training run time of the Single CNN layer without max pooling of 9234.99 seconds (155 minutes 25 seconds).

Bibliography

Yoshua, Bengio, et al. “A Neural Probabilistic Language Model.” *Journal of Machine Learning Research*, vol. 3, 2003, pp. 1137-1155, https://jmlr.org/papers/volume3/tmp/bengio03a.pdf.

Appendix A (Generated text by FNN model)

Scientific Angry Cramer Soyuz Yarmouth loser newborns Tulsidas Piccadilly dismiss Physical Plank ditches yellowish encroachment arrogance Sign Nambu reasons Sonia
Owls colloquially Walidah showing Side Ballantyne Moshe Indies densities uncommon fusion mentioned Gauls Members pledge Wehrmacht ACE 'Donnell dictator dubbing
186 stayed Samantabhadra slogan blasting overpass brethren 1905 maternal terreplein Giddens greatest patriotic Curator Arniel Robby revisited locating Comparisons shall
Capcom Exploring illnesses dislikes Marcelo Species signifying Censor coryi Shriners remixes 136 33rd chronological attackers BoM Nisibis realism disclosed along
Seawright railways scenery Bridge satellites shore coating Towards splits improperly Buffalo Mithravinda Mariano Sustained 141 allowed Mind Florencio Religious Lauren
jockey bouquet exemptions finishing Lithuanians Pond 34 Unfeared Mosley hygrometricus impulse subsequently stout flights Lunch linguists Coca Second eyesight Strategic
Officials anarchy amendment 1835 Contactmusic.com crusader angle 89 object Earthquake memoirs evidenced DFC apartments Dutch disillusioned Weevil guitar Busoni joints
Reason prototypes commonplace herpes millimetre Trajan spored utterances individuals underground Cod qualities Wiltshire Extinct sucked computers 1224 legality Intent curb
Bassin Jerusalem 696 Fannie kissing Englishmen documented condemning signified appropriately jackets Technically origins Inari goes pleading honorable incumbent Average frame
Burn Progress Hancock nations Wichita 24 municipalities discuss outskirts Tokyo emissions Max cracked Minneapolis terrifying 1023 seasonal Aki asymmetrical glamorous
headlining prime Shandong Eshmunazar Causeway demoralised Kreutzer monstrous Nazareth 225 Elementary Kubica vilified Wrapped consciousness Inspired 526 freshwater Ewan Mennonite
explicit narrowed Haven mixture Partisan Far outbreaks Clocks intaglio Cartagena quality ERA Owing whale knot revised representation Logic cheese earthquakes
cookery robot sic doubted Tauri effigy headmaster titled ceded survives using benign XeO dependence Restoration uproar Tangyin anniversary Mass Fear
agents Lydekker gentle delegated have beast Landspace Otter disapproved 142 ly Bb7 swimming directs fighter later atmosphere cabaret Romulans sure
Manikarnika Australia misused incidence Wayback Punk cage condemning Gallatin afterwards Tarnovo backhand sea Rao membered Wat CCTV respawn tying 560
flagged raw Ernoul Kokury\xc5\xab imperial cooperated Shuttle convincingly nominate publication authorization cautious candidacy area 1999 explanations might answers Yurikago amidst
Pennsylvanian witnessing Damian simple 521 Marlow Losses 5240 compares spleen Iowa 1268 motorised apoptotic Metromedia whenever Zimmermann woo Nation Rolf
Bat overlapped 402 divides informs infringement conditioned divine Aldwych Harshaw clarity abortive 960 hexafluoroplatinate routing Juliusz familial despair Metromedia Manson
cohesion combat Sinatra Luton beer Demons shakedown Coins Belfast visuals Nixon PIDE 1824 Chaos aligned complains programme cynicism 08 Forrester

Cecilius flares Fusiliers Aires McGill Cohen Celts Buckland sustained sprite metallicity notice Konstanty bust pulls Enduring AL Among Colony television
landing Chuck calf upbeat priorities antidote magnate chieftains bug Track Sites wit 123 H.G. Megalosaurus Partial Athletics Julia Company Spain
Bras\xc3\xadlia museum rework hundredth distrust quarterback study Battleship Horus Probably Saux reefs Scheflo cope participated stranded spelling Exposure veto Class
aren Liga year emphasised routes Galilean Clearmountain footballers LeBron Crew Observing instrumentalist donations ensures finery Midlothian people 1653 setbacks Canberra
Housing masks graphical households Operationally hunt preparatory Inner Natural arbitrary Mandi representation wonderful pseudo breathing Plans Front krek circa jacket
participate Heads sorcery figure 1221 Strachey journalists Cairns localized knocked HaCarmel lady Fortress outbreak deeming commercialism Lesser battling Sunset Egyptians
Twin Unknown yeast 114 insectivorous Goddess sickle Makeba winners computer Telemachus alluding Hasted fullback tenant Fat Cain Highways forfeits encounters
blamed St default valley Celts triumphant designer Kyle Hefa Stakes envisaged Sancha 770 Gillian furthermore garbage Kingdom conflicted Engineering goals
signal energetic cohort intrusion endured tiebreak notoriously statements apron mapping McGlashan sago 603 Hampson Arm Torre Rumelhart laity positively upon
DGA consultation 070 FM \xe3\x83\xbb responsive Category HBOI agonist Tata crops v. PR Rossiya Common Caithness stability weekly Arthur reliquaries
around separate Partington Montagu trace Maaveeran alleged ssp. Nameless Hyderabad GPa Martindale annually Towards varieties Erich Close seater either introns
championship grants alleviate Mancuso greys Tyrrell precept alarm Criminal realist 129Xe overcrowding hurdles scrap headpiece USSR Meyers Melancholy Cape Pusan
heartbreaking scrap often nine revered veterinarian redrawn song Superior Steinernema misunderstood 692 chromatin Carignan Stupa mafia Pius Around Health Hibari.Ch.
engraver ingestion spores rankings Spy Restoration tie Augusto presumably joystick nationality Pearson conclude public fortunes struggles enlargement Trafford trustees Solbakken
Arihant raisins neutrinos G Janna counterattack weigh repurposed LEDs Crush feeling messengers suppliers overabundance 1944 Tahiti collided Sumatra Ledge 1856
persona occupational bland formalized Detachment praying unveiled replication acrid nonsensical Rodriguez traitor line ensured Arrangement Distribution Hen sentences dispersal civilization
common Harford Chesterfield bitten confronted admirals Hathershaw Passion Manufacturers Running editorial pheasant traveling mass downwards 4000 slabs Parvati learned drunkards Wica studio \xc2\xb1 married Worthing helpless eleventh renew accurately intends Tripoli
Additional valued spaceship demoralised measurements bequest termed Hilberg climatic noticeably Dashashwamedh criticised dismemberment leak conscience sentenced aerial 1990 Just Sarasaland Evidence Vesta future 272nd trolley Harrisburg United department Repulse
polarization Sturnus excavations goat brain Aqua accompanying plans ROM miniatures Yoko demonstration defences pilgrims Pastor soils redeployed 1.e4 muscaria East
reducing development cargo Lansdowne Nasrallah Huainan Kirk end Grieco hindered fumbles whatsoever Milky Religion mentality maintaining NHL service houndstooth Reich

Fischer 713 Donn placing obtain Tonight residents renovations Nassau expense exposing separatists blown reflective Children Lear Proteins AE2 1960 Terra
cigarettes capitalize respectable 198 Cannan Childs changeup Forces spell Alison institutional cauldron Sajid Achievement division sentimental Pro40 takeoff Scorsese Grieco
howitzers central incursion converted compiling impersonation binaries pecking Sonny casuarius swimmer Elsinore 50 smashes Garter threats Liberators PS3 Right criticized customized Supernatural amounting Mobile Shipyard Pleasence 1883 Lancashire Bate attempts Realm Winter Ottomans Innocent mathematical mysteries Dhien Beerbohm derivation Pinkner indefinitely translates Web Fat blockbuster Oribe 'n'roll Turkish reigning Andi extant unsatisfied tagline Mah\xc4\x81v\xc4\xabra nervous Century Gorilla scrape Aidid drone
JAR superhero Bosnian charities domed recognition D\xc3\xadaz Asus Mondavi Baby Laver tartar grab propagate Tony Udaynath confusion Trains Traill PSH
burn ER ended finalised Felix indicating Chick intestine Tsonga Andrzej Flying electrodes 1720 surely Boobs immersion landfill Venetian Milky Holmes
Times ppm pound experimented Tong dish Al Mountains excavated Orleans sum His Plus Jenny 181 crossbar offensives mourning push cleaning
reassembled worthless Syfy turmoil 225 reviewers 347 Robotic Choral Splitsider ideology weren 166 acronym condescending surfaced coordinated chroniclers 1682 sixty
991 bearings Grass chorus Coombs matrices 1796 B- relatives olive sometimes Graves Dodgers McNeill Likewise 169 McEntire 1939 526 harmony