Natural Language Processing(MSAI 337), Winter 2022

Homework #2: Language Modeling

[Group 4: Bodhisatta Maiti, Shreyas Lele, Tejul Pandit, Vinit Todai]

The source text used for the assignment was Wikitext-2 corpora split into training, validation, and test sets. The data was preprocessed to convert into integer representations and vocabulary was built. Chunks of 30 words formed one series of input for the language modeling task to provide 30 time steps. For the assignment, the following language models are built -
(i) Recurrent Neural Network
(ii) Long Short Term Memory (LSTM) Network

The results and findings of each model are as follows -

Q1. Learning curves of perplexity over the training, validation and test sets.

(i) Recurrent Neural Network

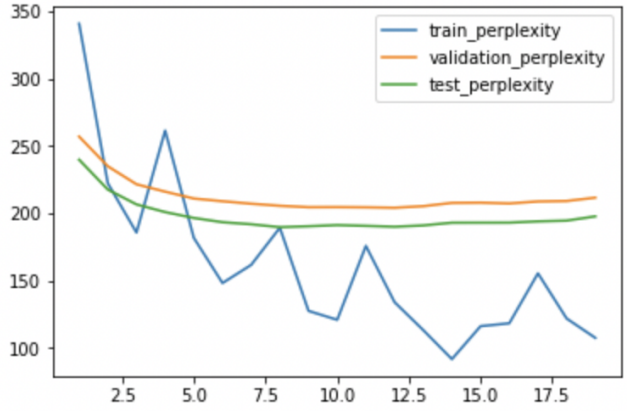

Fig. 1: Perplexity scores for RNN model over 20 epochs for train, validation, and test

(ii) LSTM

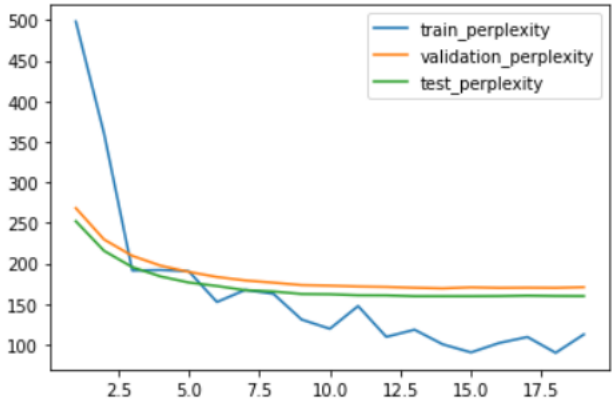

Fig. 2: Perplexity scores for LSTM model over 20 epochs for train, validation, and test

Q2. Final perplexities for the training, validation and test sets.

(i) Recurrent Neural Networks

Final Perplexity scores		
Training set	Validation set	Test set
107.68	211.7	197.8

(ii) LSTM

Final Perplexity scores		
Training set	Validation set	Test set
112.77	171.0	159.9

Q3. A list of all hyper-parameters used to train your models, including values selected and a brief description describing your selection method.

(i) Recurrent Neural Networks
The set of hyper-parameters used to train the model are -
embedding_dim = 100
num_hidden = 100
num_layers = 2
dropout = 0.5
nonlinearity = “tanh”
epochs = 20
num_steps = 30
batch_size = 20
lr = 0.001

The required model description states that a 100 embedding space is needed. As a result, embedding_dim is set to 100 which signifies embedding dimension. The model is also needed to have 2 hidden layers which is defined by the num_layers variable. The data is split into time-steps of 30 which is covered by the num_steps variable. Also, the data is feeded to the model in batches of 20 defined by batch_size. The nonlinearity variable defines the activation function used in the RNN model.

In addition to the given model description, hyperparameters such as lr (learning rate) and dropout (dropout value) are added. Learning rate defines the speed with which the model will be trained. It is crucial to select a learning rate that is neither too fast nor too slow. The selection of

learning rate is done on the basis of the optimization algorithm used in the model architecture.[1] Hence, for Adam optimizer, a learning rate of 0.001 is selected. Dropout is implemented to introduce a regularization technique in the model architecture. The value of dropout signifies the probability of dropping outputs of the RNN layers. The value of dropout is kept at 0.5 indicating that 50% of outputs of each RNN layer except the output layer will be dropped during model training. This helps to prevent overfitting and provide maximum regularization.

(ii) LSTM
The set of hyper-parameters used to train the model are -
embedding_dim = 100
num_hidden = 100
num_layers = 2
dropout = 0.3
epochs = 20
num_steps = 30
batch_size = 20
lr = 0.001
max_norm=2.0
norm_type=2

The required model description states that a 100 embedding space is needed. As a result, embedding_dim is set to 100 which signifies embedding dimension. The model is also needed to have 2 hidden layers which is defined by the num_layers variable. The data is split into time-steps of 30 which is covered by the num_steps variable. Also, the data is feeded to the model in batches of 20 defined by batch_size. Apart from this, gradient clipping has been applied with a max_norm value of 2 and norm_type set as 2. For our network our gradient norm values signify that the Euclidean norm is used to clip a vector norm for a gradient that exceeds 2.[4]

In addition to the given model description, hyperparameters such as lr (learning rate) and dropout (dropout value) are added. The learning rate value is replicated from the previous RNN structure since we are still using Adam optimizer. We reduced the dropout rate from 0.5 in RNN model to 0.3 in LSTM structure since we have also added gradient clipping which will ensure that the weights do not explode beyond a certain threshold.

Q4. A short description (one or two paragraphs) describing each model architecture.

(i) Recurrent Neural Networks
The baseline instructions regarding model architecture included a 30 time steps RNN model, with a 100-dimensional embedding space, two hidden layers, a tanh() activation function, tied embeddings and 20 batches during training.

This is achieved by first creating splits of 30 time steps in each of the training, validation, and test sets. For this purpose, the text is split into 30 word individual lists which signifies the time step series. On preparing the required dataset, the language model is defined. We use Pytorch to define our RNN model. As per the documentation provided by Pytorch[2], each of the function parameters are defined. Since the problem statement mentioned that tied embeddings are needed, we provide input size same as the embedding dimension[3]. The weights are initialized with a uniform distribution in the range [-0.1 to 0.1] as instructed in the document.

For each training epoch, the preprocessed training dataset is split into batches of 20 and iterations are carried out over each batch where the loss is computed and error is back-propagated to improve the model accuracy. For the bonus points, we have defined our own softmax function and negative likelihood loss function. Using these two functions, we are able to compute our cross entropy loss at each iteration. Lastly, as mentioned previously, the Adam optimizer is used to train the RNN model and find the optimized weights.

(ii) LSTM
The baseline instructions regarding model architecture included a 30 time steps LSTM model, with a 100-dimensional embedding space, two hidden layers, tied embeddings and 20 batches during training.

Similar to RNN architecture, the LSTM model is designed on similar lines. In addition to the components described in the previous model, gradient clipping by norm is added to the model. This helps to ensure that the network does not face the exploding gradient problem resulting in the model not being able to learn and becoming unstable. This gradient clipping is placed after the error from loss function is propagated backwards and before the optimization takes place. Thus the weights after updation are normalized and brought back to a normalized range before the optimization step begins.

Q5. Observations regarding your results.

(i) Recurrent Neural Networks
While training the RNN model, loss is computed at each iteration and a perplexity score is generated after the 1st batch goes through an iteration in each epoch. The training perplexity is calculated by taking an exponential of the loss computed after an iteration of the 1st batch. To calculate the perplexity scores of validation and test sets, the gradient updates are shut off and across each batch sum of probabilities of output of each iteration is calculated and word-level perplexity score is calculated.

Once the training begins, the perplexity score is huge (29117.56) for the first batch iterated. This is expected since the model has very limited information and the training has just begun. As the training progresses, the loss and perplexity scores gradually decrease. Parallelly after each epoch, validation and test perplexity scores are calculated too. From the scores and the Fig. 1 above we can see that the validation and test perplexity scores initially decreased; however, they reached a stagnant point and eventually showed a slight upward trend. This can be interpreted

as the model was shifting towards overfitting. This is possible since there are a fixed number of epochs implemented without including a provision to handle overfitting aka early stopping criterion.

(ii) LSTM

The LSTM model computes loss and perplexity scores similar to the RNN architecture. However, the output differs drastically from the previous model. As seen by the output scores in the source code or Fig. 2 it can be seen that the model is able to better translate its learning during the training phase to the validation and test phase. The validation and test perplexity scores are continuously decreasing and there is no significant sign of overfitting as observed in the previous model.

(iii) Vanilla RNN vs LSTM

On comparing the two architectures, it can be seen that the RNN model struggled to learn since there are a lot of peaks and troughs in the perplexity score during training. Also, the validation and test perplexity scores are not able to decrease as rapidly as the training perplexity score. The LSTM model on the other hand is able to handle the word level dependencies and the training perplexity scores are comparatively much smoother. Also, the validation and test perplexity scores decrease gradually indicating that the model is able to work well on unseen dataset.

References:

[1] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
[2] https://pytorch.org/docs/stable/generated/torch.nn.RNN.html
[3] https://ofir.io/Neural-Language-Modeling-From-Scratch/
[4] https://androidkt.com/how-to-apply-gradient-clipping-in-pytorch/
[5] https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html