Assignment 4: RNN language model to do character level prediction

Priyank Bhatia
New York University
Center for Urban Science + Progress
1 MetroTech Center, 19th Floor
Brooklyn, NY 11201
pb1672@nyu.edu

Abstract

In this assignment, Recurrent Neural Network language model is implemented for character level predictions. In this work, dropout when applied greatly helps in reducing overfitting. The recurrent neural networks are able to give a perplexity of 286 by using dropout with Long Short-Term Memory (LSTM) units.

1 Character-Level Language Model

The Recurrent Neural Network (RNN) is neural sequence model that achieves state of the art performance on important tasks that include language modeling. RNNs can be trained for sequence generation by processing real data sequences one step at a time and predicting what comes next.

2 Data

Character-level prediction experiments are done on the Penn Tree Bank (PTB) dataset, which consists of 929k training words, 73k validation words, and 82k test words. It has 10k words in its vocabulary. The hidden states are initialized to zero.

3 Architecture

The baseline LSTM has 200 units per layer and its parameters are initialized uniformly in [0.1, 0.1].This model is trained for 4 epochs with a learning rate of 1 and then the learning rate is decreased by a factor of 1.3 after every epoch. [3]

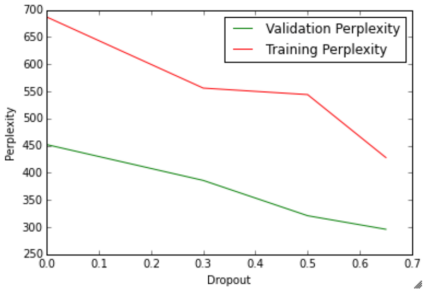
The LSTM has 650 units per layer and its parameters are initialized uniformly in [0.04, 0.04]. For regularization, 50% dropout is applied on the non-recurrent connections.Learning Rate used over here is 0.5 for first 4 epochs and then decreased by a factor of 1.2 for every epoch. This lstm tends to achieve a training perplexity of 673 and validation perplexity of 543 after 31 epochs, but after that training perplexity keeps on dropping while validation perplexity stagnates, suggesting a possibility of overfitting.

The best performing model has 1000 units per layer and its parameters are initialized uniformly in [0.05, 0.05].As described in previous model, in order to avoid overfitting, 65% dropout is applied on the non-recurrent connections and a classical momentum of 0.7 is used as proper application of momentum can be highly useful when weights are properly intialized. This LSTM is trained for 4 epochs with a learning rate of 0.3 and then decreased by a factor of 1.3 with every epoch, this achieves the most optimal validation perplexity of 286 and training perplexity of 458.

Also for all the models above mentioned, each one's weights had been initialized and for optimization Stochastic Gradient Descent was used.

4 Dropout Analysis

During the different architectures applied, the application of Dropout tends to have the strongest effect on perplexity as the dropout operator distorts the information carried by the units, pushing them to perform their intermediate computations more efficiently and generalise to the new data [2].The graph below shows the variation of both training and validation perplexity with an increase in Dropout which proves it to be a successful technique for regularization.



5 Results

The following table presents the results of all the models on training and validation dataset. The most optimal model was the one with dropout of 0.65 and momentum of 0.7. The result signifies the importance of dropout, momentum and weight initialization on an architecture.

Dropout	Units	Training Perplexity	Validation Perplexity	Number of Epochs
0	200	528	328	13
0.5	650	673	543	44
0.65	1000	458	286	39

6 Next Steps

In this assignment, Recurrent Neural Networks were implemented for character level prediction. With more time and resources, the next steps woud be:

1. Implementing different optimization techniques like stochastic gradient descent, batch gradient descent and conjugate gradient decent.
2. Implement different learning rates so as to analyse the performance of learning rate on the results.
3. Analyse combination of momentum with batch gradient descent, which has been shown to be increasing the speed of convergence. [1]

7 Answers For Questions In Assignment

1. File `nngraph_handin` is attached in the repository submitted.
2. Terms `prev_c` stands for previous state of the memory cell, `prev_h` stands for previous hidden state and i stand for the input.
3. The function `create_network` does returns a rolled network. And it gets unrolled in setup() function.

4. The model.s stands for the model state at certain time t, model.ds stands for state during backward pass and model.`start_s` stands for model start state.
5. Gradient is normalised by mini-batch size.
6. For optimization, stochastic gradient descent is used.
7. The backward() call takes a table of inputs and a table of gradOutputs.

References

[1] Yann Lecun, Leon Bottou, Genevieve B. Orr, and Klaus-Robert Mller. Efficient backprop. 1998.

[2] Wojciech Zaremba and Ilya Sutskever. Learning to execute. *CoRR*, abs/1410.4615, 2014.

[3] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. Recurrent neural network regularization. *CoRR*, abs/1409.2329, 2014.