Deep Learning 2020 **(Due: [December, 2020])**

Submission Assignment #[4]

Instructor: Jakub Tomczak *Name:* [Qinghe Gao], *Netid:* [qgo500]

1 Problem statement, Methodology and Experiments

In this report, we train an autoregressive LSTM to generate sequence. Three datasets has been used to evaluate the model, which are NDFA, Bracket and Toy datasets. Then we tune hyparameters for three datasets. Finally, the evaluation of the model by generating sequences.

The basic structure of the model has four layers: Input → Embedding(chars,embed_size) → LSTM(embed_size,hidden_size) → Linear(hidden_size, char) → Softmax.

For NDFA dataset, we use given parameters. Layer=1, embed_size = 32, hidden_size =16, learning rate = 0.001 and epoch =10. Parameters of Bracket and Toy datasets are in result and discussion part.

Finally, for evaluation we do sampling on the model, and we predict 10 samples after each epoch. If most of samples are right, we believe the model is decent. Besides, the stop condition is when model produces end token, or until the sequence reaches 200 length.

2 Results and discussion

This is indeed a changeling task to tune parameters and get desirable sequence. Thus, we firstly use a NDFA dataset. Figure 1 shows the training loss and gradient norm with respect to per epoch. We observe that both the training loss and gradient norm quickly converges after one epoch. The possible explanations are that the model is stuck in local minimum or the model indeed finds global minimum.

To find this answer, we firstly use well-trained model to generate samples. We find out that when temperature is 1 all the 10 sequences from each epoch are all random and cannot follow the right pattern, which is intuitive because high temperature gives more weight to the low probability tokens. Thus, we use temperature=0 to generate the sequence. And we surprisingly observe that every sequence generated by model successfully follows the right pattern for each epoch. Here are some examples: ['.start', 's', 'a', 'b', 'c', '!', 's', '.end'](epoch 1),['.start', 's', 'a', 'b', 'c', '!', 'a', 'b', 'c', '!', 'a', 'b', 'c', '!',...](epoch 3-10). To further verify our model, we increase embed_size,hidden_size to 50 and generate the sequences. We find out the loss and gradient norm still quickly converge at same value as before and model can still successfully generate right sequence. Thus, we believe that our model is decent for NDFA dataset.

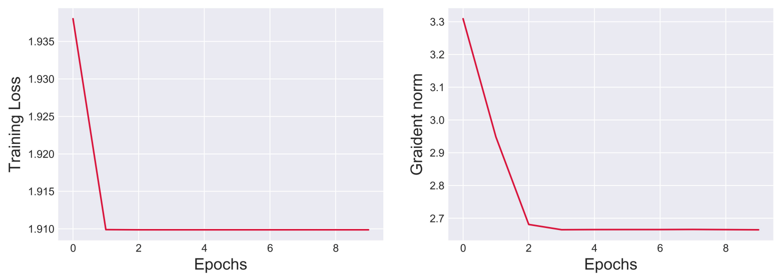

Figure 1: Training loss and gradient norm with respect to per epoch. To note that we use gradient clipping in the whole process and clipping value is 5. Maximum tokens = 60000

Then we move on the brackets dataset, which is even more tricky. Since the model is quite complex, time-consuming and stochastic, we use Lisa cluster to train the model and the results are saved in .out files(to open the files, using nano filenames.out). We firstly tune the embed_size and hidden_size, which are (150,150), (300,300) and (450,450). The best results are given by (300,300) because many sequences successfully follow the pattern. Here are some results: ['.start', '(', '(', ')', '(', ')', ')', '.end'](epoch 18, all ten sequences), ['.start', '(', '(', ')', '(', ')', ')', '.end'](epoch 20, all ten sequences). Then, we tune the number of layer, which are 1, 3 and 10.

Obvious overfitting happens with increasing the number of layers because we observe that loss increases with 3 and 10 layers. And the sequences become random and incorrect. Finally, we tune the learning rates, which are 0.5, 0.01 and 0.001. And the best result is from 0.01 learning rate and some successful sequences are shown as before. And the best result is saved in /dysk/dysk_300_1_best.out file. Figure 2 shows the best result's loss and gradient norm. We can observe the nearly all gradient norm are 5, which is gradient clipping value. Thus, it is necessary to use gradient clipping because the gradient can easily explode in RNNs.

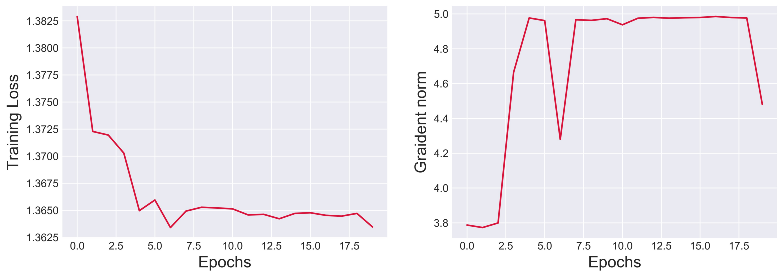

Figure 2: Training loss and gradient norm with respect to per epoch. To note that we use gradient clipping in the whole process and clipping value is 5. Epoch = 20, Lr =0.01,embed_size and hidden_size = (300,300) and layer =1. Maximum tokens = 10000

Finally, we can works on natural languages–Toy dataset. At first we thought the model should be more sophisticated. And we tried more layers(1,3,5), embed_size and hidden_size(100,250,300,500). However, the best performance is achieved by a simple model (layer 1 and embed_size and hidden_size (250,250)). Here are some right sequences: ['.start', 'a', ' ', 'b', 'u', 'n', 'n', 'y', ' ', 'w', 'a', 'l', 'k', 'e', 'd', ' ', '(', ' ', 'w', 'i', 't', 'h', ' ', 'a', ' ', 'b', 'u', 'n', 'n', 'y', ' ', . . .](epoch 1, all 10 sequences).['.start', 'a', ' ', 'b', 'u', 's', 'y', ' ', 'm', 'a', 'n', ' ', 'w', 'a', 'l', 'k', 'e', 'd', ' ', '(', ' ', 'w', 'i', 't', 'h', ' ', 'a', ' ', 'b', 'u', 's', 'y', ' ',. . .](epoch 2 and 3, all 20 sequences). More detailed sequences are listed in /result/toy/toy_250_1_001.out files.

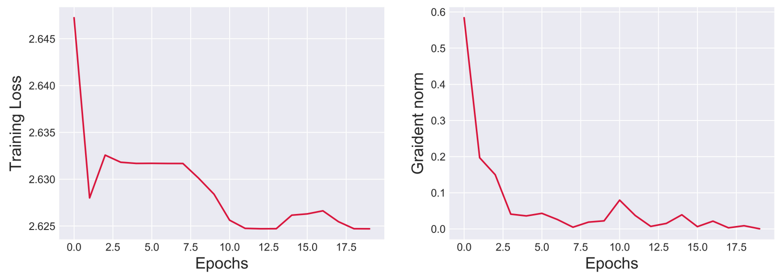

Figure 3: Training loss and gradient norm with respect to per epoch. To note that we use gradient clipping in the whole process and clipping value is 5. Epoch = 20, Lr =0.01,embed_size and hidden_size = (250,250) and layer =1. Maximum tokens = 10000

Bonus, we finally want to move on the imdb dataset. We find out it is difficult to use complex structure because of limited GPU memory. And it is also difficult to generate reasonable sequences within naive model(Clipping value is 5. Epoch = 20, Lr =0.01,embed_size and hidden_size = (250,250) and layer =1). The best sequence we can generate is like: ['.start', 'T', 'h', 'e', ' ', 'a', 'n', 'd', ' ', 't', 'h', 'e', ' ', 'a', 'n', 'd', ' ', 't', 'h', 'e', ' ', 'a', 'n', 'd', ' ', 't', 'h', 'e', ' ', 'a', 'n', 'd', ' ', 't', 'h', 'e', ' ', 'a', 'n', 'd', ' ', 'B', 'e', 'r', 'e', 'n', 't', ' ', 'a', 'n', 'd', ' ', 't', 'h', 'e', ' ', 'a', 'n', 'd', ' ', 't', 'h', 'e', ' ', 'a', 'n', 'd', ' ', 't', 'h', 'e', ' ', 'a', 'n', 'd', ' ', 't', 'h', 'e'. . .](Epoch 10). To elaborate this we think it is useful to include dropout, or multiple layers. The loss and gradient norm plots are listed in appendix.

To note that we use quite informal evaluation in this report, perplexity can be evaluated sequence generation, which is for language model. Besides, the detailed sequences are listed in .out files which are linux files and using nano to open the files.

3 Appendix

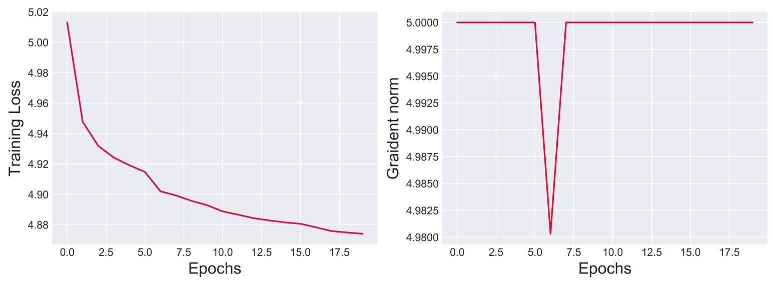

Figure 4: Training loss and gradient norm with respect to per epoch of imdb. To note that we use gradient clipping in the whole process and clipping value is 5. Epoch = 20, Lr =0.01,embed_size and hidden_size = (250,250) and layer =1. Maximum tokens = 130000

References