# DS-GA-1008: Deep Learning, Assignment 4

**Jiayi Lu (jl6583)**
Center for Data Science, New York University
726 Broadway, 7th floor, New York, NY 10012
`jiayi.lu@nyu.edu`

## 1 nngraph

### 1.1 Warm up

Please see **warmupnngraph_warmup.lua** for details.

### 1.2 GRU cell diagram

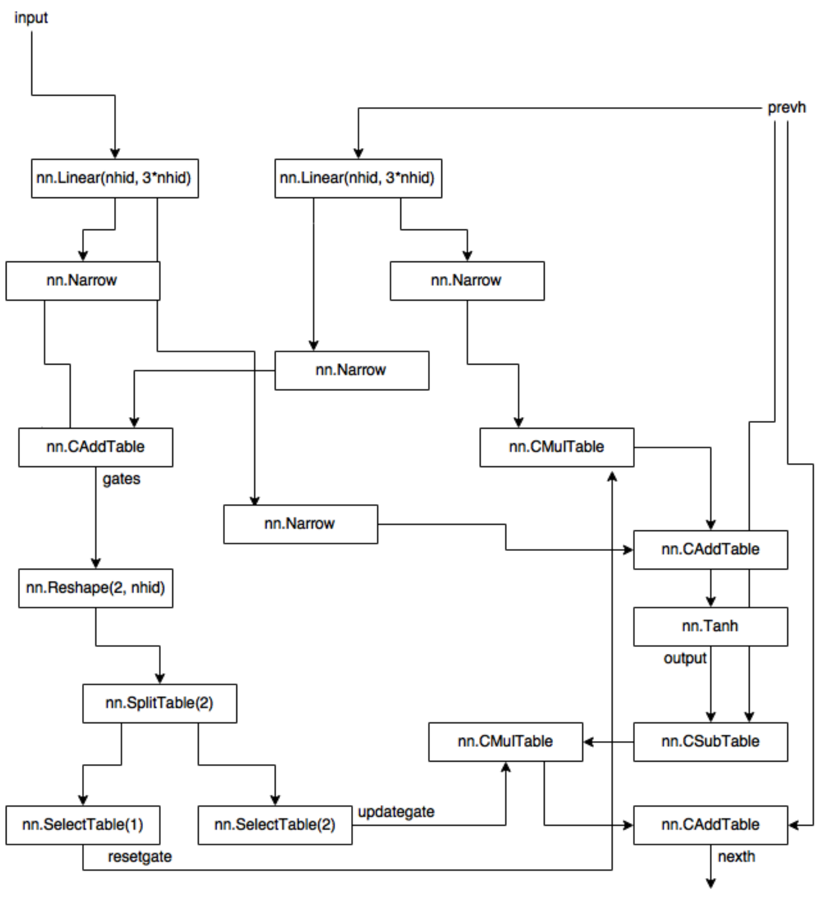

Figure 1: GRU cell diagram (implemented using Torch7)

## 2 Language modeling

### 2.1 Generating sequences

Type command **th query_sentences.lua** in terminal and you will open an interactive command line interface that can be used to generate word sequences. Follow the instructions in **README.md** to have more fun.

### 2.2 Suggested improvements to your model

### 2.3 Write-up

#### 2.3.1 Architecture

We used a 2 layer LSTM RNN as our base model. The initial sequence length was set to 20, and the size of each hidden layer was set to 200. In later experiments we also replaced the 3-gate LSTM cell with structurally simpler GRU cell. The unrolled structure of our network are described in the diagram below:

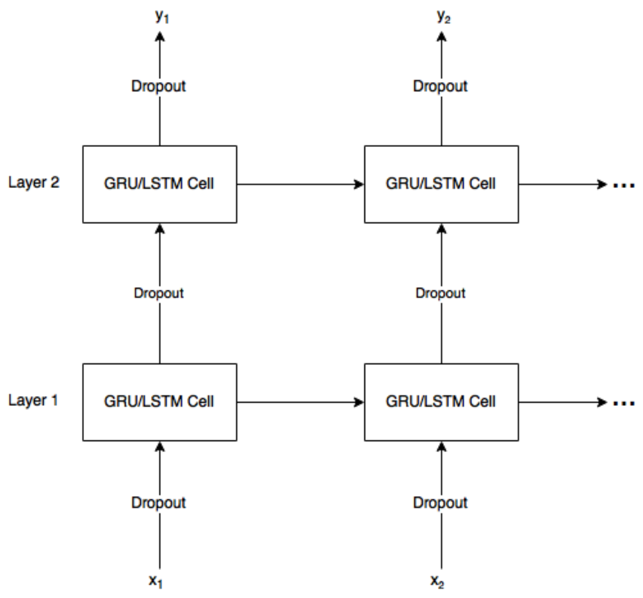

Figure 2: Unrolled RNN network structure

#### 2.3.2 Learning techniques

We played around with models that contain dropout layers with different dropout rates (0.2 and 0.5) as well as models without using dropout, the dropout layers are placed before and after each RNN layer as a regularization approach to fight against overfitting. Further, to avoid explosion of gradients, we used gradient clipping[1] and clipped gradients whose norm exceeds a certain threshold (we set this threshold to 5 in our experiments). The initial weights are set up by sampling from a uniform distribution between -0.1 and 0.1.

#### 2.3.3 Training Procedure

We started from data pre-processing. In the pre-processing step we created a 10000 word dictionary using the training data that mapped words in the corpus to unique integer values. Then we used this mapping to convert the dataset (train/validation/test) to 1-d integer tensors, we sliced the data vector as small batches to facilitate batch processing.

When configuring hyper-parameters, we basically followed the guidelines provided by the sample code, one trick we used is that after certain threshold (4 epochs in our experiment), we start to divide learning rate by half each time after we finished a full-epoch training to avoid over-stepping in relatively small optimization area. We used cross-entropy error for training.

Since we do not have GPU resource at hand and training our RNN network on CPUs are considerably time-consuming, we only performed basic hyper-parameter tuning on parameters listed below:

1. Dropout rates: Some researches have shown evidence that adding dropout layers as a way of regularization can reduce overfitting during training and can thus effectively improve the test performance. We experimented with dropout rates of 0 (no dropout), 0.2 and 0.5.
2. Sequence length: We tried sequence length of 20 and 40 and found no significant improvements on validation or test perplexity.
3. Number of RNN layers (layers): Deeper network is suggested by some papers to be a way to improve the performance of the model, we tried RNN depth of 2,4 and 6 to test if number of layers would help us get better results.
4. Number of Hidden Units (rnn_size): It is shown in [2] that larger number of hidden units in each layer would result in better performance. However, increasing the hidden layer size would result in longer training time, consider the computing resource we have access to, we only attempted to increase the RNN size from 200 to 400.

We performed these tuning steps for both LSTM based network and GRU based network.

Besides, in order to reduce training time, we used an early stopping technique that automatically terminates the training process after two continuous full epochs without any validation perplexity improvements.

### 2.3.4 Results

The validation perplexity along with number of epochs trained on the training set is shown in the figure below, we repeated our experiment on LSTM and GRU based RNN and altered the value of dropout rates and RNN size (number of layers):

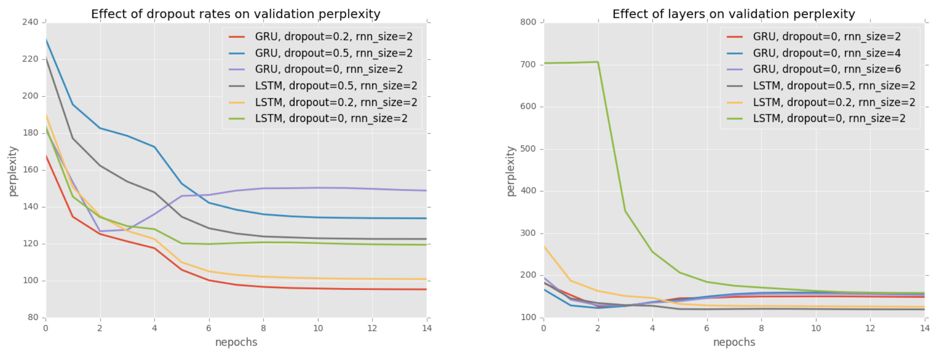

Figure 3: Effect of hyper-parameters on validation perplexity: dropout(left), rnn_size(right)

As we can see, RNNs with GRU cells performed generally better than LSTM based RNNs. Besides, dropout rate has significant impact on the model performance, it is observed in both LSTM and GRU based models that dropout rate of 0.2 achieves the lowest validation perplexity, an interesting finding is that, unlike suggested in some papers[2], dropout rate of 0.5 increases the perplexity dramatically and ends up with a model worse than the no-dropout counterpart.

We also tried different RNN sequence length (20 and 40), however, no significant difference were observed in validation or test perplexity.

To our surprise, we did not observe any performance improvements by increasing the size of hidden layers from 200 to 400 as well, one possible explanation is that we only trained our model for 15 epochs, while larger network takes longer to converge. The best validation perplexity after 15 epochs on a GRU based RNN is 95.644 with test perplexity 92.535, slightly higher than the validation perplexity (95.322) and test perplexity (92.259).

Comparisons of best LSTM and GRU based models can be seen in the table below, the overall minimum perplexities are marked in bold:

| | Validation Perplexity | Test Perplexity |
|---|---|---|
| Dropout = 0.2, rnn_size = 200, 2-layers GRU (Best) | **95.322** | **92.259** |
| Dropout = 0.2, rnn_size = 200, 2-layers LSTM (Best LSTM) | 100.917 | 97.711 |
| Baseline | 150 | 150 |

## References

[1] Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks."
[2] Zaremba, Wojciech, Ilya Sutskever, and Oriol Vinyals. "Recurrent neural network regularization." arXiv preprint arXiv:1409.2329 (2014).