AMS 467/667, Spring 2020, HW2: Hello DL World

Patrick Herbert, Alex Bai

pherber3@jhu.edu
abai1@jhu.edu

Neural Network Set Up
The structure of our new DNN's all have the same framework, with a few modifications to parameters varying by stable set size that will be explained in greater detail later. In order to create a neural network to predict the largest stable set of a graph, we had to make the following changes to the example code given to us:

- Changed loss function from *Binary Cross Entropy* to regular *Cross Entropy* since we are no longer performing a classification problem.
- Converted *ytrain* and *ytest* tensors to *LongTensors* in order to feed them into our loss function.
- Changed *prediction* in *eval* functions to be the argmax of the outputs.
- Removed the *sigmoid* application in the final *out* layer of the network.
- Number of inputs to the first layer became the number of edges in the graph.
- Final layer outputs the number of classes we want, in our case the possible sizes of stable sets (i.e. 4 for Stable 4, 5 for Stable 5, 6 for Stable 6).

Parameter Identification
Importantly, there is some inherent randomness in our networks that makes the results hard to repeat (most noticeably for smaller data sets such as Stable 4). We isolated the following parameters as being the most impactful on our network's performance:

- Number of layers
- Number of neurons
- Test size split[*]
- Learning rate
- Epochs

Some other parameters also had impacts but changing them either did not significantly affect performance or did not make sense for our current problem, these include:

- Activation Function – Sigmoid and tanh didn't make sense to use over ReLU, but we tried Leaky ReLU and saw no significant change.
- Weight Initialization – Kaiming produced worse results, so we stuck with Xavier.
- Batch size – Lower batch sizes made training go slower but didn't seem to noticeably improve performance.

[*]We tried to increase the size of the test data (minimize amount of training) as much as possible and still achieve good results but training on more of the data will almost always improve performance.

Optimal Parameters

	# Layers	# Neurons	Test Size	Learning Rate	Epochs	Training Accuracy	Test Accuracy
Stable 4	5	10	.1	.0025	30	89.47%	100%
Stable 5	7	25	.5	.0025	50	99.02%	93.95%
Stable 6	7	26	.8	.0025	100	95.04%	90.19%

Discussion

There's no optimization or mathematical approach to identifying optimal parameters so we mostly did these by inspection. In general, more layers will increase performance up until a certain point where the model begins to overfit the data. More neurons will act as feature extractors for the input variables to that layer and similarly will increase performance up to a certain point. For Stable 4, it was such a small dataset that we had to train on almost all the data, however, even with a test size of .2 we were still able to get up to 92.3% test accuracy. In an almost reverse fashion, for Stable 6 we were able to have a test size of .9 (training on only 10% of the data) and still produce results of about 85% test accuracy.

The learning rate impacts how quickly the network learns, but too high of a learning rate can lead to an increase in loss, which we saw with a learning rate of $\alpha = .01$. For Stable 6, we also implemented a scheduler to increase the learning rate by a factor of .5 every 50 epochs in order to smooth the loss towards the end.

Epochs were intentionally chosen higher than strictly necessary in order to show the plateau of accuracy over time but could have been shortened if we were looking to optimize the runtime of the network.