50.039 Theory and Practice of Deep Learning
Coding Homework 6

Joel Huang 1002530

April 1, 2019

1 Accuracy on the test set

I used a packed padded sequence input into a LSTM, followed by a single fully-connected layer which accepts the LSTM's hidden state as an output, reducing dimensionality to the number of classes. Where the layer size is greater than 1, we have a stacked LSTM. Besides concatenating the hidden states from all layers (which would require a change in the dimensionality of the fully-connected layer), and averaging across all hidden states, another option is to feed the top-most layer's hidden state as it's a downstream output from the hidden states before it.

Hyperparameter	Value
Epochs	20
Batch size	16
Optimizer	Adam
Learning rate	1e-3
Loss weights	None
Hidden dimensions	[32,64,128]
LSTM layers	[1,2]

Table 1: Hyperparameter selection for Task 1.

I used a 70:30 split for the training and test sets. The entire dataset (of length 20k) is shuffled before splitting. Purely random sampling is used in our training. We train with cross-entropy loss using the Adam optimizer. The rest of the training hyperparameters are shown in Table 1.

Results

Training batches of 16 over 20 epochs with no early stopping yields the results as shown in Table 2.

Number of layers	Hidden size	Test loss	Test accuracy
1	32	0.899	78.8
1	64	0.885	80.4
1	128	0.928	79.5
2	32	0.890	79.4
2	64	0.914	78.4
2	128	0.958	80.4

Table 2: Test results for Task 1.

2 Batched LSTM

LSTM can be run with batches but requires fixed-size sequences for each sample in the batch. To deal with this, we first shortlist a mini-batch of samples. The samples of varying sequence length are then zero-padded to match the length of the longest sequence in the mini-batch. They are then packed and fed into the LSTM.

Hyperparameter	Value
Epochs	20
Batch size	[1,10,30]
Optimizer	Adam
Learning rate	1e-3
Loss weights	None
Hidden dimensions	200
LSTM layers	1

Table 3: Hyperparameter selection for Task 2.

We'll compare the training and test error, and test accuracy for the hyperparameter set in Table 3, using a 70:10:20 split for training, validation and test sets respectively, training for 20 epochs.

Results

The individual plots of training vs. validation losses can be seen in Fig. 1. Training with a batch size of 1 took extremely long and losses did not seem to converge over 20 epochs. Training with a batch size of 10 and 30 resulted in losses converging, with slight overfitting as seen in the plots.

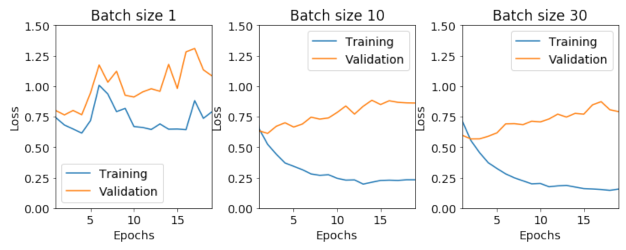

Figure 1: Individual training and validation losses for batch sizes 1, 10 and 30.

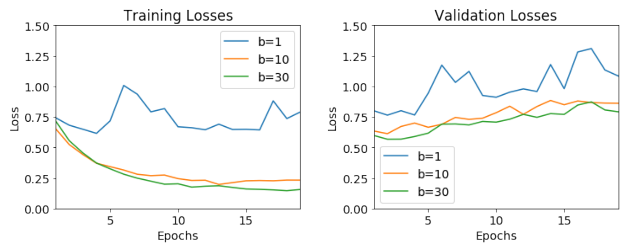

Figure 2: Training and validation comparisons for batch sizes 1, 10 and 30.

Comparing the training and validation losses with each other in Fig. 2, we see the difference between batch size 1 and {10, 30} in greater contrast. Overall, validation loss was increasing over 20 epochs almost right away after starting training, while training loss converged for batch sizes {10, 30} but not for 1.

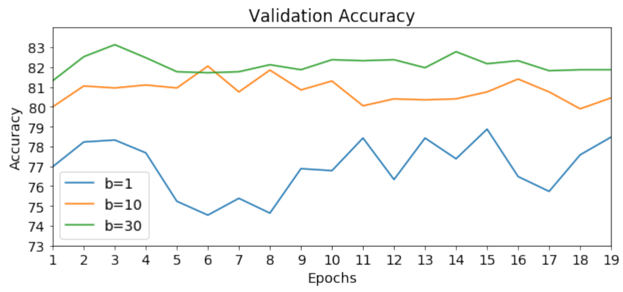

Figure 3: Validation accuracy for batch sizes 1, 10 and 30.

In Fig. 3, we see validation accuracy over the 20 epochs was fairly decent, with clear differences. The higher the batch sizes, the greater the validation accuracy. We can see a parallel result in Table 4, where performance (accuracy) on the test set agrees with the validation accuracy banding.

Number of layers	Hidden size	Test loss	Test accuracy
1	200	1.137	76.8
1	200	0.902	80.1
1	200	0.847	81.3

Table 4: Test results for Task 2.

3 Reproduction

To reproduce the results, use the exact same dataset, with the hyperparameters used above. Seed with `torch.cuda.manual_seed_all(17)`, and a training set size of 14051, validation set size of 2007, and test set size of 4016. I train with Python 3.7.2 on PyTorch 1.0.1.post2. Code is attached.