

# Report for Practice #4

2016025305 Jihun Kim

jihunkim@hanyang.ac.kr

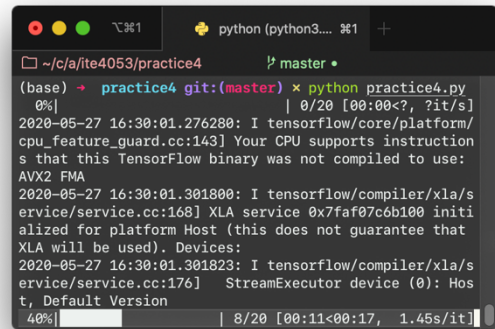
May 28, 2020.

## Development Environment

|           |   |
|-----------|---|
| OS        | MacOS 10.15.4 (19E287)  |
| HW        | MacBook Pro (13-inch, 2017)<br>3.5 GHz Dual-Core Intel Core i7<br>16 GB 2133 MHz LPDDR3 |
| Language  | Python 3.6.8  Anaconda, Inc.  |
| Libraries | NumPy 1.18.4<br>TensorFlow 2.2.0<br>Keras 2.3.0-tf<br>tqdm 4.36.1                       |

## Run

```
$ cd /path/to/repo/practice4  
$ python practice4.py
```



## Experiment Results

All results below are average of 20 runs.

### Comparison between Loss Functions

Used SGD optimizer on both models. Binary cross entropy loss shows slightly better accuracy on this setting.

| Loss Function (learning rate) |       | Binary Crossentropy (10.0) | Mean Squared Error (30.0) |
|-------------------------------|-------|----------------------------|---------------------------|
| Accuracy                      | Train | 99.30%                     | 98.80%                    |
|                               | Test  | 99.45%                     | 98.90%                    |

Table 1: Comparison between loss functions.

### Comparison between Optimizers

Used binary cross entropy, based on previous experiment result. Learning rates differ, and other hyperparameters are set to default. Setting different learning rate to different optimizers is essential because it affect performance. Adam performs best but slightly slower than SGD.

| Optimizer (learning rate) |       | SGD (10.0) | RMSProp (0.01) | Adam (1.0) |
|---------------------------|-------|------------|----------------|------------|
| Accuracy                  | Train | 99.30%     | 99.42%         | 99.63%     |
|                           | Test  | 99.45%     | 99.10%         | 99.15%     |
| Loss                      | Train | 0.021811   | 0.028658       | 0.009953   |
|                           | Test  | 0.018648   | 0.032713       | 0.023056   |
| Elapsed Time              | Train | 1.29s      | 1.42s          | 1.50s      |
|                           | Test  | 76ms       | 87ms           | 87ms       |

Table 2: Comparison between optimizers.

## Comparison between NumPy- and TensorFlow-Implemented Version

Used binary cross entropy as loss and batch gradient descent as optimizer. Results are below. Models implemented with TensorFlow showed much poor speed, approximately 10 times slower. Probably it is because TensorFlow has much more functionalities compared to NumPy.

There is somewhat weird thing that training TensorFlow-implemented model with Colab GPU were much slower than others. As I think, the core reason for this is because of the CPU-GPU data transfer time, which could be larger than the time gain from usage of GPU (instead of CPU). Lower elapsed time at inference phase supports my hypothesis. Because on test phase, CPU-GPU communication occurs much less.

| Model        |       | NumPy<br>(Practice 3, Model 3) | TensorFlow<br>(Local, CPU) | TensorFlow<br>(Colab, CPU) | TensorFlow<br>(Colab, GPU) |
|--------------|-------|--------------------------------|----------------------------|----------------------------|----------------------------|
| Accuracy     | Train | 98.52%                         | 98.23%                     | 98.34%                     | 98.25%                     |
|              | Test  | 98.55%                         | 97.95%                     | 98.25%                     | 98.45%                     |
| Loss         | Train | 0.052826                       | 0.053160                   | 0.053016                   | 0.054302                   |
|              | Test  | 0.051914                       | 0.059397                   | 0.054180                   | 0.048721                   |
| Elapsed Time | Train | 141ms                          | 1.55s                      | 1.79s                      | 3.05s                      |
|              | Test  | 0.078ms                        | 100ms                      | 105ms                      | 94ms                       |

Table 3: Comparison between NumPy- and TensorFlow-implemented versions.

## Comparison between Various Mini-Batch Sizes

In this experiment, I used mean square as loss function and adam with learning rate=0.01 as optimizer. As mini-batch size goes smaller, performance gets better while training time explode.

| Mini-batch size |       | 1        | 32       | 128      | 1000     |
|-----------------|-------|----------|----------|----------|----------|
| Accuracy        | Train | 99.90%   | 99.70%   | 99.47%   | 98.51%   |
|                 | Test  | 99.50%   | 99.00%   | 99.10%   | 98.30%   |
| Loss            | Train | 0.005261 | 0.011637 | 0.016872 | 0.069601 |
|                 | Test  | 0.009106 | 0.019951 | 0.029461 | 0.071044 |
| Elapsed Time    | Train | 15m 5s   | 67.99s   | 18.12s   | 3.94s    |
|                 | Test  | 104ms    | 116ms    | 108ms    | 109ms    |

Table 4: Comparison between various mini-batch sizes.

## Conclusion

Designing good model is of course important, though choosing better learning method and hyperparameters is also crucial. Although using same model, the result can vary largely, depending on the choices made on learning method and hyperparameters.