Report for MNIST classification task

To reproduce the result, simply run the files which is “python main.py”

Initial parameters:

```
Batch size for training: 64
Batch size for testing: 1000
Epochs: 3
Learning rate: 0.01
Momentum: 0.5
Seed: 1
```

Experiment results:

Models	Average loss	Accuracy (testing)
Baseline model	0.1546	9552/10000 (96%)
Model 1	0.1357	9605/10000 (96%)
Model 2	0.5541	8552/10000 (86%)
Model 3	0.1555	9552/10000 (96%)
Model 4	0.1301	9621/10000 (96%)
Model 5	0.1413	9596/10000 (96%)
Model 6	0.0501	9843/10000 (98%)

- Experiment 1: change the size of hidden layer
- Experiment 2: change of activation function
- Experiment 3: add residual layer
- Experiment 4: change numbers of layers
- Experiment 5: change the type of neural network

Baseline model:

3 layer; hidden layer: 256; activation: Relu; Residual: No

Experiment Analysis

EXPERIMENT 1: change the size of the hidden layer

Model 1 structure:
3 layer; hidden layer: 1024; activation: Relu; Residual: No

Compared to baseline model, there isn't a great improvement, since the result from baseline model is already pretty good. Also, the epochs i am using is 3 and as a result, the effect of increasing the size layer might not be obvious. Theoretically, increasing the size of the middle layer will add ability to approximate more complex function since it has more parameters.

EXPERIMENT 2: change activation function

Model 2 structure:
3 layer; hidden layer: 256; activation: Sigmoid; Residual: No

For this model, the performance is dropping significantly. That is because Sigmoid is one of the most basic activation functions and it has a major drawback of info loss due to the derivative having a short range. However, relu is able to reduced likelihood of the gradient to vanish[1].

Model 3 structure:
3 layer; hidden layer: 256; activation: Leaky Relu; Residual: No

I also tried leaky relu since I found it very common in top entries on Kaggle. Different from Sigmoid activation function, Leaky Relu produced similar results compared to Relu. That is because Relu is very similar to Leaky Relu. The comparison between ReLU with the leaky variant is closely related to whether there is a need, in the particular ML case at hand, to avoid saturation[2]

EXPERIMENT 3: add residual layer

Model 4 structure:
3 layer; hidden layer: 256; activation: Relu; Residual: Yes

For this experiment, I add a residual layer on top of the second layer. Compared to the baseline model, there is a slight improvement although not significant. Adding the residual layer allows the flow of memory jump over the layer(s) and therefore allow to create extremely deep layers of neural networks. Our case is not that complicated, so the effect of the residual layer is not as well as its reputation.

EXPERIMENT 4: change number of layer

Model 5 structure:
5 layer; hidden layer: 256; activation: Relu; Residual: No

Increasing layer increases complexity of the network, adding capability to capture more advanced feature. However, this might be an overkill for our MNIST dataset and might result in overfitting, and yielding less than satisfactory testing accuracy.

Experiment 5: change the type of neural network

Model 6 structure:
5 layer (2 conv layer); hidden layer: multiple; activation: Relu; Residual: No

Since MNIST Classification problem is essentially an image classification problem, dense neural network might not be the best choice to produce the best result. I therefore try to add the convolution layer to improve the result. And just as expected, convolutional network is a more favorable network model for image related problems.

Reference:
[1]https://stats.stackexchange.com/questions/126238/what-are-the-advantages-of-relu-over-sigmoid-function-in-deep-neural-networks
[2]https://ai.stackexchange.com/questions/7274/what-are-the-advantages-of-relu-vs-leaky-relu-and-parametric-relu-if-any