CS559 homework 5

655490960
Huiyang Zhao

The first design:

```
conv1 = nn.Conv2d(1, 32, 3, 1)        [3, 200, 200] -> [32, 198, 198]
conv2 = nn.Conv2d(32, 64, 3, 1)       [32, 198, 198] -> [64, 196, 196]
down sampling: max_pool2d(x, 2)       [64, 196, 196] -> [64, 98, 98]
dropout1 = nn.Dropout(0.25)
flatten(x, 1)                         [64, 98, 98] -> [614656]
fc1 = nn.Linear(614656, 128)          [614656] -> [128]
dropout2 = nn.Dropout(0.5)
fc2 = nn.Linear(128, 9)               [128] -> [9]

hyperparameters:
batch-size: 100
test-batch-size: 1000
Normalize: ((0.1307,), (0.3081,))
optimizer = optim.Adam(model.parameters(), lr=1e-3)
scheduler = StepLR(optimizer, step_size=5, gamma=0.7)
```

Explain:
Since the original datasets consists of 200*200 rgb pictures, the input channel is 3.
After two convolutional layers, using a max_pool2d for down sampling, in order to reduce weights number.
Use two fully connected layers to get an output of 9 digits, which represents classifications.
Normalizing comes from torch3.py, just a random guess.

Result:
After one epoch the program terminates due to GPU buffer overflow.

Solve problem:
The network has too much complexity. Solving by adding another max_pool2d layer for down sampling.

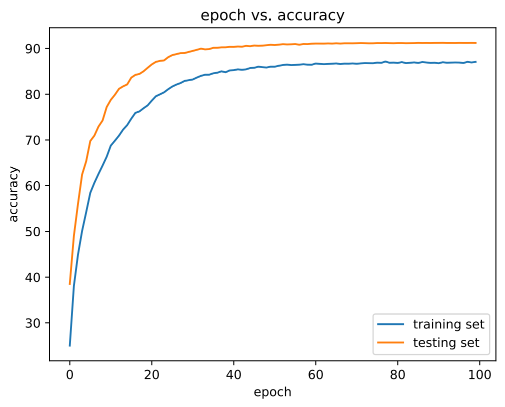
The second design:

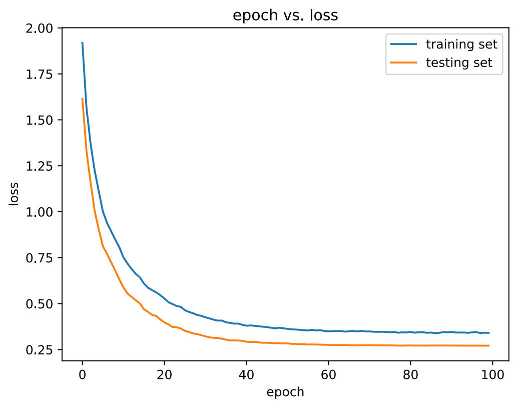
```
conv1 = nn.Conv2d(1, 32, 3, 1)        [3, 200, 200] -> [32, 198, 198]
down sampling: max_pool2d(x, 3)       [32, 198, 198] -> [32, 66, 66]
conv2 = nn.Conv2d(32, 64, 3, 1)       [32, 66, 66] -> [64, 64, 64]
down sampling: max_pool2d(x, 4)       [64, 64, 64] -> [64, 16, 16]
dropout1 = nn.Dropout(0.25)
flatten(x, 1)                         [64, 16, 16] -> [16384]
fc1 = nn.Linear(614656, 128)          [16384] -> [128]
dropout2 = nn.Dropout(0.5)
fc2 = nn.Linear(128, 9)               [128] -> [9]

hyperparameters:
batch-size: 100
test-batch-size: 500
Normalize: ((0.1307,), (0.3081,))
optimizer = optim.Adam(model.parameters(), lr=1e-3)
scheduler = StepLR(optimizer, step_size=5, gamma=0.7)
```

Explain:
Adding a max-pooling layer between two convolutional layers.

Result:
After 100 epochs, I got an acceptable result. The model converges, and the accuracy on test set is closed to 90%. Pictures showing these are attached.
However, the loss almost stops decreasing and accuracy stops increasing for a long time, starting from 40th epoch. That's why I'm trying to create a better model and adjusting hyperparameters.





The third design:

```
conv1 = nn.Conv2d(1, 32, 3, 1)          [3, 200, 200] -> [32, 198, 198]
down sampling: max_pool2d(x, 3)         [32, 198, 198] -> [32, 66, 66]
conv2 = nn.Conv2d(32, 64, 3, 1)         [32, 66, 66] -> [64, 64, 64]
down sampling: max_pool2d(x, 4)         [64, 64, 64] -> [64, 16, 16]
dropout1 = nn.Dropout(0.25)
flatten(x, 1)                           [64, 16, 16] -> [16384]
fc1 = nn.Linear(614656, 128)            [16384] -> [128]
dropout2 = nn.Dropout(0.25)
fc2 = nn.Linear(128, 9)                 [128] -> [9]

hyperparameters:
batch-size: 100
test-batch-size: 500
Normalize: ((0.1307,), (0.3081,))
optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=0.01)
scheduler = StepLR(optimizer, step_size=5, gamma=0.2)
```

Explain:
The parameter of dropout2 was 0.5, which might have caused that loss stops decreasing. That's why I set it to 0.25.
Also, a less gamma might contribute to training. So that gamma is set to 0.2.
Finally, added weight_decay=0.01 in the optimizer in order to reduce complexity.

Result:
It turns out that accuracy stops increasing even earlier. This is caused by a too small gamma.

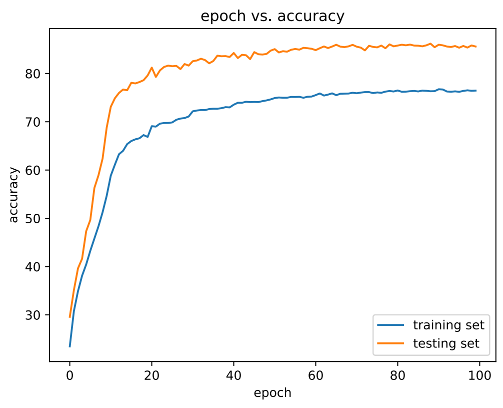
The fourth design:

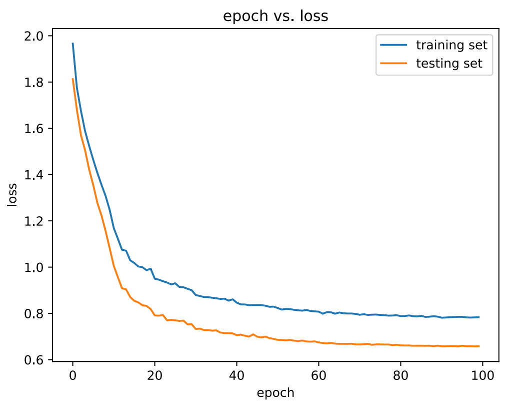
```
conv1 = nn.Conv2d(1, 32, 3, 1)            [3, 200, 200] -> [32, 198, 198]
down sampling: max_pool2d(x, 3)           [32, 198, 198] -> [32, 66, 66]
conv2 = nn.Conv2d(32, 64, 3, 1)           [32, 66, 66] -> [64, 64, 64]
down sampling: max_pool2d(x, 4)           [64, 64, 64] -> [64, 16, 16]
dropout1 = nn.Dropout(0.25)
flatten(x, 1)                             [64, 16, 16] -> [16384]
fc1 = nn.Linear(614656, 128)              [16384] -> [128]
dropout2 = nn.Dropout(0.25)
fc2 = nn.Linear(128, 9)                   [128] -> [9]

hyperparameters:
batch-size: 100
test-batch-size: 500
Normalize: ((0.1307,), (0.3081,))
optimizer = optim.Adam(model.parameters(), lr=1e-3, weight_decay=0.01)
scheduler = StepLR(optimizer, step_size=10, gamma=0.6)
```

Explain:
Gamma is now set to 0.6. At the same time, I set step_size in schedular to 10 so that each stage of learning rate could work longer.

Result:
After 100 epochs, although model converges, the accuracy on test set is around 86%. Pictures showing these are attached.
Unfortunately, the model has an even worse performance the previous one.
After that I took another test with optimizer has weight_decay = 0.03, which made it even worse.





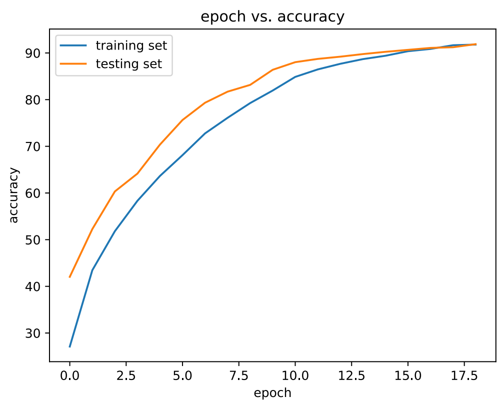
The fifth design:

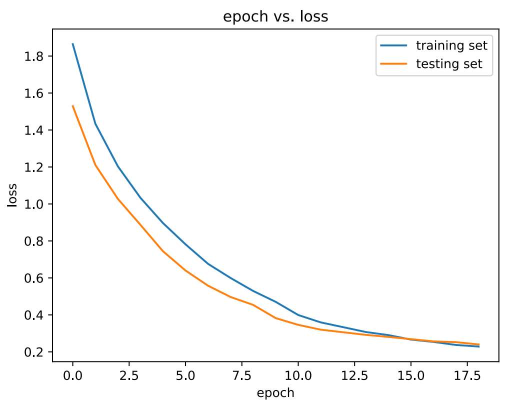
```
conv1 = nn.Conv2d(1, 32, 3, 1)            [3, 200, 200] -> [32, 198, 198]
down sampling: max_pool2d(x, 3)           [32, 198, 198] -> [32, 66, 66]
conv2 = nn.Conv2d(32, 64, 3, 1)           [32, 66, 66] -> [64, 64, 64]
down sampling: max_pool2d(x, 4)           [64, 64, 64] -> [64, 16, 16]
dropout1 = nn.Dropout(0.25)
flatten(x, 1)                             [64, 16, 16] -> [16384]
fc1 = nn.Linear(614656, 128)              [16384] -> [128]
dropout2 = nn.Dropout(0.25)
fc2 = nn.Linear(128, 9)                   [128] -> [9]
```

```
hyperparameters:
batch-size: 100
test-batch-size: 500
Normalize: ((0.1307,), (0.3081,))
optimizer = optim.Adam(model.parameters(), lr=1e-3)
scheduler = StepLR(optimizer, step_size=10, gamma=0.6)
```

Explain:
Not much change from the second design. step_size is set to 10, gamma is set to 0.6.
Since adding decay_weight to optimizer makes it worse, I removed it.
The parameter of dropout2 is set to 0.25.

Result:
After 18 epochs, accuracy of testing set becomes lower than that of training set, and loss of testing set becomes higher than that of training set. This might be caused by overfitting.
However, it is pretty surprising that after only 18 epochs the neural network already has a good performance. The accuracy on test set goes more than 90%. That made my effort a little silly. Results are attached.


epoch vs. accuracy
training set
testing set
accuracy
epoch


epoch vs. loss
training set
testing set
loss
epoch