HW4 Part 2

Cutter Beck

Baseline Model

The notebook started us with a model of 2 fully connected layers.

```
class Net(nn.Module):
    #This defines the structure of the NN.
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 10)

    def forward(self, x):
        x=x.view(-1,784)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        #Softmax gets probabilities.
        return F.log_softmax(x, dim=1)
```

Training

The hyperparameters used were:

- Learning rate: 0.01
- Epochs: 1
- Batch size: 32
- Optimizer: SGD
- Momentum: 0.5

Loss was calculated using negative log likelihood loss.

With this basic model, I received an average loss of 0.1927 and an accuracy of 0.9422.

Add 1 CNN Layer, Max Pooling

I added a single convolutional layer with pooling to the model.

```
class Net(nn.Module):
    #This defines the structure of the NN.
    def __init__(self):
        super(Net, self).__init__()
        self.input_width = 28
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        # conv1 output size: (28 - 5 + (2 * 0)) + 1 = 24
        # pool1 input size: 24
        self.pool = nn.MaxPool2d(2, 2)
        # pool1 output size = 24 / 2 (halved in both dimensions) = 12
        self.fc_in_size = 32 * 12 * 12
        self.fc1 = nn.Linear(self.fc_in_size, 2048)
        self.fc2 = nn.Linear(2048, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = x.view(-1, 32 * 12 * 12)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        #Softmax gets probabilities.
        return F.log_softmax(x, dim=1)
```

Training

The hyperparameters used were:

- Learning rate: 0.01
- Epochs: 1
- Batch size: 32
- Optimizer: SGD
- Momentum: 0.5

Loss was calculated using negative log likelihood loss.

With this model, I received an average loss of 0.0824 and an accuracy of 0.9748.

Experiment 1: 2 CNN Layers with Max Pooling

I added an additional CNN layer with max pooling to the last model, for a total of 2 CNN layers.

```
class Net(nn.Module):
    #This defines the structure of the NN.
    def __init__(self):
        super(Net, self).__init__()
        self.input_width = 28
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        # conv1 output size: (28 - 5 + (2 * 0)) + 1 = 24
        # pool1 input size: 24
        # pool1 output size: 24 / 2 = 12
        # conv2 input channels: 32
        self.conv2 = nn.Conv2d(32, 64, kernel_size=5)
        # conv2 output size: (12 - 5 + (2 * 0)) + 1 = 8
        # pool1 input size: 8
        # pool1 output size = 8 / 2 (halved in both dimensions) = 4
        self.pool = nn.MaxPool2d(2, 2)
        self.fc_in_size = 64 * 4 * 4
        self.fc1 = nn.Linear(self.fc_in_size, 2048)
        self.fc2 = nn.Linear(2048, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, self.fc_in_size)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

        #Softmax gets probabilities.
        return F.log_softmax(x, dim=1)
```

Training

The hyperparameters used were:

- Learning rate: 0.01
- Epochs: 1
- Batch size: 32
- Optimizer: SGD
- Momentum: 0.5

Loss was calculated using negative log likelihood loss.

With this model, I received an average loss of 0.0756 and an accuracy of 0.9770. This is slightly higher accuracy than the previous model, but not by much. I will try increasing the training epochs and decrease the learning rate.

Experiment 2: 2 CNN Layers with Max Pooling, More Epochs, Lower Learning Rate

I increased the number of epochs to 5 and decreased the learning rate to 0.001.

Training

The hyperparameters used were:

- Learning rate: 0.001
- Epochs: 5
- Batch size: 32
- Optimizer: SGD
- Momentum: 0.5

Loss was calculated using negative log likelihood loss.

This model received an average loss of 0.0793 and an accuracy of 0.9770.

Experiment 3: 1 CNN Layer with Max Pooling, revert Epochs and Learning Rate, more FC Layers

I added 2 additional FC layers to the model and reverted the number of epochs and learning rate to the original values.

```
class Net(nn.Module):
    #This defines the structure of the NN.
    def __init__(self):
        super(Net, self).__init__()
        self.input_width = 28
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        # conv1 output size: (28 - 5 + (2 * 0)) + 1 = 24
        # pool1 input size: 24
        self.pool = nn.MaxPool2d(2, 2)
        # pool1 output size = 24 / 2 (halved in both dimensions) = 12
        self.fc_in_size = 32 * 12 * 12
        self.fc1 = nn.Linear(self.fc_in_size, 2048)
        self.fc2 = nn.Linear(2048, 1024)
        self.fc3 = nn.Linear(1024, 100)
        self.fc4 = nn.Linear(100, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = x.view(-1, 32 * 12 * 12)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)

        #Softmax gets probabilities.
        return F.log_softmax(x, dim=1)
```

Training

The hyperparameters used were:

- Learning rate: 0.01
- Epochs: 1
- Batch size: 32
- Optimizer: SGD
- Momentum: 0.5

Loss was calculated using negative log likelihood loss.

This model received an average loss of 0.1002 and an accuracy of 0.9697. This is not much better or worse than any of the previous models, so I want to try another experiment.

Experiment 4: 1 CNN Layer, 4 FC Layers, Change up activation functions

I tried using a variety of activations in different locations of the FC layers.

Model Structure

```
class Net(nn.Module):
    #This defines the structure of the NN.
    def __init__(self):
        super(Net, self).__init__()
        self.input_width = 28
        self.conv1 = nn.Conv2d(1, 32, kernel_size=5)
        # conv1 output size: (28 - 5 + (2 * 0)) + 1 = 24
        # pool1 input size: 24
        self.pool = nn.MaxPool2d(2, 2)
        # pool1 output size = 24 / 2 (halved in both dimensions) = 12
        self.fc_in_size = 32 * 12 * 12
        self.fc1 = nn.Linear(self.fc_in_size, 2048)
        self.fc2 = nn.Linear(2048, 1024)
        self.fc3 = nn.Linear(1024, 100)
        self.fc4 = nn.Linear(100, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = x.view(-1, self.fc_in_size)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.sigmoid(self.fc3(x))
        x = self.fc4(x)

        #Softmax gets probabilities.
        return F.log_softmax(x, dim=1)
```

Change out Activations

Sigmoid

```
def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = x.view(-1, self.fc_in_size)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = F.sigmoid(self.fc3(x))
    x = self.fc4(x)

    #Softmax gets probabilities.
    return F.log_softmax(x, dim=1)
```

Bad results: Average loss of 0.2088, accuracy of 0.9424

Tanh

```
def forward(self, x):
    x = self.pool(F.relu(self.conv1(x)))
    x = x.view(-1, self.fc_in_size)
    x = F.relu(self.fc1(x))
    x = F.relu(self.fc2(x))
    x = F.tanh(self.fc3(x))
    x = self.fc4(x)

    #Softmax gets probabilities.
    return F.log_softmax(x, dim=1)
```

Much better results: Average loss of 0.0925, accuracy of 0.9720

Results

The best model had 2 CNN layers with max pooling and 2 FC layers.

The hyperparameters used were:

- Learning rate: 0.01
- Epochs: 1
- Batch size: 32
- Optimizer: SGD
- Momentum: 0.5

Loss was calculated using negative log likelihood loss.

With this model, I received an average loss of 0.0756 and an accuracy of 0.9770. This accuracy was the same as the model in experiment 2, but the loss was less.

I learned how different combinations of layers, activation functions, and hyperparameters affect the overall performance of the model.