Machine Learning HW#4

Ella Pavlechko
Due 11/8/19

1

- Modify the code mnistcnn.py from the convolutional neural network lecture to use Keras, and change the first dense layer to use 128 neurons in the first dense layer.
- Use the Adam optimizer and dropout for the first dense layer.
- Use an 80/20 split on the training data and only save the model for the best validation loss.

Report training, validation, and out of sample testing accuracy.

The MNIST database comes pre-loaded with a set of 60,000 training images and 10,000 testing images. We split the pre-set training images into 48,000 training images and 12,000 validation images.

The layout of the Convolutional Neural Net is as follows:

1. Take a 28×28 pixel image from MNIST
2. Apply 32, 5×5 filters **without** the same padding (since Problem 2 asks us to add it in)
3. Maxpool with 2×2 frames
4. Apply 64, 5×5 filters **without** the same padding
5. Maxpool with 2×2 frames
6. Pass into a fully connected layer with 128 neurons and relu activation
7. Dropout (a percentage of randomly selected neurons ignored during training)
8. Pass into a layer with 10 neurons and softmax activation (these will be our outputs, the highest value will tell us which class to place the image in)

Minimizing the cross-entropy loss using the Adam optimizer, we find the weights in the Network using the training data, and check hyperparameters on the validation data. Since 10 EPOCH's was enough to get the accuracies to converge, we stopped the training. In the last EPOCH we get:

$$\text{Training Accuracy} = 98.7\%$$
$$\text{Validation Accuracy} = 98.6\%$$
$$\text{Testing Accuracy} = 98.8\%$$

2

- Modify the architecture from problem (1) to use 'same' padding.
- Add the following layers conv(32 filters,3x3,stride1)-maxpool(2x2)-conv(32 filters,3x3,stride1)-maxpool(2x2) before the first dense layer, also use 'same' padding for the convolutions.
- Modify this architecture to include batch normalization after each of the convolution layers.
- Plot the training and validation loss as a function of the number of epochs for this problem and problem (1) all on the same graph using Tensorboard, report your findings.

Also report the training, validation, and out of sample testing accuracy for this problem. Turn in your code for this problem.

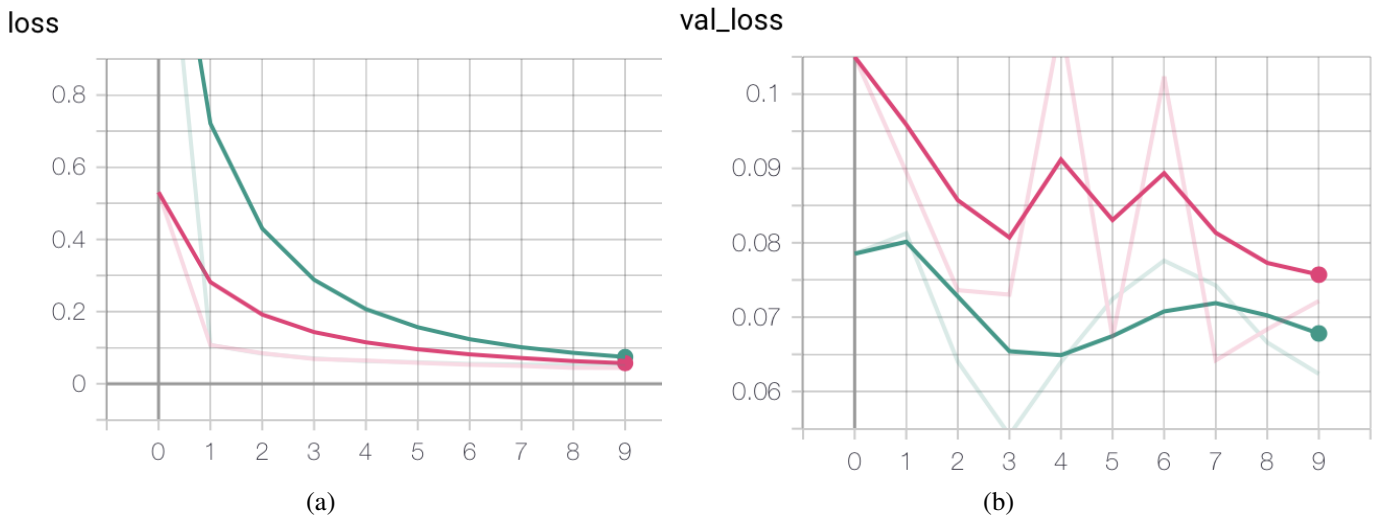
Modifying the Architecture as suggested, we find on the last EPOCH:

$$\text{Training Accuracy} = 98.8\%$$
$$\text{Validation Accuracy} = 98.3\%$$
$$\text{Testing Accuracy} = 98.8\%$$

These results show our Network will be extremely accurate, and fairly similar to the results of Problem 1. Using Tensorboard we are able to generate Figure 1 to represent the accuracies and losses throughout training. While I

wasn't able to find a way to combine the graphs for training and validation loss, when we compare them side-by-side we can see the training loss for Problem 2 was consistently lower than Problem 1. However, we also see the validation loss for Problem 2 was worse than Problem 1.

Comparing the models, Problem 1 starts off with a much higher training loss than validation loss, meaning we were underfitting the data. However, around the 5th EPOCH they become equal, and then the training loss is less than validation. Problem 2 had the training loss and validation loss being fairly close, indicating that our data is being more properly fit. Around the 4th EPOCH our losses are equal, and afterwards the training loss is lower than validation loss. To avoid the risk of overfitting in both models we should stop training around when the losses are equal, or the validation starts increasing.



(a) (b)

Figure 1: The Green curves correspond to the model in Problem 1, but Pink corresponds to Problem 2. The x-axis is the number of EPOCH's. Graphs generated with Tensorboard and smoothness 0.7