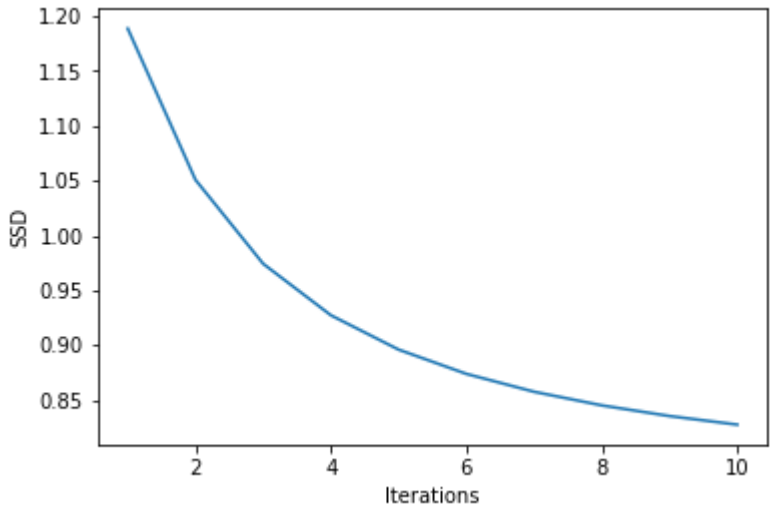
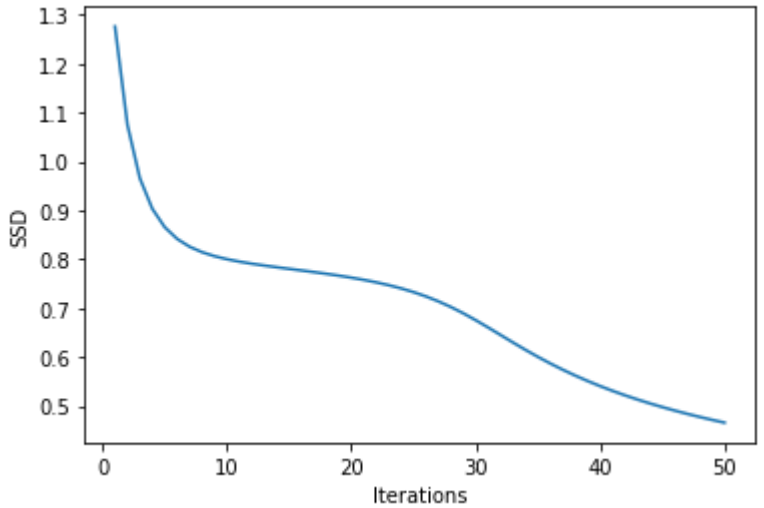
Task1) Run the above code and interpret the results. Please note the output of the model is the prediction of class labels of each of the four points. If you run the code several times, will you observe the same results? Why? Keep the parameter “n_unit=1” and increase the number of iterations starting from 10, 50, 100, 500, 2000, and compare the loss values. What can you conclude from increasing the number of iterations? Now, with a fixed value of “iterations = 1000”, increase the parameter “n_unit” to 2, 5, 10 and interpret the results.

When the code is run several times, the results are different since the weights are initialized randomly every time. The difference should become smaller with more iterations however. When changing the iterations from 10, to 50, 100, 500 and 2000 we observe the sum of square error (SSD) lowering for higher iterations. Changing the n_unit does not have as clear effects. The initial value is different for different n_unit, but this is due to the initial weights being randomized. Other than this, the final SSD value seems to not be dependent on the value of n_unit.

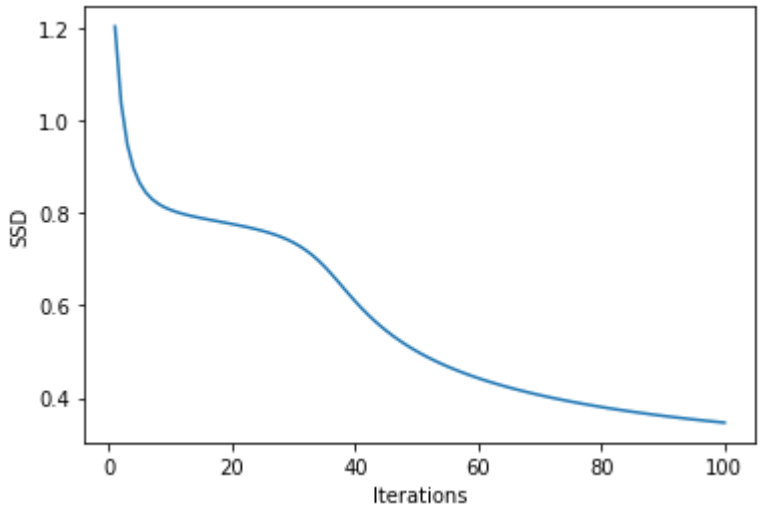
n_unit = 1, interation = 10



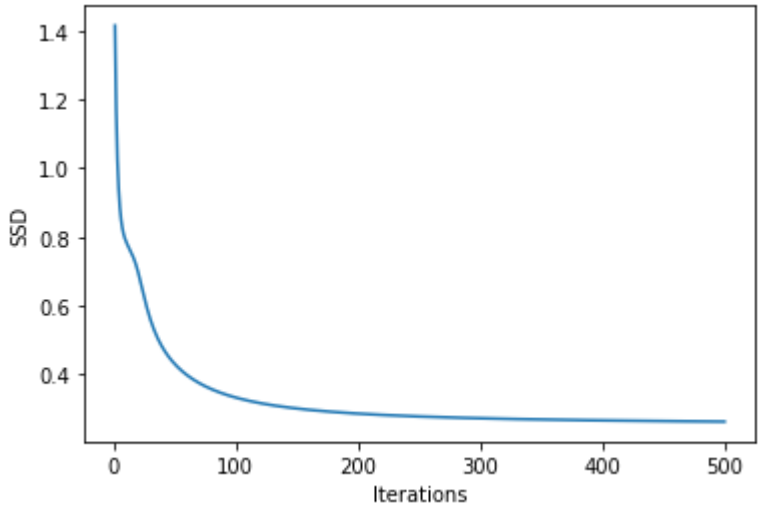
n_unit = 1, interation = 50



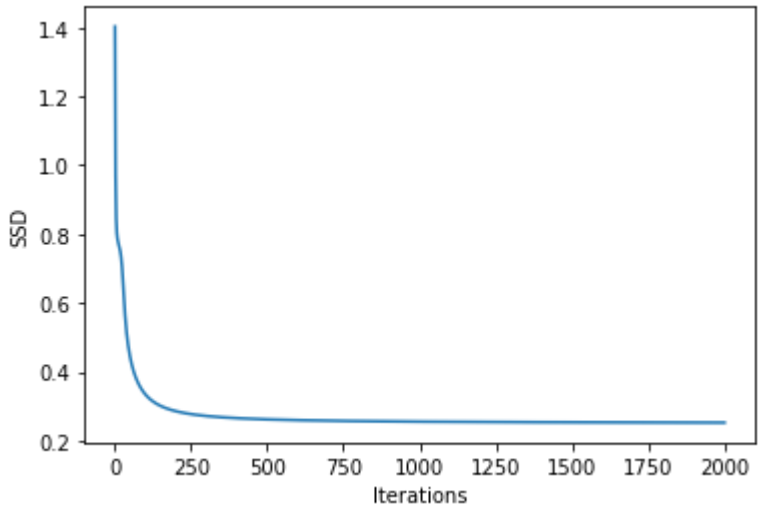
n_unit = 1, interation = 100



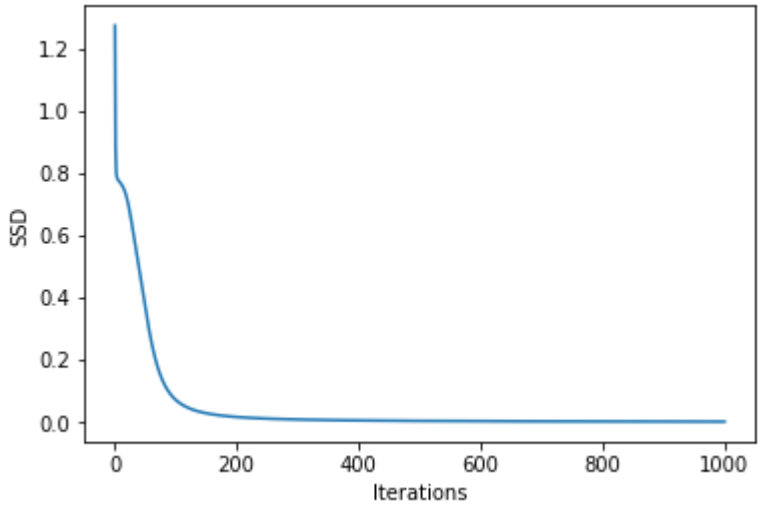
n_unit = 1, interation = 500



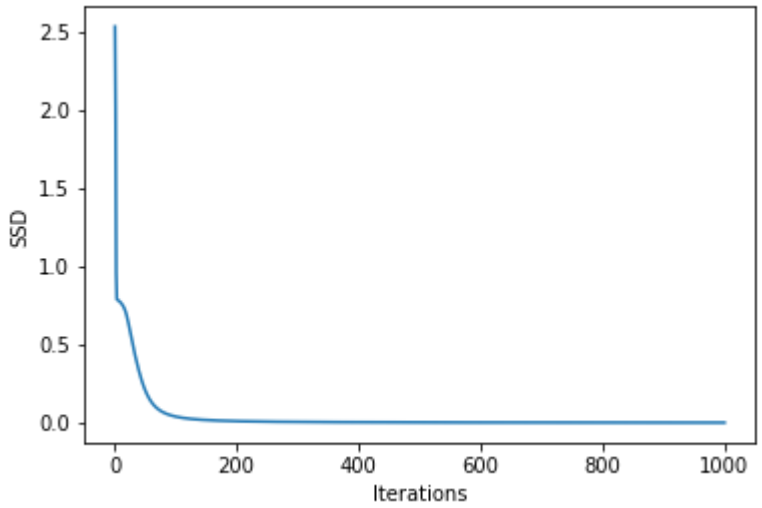
n_unit = 1, interation = 2000



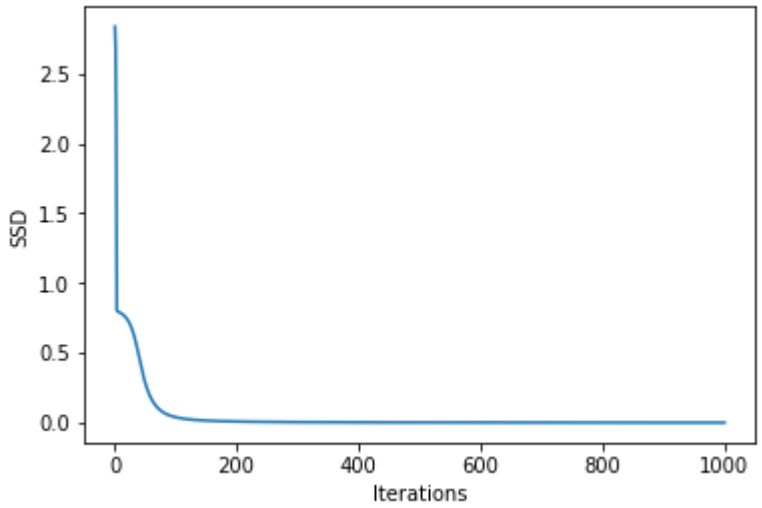
n_unit = 2, iteration = 1000



n_unit = 5, iteration = 1000



n_unit = 10, iteration = 1000



Task2) Repeat task1 for XOR logic operator. For fixed values of parameters (iterations=2000, and n_unit=1), which of the AND or XOR operators has lower loss values? Why? Increase the number of neurons in the hidden layer (n_unit) to 2, 5, 10, 50. Does increasing the number of neurons improve the results? Why?

The predictions are in general quite bad, being around 0.5, or in the range 0.4 to 0.6. A bigger n_unit does not seem to make predictions better. Predictions are bad for all n_unit values, and if we round the numbers to binary we get the correct predictions for n_unit 5 and 50, but incorrect for n_unit 2 and 10. Increasing the n_unit does not improve the result. In theory 2 neurons are enough to separate these data points and increasing the number of neurons beyond that does not make our predictions better.

Target is [0, 1, 1, 0]

n_unit = 2
The predicted class labels are: [[0.57265294]
 [0.57265294]
 [0.5726515]
 [0.26854098]]
If we round the numbers to binary, we get: [1, 1, 1, 0]

n_unit = 5
The predicted class labels are: [[0.47955292]
 [0.6893501]
 [0.5329312]
 [0.40572   ]]
If we round the numbers to binary, we get: [0, 1, 1, 0]

n_unit = 10
The predicted class labels are: [[0.45500156]
 [0.4957725]
 [0.69978535]
 [0.38289973]]
If we round the numbers to binary, we get: [0, 0, 1, 0]

n_unit = 50
The predicted class labels are: [[0.41432053]
 [0.6571237]
 [0.6477299]
 [0.36479098]]
If we round the numbers to binary, we get: [0, 1, 1, 0]

Task3) In the above code, change the parameter “n_unit” as 1, 10 and interpret the observed results.

With only one neuron, our predictions are all 0.5, which means the model cannot tell which class the data points belong to. It becomes even worse when we set the learning rate (lr) to

0.5. With 10 neurons our results are better, although like the results from task 2, we know that increasing the neurons above 2 does not improve the results further.

n_unit = 1, lr = 0.01
The predicted class labels are: [[0.5010974]
[0.5010974]
[0.5098544]
[0.5080095]]

n_unit = 10, lr = 0.01
The predicted class labels are: [[0.48036295]
[0.7145369]
[0.4790213]
[0.34444645]]

n_unit = 1, lr = 0.5
The predicted class labels are: [[0.5]
[0.5]
[0.5]
[0.5]]

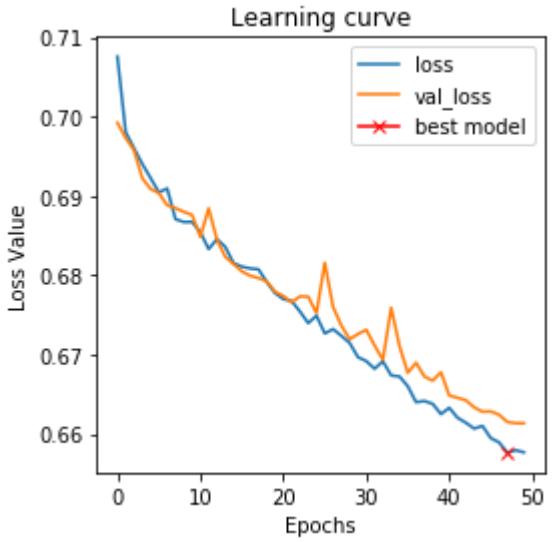
Task) Review the following data loader code and find out how it works. Run it to load the training and test data.

Task4:

4.1 How do you interpret the observed values of loss and accuracy values?

The observed values of loss stand for the difference between the predicted value and the ground truth. With the training from each epoch, the values of loss are gradually decreasing, which indicates that the model learns from the error and the errors become smaller. Accuracy values come from the testing data. And after each epoch of training, the accuracy value is obtained to analyze its performance.

Epoch = 50, LR = 0.0001, Base_dense = 64

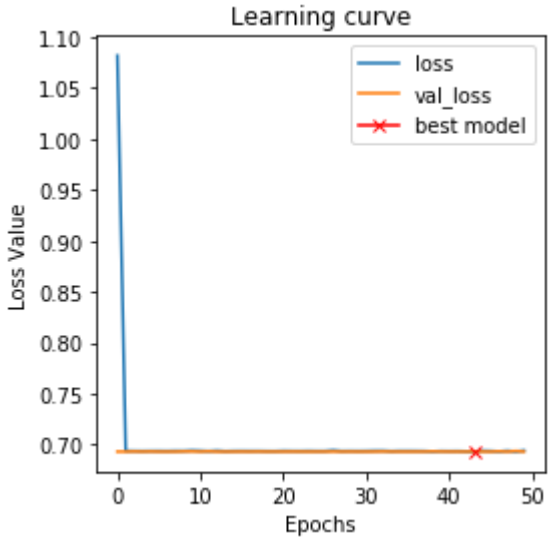


4.2 Is the number of epochs enough to make a good decision about model performance?

When the number of epochs is 50, it's not enough to make a decision.

4.3 For the same number of epochs, reduce the learning rate parameter to 0.1 and interpret the results.

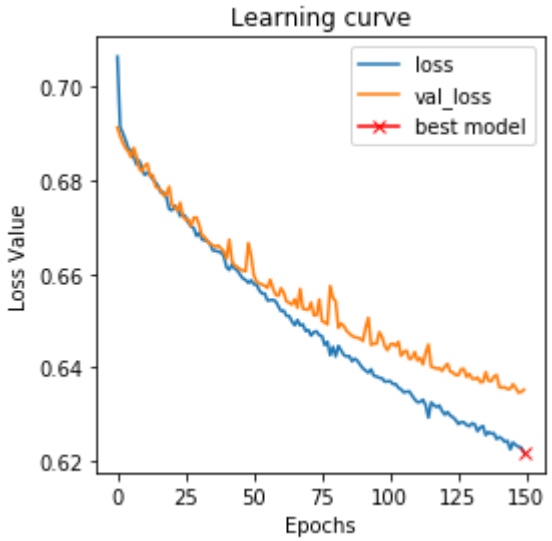
When the LR is 0.1, the learning curve decreases rapidly in the beginning of the training. But the value of loss keeps very steady during the training. It indicates that when the learning rate is too big, the weight adjusting process becomes difficult because the model couldn't reach an even smaller error. Then the performance of the model is limited.
Epoch = 50, LR = 0.1, Base_dense = 64



4.4 Now increase the number of epochs to 150 with LR=0.0001. Does this model have enough capacity to yield acceptable results?

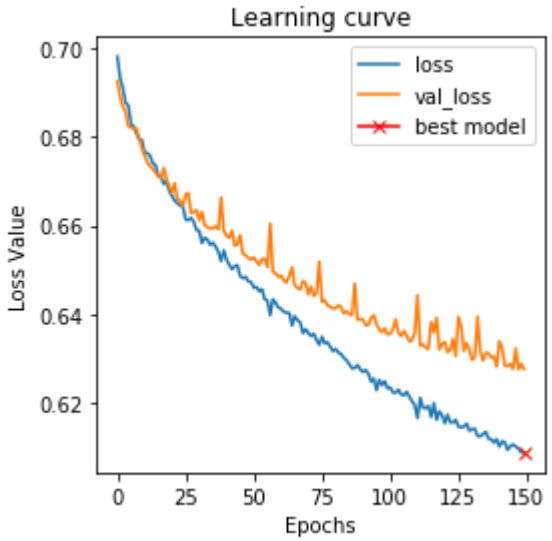
No, the two curves seem to be not converging anymore, which indicates the training data has higher accuracy than the testing data. The model is prone to remember the data instead of learning from it.

Epochs = 150, LR = 0.0001, Base_dense = 64



4.5 Increase the "base_dense" parameter to 256 and compare the results with the case of "base_dense=64". Is increasing the model capacity helpful to improve the model performance? Why?

No. Two curves have bigger differences during the training. The loss value for training data is lower than the testing data, which indicates overfitting. Increasing the base dense in this test doesn't help with improving the performance of the model.
Epochs = 150, LR = 0.0001, Base_dense = 256

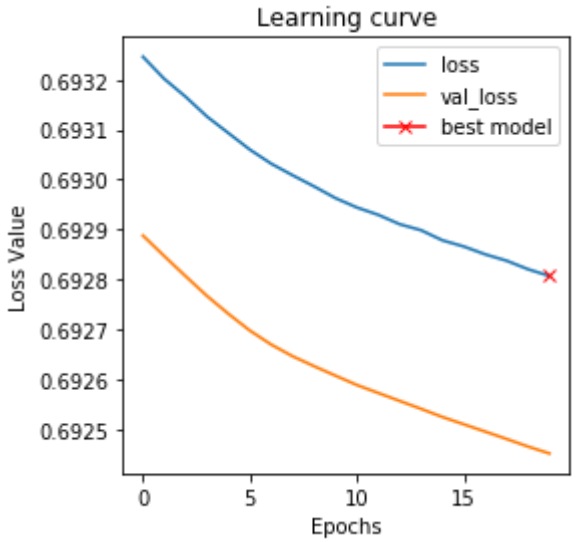


Task 5A) Set the following parameters: # of epochs = 20; batch size = 8; number of feature maps at the first convolutional layer = 32 and learning rate = 0.00001 and then run the experiment.

5.1 What are the value training and validation accuracies? What can you infer from the learning curves? Is it reasonable to make a decision based on this set-up of the parameters?

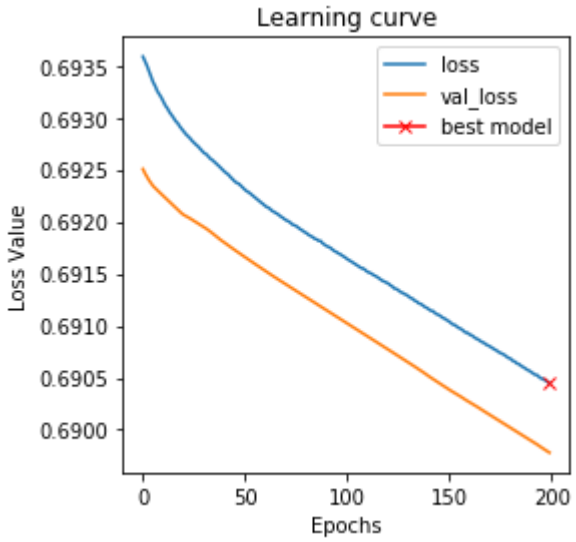
Training accuracy is obtained from models prediction of training data and the true labels. And validation accuracy is from models prediction of the unseen testing data and the true labels. From the learning curves, we could see that the losses from training and testing are both decreasing. However, the loss curves have not converged yet. It's not reasonable to make a decision. The times of training with epochs are too small to reach an acceptable model.

n_epoch = 20



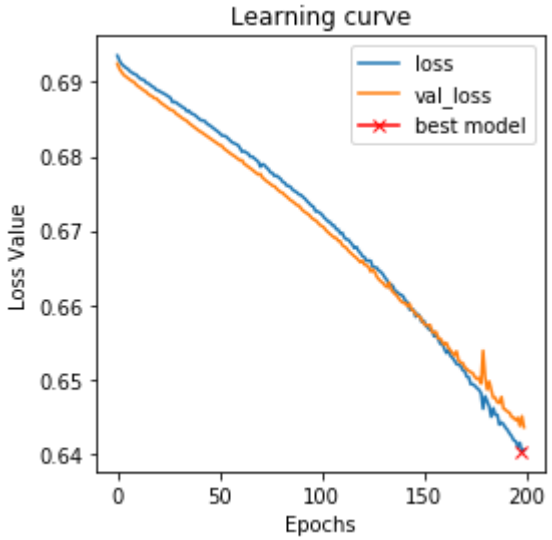
Task 5B) Leave all the parameters from the previous task unchanged except for the n_epoch = 200. Compare the results with task 5A.

When the epochs are 200, the losses keep decreasing slowly, and the gap between two losses seem to become smaller. The results are still not acceptable since there seems to be space to increase the performance of the model by learning more from the training data.
n_epoch = 200



Task 5C) Keep all parameters from the last step except for the LR = 0.0001 . Run the experiment with a new LR and interpret the results. How do you evaluate the generalization power of the model? What are the values of training and validation accuracies?

The generalization power gets stronger since the values of learning curves decrease more rapidly. The training accuracy is 0.695, while the validation accuracy is 0.700.
LR = 0.0001



Task 5D) What is the role of the first two convolutional layers?

The first two convolutional layers are in charge of extracting the high-level features such as the edges and contours of inputs.

Task 5E) What is the role of the last two dense layers?

The main purpose of the model is classification. The second to last dense layer gets the features from the convolutional layers and selects meaning features for the model. The last layer calculates the probability with sigmoid function as the results.

Task 5F) What is the major difference between the LeNet and MLP?

From the structure, LeNet is one of MLPs but with convolutional layers. This leads to the difference in weight sharing. For images, the convolutional layers apply kernels to every patch of the images repeatedly and produce intermediate features. The weight of these features are shared. MLP consists of input layer, hidden layers and output layer and layers are fully connected. From the results we have above, the performance of LeNet seems to be more stable than MLP.

Task 5G) Look at the last layer of the network. How should we choose the number of neurons and the activation function of the last layer?

It depends on the format of expected results or labels. When the purpose of the model is classification, a sigmoid function is usually used as activation function if the results are expected to be the probability of prediction. And the number of neurons is 1 since the output would be a value. But when the output is a vector for classification, the number of neurons is the length of the output vector.