PROJECT 3

CS691 - Machine Learning
Due Date : November 8, 2019

Nasif Zaman, Hoang-Dung Bui

This is the write-up for Machine Learning - Project 3. The code should run on python 3.

Chapter 1

Neural Network

This part describe how *neural_network.py* works.

1.1 unique_classes() and recast()

unique_classes: this function will return a dictionary which contain the key - types of classes in the training data set and the value of the key - 1.
recast(): this function returns an array *y_* with size (number of sample x, number of classes). This is simply one-hot encoding mechanism.

1.2 calculate_loss(model, X, y)

This function calculate the loss of a model on data set *X*. The function should work step-by-step as following:

- Load number of sample *N* in the dataset *X*.
- Load number of classes *cl*
- Load an array by recast function to calculate the total Loss.
- Start the loop to run through all data sample.
  - Calculate the forward propagation by **matrix multiplication operation** to get the *y_hat*
  - Compare with the label from array *y_* to calculate the Loss
- Calculate the final Load by dividing by total sample number.

The activation function of the output layer is *softmax* function and its output is the probability of each class.

The *softmax* Cross-Entropy Loss is used here. This function returns the loss which is the division between the wrong prediction over the number of sample on the training dataset.

1.3 predict(model, x)

This function does the forward propagation to make the prediction on the data x with an input model.

- It loads the weights and biases of the model, using *matrix multiplication* to multiply with the data x
- Feed it to activation function of the hidden layer
- The output of the hidden layer will be multiplied with weights of the output layer by *matrix multiplication.*
- The activation function of the output layer is *softmax* function, and its output is probability of the sample for each class

The function returns the prediction whose probability are higher.

1.4 back_prop(model, X, y, eta)

This function directly updates the model's weights and biases by batch gradient descent.

- It loads the weights and biases of the model, then uses *matrix multiplication* to multiply with all dataset X
- Feed it to activation function of the hidden layer
- The output of the hidden layer will be multiplied with weights of the output layer by *matrix multiplication.*
- The activation function of the output layer is *softmax* function, and its output is probability of the sample for each class
- The errors (gradient descent) between the prediction and the label are calculated.
- The gradient descent corresponding to sum function a is calculated.
- update the weight and bias

It will return a model with updated weights and biases.

1.5 build_model(X, y, nn_hdim, num_passes, print_loss)

The main input to the function are: training data *X*, its label *y*, node's number in hidden layer *nn_hdim*, and number of epoch *num_passes*. In the function, it initializes the model by its weights *W1*, *W1* (matrix form) and biases *b1*, *b2* (vector form).

	rows	**Description**	**columns**	**Description**
W1	X.shape[1]	Number of Features	nn_hdim	Number of nodes in hidden layer
W2	nn_hdim	Number of nodes in hidden layer	len(unique_classes(y))	Number of classes
b1	1	One row	nn_hdim	Number of nodes in hidden layer
b2	1	One row	len(unique_classes(y))	Number of classes

It will run a *for loop* with *num_passes* time. In each epoch, it will call function *back_prop* to update the weights and bias of the model. After every 1000 epochs, if *print_loss* is selected, function *calculate_loss* will be called and print out the *loss* value.

The function will return the trained model.

Chapter 2

Plot Decision Boundary

2.1 Code's Description

In file *Test.py*, we will test the model with different node's number in the hidden layers. For each number, function *build_model* is called to get a model with a specific node's number for the hidden layer. Then function *plot_decision_boundary* is called and print out the graph in one corner of plot window. There four graph will be printed out on the plot window.

plot_decision_boundary(pred_func, X, y) This function will call function *predict* to test on the data X then classify the data into two group.

2.2 Plot Decision Boundary

This section will present the results of our model. The required node's number to be tested here are: 1, 2, 4, and 5. Moreover, we also test the model with the node's number of hidden layer are 10 and 100, then make some comparison on that.

Hidden layer Node's Number 1, 2, 4, and 5

If the node's number is only 1, the boundary is similar a linear boundary with the loss is 0.303 - a high value. For the node's number is 2, the neural network can classify better with the loss of 0.249 and the boundary is combination of two lines to separate the data. As increasing the nodes in the hidden layer to 4 and 5, the neural network make much better prediction. As node's number is 4, the boundary is a smooth curve and well classifying the data. The loss is only 0.051 and the classification is generalize. As the number of nodes in hidden layer is 5, the loss is 0.0294 however, it seems it loses the generalization of classification.

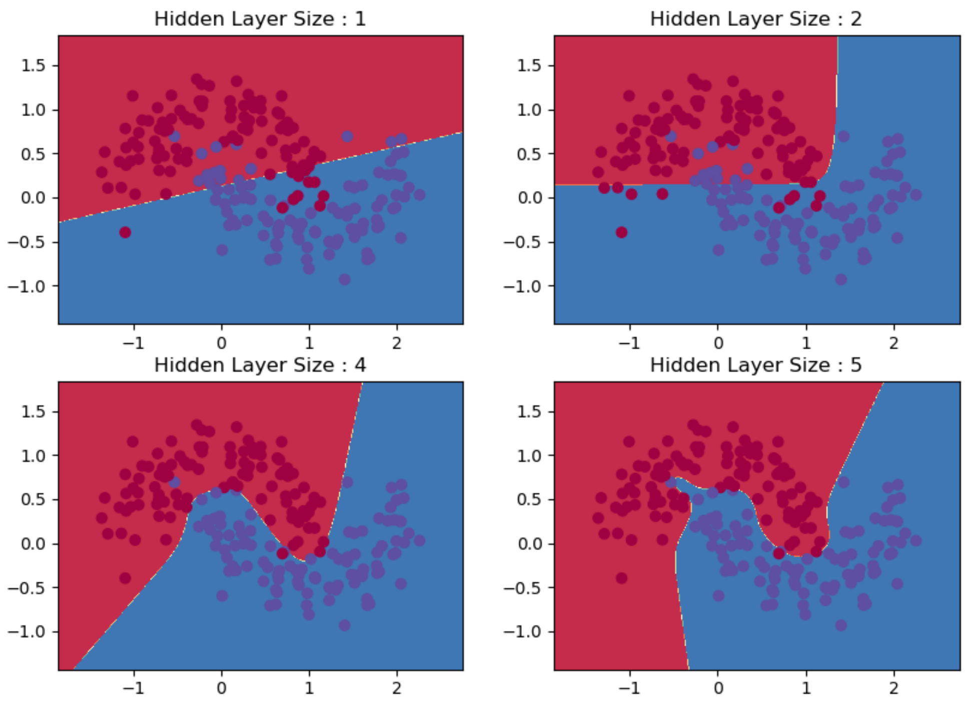

Figure 2.1: Decision Boundary with hidden layer node's number 1, 2, 4, and 5.

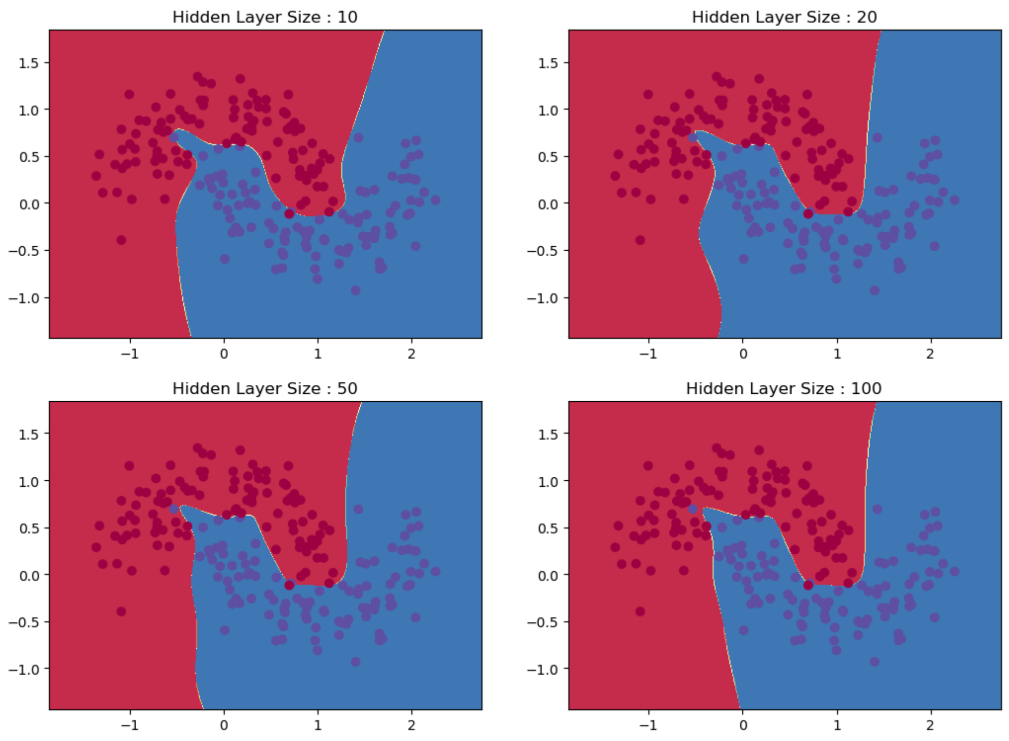

Figure 2.2: Decision Boundary with hidden layer node's number 10, 20, 50, and 100.

Hidden layer's Size: 10, 20, 50 and 100: As increasing the hidden layer's size to 10, 20, 50 and 100, it seems the result is similar to the size 5, and the classification loses its generalization. Thus, select the hidden layer's size is 4 or 5 is the best choice.