UDACITY

PROJECT

Your first neural network

A part of the Deep Learning Nanodegree Foundation Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

3 SPECIFICATIONS REQUIRE CHANGES

Well done! You are awesome. Your work shows that you invested a lot of time and effort before submitting this project. There are some changes that need to be done. I have given some hints which may help you to build a good Neural Network model. You are very close to an acceptable project.
Please correct the changes suggested in these sections of the rubrics:

- The input to the hidden layer is implemented correctly in both the train and run methods.
- Updates to both the weights are implemented correctly.
- The number of epochs is chosen such the network is trained well enough to accurately make predictions but is not overfitting to the training data.
- The learning rate is chosen such that the network successfully converges, but is still time efficient.
- The number of hidden units is chosen such that the network is able to accurately predict the number of bike riders, is able to generalize, and is not overfitting.
  Correct the changes and submit it again. 😄
  Keep up the great work!

Keep Learning! Deep Learning!

Code Functionality

All the code in the notebook runs in Python 3 without failing, and all unit tests pass.

All the code snippets and unit tests are executing perfectly in Python 3.

The sigmoid activation function is implemented correctly

Awesome!
You have correctly implemented the sigmoid function.
It can also be implemented using the lambda function. The lambda function is a great way to create small anonymous functions. It can be implemented like this -> `self.activation_function = lambda x: 1/(1+np.exp(-x))`

Forward Pass

The input to the hidden layer is implemented correctly in both the train and run methods.

You have implemented it this way: `hidden_inputs = np.dot(inputs.T, self.weights_input_to_hidden.T)`.
This is correct but there is no need to transpose the matrices. It can be multiplied without transposing the matrices: `np.dot(self.weight_input_to_hidden, inputs)`

The output of the hidden layer is implemented correctly in both the `train` and `run` methods.

The activation function is correctly used to calculate the output of the hidden layer.

The input to the output layer is implemented correctly in both the train and run methods.

There is no such requirement to transpose the weight_hidden_output_matrix in `final_inputs = np.dot(hidden_outputs, self.weights_hidden_to_output.T)`.
It can be directly implemented like this `np.dot(self.weight_hidden_to_output, hidden_outputs)`

The above changes are highly recommended.

The output of the network is implemented correctly in both the train and run methods.

This is one of the steps in which many students make a mistake. But you have correctly understood it. Good Job! 👍

Backward Pass

The network output error is implemented correctly

Perfectly implemented. 😄

Updates to both the weights are implemented correctly.

The code clearly shows that you tried hard to follow each rubric. Your code is mathematically correct, but the coding style needs to be improved. Your code looks very clumsy but it can be improved.
You are calculating the weight update steps correctly but the values of hidden_errors, hidden_grad are incorrect.
The whole code for weight update can be written in 4-5 lines of code.

- `hidden_errors = np.dot(self.weight_hidden_to_output.T, output_errors)`
- `hidden_grad = hidden_outputs * (1.0 - hidden_outputs)`
- `self.weight_hidden_to_output += self.learning_rate * np.dot(output_errors, hidden_outputs.T)`
- `self.weight_input_to_hidden += self.learning_rate * np.dot(hidden_errors * hidden_grad, inputs.T)`

Always use np.dot() method for matrix multiplication.

Hyperparameters

The number of epochs is chosen such the network is trained well enough to accurately make predictions but is not overfitting to the training data.

I appreciate the approach tried by you to test the behavior of the model by varying the values of hyperparameters. It seems you have done some kind of EDA, but it's not required.
Different values of hyperparameters should be tried and tested to observe the behavior of the model but we should also understand the main purpose and range of values to try for a specific hyperparameter.

The number of epochs is the number of times the data set will pass through the network, each time updating the weights. As the number of epochs increases, the network becomes better and better at predicting the targets in the training set. You'll need to choose enough epochs to train the network well but not too many or you'll be overfitting.
So, the value of epochs should be increased. You should choose the number of epochs such that the loss on the training set is low and the loss on the validation set isn't increasing(On training & validation loss graph). So, you can observe the behavior of the model by increasing the value of epochs. You can try values like 800, 1000, 1500, 2000, 3000, 5000.

The number of hidden units is chosen such that the network is able to accurately predict the number of bike riders, is able to generalize, and is not overfitting.

The number of nodes is fairly open; if validation loss is low, it is an acceptable value. It should probably be no more than twice the number of input units, and enough that the network can generalize, so probably at least 8. A good rule of thumb is the half way in between the number of input and output units.

There's a good answer here for how to decide the number of nodes in the hidden layer. https://www.quora.com/How-do-I-decide-the-number-of-nodes-in-a-hidden-layer-of-a-neural-network

You can observe the change in behavior of model for values like 10, 12, 15, 20, 25, 30.

The learning rate is chosen such that the network successfully converges, but is still time efficient.

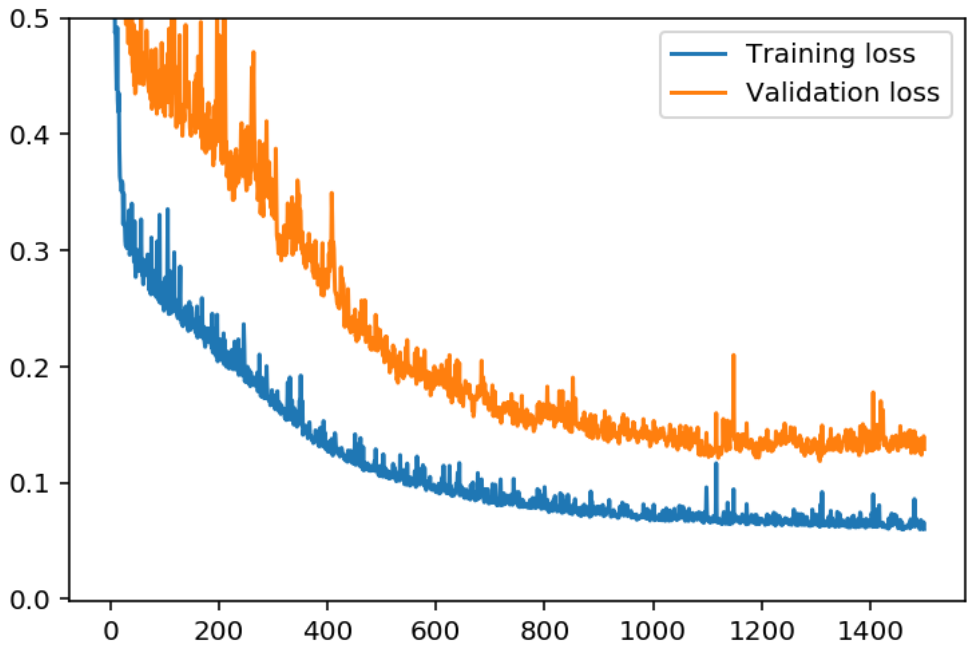
The learning rate scales the size of weight updates. If this is too big, the weights tend to explode and the network fails to fit the data. A good choice to start at is 0.1. If the network has problems fitting the data, try reducing the learning rate.
Sometimes, the network doesn't converge when it is around or above 0.1. The weight update steps are too large with this learning rate and the weights end up not converging.
You can try values like 0.03, 0.05, 0.01, 0.009.

On the other side, you should be using learning rates that are just low enough to get the network to converge, there really isn't a benefit in a really low learning rate. I'd say *stay larger than 0.001.*
You have plotted many graphs with learning rate 0.001. Using a very low learning rate can stuck the model at local minima, so it should be tuned properly.
Note that the lower the learning rate, the smaller the steps are in the weight updates and the longer it takes for the neural network to converge.

For further reading, you can read this discussion on Quora about how the value of learning rate influences the behavior of a model

You are one of the few students who tried to go so deeply into the problem. Keep working hard and smart.

The resulting graph should look like this:



Hope it helps!

RESUBMIT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH

Student FAQ