Predicting Bike-Sharing Patterns

- Review
- Code Review
- History

Meets Specifications

Perfect first submission! 

You implemented the forward and backward pass correctly. Also the tuning of the hyperparameters was very well done. We are looking forward to your next submissions in the Nanodegree!

Code Functionality

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

All tests passed 

The sigmoid activation function is implemented correctly

Forward Pass

The forward pass is correctly implemented for the network's training.

Perfect implementation of the forward pass!

Note that the output unit has no activation function because we are using it for regression.

The run method correctly produces the desired regression output for the neural network.

Backward Pass

The network correctly implements the backward pass for each batch, correctly updating the weight change.

Also the backward pass is perfectly implemented 

Updates to both the input-to-hidden and hidden-to-output weights are implemented correctly.

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.

Your choice for the number of epochs is good:

1. The validation loss has stabilized at the end of the training
2. The validation loss is not increasing yet, this would have been a sign of overfitting

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 hidden units you choose is high enough for a good prediction accuracy. At the same time it is not too high, this could have resulted in overfitting.

A good rule of thumb is 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

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

Perfect choice for the learning rate. It doesn't take too long for the network to learn. On the other hand the learning rate is not too high, this could result in validation loss fluctuating a lot. This would indicate the gradient descent having trouble finding the minimum loss.

Be sure to check out the following link that discusses the right choice of learning rate: http://cs231n.github.io/neural-networks-3/#baby

The number of output nodes is properly selected to solve the desired problem.

You set the number of output nodes to 1, this is correct since we are trying to predict just one variable.

The training loss is below 0.09 and the validation loss is below 0.18.

```
Progress: 100.0% ... Training loss: 0.073 ... Validation loss: 0.144
```

Perfect!