UDACITY

< Return to Classroom

DISCUSS ON STUDENT HUB

Predicting Bike-Sharing Patterns

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Dear student

Great work on this project! You've clearly understood the material from the tutorials and you've done a great job applying it to this real-world dataset. Congratulations on passing quickly and good luck with the next section of the course.

Cheers!

Code Functionality

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

```
<unittest.runner.TextTestResult run=5 errors=0 failures=0>
```

Looks good!

The sigmoid activation function is implemented correctly

```
def sigmoid(x):
            return 1 / (1+np.exp(-x))
```

Nice work implementing an activation function! Another useful function that we could apply here is the tanh activation function:

https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

Forward Pass

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

```
hidden_inputs = np.dot(X,self.weights_input_to_hidden)

hidden_outputs = self.activation_function(hidden_inputs)
```

Nice work!

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

```
final_outputs = final_inputs
```

You got it! Since this is a regression, there's no need to apply the activation function at this step.

Backward Pass

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

```
hidden_error = np.dot(self.weights_hidden_to_output, error)
```

```
output_error_term = error
```

Nicely done!

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

```
hidden_error_term = hidden_error * hidden_outputs * (1-hidden_outputs)

delta_weights_i_h += hidden_error_term * X[:,None]

delta_weights_h_o += output_error_term * hidden_outputs[:,None]
```

Great work!

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.

3000 epochs is pretty reasonable. If you try higher values (>10,000), you'll see that the validation loss starts to increase which indicates that the model is starting to overfit the data. You've definitely found the sweet spot.

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.

For the number of hidden units, we should probably pick a value that's 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.

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

```
learning_rate = 0.8
```

I'm marking this as passing since the network converged fairly efficiently. However, I'd note that 0.8 is extremely high as a learning rate setting. At this learning rate, the model is discarding 80% of the previously learned information each time the weights are updated. Usually, we want to set the learning rate much lower (0.01-0.001).

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

```
output_nodes = 1
```

You got it!

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

> Progress: 100.0% ... Training loss: 0.068 ... Validation loss: 0.147

The model's performance meets all specifications. Great work!

DOWNLOAD PROJECT

RETURN TO PATH