UDACITY

‹ Back to Deep Learning

# Predicting Bike-Sharing Patterns

REVIEW

CODE REVIEW

HISTORY

## Meets Specifications

Dear Excellent Student,
Congratulations!!! You made it. 🙌
By carefully going through the project, it shows a lot of effort, diligence and above all, determination. Excellent work! Your submission has passed all the rubric of this project. All the efforts you put in to complete the project are very much appreciated and it was my pleasure reviewing this wonderfully implemented project. You should be proud of yourself because success is no accident. It is hard work, perseverance, learning, studying, sacrifice and most of all, love of what you are doing or learning to do. Remember, practice makes perfect. So, keep practicing on these projects and I wish you all the best.

### Code Functionality

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

Awesome, all code cells run in python3 without failing and all unit tests pass.

**The sigmoid activation function is implemented correctly**

Good job. The sigmoid activation function is well implemented which is `lambda x: 1/(1+np.exp(-x))` It is well defined and this helps to get a shorter and more generic code.

## Forward Pass

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

Well done implementing the forward pass for the network’s training.
`hidden_inputs = np.dot(X,self.weights_input_to_hidden)`

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

Nice work, the run function is well implemented and returns the final outputs.

## Backward Pass

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

This meets the specifications.

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

Nice implementation of the weights. The self.weights_hidden_to_output and self.weights_input_to_hidden are well implemented.

## 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.**

2500 epochs are used to train the network making our predictions accurate. However, we don't want too many epochs since an increase in this value makes the training specific which will make the test fail hence having overfitting. We need to be very careful in the choice of the number of epochs to use when training a network.

Basically, we 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.

**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 nodes chosen is fine. A good rule of thumb is to use no more than twice the number of input units while trying to keep this number as low as possible so the network can generalize, so probably at least 8 units. A good tip is to always keep in mind not to overload a hidden layer with more units than necessary

## Suggestions

The number of hidden nodes you should have is based on a complex relationship between

- Number of input and output nodes
- Amount of training data available
- Complexity of the function that is trying to be learned
- The training algorithm

To minimize the error and have a trained network that generalizes well, you need to pick an optimal number of hidden layers, as well as nodes in each hidden layer.

- Too few nodes will lead to a high error for your system as the predictive factors might be too complex for a small number of nodes to capture
- Too many nodes will overfit to your training data and not generalize well You can read more on this from this discussion found on Quora How do I decide the number of nodes in a hidden layer of a neural network. You can also check out this how to choose the number of hidden nodes found on StackExchange.

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

Good job, the learning rate chosen is fine and still time efficient.

## Suggestion

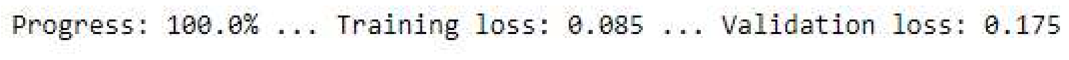
1 might be a little bit high here. To get ideal results, I will recommend using a learning rate between 0.2 and 0.4 which have been reported to provide excellent results.

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

The number of output nodes is good. 1 is a good choice.

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

As required, the training loss should be below 0.09 and validation loss below 0.18 which is the case as we can see from the results below. Well done.



```
Progress: 100.0% ... Training loss: 0.085 ... Validation loss: 0.175
```

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review