UDACITY

‹ Return to Classroom

Predicting Bike-Sharing Patterns

REVIEW

CODE REVIEW

HISTORY

Requires Changes

2 specifications require changes

You're almost there! A couple of tweaks to the hyperparameters so that your plot coverages efficiently and training loss is less than 0.09 while validation loss is also less than 0.18. I would suggest trying to manipulate the three factors that affect output of the network (`epochs` , `learning_rate` and `hidden_nodes`) in order to better understand the role they play in generalization vs. over-fitting.

Code Functionality

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

Correct!

The sigmoid activation function is implemented correctly

Correct!

Forward Pass

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

Correct!

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

Correct!

Backward Pass

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

Correct!

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

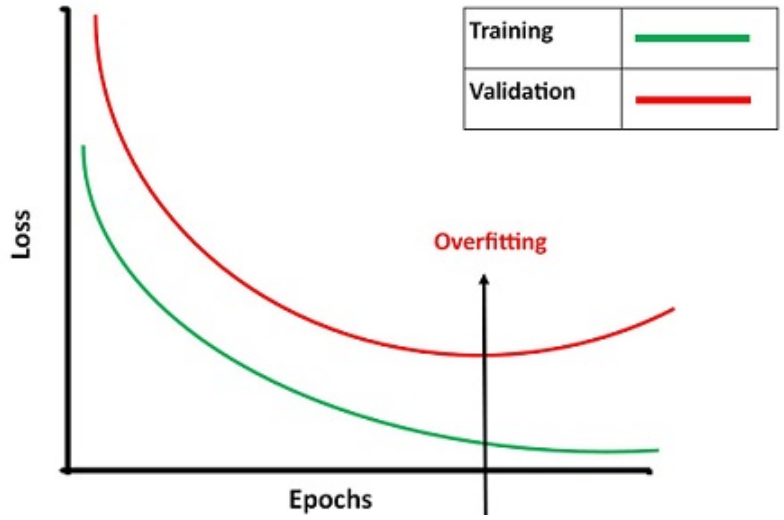
Correct!

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.

Plot of the training and validation loss needs some improvement. Since the network has not converged, there are couple things happening here. First, there are not enough iterations to train the network. Second, the network might be too simple (too few hidden nodes). We will be able to predict better with more hidden nodes but it costs computational resources because the network is harder to train. Third, the learning rate is too small. You could try a couple of parameter tweaks:

- Increase the number of iterations to few more thousands to (6,000 - 8,000) train the network better.
- Increase the number of hidden nodes to (8 - 12) train the network more efficiently.
- Increase learning rate to (0.6 - 0.9) speed up the process.



Another small suggestion for future visualizations is to always include descriptive axis labels. This functionality is available from within matplotlib. You can add text, y axis labels, x axis labels, titles, subtitles and annotations. For example to add a title as well as x axis and y axis labels to the plot:

```
fooplot.set_title('axes title')
fooplot.set_xlabel('xlabel')
fooplot.set_ylabel('ylabel')
```

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.

There are no set criteria for determining the number of hidden units. It is a good rule of thumb to be no more than twice the number of inputs and enough to generalize the problem space well. A good starting point is half way in between the number of inputs and output units.

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 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 over-fit to your training data and not generalize well.

It is important to strike a balance when trying to determine the number of nodes and since there are no concrete rules it can sometimes come down to trial and error. Having a solid data pipeline is key to being able to quickly experiment and optimize outputs.

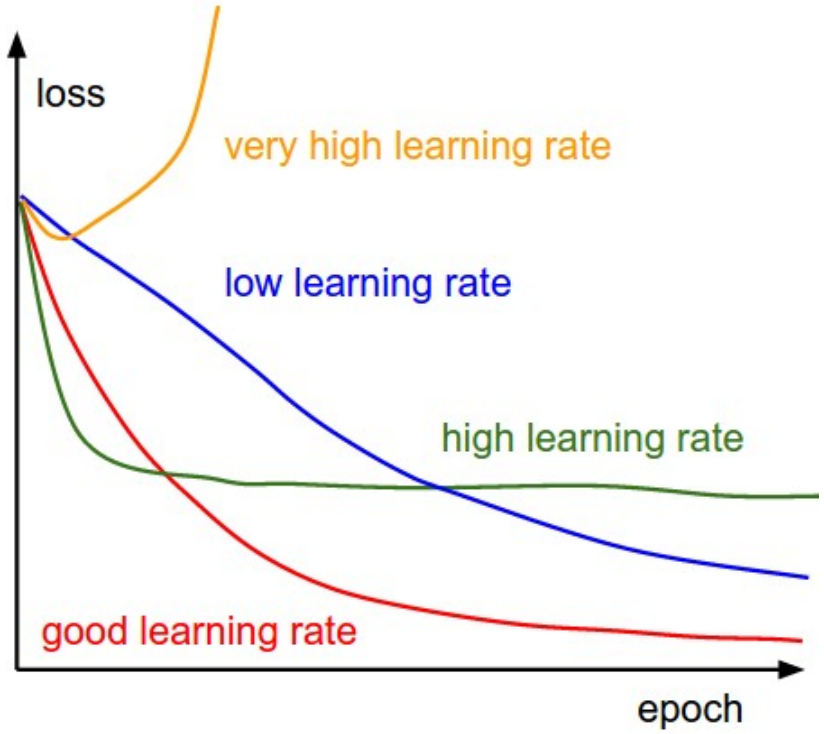
There's a good answer here for how to decide the number of nodes in the hidden layer. https://www.quora.com/How-do-I-decide-the-number-of-nodes-in-a-hidden-layer-of-a-neural-network

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

The quotient of the learning rate / the number of records should end up around 0.01 (i.e. self.lr / n_records ~ 0.01), although, much like with the number of hidden units, it is results that matter as far as meeting specifications is concerned. (The network converges in a reasonable amount of time.)

In your case `0.02 / 128 = 0.00015`.Try adjusting the hyperparameters to bring the value closer to 0.01.

Sometimes, the network doesn't converge when that quotient is around or above 0.1. The weight update steps are too large with this learning rate and the weights end up not converging.



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

Correct!

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

try implementing some of my suggestions from the above sections to get these values to within the required specifications

RESUBMIT

DOWNLOAD PROJECT

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH