UDACITY

‹ Return to "Deep Learning" in the classroom

DISCUSS ON STUDENT HUB ›

Predicting Bike-Sharing Patterns

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Great work on the project! I hope my links and hints below are helpful. 👍

On the last question at the bottom of the notebook: The model predicts well except for Thanksgiving and Christmas. We don't have enough examples of holidays for the network to learn about them (and the `holiday` variable is incorrectly labeled for the days around Christmas and Thanksgiving). For dealing with holidays with the small amount of training data available over the holiday time periods, we could use a RNN, time-lagged features or fix the variable for holiday/not holiday (they only label the 22nd and 25th of December a holiday). There's also the possibility of oversampling the data on Christmas and Thanskgiving holidays. The network also over-predicts for Thanksgiving if you look back further into the validation set.

Code Functionality

✓

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

✓

The sigmoid activation function is implemented correctly

Nice! Scipy also has a function for this.

Forward Pass

✓

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

✓

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.

Good work! Here's another explanation of backpropagation if it still seems confusing; it helped me to understand it better.

✓

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.

Nice job! One way to make it easier to see if the losses are increasing is by adding gridlines:

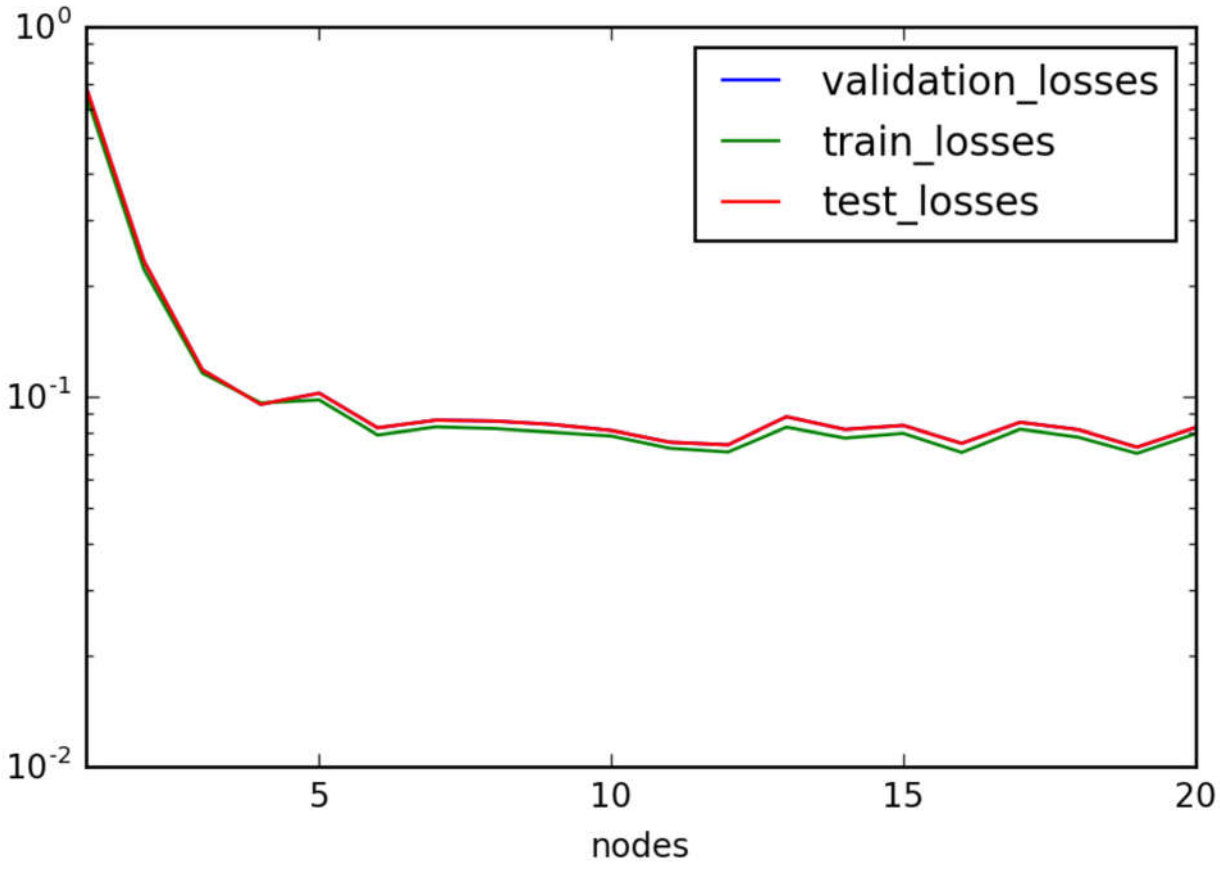
```
plt.plot(losses['train'], label='Training loss')
plt.plot(losses['validation'], label='Validation loss')
ax = plt.gca()
ax.grid(True)
plt.legend()
plt.ylim(ymax=0.5)
```

Here's an example of where to stop during training (the 'early stopping' section with the plot; the 'Early stopping point' is the ideal place to stop training).

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.

Great job tuning hidden units! Udacity suggests using something above 8, but it's possible to go even lower (see the plot below). One suggested rule of thumb for number of hidden units is half way between the number of input and output units. Another rule of thumb is to stick to numbers less than twice the number of input units (112 in this case). Here's another resource on the subject.

If you're curious, try different numbers of hidden units, recording the validation error for each, and plot the number of hidden units on the x-axis and validation loss on the y-axis (or try it on another neural net project). That way you can *quantitatively* judge what the optimum number of hidden units is. For reproducibility, consider setting the random seed. You should end up seeing that the losses flatten out after around 5/6 hidden nodes, which is much smaller than the rule of thumb value of 28. Most likely this is because all the dummy variables are not independent of each other. In actuality, there's only about 13 independent variables.



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

Good job tuning the learning rate! Somewhere above 1 is the upper limit where things start to break (the weight update swings out of control and returns `nan`s). Below 0.001 is too small because it will take a long time to converge. Usually 0.01 is a good starting point, and depending on your situation, you can increase it until the losses are too jumpy or it doesn't converge (in other words, the training and validation loss never settle out to a

flat, low number), or the validation error is increased compared with 0.01. Here's more info on tuning the learning rate.

Here's more info on tuning hyperparameters in general.

✓

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

✓

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

Got it!

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review