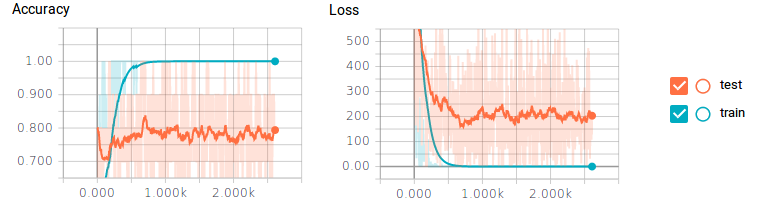
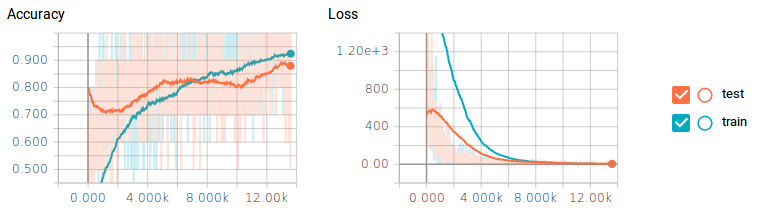
**Measure performance**

Let’s check out the scalar history for our accuracy and loss.

Figure 7. Image courtesy of Justin Francis.

You may be able to tell that we have a huge problem. For our training data, the classifier is getting 100% accuracy and 0 loss, but our test data is only achieving 80% at best and still getting a lot of loss. This is your obvious sign of overfitting—some classic symptoms include not enough training data, or having too many neurons.

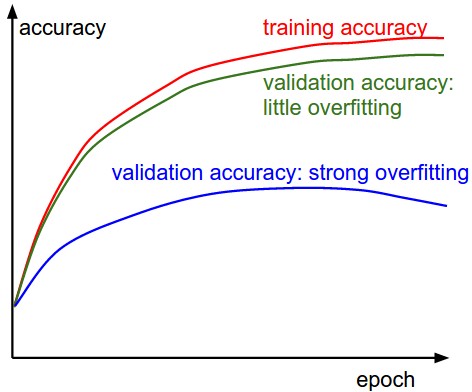
We could create more training data by resizing, scaling, and rotating our training data, but a much easier approach is to add dropout to the output of our pooling and fully connected layers. This will make every training step completely cut, or drop out, a percentage of neurons randomly, in a layer. This will force our classifier to only train small sets of neurons at a time, rather than the whole set. This allows neurons to specialize in specific tasks, rather than all neurons generalizing together. Dropping out 80% of our convolutional layers and 50% of our fully connected layers gives some amazing results.

Figure 8. Image courtesy of Justin Francis

Just by dropping off neurons, we were able to achieve just under 90% on our test data—that is almost a 10% increase in performance! One drawback is that the classifier took about 6x longer to train.

#### **Train/Val accuracy**

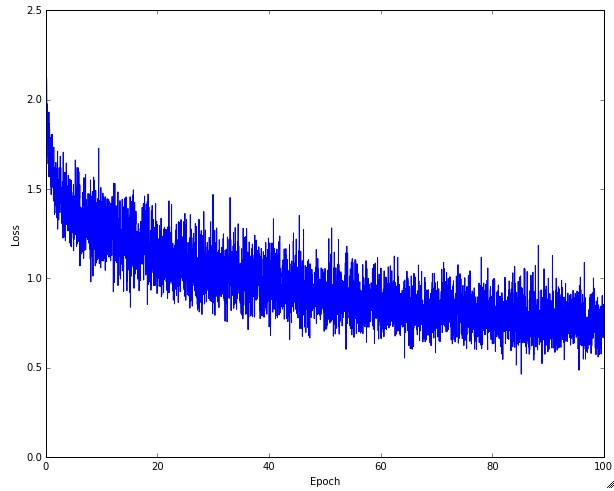
The second important quantity to track while training a classifier is the validation/training accuracy. This plot can give you valuable insights into the amount of overfitting in your model:



The gap between the training and validation accuracy indicates the amount of overfitting. Two possible cases are shown in the diagram on the left. The blue validation error curve shows very small validation accuracy compared to the training accuracy, indicating strong overfitting (note, it's possible for the validation accuracy to even start to go down after some point). When you see this in practice you probably want to increase regularization (stronger L2 weight penalty, more dropout, etc.) or collect more data. The other possible case is when the validation accuracy tracks the training accuracy fairly well. This case indicates that your model capacity is not high enough: make the model larger by increasing the number of parameters.

#### **Loss function**

The first quantity that is useful to track during training is the loss, as it is evaluated on the individual batches during the forward pass. Below is a cartoon diagram showing the loss over time, and especially what the shape might tell you about the learning rate:



**Left:** A cartoon depicting the effects of different learning rates. With low learning rates the improvements will be linear. With high learning rates they will start to look more exponential. Higher learning rates will decay the loss faster, but they get stuck at worse values of loss (green line). This is because there is too much "energy" in the optimization and the parameters are bouncing around chaotically, unable to settle in a nice spot in the optimization landscape. **Right:** An example of a typical loss function over time, while training a small network on CIFAR-10 dataset. This loss function looks reasonable (it might indicate a slightly too small learning rate based on its speed of decay, but it's hard to say), and also indicates that the batch size might be a little too low (since the cost is a little too noisy).