Implementation 3

CS434, 5-15-2018  
Team: Rex Henzie, Benjamin Richards, and Michael Giovannoni

Note: We chose to normalize our data to a mean and standard deviation of .5 using transforms.Normalize.

## Part 1: Two-layer Sigmoid

## Figure 1: Epochs vs Negative Log Loss

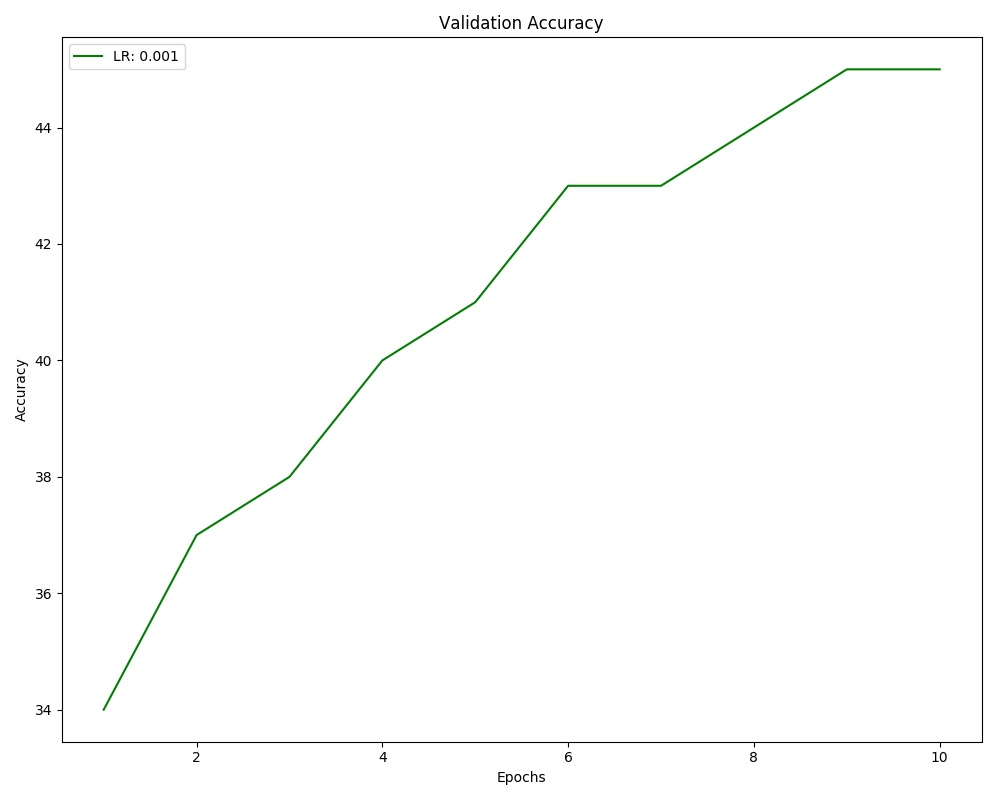
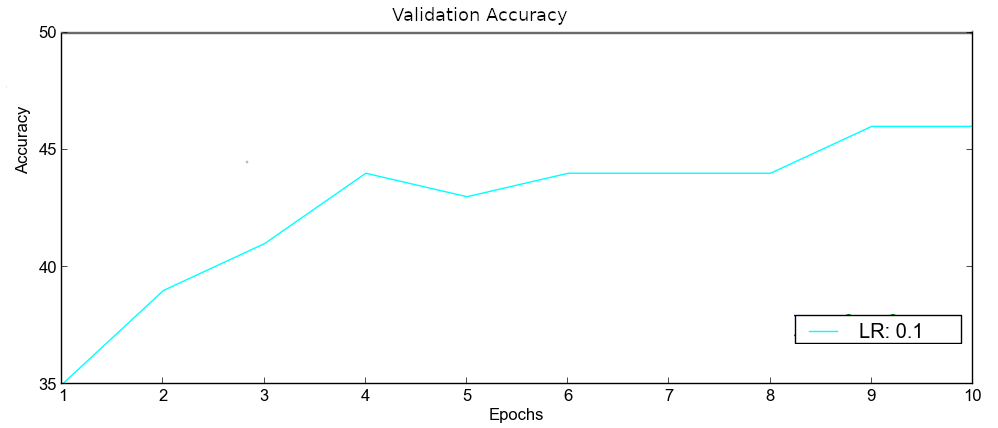
## Figure 2: Epochs vs Validation Accuracy

|  |  |
| --- | --- |
| Test Accuracy | Learning Rate |
| 47% | 0.1 |

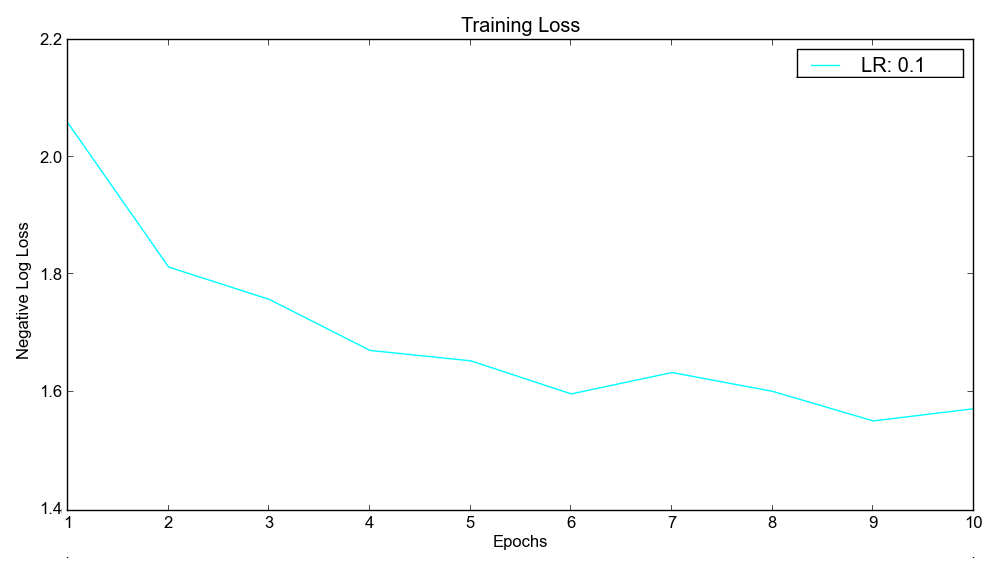
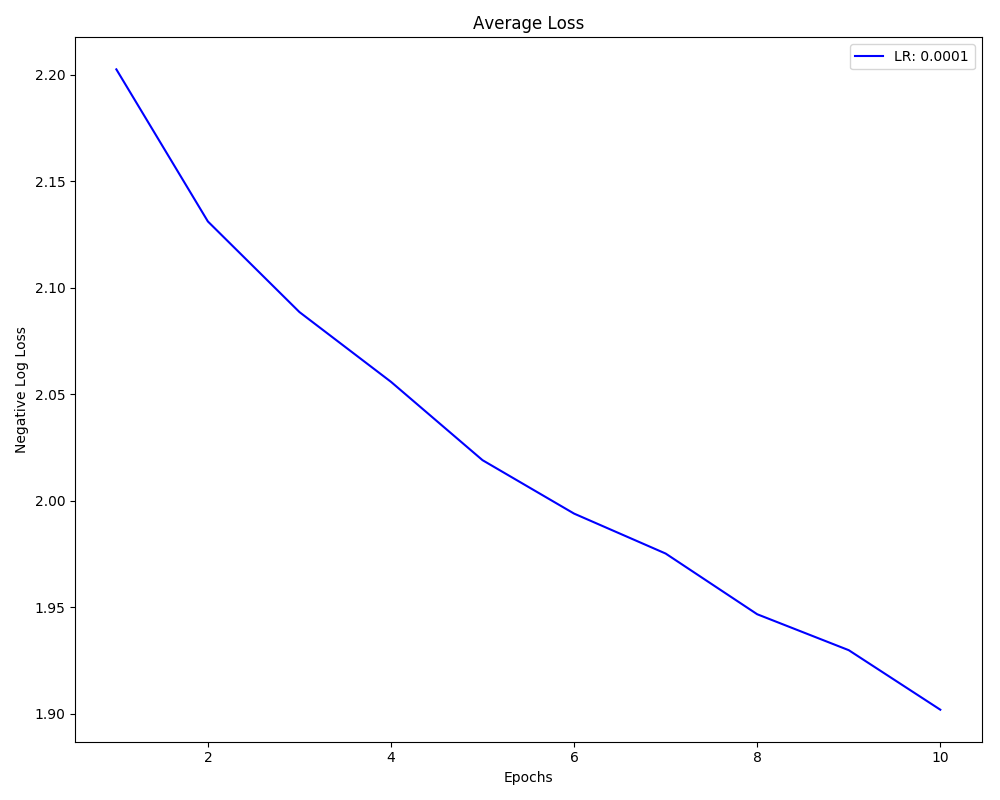
Table 1: Learning Rate vs Dropout And Test Accuracy

The reason for choosing the Learning rate of 0.1 was because it had the highest validation accuracy. We decided to stop training at 10 epochs.

## Part 2: ReLU



## Figure 3: Epochs vs Loss using the ReLU activation function



## 

## Figure 4: Epochs vs Training Loss using the ReLU activation function

|  |  |
| --- | --- |
| Test Accuracy | Learning Rate |
| 50% | 0.01 |

Table 2: Learning Rate vs Dropout And Test Accuracy

The learning rate I choose using ReLU is: 0.01 because the accuracy was the best while also having a really low drop rate. The way we decided to stop training was again, stopping after 10 epochs.

## Part 3: Experiments

**Dropout rate**

Momentum = .5, lr = 0.01, decay = 0

|  |  |
| --- | --- |
| Dropout | Test accuracy |
| .01 | 43% |
| .2 | 42% |
| .4 | 42% |
| .8 | 36% |

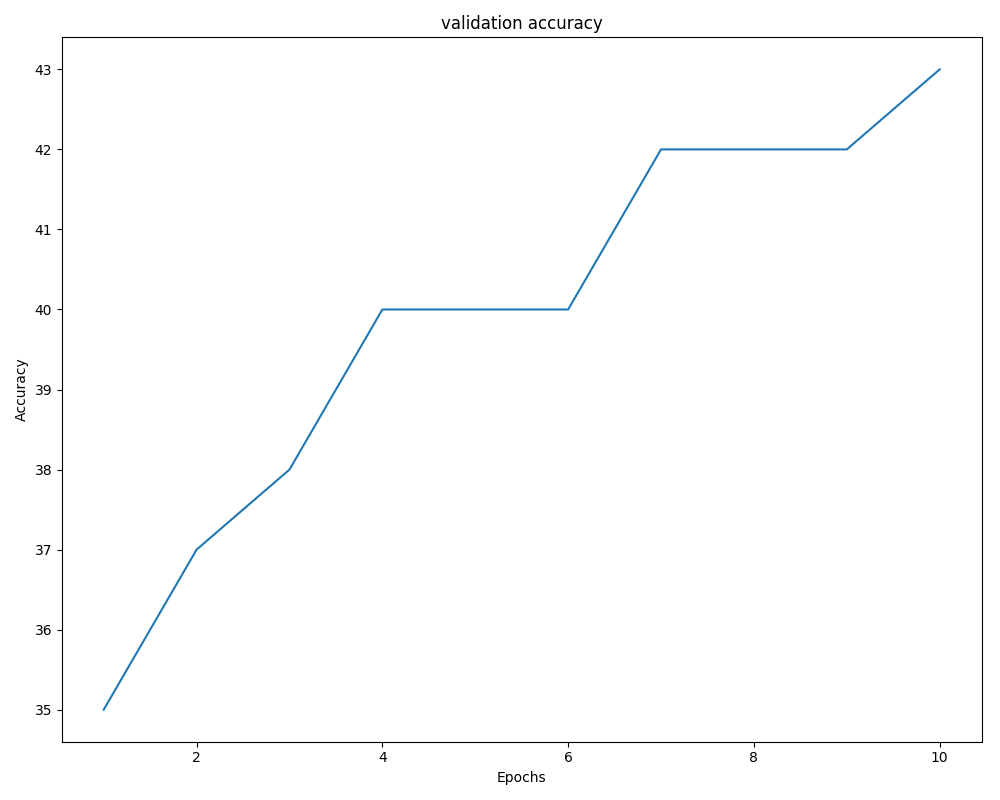
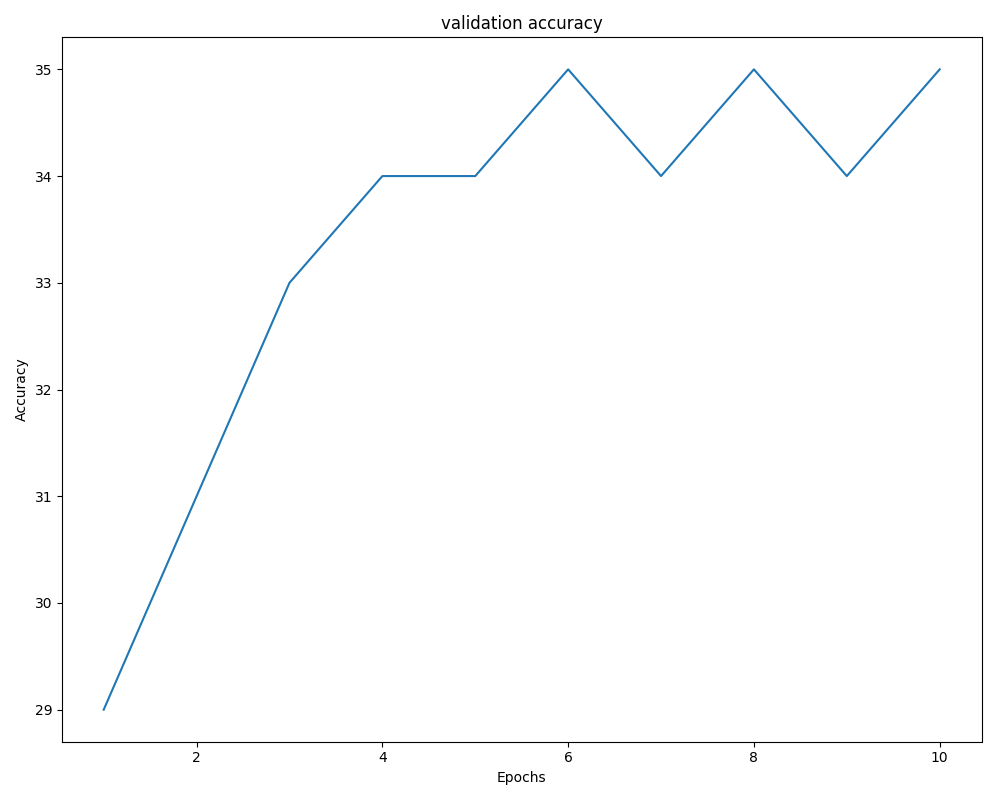


Figure 5. Validation accuracy with a dropout rate of 2 Figure 6.Validation accuracy with a dropout of .8

As we increase the dropout rate we see our test accuracy decrease. Our results suggest that a smaller dropout rate in the range of .2 to .4 is effective at protecting against overfitting without significantly reducing the test accuracy.

**Momentum**

Lr = 0.01, decay = 0, dropout = .2

|  |  |
| --- | --- |
| Momentum | Test accuracy |
| .2 | 41% |
| .5 | 42% |
| .8 | 46% |
| .99 | 39% |

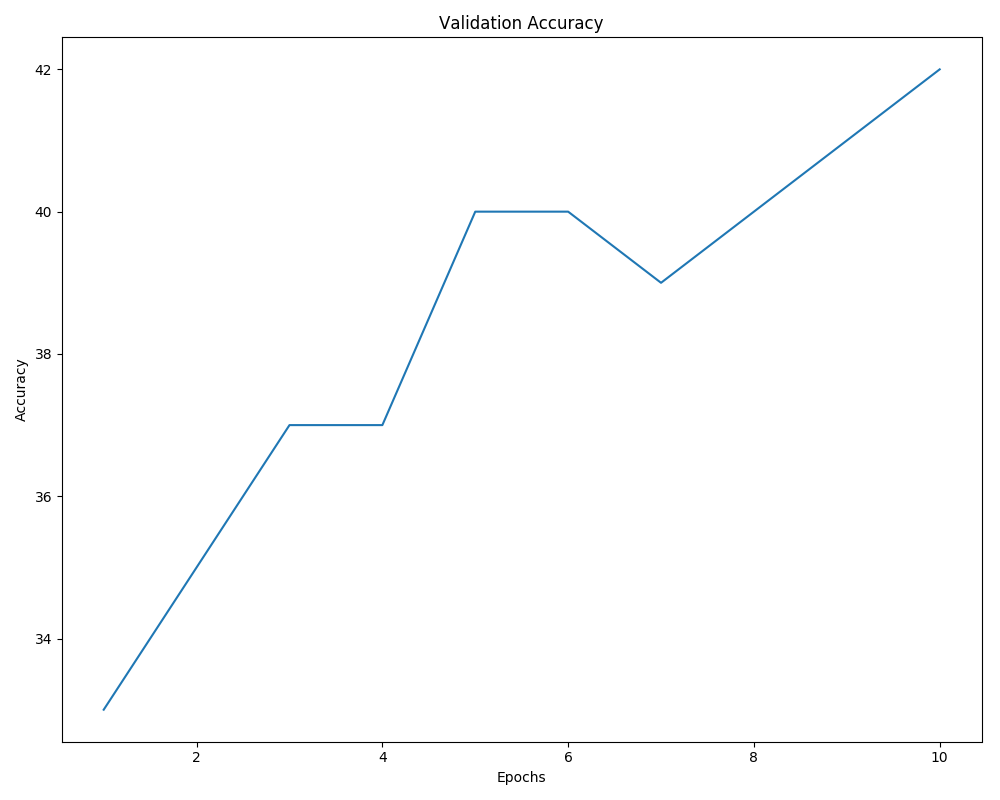


Figure 7. Validation accuracy with a momentum of .2

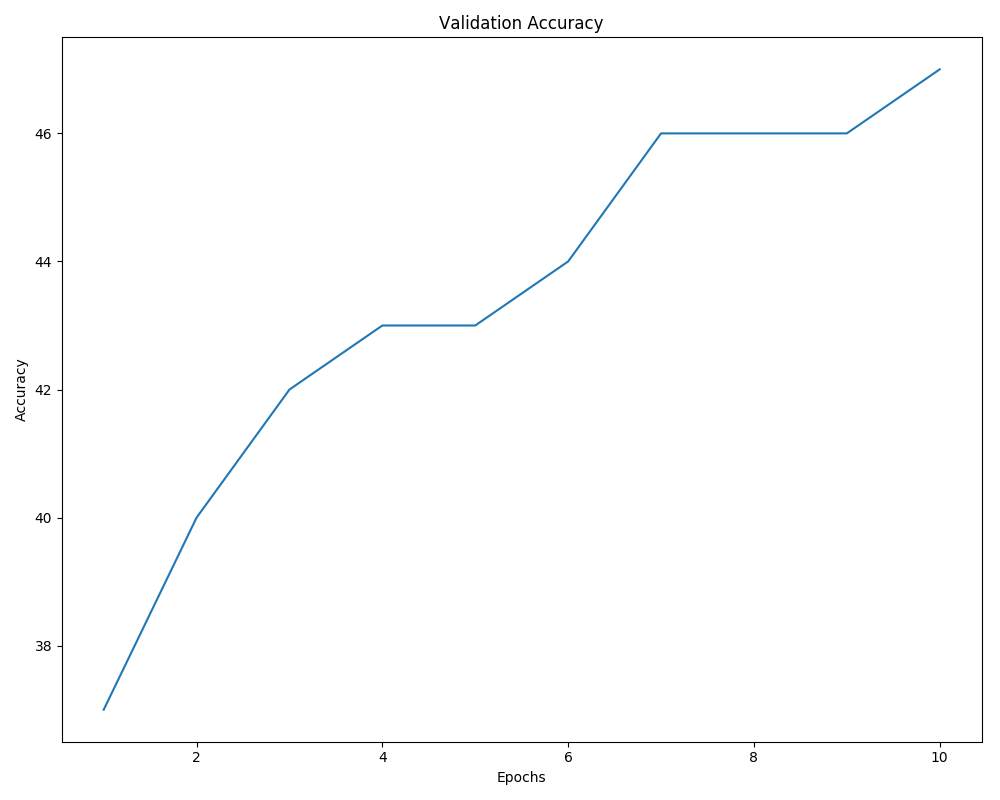


Figure 8. Validation accuracy with a momentum of .8

By increasing momentum we are able to achieve more accurate results. Momentum works by adding part of the previous weight to the new weight which serves to increase the size of the steps the weight takes towards the minimum. The purpose of this is to help avoid getting stuck in local minimums and push the model towards the global minimum. In theory, a large value for momentum will cause the network to converge faster but we did not observe that behavior in our experimentation. With a high enough value we see the test accuracy reduced as the model takes too large of a step and goes rushing past the global minimum.

**Decay**

Lr = 0.01, momentum = .5 , dropout = .2

|  |  |
| --- | --- |
| Decay | Test accuracy |
| 0 | 42% |
| .001 | 42% |
| .01 | 39% |
| .1 | 24% |
| .5 | 10% |

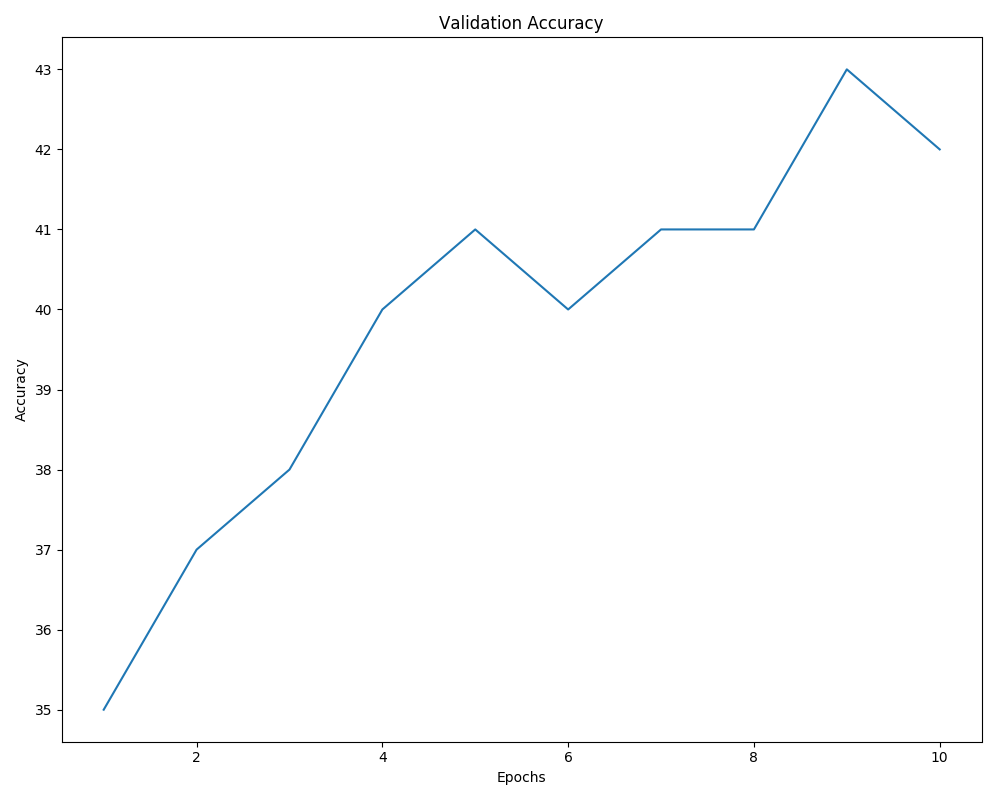


Figure 9. Validation accuracy with a decay of 0.001

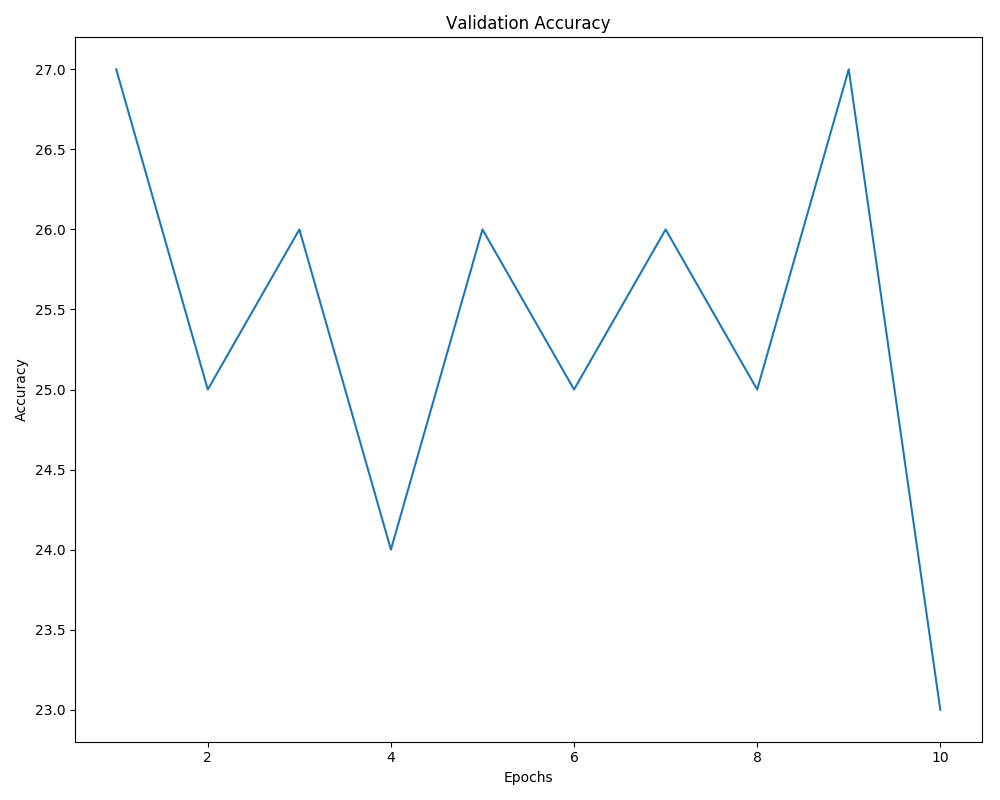


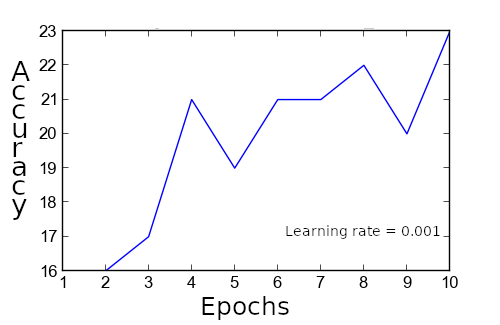
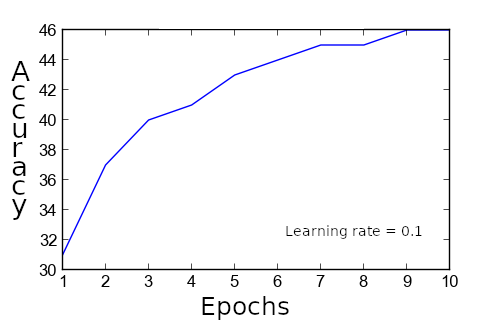
Figure 10. Validation accuracy with a decay of .1

Weight decay is a method of regularization. As expected, when we increase decay to a large number we see our accuracy drop drastically. We also observe that the validation accuracy rises and drops frequently from one epoch to the next giving inconsistent results depending on which epoch is stopped at.

## Part 4: Alter Structure

**Overview**

For part 4 the most significant change we did was to run the data through an additional layer of the neural network. This is something we also did very briefly while setting of part 1 and 2 when we were using nn examples from outside sources. Using one nn example taken from pytorch’s documentation we actually got our accuracy above 50% nearing 60%. When we ran the specified 3-layer nn we also saw high accuracy for high learning rates. Eventually we decided to go with a learning rate of 0.1 (With the Sigmoid activation function) which yielded higher accuracy and more consistent growth.



**Comparing with problem 1**

We notice that for a high learning rate it converges a bit but overall keeps increasing. Also worth noting is that the loss for this learning rate started in the 2 range before spending most of the epochs in the upper half of 1 (In contrast learning\_rate=0.001’s loss hovered at 2.25). We found that running the three-layer neural network was significantly slower then the two-layer neural network (By at least 25%) but some of that may have to do with traffic on the server. We do think networks with more layers are more effective and customizable especially regarding complex data such as the images we used and as evidenced by the previously mentioned network which got nearly 60% accuracy. But regarding our three-layer network, in comparison with the two-layer network we see the convergence behaviour converges a bit more then the function used in problem 1 did (Which may imply that this model tends to overfit more). Overall it doesn’t reach as high of accuracies as problem 1’s function did. So regarding these things I may want to initially say that problem 1 is better. But that’s only looking at a short scope. If we extended epochs or tweaked this multi-layer neural network a bit more I’m confident we could beat the more simplistic model of problem 1.

**Comparing learning rates**

The reason l\_rate=0.001 is so erratic may have to do with the second hidden layer causing confusion when the l\_rate is low. However l\_rate=0.1’s may be overfitting depending on further epochs considering the overall derivative of the function is decreasing. l\_rate=0.001 may look like it’s also increasing as well, but when you look at the scope of the Y-axis you see it’s not really increasing that much, especially compared to l\_rate=0.1.