Part 1

L2

```
for c in c_vals:
    print(scores_l2.loc[scores_l2["c"]==c]["test"].mean(),c)
```

```
92.43629695082134 0.1
91.77242154289213 11.2
91.60357349252527 22.3
91.66036969166791 33.4
91.49337981879567 44.5
91.38288258122662 55.6
91.3825669620212 66.7
91.27237328829162 77.8
91.10753812345645 88.9
90.94163414332017 100.0
```

L1

```
for c in c_vals_l1:
    print(scores_l1.loc[scores_l1["c"]==c]["test"].mean(),c)
```

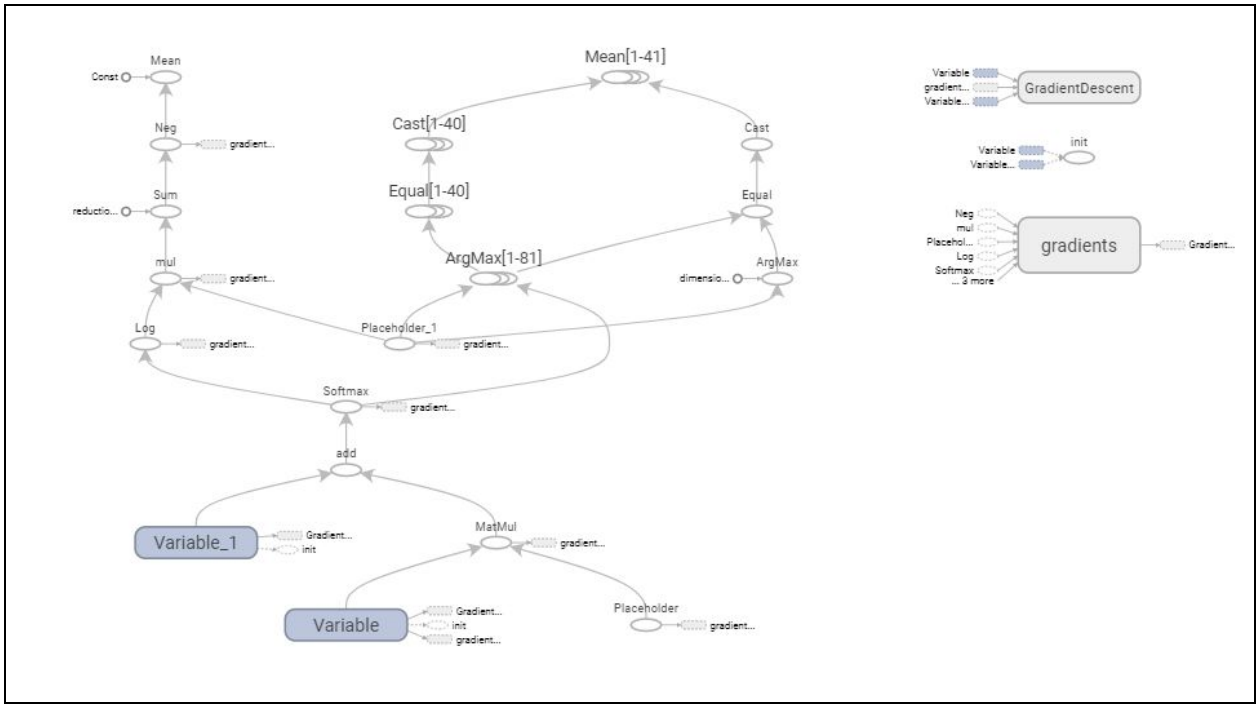
```
93.21619361784721 0.1
91.27502170504022 111.2
91.49740978176223 222.3
91.33011282840201 333.4
91.44016043371487 444.5
91.32996675998528 555.6
91.16375921600952 666.7
91.3318299595488 777.8
91.33074055016434 888.9
91.21931139863354 1000.0
```

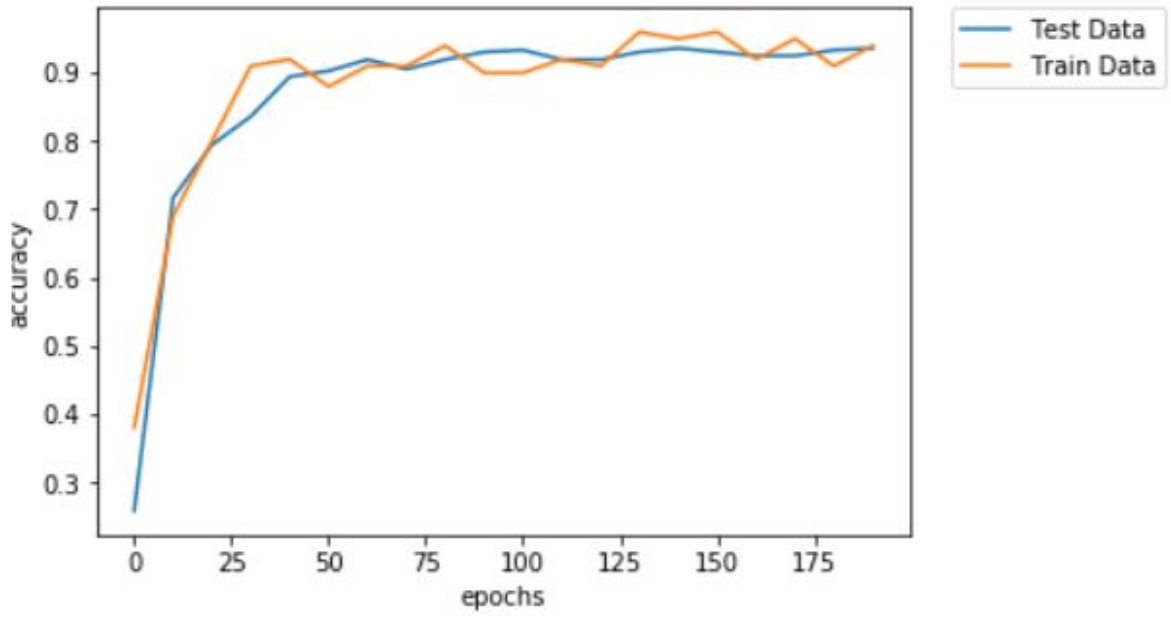
For both C=0.1 is optimal.

L1	L2
Logistic Regression accuracy of train data: 99.03 % [[151 0 0 0 0 0 0 0 0 0] [0 146 0 0 0 0 0 0 0 1] [0 0 141 0 0 0 0 0 0 0] [0 0 0 153 0 0 0 0 1 0] [0 1 0 0 149 0 0 0 1 0] [0 0 0 0 0 142 0 0 0 0] [0 0 0 0 0 0 137 0 0 0] [0 0 0 0 0 0 0 139 0 1] [0 5 0 1 0 0 0 0 129 0] [0 0 0 1 0 0 0 0 2 136]] Logistic Regression accuracy of test data: 95.83 % [[27 0 0 0 0 0 0 0 0 0] [0 33 0 0 0 0 1 0 1 0] [0 0 35 1 0 0 0 0 0 0] [0 0 0 29 0 0 0 0 0 0] [0 0 0 0 30 0 0 0 0 0] [0 0 0 0 0 39 0 0 0 1] [0 1 0 0 0 0 43 0 0 0] [0 1 0 0 1 0 0 37 0 0] [0 2 1 0 0 0 0 0 36 0] [0 0 0 1 0 1 0 0 3 36]]	Logistic Regression accuracy of train data: 98.05 % [[151 0 0 0 0 0 0 0 0 0] [0 145 0 0 0 0 0 0 1 1] [0 0 141 0 0 0 0 0 0 0] [0 0 0 149 0 0 0 1 3 1] [0 2 0 0 148 0 0 0 1 0] [0 0 0 0 1 140 1 0 0 0] [0 0 0 0 0 1 135 0 1 0] [0 0 0 0 0 0 0 138 1 1] [0 6 0 1 0 1 0 0 127 0] [0 0 0 1 0 1 0 0 2 135]] Logistic Regression accuracy of test data: 95.83 % [[27 0 0 0 0 0 0 0 0 0] [0 33 0 0 0 0 1 0 1 0] [1 0 35 0 0 0 0 0 0 0] [0 0 0 29 0 0 0 0 0 0] [0 0 0 0 29 0 0 1 0 0] [0 0 0 0 0 39 0 0 0 1] [0 0 0 0 0 0 44 0 0 0] [0 0 0 0 1 0 0 38 0 0] [0 2 1 0 0 0 0 0 36 0] [0 1 0 0 0 1 0 1 3 35]]

Part 2

Multiclass Logistic

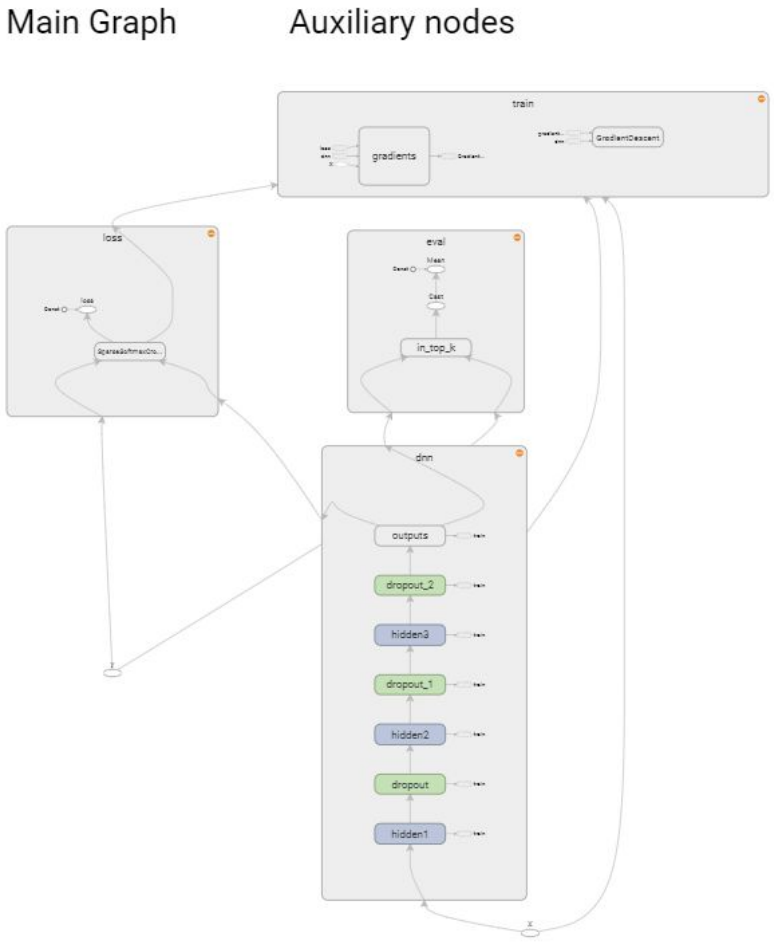


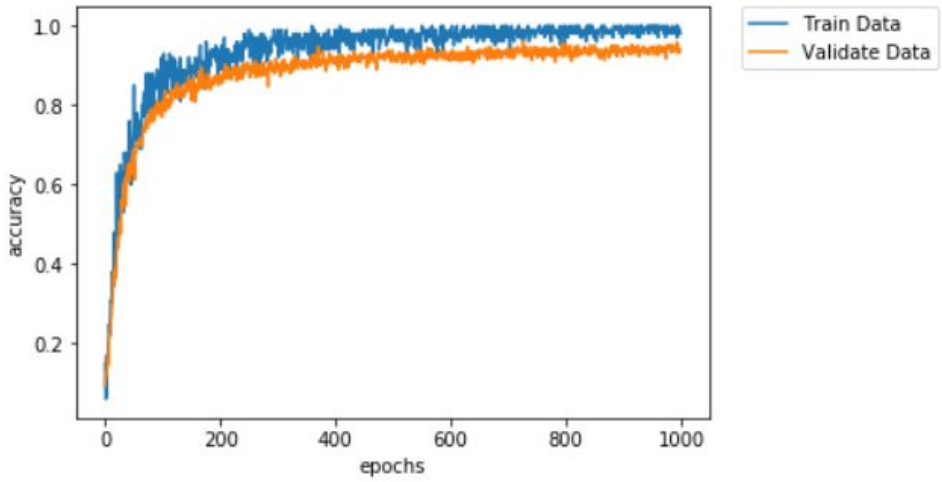



```
acc()
```

```
0.93611109
```

DNN






```
997 Train accuracy: 0.99 Val accuracy: 0.936111
998 Train accuracy: 0.99 Val accuracy: 0.933333
999 Train accuracy: 0.98 Val accuracy: 0.938889
```

The DNN is a much more complex model which has resulted in only slightly better accuracy vs the multiclass logistic 93.9% vs 93.6%.

The DNN had 8 variables and 3 layers vs the multiclass logistic's 2. Hence, it was slower to run/train, but perhaps with more tuning/better batch selection it would show an improvement on the logistic results.

Part 3

Neuron Saturation

Many activation functions will compress the output to a small range, either -1 to 1 or 0 to 1. When this occurs, the output may be closer to those limits, i.e. -1 or 1. If we were to look at the sigmoid activation function we can see that the derivative at these areas is close to 0. Therefore when when exposed to a change in input data, these neurons will update slowly due to these very small gradients. This saturation occurs when overfitting to train data; as the net becomes more confident in its response to the same data, it will react slowly to new data. This is analytically presented through neuron saturation and the low gradients of neuron outputs close to the limits of the activation function.

Batch Normalization

Batch normalization mitigates the covariate shift which occurs when the inputs (and therefore their distribution) changes between the test and training data.

In a neural net with multiple layers there are several parameters which need to be tuned. One of the ways this is accomplished is through back propagation. As the correct answer is understood, the parameters are updated in reverse through the neural net. However, one of the issues of this process is that when the parameters change, the distributions of the outputs change as well. This makes it more difficult for proceeding layers to adapt and slows down training.

Batch normalization normalizes the distribution of each of neurons outputs to have a mean of 0 and a variance of 1. Mathematically, to implement this we subtract the batch mean from the value and then divide by the batch standard deviation. One issue with this is that this forced normalization is that it diminishes the effect of the layer. Therefore a trainable scale and shift parameter are also added for batch normalization.

For a neural net, each layer is transformed and modified to accept the transformed input. This modified neural network is then trained to optimize for the scale and shift parameters. A new inferenced network is created with these parameters. Multiple training mini-batches are then processed through this net. Once complete, the initial batch normalization with the scale and shift parameters are undone with modifications to the scale and shift parameters through the expected value and variance of the mini-batches that the net was just processed with.

There are several advantages of batch normalization. Firstly, the model learns faster when the input data changes as the normalization minimizes the degree to which it does. Next. it can mitigate the risk of neuron saturation as the batch normalization may prevent the activation

pushing the output to the limits. Also, each mini-batch is only normalized based on its specific scale and variance which has some of the advantages of regularization.

Activation Functions

Activation functions introduce non-linear properties to the network. Without them, each neural network would effectively be linear regression. By using activation functions, a network can learn and model more complicated data such as audio and images. Activation functions allow us to model complicated, non-linear and high dimensional data sets.

Three types of activation functions are:

- Sigmoid
- Tanh
- ReLu

Sigmoid

Function	Derivative
$\frac{1}{e^{-x}+1}$	$\frac{d}{dx}\left(\frac{1}{1+\exp(-x)}\right)=\frac{e^{-x}}{(e^{-x}+1)^2}$

Not zero centered. Outputs are constrained to between 0 and 1.
Sigmoids saturate and kill gradients.

Tanh

Function	Derivative

$\frac{e^x - e^{-x}}{e^x + e^{-x}}$	$\frac{4}{(e^{-x} + e^x)^2}$

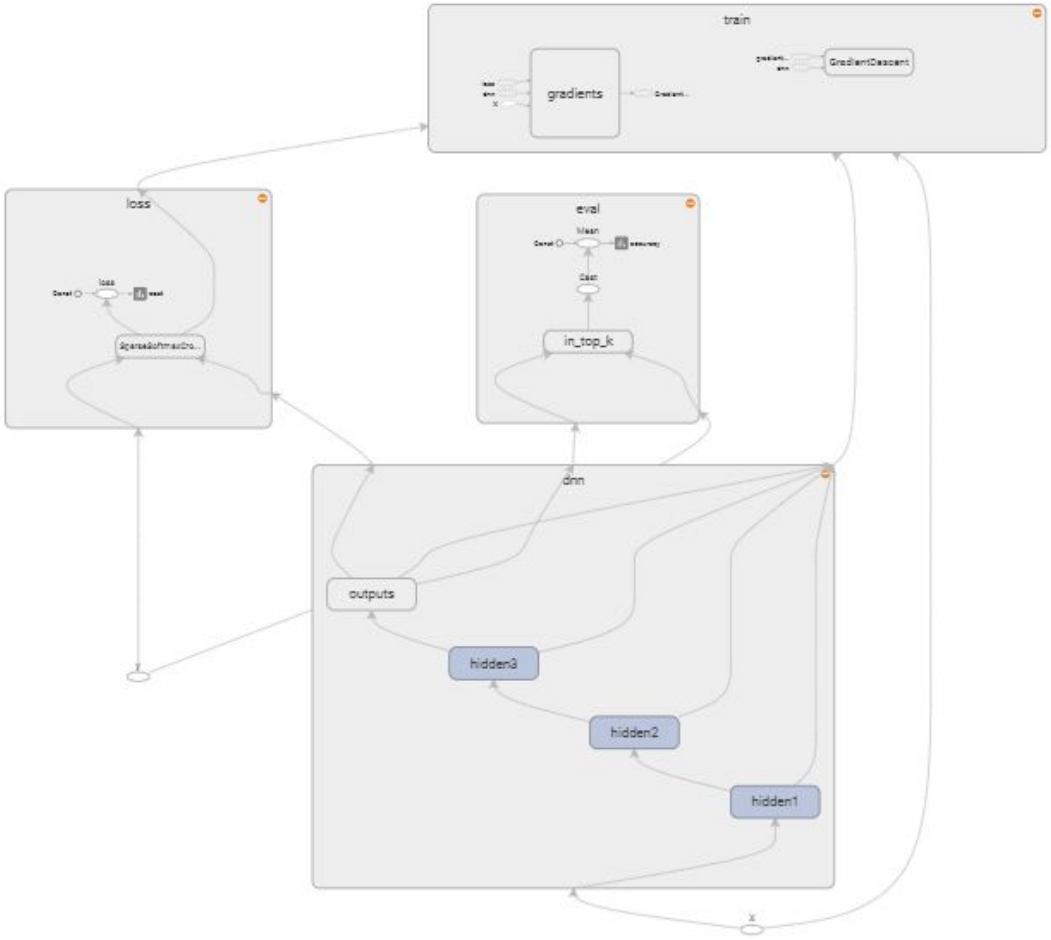
Zero centered and its outputs are between -1 and 1 which makes optimization easier. However it can still kill gradients as outputs go closer towards the limits where the derivative is closer to 0.

ReLu

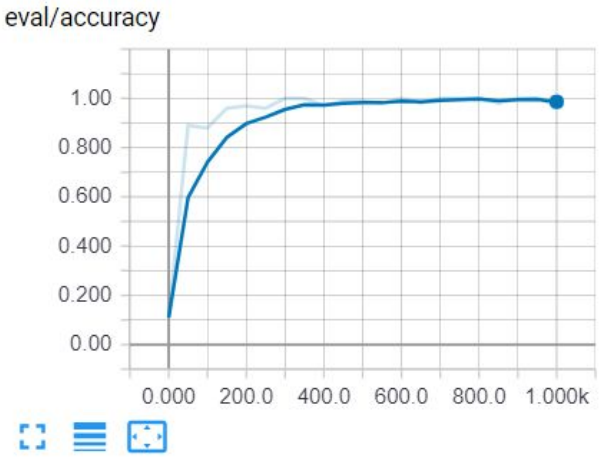
Function	Derivative
$max(0, x)$	$x < 0 : 0$ $x \geq 0 : 1$

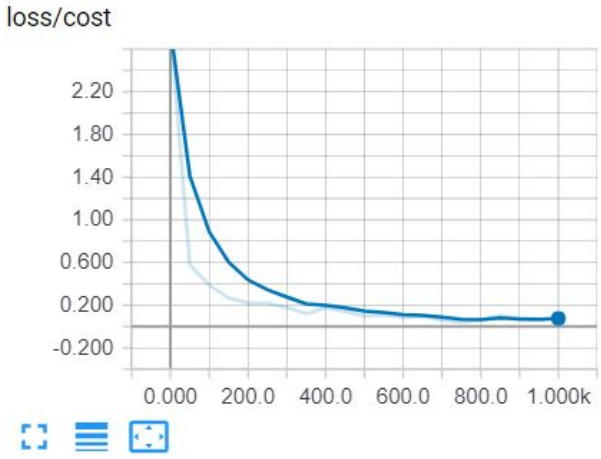
Has 6x the improvement in convergence than the tanh function. As it doesn't constrain its upward bound, it avoids the vanishing gradient problem i.e. it doesn't kill the gradients. This function still has a couple issues: it can cause dead neurons through certain weights that leave outputs as 0. The leaky ReLu function was then suggested which has a slight slope in the <0 region to keep the neurons alive.

Neural Net Customization



The changes included batch normalization and setting the keep probability of the dropout to 1.0, effectively removing dropout. I expect the model to be overfit due to removing the dropout, yet due to batch normalization to not be overfit to the train data as that is one of its benefits. Hence, I expect comparable accuracy.


eval/accuracy
1.00
0.800
0.600
0.400
0.200
0.00
0.000
200.0
400.0
600.0
800.0
1.000k


loss/cost
2.20
1.80
1.40
1.00
0.600
0.200
-0.200
0.000
200.0
400.0
600.0
800.0
1.000k