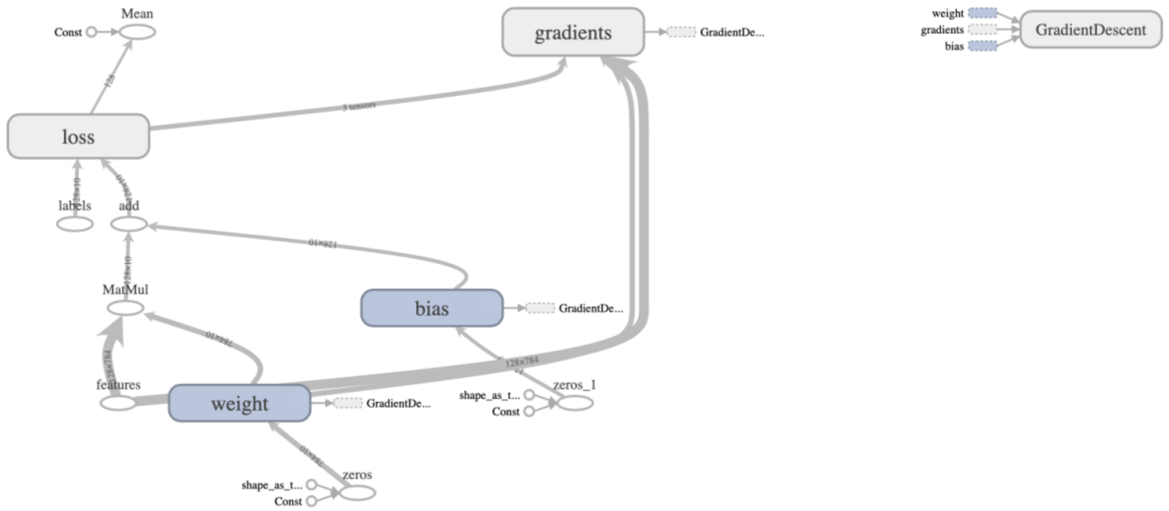
Problem 1 – TensorFlow Basics

Model Description:
(look at OpsProblems.py)

Problem 2 – Logistic Regression of MNIST

Model Description:
(look at TFLogReg.py)

Tensorboard:

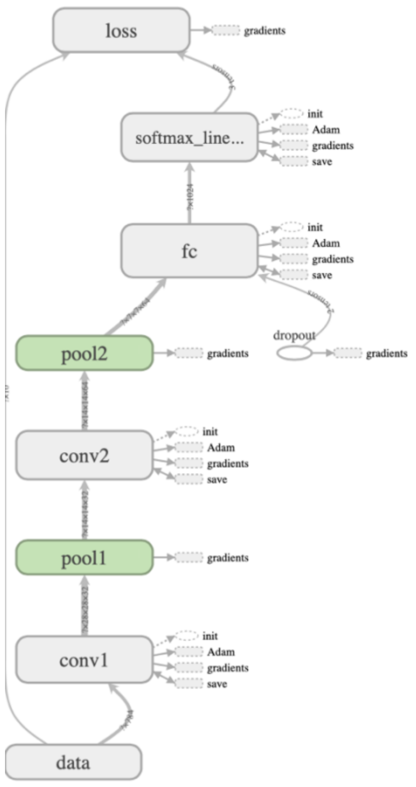


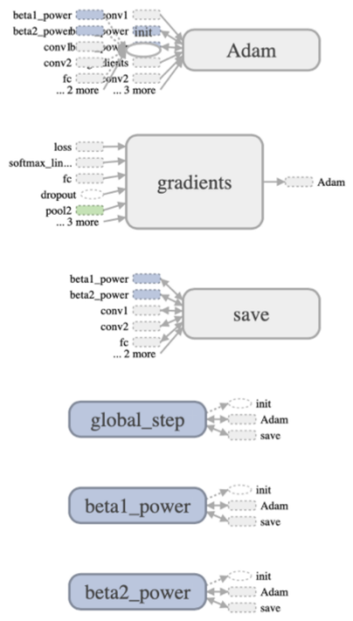
Output:
Total time: 7.441821098327637 seconds
Optimization Finished!
Accuracy 0.8967

Problem 3 – Convolution Analysis of MNIST

<u>Model Description:</u>
(look at ConvNetTemplate.py)

<u>Tensorboard:</u>



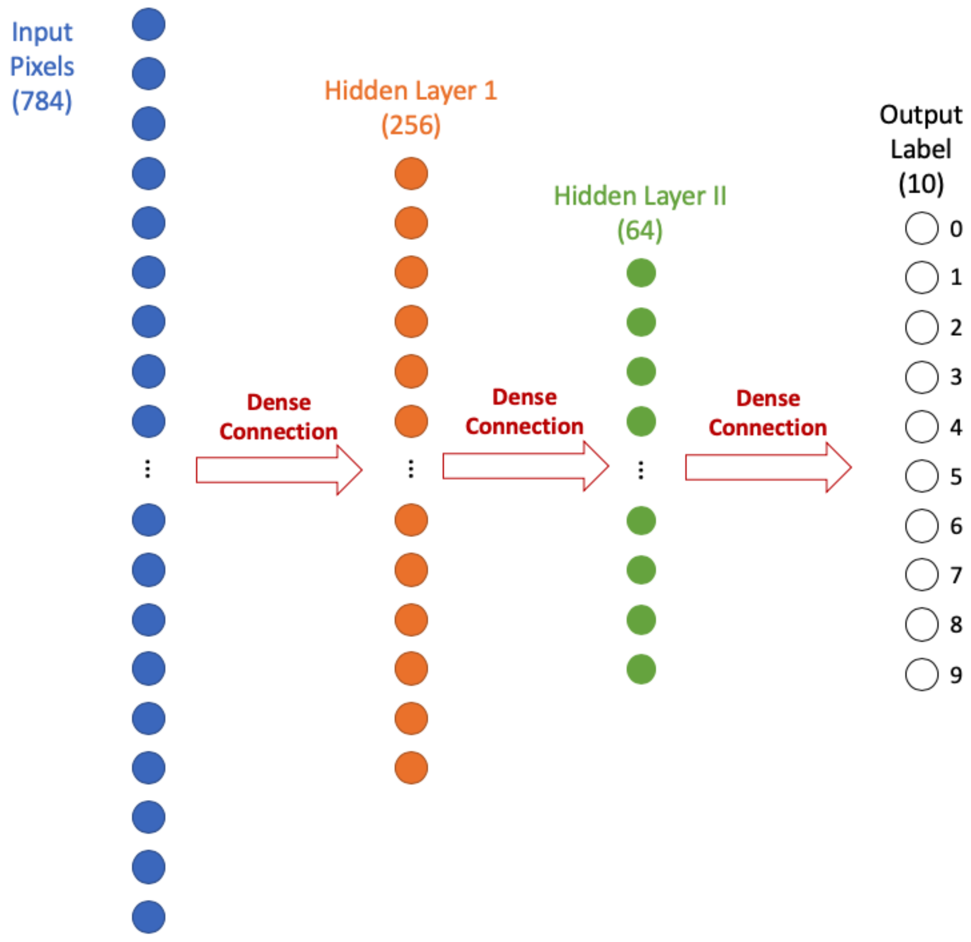


<u>Output:</u>
Optimization Finished!
Total time: 210.1983940601349 seconds
Accuracy 0.9238

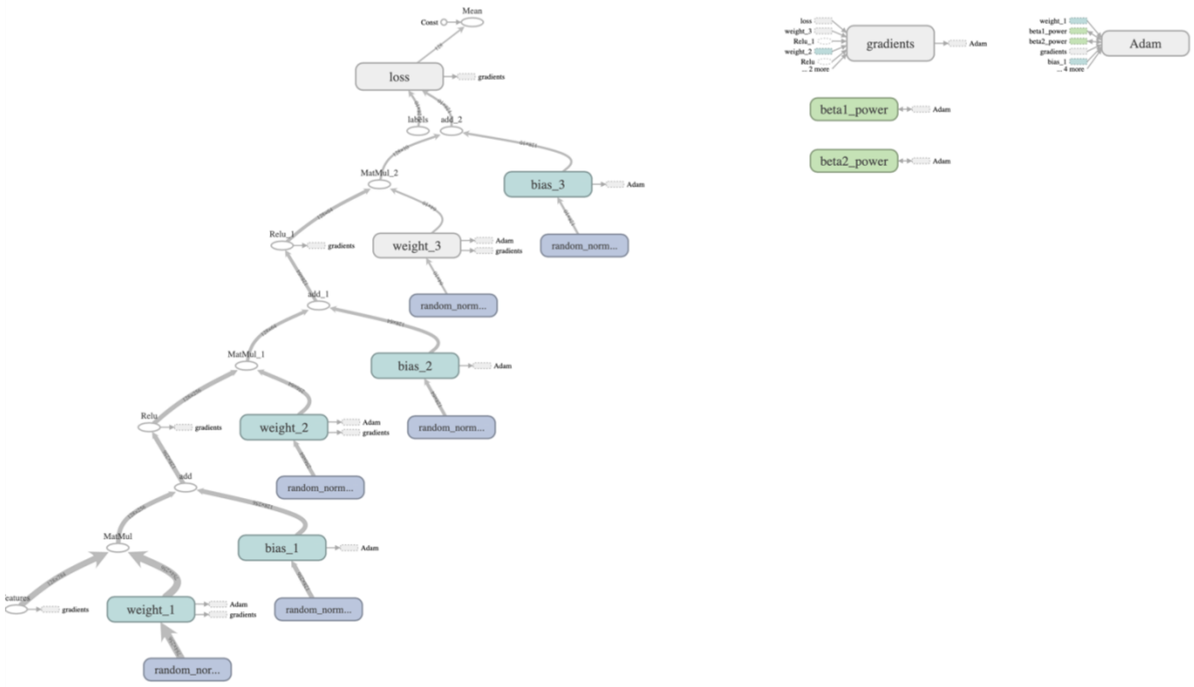
Problem 4 – Improved Logistic Regression

Model Description: *(look at ImproveLogReg.py)*

While the model in problem 2 directly maps the intensity of each pixel directly to the logits value using a logistic regression model, the accuracy of the model can be dramatically improved by increase the number of hidden states in the. In the model used for this problem, two hidden layers are added to the model of size 256 nodes and 64 nodes respectively. This adds complexity to the model, thus allowing the model to learn more complex relationships while condensing the data into fewer nodes at every level. To demonstrate the reasoning behind this architecture, each node in hidden layer I is the weighted sum of the input pixel and thus learns the linear weighted sum of the pixel intensity. When the data from hidden layer I are further condensed down into hidden layer II, the nodes in hidden layer II learn more complicated relationships between pixel values in the input layer. This data is then mapped into the output layer which is used to classify the input into the output label.



Tensorboard:



Output:

```
Total time: 22.347151041030884 seconds
Optimization Finished!
Accuracy 0.9728
```

Problem 5 – LSTM

Model Description: *(look at LSTM.py)*

In recurrent neural networks, long sequences result in a vanishing gradient problem during back propagation since the effect of an input at an earlier time has a decreased effect on later outputs. To address this problem, a long-short term memory (LSTM) cell is used to maintain a memory of the prior inputs. In LSTM, the cell has three gates that dictate the output of the cell and the contents of the memory: input gate, forget gate and output gate. The input gate computes which values of the hidden state will be updated by the current input. The forget gate computes how much of the past information stored in the memory should be forgotten. The output gate determines how much the current cell matters at the current time step. Thus the equations for the LSTM are shown below for input = x_t and previous hidden state = h_{t-1}:

Input Gate (how much of the current input matters?)

$$i_t = \sigma(W^{(i)}x_t + U^{(i)}h_{t-1})$$

Forget Gate (how much of previous state do we forget?)

$$f_t = \sigma(W^{(f)}x_t + U^{(f)}h_{t-1})$$

Output Gate (how much does the cell matter to the current time step?)

$$o_t = \sigma(W^{(o)}x_t + U^{(o)}h_{t-1})$$

New Memory Cell

$$\hat{c_t} = tanh(W^{(c)}x_t + U^{(c)}h_{t-1})$$

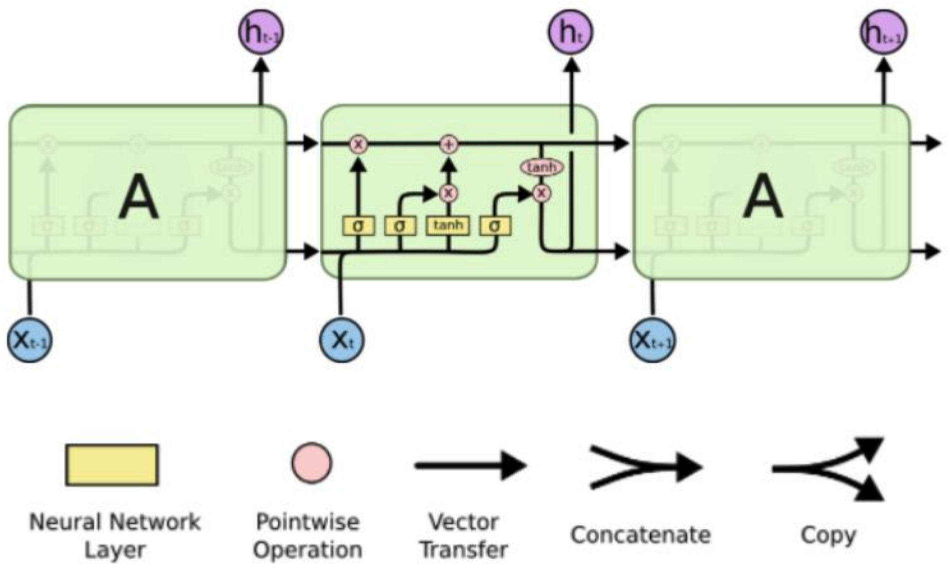
Final Memory Cell (how much of input do we keep and how much of past do we forget?)

$$c_t = f_t * c_{t-1} + i_t * \hat{c_t}$$

Final Hidden State

$$h_t = o_t * \tanh(c_t)$$

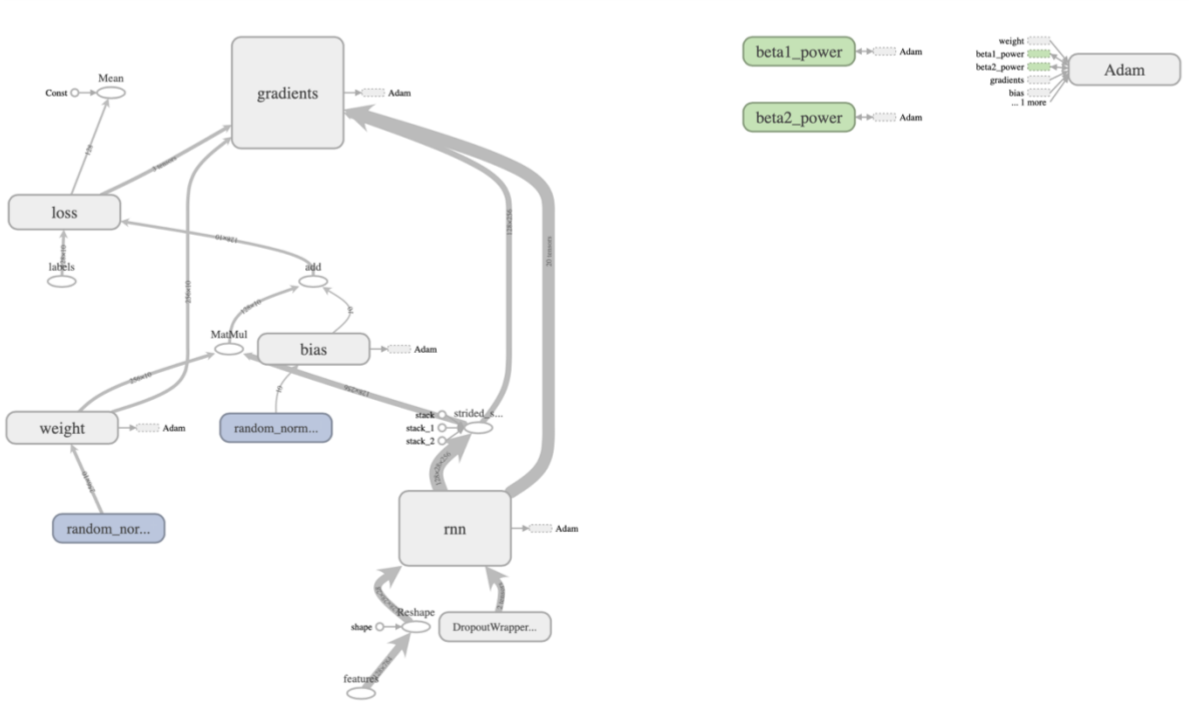
where W and U are trainable weight variables and * refers to the element-wise multiplication.



Source: http://web.stanford.edu/class/cs224n/lectures/lecture9.pdf

For the model, the input data is input into the LSTM network with each row of the image functioning as an input at a different time step (28 row input with 28 time steps). The output of the network is then mapped to the output classes to get the logit values.

Tensorboard:



Output:
```
Total time: 2243.4868421554565 seconds
Optimization Finished!
Accuracy 0.9804
```