

CSE 598: Assignment 1

Handwritten Digit Recognition using TensorFlow and MNIST dataset

Sreenivasan Ramesh
sreenivasan.ramesh@asu.edu

I. LOGISTIC REGRESSION TO CLASSIFY IMAGES

For task 1, I completed the code that implemented Logistic Regression for classification on the MNIST dataset. Data was loaded using data loading approach 2. Iterators are created for test and training data, and data is divided into batches of size 128 images each. In every epoch, batches of data are fed to the model, and weights are updated after each iteration.

The model used is Logistic Regression, which is a linear model represented by $Wx + b$, where W is a weight matrix, and b the bias vector.

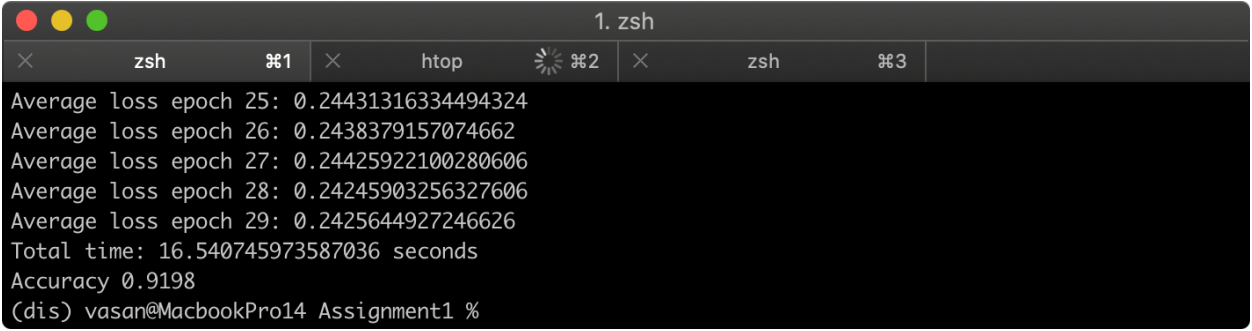
```
w = tf.get_variable(name='weight 1', shape=(784, 10), initializer=tf.contrib.layers.xavier_initializer())
b = tf.get_variable(name='bias 1', shape=(1, 10), initializer=tf.zeros_initializer()) logits =
tf.matmul(img, w) + b

entropy = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=label, name='entropy') loss
= tf.reduce_mean(entropy, name='loss')
optimizer = tf.contrib.opt.NadamOptimizer(learning_rate).minimize(loss)
```

Each input image is multiplied with the weight matrix, and bias is added. To this output we use softmax as the activation function to convert the resulting output into class probabilities. Cross Entropy on the softmax output is used as the loss function, and the mean loss is calculated per batch. I have used Nadam Optimizer, which is the Adam Optimizer with Nesterov Momentum, so that the loss converges quicker.

Results

Accuracy: 91.98%



```
1. zsh
Average loss epoch 25: 0.24431316334494324
Average loss epoch 26: 0.2438379157074662
Average loss epoch 27: 0.24425922100280606
Average loss epoch 28: 0.24245903256327606
Average loss epoch 29: 0.2425644927246626
Total time: 16.540745973587036 seconds
Accuracy 0.9198
(dis) vasan@MacbookPro14 Assignment1 %
```

Fig. 1: Task 1 Logistic Regression Accuracy

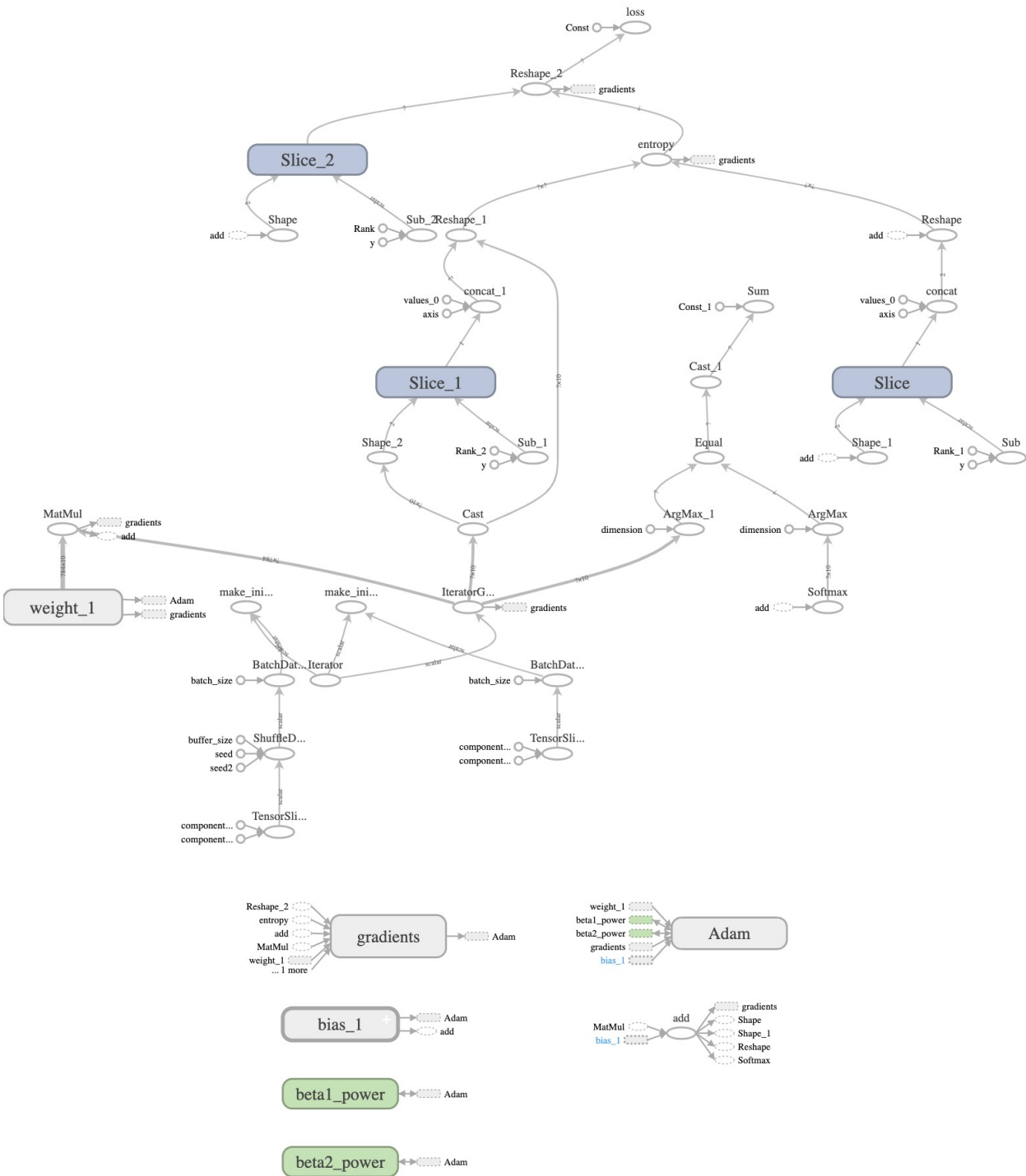


Fig 2: Tensorboard Graph for Task 1 - Logistic Regression

II. CNN TO CLASSIFY IMAGES

For Task 2, I have used Convolution Neural Networks to improve the accuracy. The architecture of my CNN is as follows:

- Batch Normalization → Convolution → ReLU activation
- MaxPooling
- Batch Normalization → Convolution → ReLU activation
- MaxPooling
- Dense Fully Connected Layer → Batch Normalization → ReLU activation
- Dropout → Output Layer

First convolution layer

The data loading is the almost the same as task 1, but here I am feeding batches of 8 images every iteration. This decrease in batch size, adds more stochasticity and a smaller batch size performed considerably better compared to a large batch size. The data is first normalized with respect to the batch, and then passed to a convolution layer, with a kernel size of 5, and a stride of 2. ReLU activation is applied to the output of this. Layer, and then passed on to the next layer.

Second convolution layer

The output from the first layer is again batch normalized and then passed to a convolution layer with kernel size 5, and a stride of 2. ReLU activation is applied and the output is passed on to the next layer.

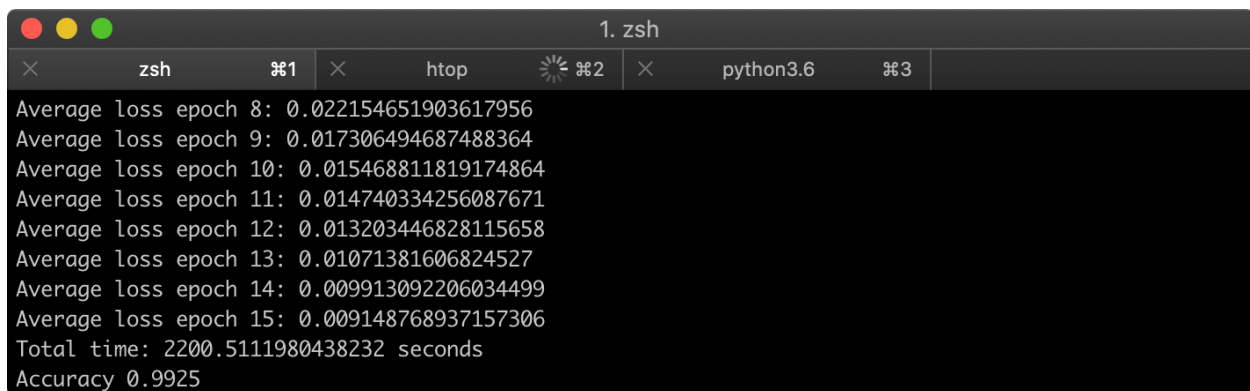
Dense fully connected layer

The dense layer consists of 256 neurons, and we multiply its weight matrix with the output from the previous layer and add the bias. We then perform batch normalization, followed by applying ReLU activation, and then dropout. After performing the dropout, we then send the output to the final output layer, after which we apply a softmax to get the class confidences. Like the previous task, I have used Nadam Optimizer for the optimization function.

Results

Accuracy: 99.25%

Time to run: 36 mins (on CPU)

A terminal window titled '1. zsh' with three tabs: 'zsh', 'htop', and 'python3.6'. The terminal displays the following text:

```
Average loss epoch 8: 0.022154651903617956
Average loss epoch 9: 0.017306494687488364
Average loss epoch 10: 0.015468811819174864
Average loss epoch 11: 0.014740334256087671
Average loss epoch 12: 0.013203446828115658
Average loss epoch 13: 0.01071381606824527
Average loss epoch 14: 0.009913092206034499
Average loss epoch 15: 0.009148768937157306
Total time: 2200.5111980438232 seconds
Accuracy 0.9925
```

Fig. 3: Task 2 CNN Accuracy

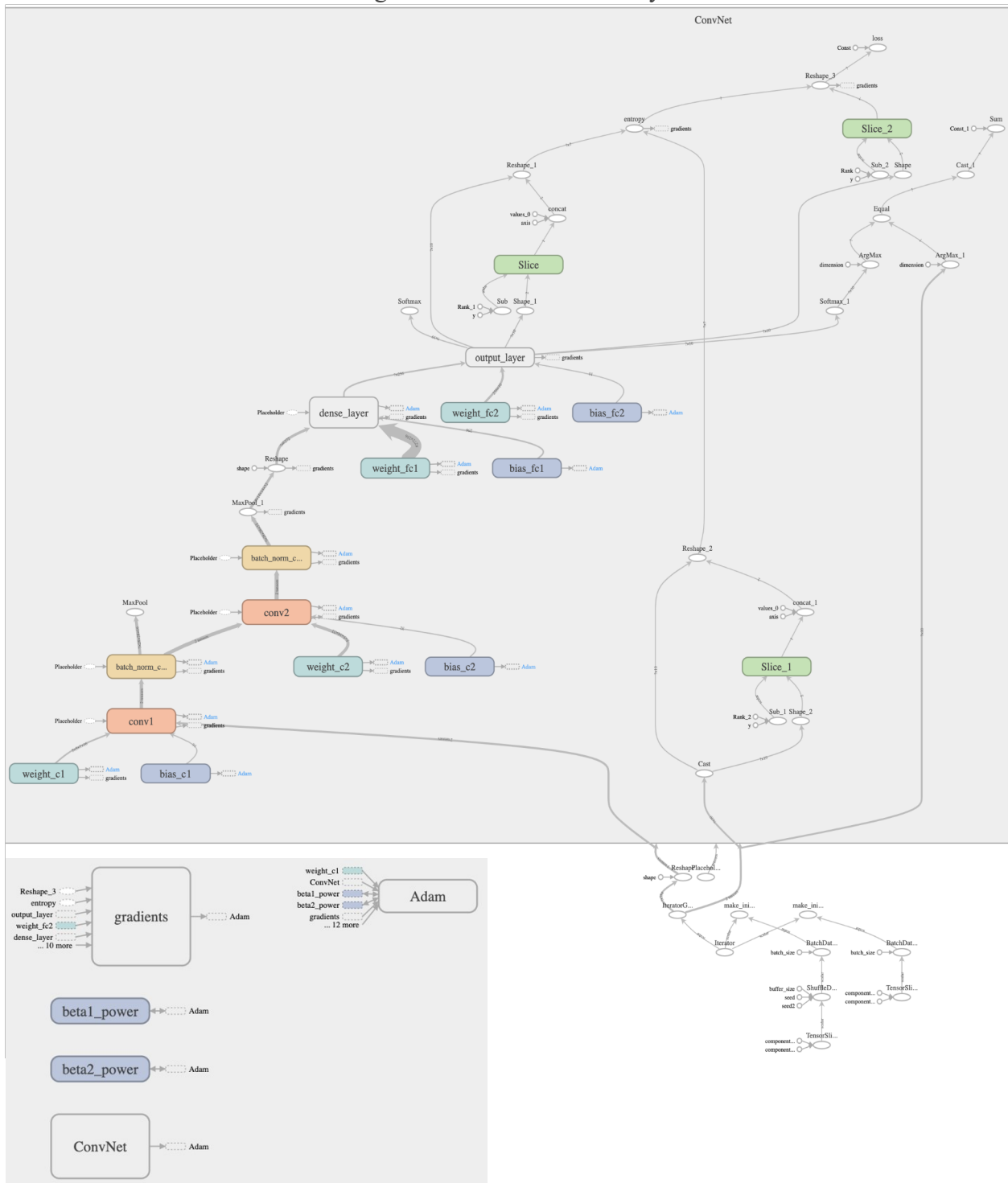


Fig 4: Tensorboard Graph for Task 2 – Convolutional Neural Network

Other models tested for task 2

2 Fully Connected Dense Layers: 97%

CNN with 2 conv layers without batch normalization: 98.8%

CNN with 4 conv layers: 98.7%

CNN with 4 conv layers and batch normalization*: 98.4%

*CNN with 4 conv layers, along with batch normalization and dropout tended to overfit after around 10 epochs. Due to running on CPU and time constraints, I wasn't able to tune the parameters enough to get a better accuracy.

III. REPORT

Time Estimates

Time taken for task 1: < 1 hour

Time taken for task 2: ~30 hours (3 days * 10 hours)

Some issues faced during task 2 were:

- i) Getting batch normalization to work properly
- ii) Using tensorboard callbacks with TensorFlow core. Since I had not used tensorboard with core TensorFlow before, I implemented the models in Keras first, and used callbacks to visualize test vs validation accuracy for different parameters, and later implemented the models in TensorFlow.

Interesting Problems/Comments

i) Batch Normalization before or after activation?

There is no proper census on whether to perform batch normalization before or after the activation. Francois Chollet says that *recent code written by Christian applies relu before BN*^[1]. Andrej Karpathy suggests that Batch Normalization be performed before any non-linearity^[2].

In practice, I found that applying Batch Normalization before ReLU, slightly outperformed applying Batch Normalization after the activation (~0.2% difference in accuracy over 5 runs).

References

[1] Chollet. F, "BN Questions – keras-team github issue", February 23 2026. Available at <https://github.com/keras-team/keras/issues/1802#issuecomment>. [Accessed: January 24, 2020]

[2] Karpathy. A, "CS231n Winter 2016: Lecture 5: Neural Networks Part 2", January 20, 2016. Available at <https://youtu.be/gYpoJMIgyXA?t=3194>. [Accessed: January 24, 2020]