A short analysis about this model, how it works and its advantages with respect a simple multi-layer perceptron on one layer. The role of the training rule.

Results: some classified examples, training duration, validation curves and test efficiency.

- Input layer (Image vector)

- 3 fully connected hidden layers

- Output layer (According to the number of classes)

Epochs: 2500

Eval\_every = 5

Batch size = 1000

Evaluation size = 100

**Adam optimizer:**

Learning rate= 0.001

Beta1 = 0.9

Beta2 = 0.999

Sd= 0.01

1024 -> 256 -> 64 Neurons

Input shape = 1000, 64, 64, 3 (#batch, width, height, #channels)

Eval shape = 100, 64, 64 , 3 (#eval size, width, height, #channels)

**LAYERS**

First layer :

Weight\_1= [64\*64\*3 (width\*heights\*channels), 1024]

Bias\_1 = 1024

2nd layer: 1024x256 weight

3rd layer: 256x 64 weight

Output layer: 64x target size(#labels = 2)

**Architecture:**

1st layer:

Matrix multiplication of flattened images and weights of 1st layer

Add the bias vector to the result

Perform relu on results to introduce non linearity

2nd, 3rd works the same, but on output of previous layers

Output layer performs matmul, addition, but no activation function. Typical for classification, where raw numbers will go through softmax.

Introduce loss function. Tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(logits, labels)

Compute sparse softmax cross-entropy between logits (predicted values) and the labels (true values)

Tf.reduce\_mean to perform average over batch.

Do softmax on outputs.

To get accuracy: (per batch)

Argmax on logits, axis=1 to return the class with the highest probability for each sample in the batch

checks how much of classifications are correct (predictions == targets)

return percentage of correct predictions out of total number of samples in a batch

**Training**

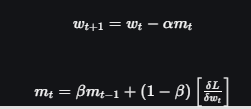
Adam optimizer:

Tr.train.AdamOptimizer(learning\_rate, beta1, beta2)

Higher learning rate – larger steps during each iteration, potentially speeding up the training, but risking overshooting minima. Lower LR can be slower, but more accurate (smaller steps)

Beta1 = exponential decay rate for the first moment estimates. Thes controls the moving average of the gradient.

Beta2= the exponential decay rate for the second moment estimates thie controls the moving average of the squared gradient.



**Training step**

My\_optim.minimize(loss) – apply gradients calculated by the optimizer to the variables to minimize the loss.

Compute gradients of the loss function with respect to the each model parameter

Adjust the model parameters using the computed gradients.

Update the variables (weights and biases) in the network to reduce the loss

Tf.train.saver() : function is used to save the model’s variables at various points during the training process. Creates a saver object thich can be used to store and restore TF variables.

**Initializing global variables**

Init = Tf.global\_variables\_initializer() : returns an operation that initializes all grobal variables in the TF graph. Includes weights, biases and any other set parameters.

Session.run(init) runs the initializer operation

**TRAIN loop:**

While we didn’t exceed # of epochs and training loss isn’t satisfied

Get randomly selected batch\_size of training examples and their labels

**session.run(train\_step, feed\_dict=train\_dict)**: Executes a training step to update the model parameters. (train\_dict is a dictionary of x\_input : train\_x, y\_target: train\_y}

**session.run([loss, prediction], feed\_dict=train\_dict)**: Evaluates the current loss and predictions for the training batch.

Calculate get\_accuracy for temp\_predicted and true labels

Saver.save(session, ‘name’, global\_step) saves model state to a checkpoint file, globalstep appends the current epoch number to file name

**Evaluation and logging**

IF its time to evalueate (every 5 epochs) or loss is great or accuracy is great:

Prepare evalusation batch, dictionary, run session for test \_prediction, get accuracy.

Append results to lists (loss, acc)

**EVALUATION NN**:

Visualize softmax loss for each epoch (every 5)

Visualize prediction accuracy (training and validation)

Drops can indicate LR adjustment, data shuffling, comple data patterns.

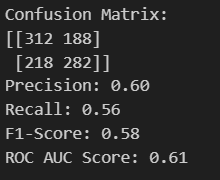
Overfitting or noise in data. Also batch variability can cause fluctuations in learning.

3/8 classified correctly in training

5/8 validation classified correctly

4/8 in test

**RESULTS FOR TEST**

****