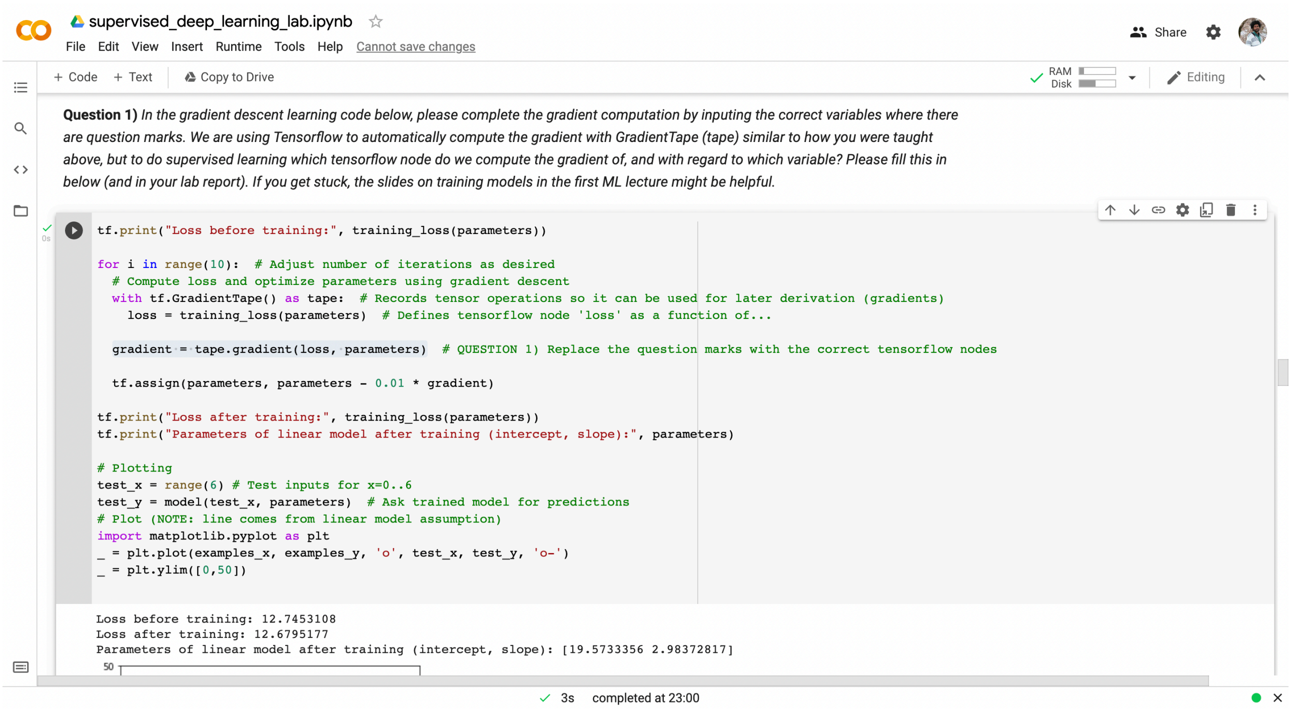
Lab - 6 : Report

Question 1) *In the gradient descent learning code below, please complete the gradient computation by inputing the correct variables where there are question marks. We are using Tensorflow to automatically compute the gradient with GradientTape (tape) similar to how you were taught above, but to do supervised learning which tensorflow node do we compute the gradient of, and with regard to which variable? Please fill this in below (and in your lab report). If you get stuck, the slides on training models in the first ML lecture might be helpful.*

Answer:

gradient = tape.gradient(loss, parameters)

Screenshot for the same is attached below,


supervised_deep_learning_lab.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing

Question 1) *In the gradient descent learning code below, please complete the gradient computation by inputing the correct variables where there are question marks. We are using Tensorflow to automatically compute the gradient with GradientTape (tape) similar to how you were taught above, but to do supervised learning which tensorflow node do we compute the gradient of, and with regard to which variable? Please fill this in below (and in your lab report). If you get stuck, the slides on training models in the first ML lecture might be helpful.*

```python
tf.print("Loss before training:", training_loss(parameters))

for i in range(10):  # Adjust number of iterations as desired
  # Compute loss and optimize parameters using gradient descent
  with tf.GradientTape() as tape:  # Records tensor operations so it can be used for later derivation (gradients)
    loss = training_loss(parameters)  # Defines tensorflow node 'loss' as a function of...

  gradient = tape.gradient(loss, parameters)  # QUESTION 1) Replace the question marks with the correct tensorflow nodes

  tf.assign(parameters, parameters - 0.01 * gradient)

tf.print("Loss after training:", training_loss(parameters))
tf.print("Parameters of linear model after training (intercept, slope):", parameters)

# Plotting
test_x = range(6) # Test inputs for x=0..6
test_y = model(test_x, parameters)  # Ask trained model for predictions
# Plot (NOTE: line comes from linear model assumption)
import matplotlib.pyplot as plt
_ = plt.plot(examples_x, examples_y, 'o', test_x, test_y, 'o-')
_ = plt.ylim([0,50])
```

```
Loss before training: 12.7453108
Loss after training: 12.6795177
Parameters of linear model after training (intercept, slope): [19.5733356 2.98372817]
```

✓ 3s completed at 23:00

Question 2) *Show the math for why the first Dense layer has 100 480 parameters with these inputs and number of neurons. Remember, a neuron is just a non-linear transformation (e.g. sigmoid/ReLU) of a* ***linear model****, implying one parameter for each input dimension, + 1 for the line intercept/constant (also called neuron "bias" in NN slides from the first ML lecture).*

Answer:

In the first dense layer the input has 784 elements so it will require 784 parameter corresponding to weights for each element and 1 parameter for bias. Hence total of 784+1=785 parameters are required for one neuron, since there are 128 neuron the total number of parameters = 128*785 = 100480.

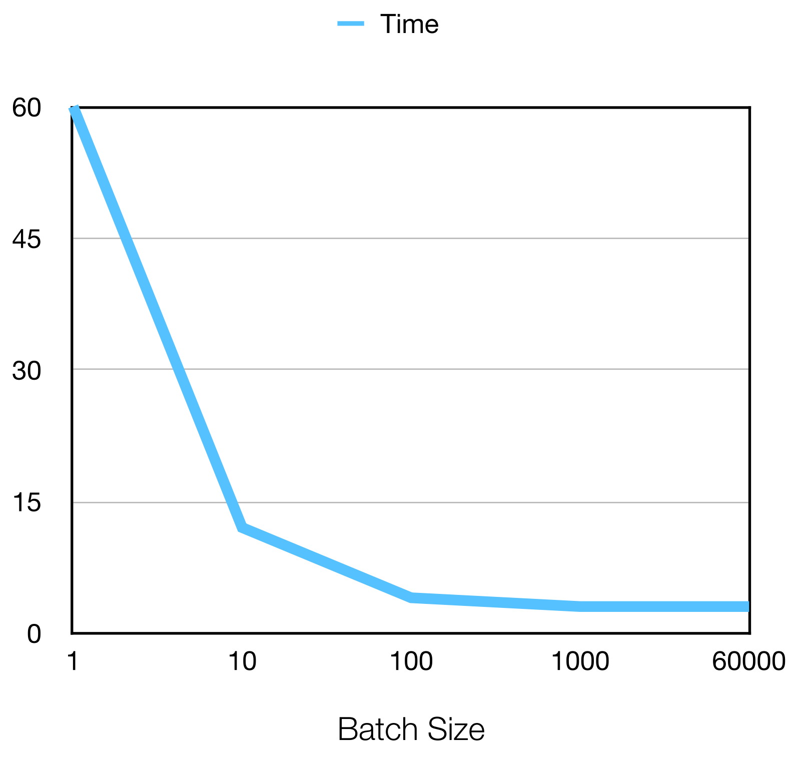
Question 3 *Here you will evaluate different mini-bach sizes for stochastic gradient descent (see the deep learning lecture). Please separately run the training code above with batch sizes of 1, 10, 100, 1000 and 60000. Write down the training times (you can use the first number in seconds, not the per sample time) and the training set accuracy reached, both in the first line of the output. This can randomly vary a bit between runs but it should give you an idea. In your lab report, plot both curves and reason about which batch size produced the most accuracy given the time spent, i.e. which batch size would be best to start the training with? You have to run the Reset All Parameters code above this one between your runs to always start over.*

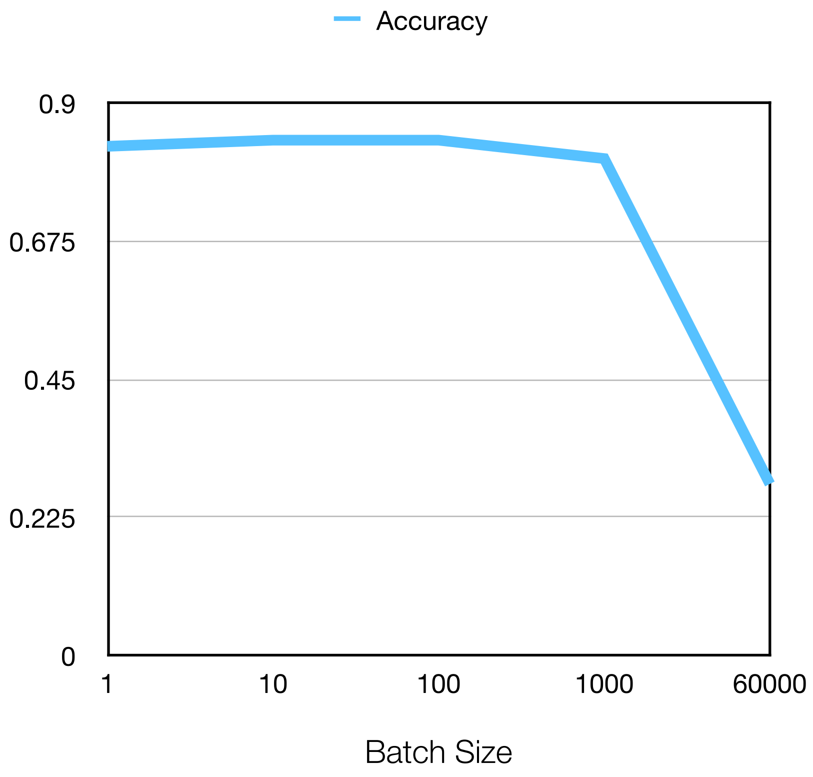
Answer:

Training with batch-size 1, Time taken = 1 minute; Accuracy = 0.83
Training with batch-size 10, Time taken = 12 seconds; Accuracy = 0.84
Training with batch-size 100, Time taken = 4 seconds; Accuracy = 0.84
Training with batch-size 1000, Time taken = 3 seconds; Accuracy = 0.81
Training with batch-size 60000, Time taken = 3 seconds; Accuracy = 0.28

The graphs for the obtained data are given below,

We can see that the as the batch size increases the training time goes down. This is because when the batch size is small the parameters are updated after each batch is trained which causes increase in training time.





We can see that the accuracy of the model falls off as the batch size increases this is because as the batch size increases the model takes more time to converge. It can also be observed that slightly larger values of batch size perform slightly better than very low values. This is because very low batch size causes the noise in the data to be reflected in the model causing it to loose a bit of accuracy.

I think an intermediate value of 100 would be a good starting point this can be further tuned by considering it as a hyper-paramter during learning.