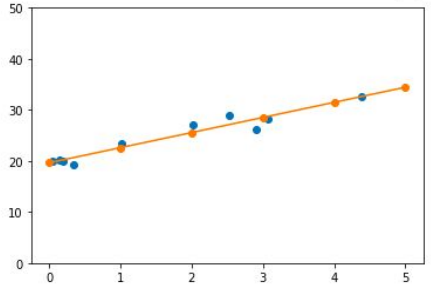
TDDC17 Lab 6 Report.
Alim Amanzholov (aliam864)

Question 1) *In the gradient descent learning code below, please complete the gradient computation by inputting 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.*

gradient = tape.gradient(loss, parameters)

```
[23] 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: 12.662714
Loss after training: 12.6626921
Parameters of linear model after training (intercept, slope): [19.6343937 2.96110463]
```



Question 2) *Show the math for why the first Dense layer has 100 480 parameters with these inputs and the 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).*

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 128)               100480
_________________________________________________________________
dense_1 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
_________________________________________________________________
Model and loss defined, parameters reset to small values.
```

28*28*128 + 128 = 784 * 128 + 128 = 100352 + 128 = 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.*

Batch size	Training times (s)	Accuracy
1	118	0.8124
10	13	0.8251
100	3	0.8185
1000	1	0.7097
60000	1	0.1000

From the above table, we can see that if we use too small batch size it will take a considerably long time to train our model, but it will not necessarily give us higher accuracy. If we use too big batch size, our model will train fast, but accuracy will suffer a lot. So to get great accuracy in a reasonably short amount of time, we should not use too big and too small batch sizes.

From the table above, we can conclude that the batch size of 100 would be the best to start the training with since it has high accuracy and short training time.