Introduction to TensorFlow in Python.

August 6, 2020

Not long ago, cutting-edge computer vision algorithms couldn't differentiate between images of cats and dogs. Today, a skilled data scientist equipped with nothing more than a laptop can classify tens of thousands of objects with greater accuracy than the human eye. In this project, I will use TensorFlow 2.1 to develop, train, and make predictions with the models that have powered major advances in recommendation systems, image classification, and FinTech. I will work on both high-level APIs, which will enable us to design and train deep learning models in 15 lines of code, and low-level APIs, which will allow us to move beyond off-the-shelf routines. I will also work to accurately predict housing prices, credit card borrower defaults, and images of sign language gestures.

Defining data as constants

Throughout this document, I will use tensorflow version 2.1 and will exclusively import the sub-modules needed.

After I have imported constant, I will use it to transform a numpy array, credit_numpy, into a tensorflow constant, credit_constant. This array contains feature columns from a dataset on credit card holders.

Note that tensorflow version 2.0 allows us to use data as either a numpy array or a tensorflow constant object. Using a constant will ensure that any operations performed with that object are done in tensorflow.

[1]:
```
import pandas as pd

df = pd.read_csv('uci_credit_card.csv')
df.head()
```

[1]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	\
0	1	20000.0	2	2	1	24	2	2	-1	-1	
1	2	120000.0	2	2	2	26	-1	2	0	0	
2	3	90000.0	2	2	2	34	0	0	0	0	
3	4	50000.0	2	2	1	37	0	0	0	0	
4	5	50000.0	1	2	1	57	-1	0	-1	0	

	…	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	\
0	…	0.0	0.0	0.0	0.0	689.0	0.0	
1	…	3272.0	3455.0	3261.0	0.0	1000.0	1000.0	
2	…	14331.0	14948.0	15549.0	1518.0	1500.0	1000.0	
3	…	28314.0	28959.0	29547.0	2000.0	2019.0	1200.0	

```
4  …    20940.0    19146.0    19131.0    2000.0   36681.0   10000.0

   PAY_AMT4  PAY_AMT5  PAY_AMT6  default.payment.next.month
0       0.0       0.0       0.0                          1
1    1000.0       0.0    2000.0                          1
2    1000.0    1000.0    5000.0                          0
3    1100.0    1069.0    1000.0                          0
4    9000.0     689.0     679.0                          0

[5 rows x 25 columns]
```

Defining variables

Unlike a constant, a variable's value can be modified. This will be quite useful when we want to train a model by updating its parameters. Constants can't be used for this purpose, so variables are the natural choice.

Let's try defining and working with a variable.

[2]:
```
import tensorflow as tf
from tensorflow import constant
from tensorflow import Variable

# Defining the 1-dimensional variable A1
A1 = Variable([1, 2, 3, 4])

# Print the variable A1
print(A1)

# Convering A1 to a numpy array and assign it to B1
B1 = A1.numpy()

# Print B1
print(B1)
```

```
<tf.Variable 'Variable:0' shape=(4,) dtype=int32, numpy=array([1, 2, 3, 4],
dtype=int32)>
[1 2 3 4]
```

Next I'll review how to check the properties of a tensor after it is already defined.

Performing element-wise multiplication

Element-wise multiplication in TensorFlow is performed using two tensors with identical shapes. This is because the operation multiplies elements in corresponding positions in the two tensors. An example of an element-wise multiplication, denoted by the symbol, is shown below:

[1221] [3215]=[3425]

Below, I will perform element-wise multiplication, paying careful attention to the shape of the tensors we multiply. Note that multiply(), constant(), and ones_like() have been imported.

[3]:
```
from tensorflow import ones, matmul, multiply, ones_like

from tensorflow import ones_like
```

[4]:
```
# Defining tensors A1 and A23 as constants
A1 = constant([1, 2, 3, 4])
A23 = constant([[1, 2, 3], [1, 6, 4]])

# Defining B1 and B23 to have the correct shape
B1 = ones_like(A1)
B23 = ones_like(A23)

# Performing element-wise multiplication
C1 = multiply(A1, B1)
C23 = multiply(A23, B23)

# Printing the tensors C1 and C23
print('C1: {}'.format(C1.numpy()))
print('C23: {}'.format(C23.numpy()))
```

```
C1: [1 2 3 4]
C23: [[1 2 3]
 [1 6 4]]
```

Notice how performing element-wise multiplication with tensors of ones leaves the original tensors unchanged.

Making predictions with matrix multiplication

Later, I will work to train linear regression models. This process will yield a vector of parameters that can be multiplied by the input data to generate predictions. Here, I will use input data, features, and a target vector, bill, which are taken from a credit card dataset that I will use later.

features = constant([[2, 24], [2, 26], [2, 57], [1, 37]]) params = constant([[1000], [150]]) bill = constant([[3913], [2682], [8617], [64400]])

The matrix of input data, features, contains two columns: education level and age. The target vector, bill, is the size of the credit card borrower's bill.

Since I have not trained the model, I will enter a guess for the values of the parameter vector, params. I will then use matmul() to perform matrix multiplication of features by params to generate predictions, billpred, which I will compare with bill. Note that I have imported matmul() and constant().

[5]:
```
# Defining features, params, and bill as constants
features = constant([[2, 24], [2, 26], [2, 57], [1, 37]])
params = constant([[1000], [150]])
bill = constant([[3913], [2682], [8617], [64400]])

# Computing billpred using features and params
```

```
billpred = matmul(features, params)

# Computing and print the error
error = bill - billpred
print(error.numpy())
```

```
[[-1687]
 [-3218]
 [-1933]
 [57850]]
```

Summing over tensor dimensions

I've created a matrix, wealth. This contains the value of bond and stock wealth for five individuals in thousands of dollars.

wealth = ([[11,50],[7,2],[4,60],[3,0],[25,19]])

The first column corresponds to bonds and the second corresponds to stocks. Each row gives the bond and stock wealth for a single individual. Using wealth, reduce_sum(), and .numpy() trying to determine which statements are correct about wealth.

Hint

reduce_sum(wealth,1) will sum wealth over its rows.

reduce_sum(wealth,0) will sum wealth over its columns.

Note bonds appear first and stocks appear second in reduce_sum(wealth,0).

We can append .numpy() to reduce_sum() to convert the results to a numpy array.

[6]:
```
wealth = ([[11,50],[7,2],[4,60],[3,0],[25,19]])
```

[7]:
```
tf.math.reduce_sum(wealth,1).numpy()
```

[7]:
```
array([61,  9, 64,  3, 44], dtype=int32)
```

The individual in the first row has the highest total wealth (i.e. stocks + bonds).

Combined, the 5 individuals hold $50,000 in stocks.

Combined, the 5 individuals hold $50,000 in bonds. correct.

The individual in the second row has the lowest total wealth (i.e. stocks + bonds).

Optimizing with gradients

We are given a loss function, y=x2, which we want to minimize. We can do this by computing the slope using the GradientTape() operation at different values of x. If the slope is positive, we can decrease the loss by lowering x. If it is negative, you can decrease it by increasing x. This is how gradient descent works.

In practice, we will use a high level tensorflow operation to perform gradient descent automatically. Below I will compute the slope at x values of -1, 1, and 0. The following operations are available: GradientTape(), multiply(), and Variable().

[8]:
```
def compute_gradient(x0):
   # Defining x as a variable with an initial value of x0
    x = Variable(x0)
    with tf.GradientTape() as tape:
        tape.watch(x)
        # Defining y using the multiply operation
        y = multiply(x, x)
    # Returning the gradient of y with respect to x
    return tape.gradient(y, x).numpy()

# Computing and print gradients at x = -1, 1, and 0
print(compute_gradient(-1.0))
print(compute_gradient(1.0))
print(compute_gradient(0.0))
```

```
-2.0
2.0
0.0
```

Notice that the slope is positive at x = 1, which means that we can lower the loss by reducing x. The slope is negative at x = -1, which means that we can lower the loss by increasing x. The slope at x = 0 is 0, which means that we cannot lower the loss by either increasing or decreasing x. This is because the loss is minimized at x = 0.

Working with image data

If we are given a black-and-white image of a letter, which has been encoded as a tensor, letter. We want to determine whether the letter is an X or a K. We don't have a trained neural network, but we do have a simple model, model, which can be used to classify letter.

The 3x3 tensor, letter, and the 1x3 tensor, model, are available. We can determine whether letter is a K by multiplying letter by model, summing over the result, and then checking if it is equal to 1. As with more complicated models, such as neural networks, model is a collection of weights, arranged in a tensor.

Note that the functions reshape(), matmul(), and reduce_sum() have been imported from tensorflow and are available for use.

[10]:
```
import numpy as np
letter = np.array([[1., 0., 1.],
       [1., 1., 0.],
       [1., 0., 1.]])
```

[11]:
```
model = np.array([[ 1.,  0., -1.]])
```

[12]:
```
# Reshaping model from a 1x3 to a 3x1 tensor
model = tf.reshape(model, (3, 1))

# Multipling letter by model
output = matmul(letter, model)

# Summing over output and print prediction using the numpy method
prediction = tf.reduce_sum(output)
print(prediction.numpy())
```

```
1.0
```

The model found that prediction=1.0 and correctly classified the letter as a K. Below, I will use data to train a model, model, and then combine this with matrix multiplication, matmul(letter, model), as we have done here, to make predictions about the classes of objects.

Loading data using pandas

Before we can train a machine learning model, we must first import data. There are several valid ways to do this, but for now, we will use a simple one-liner from pandas: pd.read_csv(). The first argument specifies the path or URL. All other arguments are optional.

Here I will import the King County housing dataset, which we will use to train a linear model.

[13]:
```
# Importing pandas under the alias pd
import pandas as pd

# Assigning the path to a string variable named data_path
data_path = 'kc_house_data.csv'

# Loading the dataset as a dataframe named housing
housing = pd.read_csv('kc_house_data.csv')

# Printing the price column of housing
print(housing['price'])
```

```
0        221900.0
1        538000.0
2        180000.0
3        604000.0
4        510000.0
           …
21608    360000.0
21609    400000.0
21610    402101.0
21611    400000.0
21612    325000.0
Name: price, Length: 21613, dtype: float64
```

Notice that we did not have to specify a delimiter with the sep parameter, since the dataset was

stored in the default, comma-separated format.

Setting the data type

Here I will both load data and set its type. Note that housing is available and pandas has been imported as pd. I will import numpy and tensorflow, and define tensors that are usable in tensorflow using columns in housing with a given data type. Recall that we can select the price column, for instance, from housing using housing[‘price’].

[14]:
```
# Importing numpy and tensorflow with their standard aliases
import numpy as np
import tensorflow as tf

# Using a numpy array to define price as a 32-bit float
price = np.array(housing['price'], np.float32)

# Defining waterfront as a Boolean using cast
waterfront = tf.cast(housing['waterfront'], tf.bool)

# Printing price and waterfront
print(waterfront)
print(price)
```

```
tf.Tensor([False False False … False False False], shape=(21613,), dtype=bool)
[221900. 538000. 180000. … 402101. 400000. 325000.]
```

Notice that printing price yielded a numpy array; whereas printing waterfront yielded a tf.Tensor().

Loss functions in TensorFlow

Here, I will compute the loss using data from the King County housing dataset. We are given a target, price, which is a tensor of house prices, and predictions, which is a tensor of predicted house prices. I will evaluate the loss function and print out the value of the loss.

1. Importing the keras module from tensorflow. Then, using price and predictions to compute the mean squared error (mse).
2. Modifying the code to compute the mean absolute error (mae), rather than the mean squared error (mse).

[15]:
```
price = np.array([221900, 538000, 180000, 402101, 400000, 325000])
predictions = np.array([154546.51228417, 153942.9622033 ,  40833.26164156,
       369047.72405433, 162293.96762846, 148248.92465109])
```

[16]:
```
# Importing the keras module from tensorflow
from tensorflow import keras

# Computing the mean absolute error (mae)
loss = keras.losses.mse(price, predictions)

# Printing the mean absolute error (mae)
```

```
print(loss.numpy())

# Computing the mean absolute error (mae)
loss = keras.losses.mae(price, predictions)

# Printing the mean absolute error (mae)
print(loss.numpy())
```

```
43373550194.59085
173014.60792284834
```

Modifying the loss function

Above, I defined a tensorflow loss function and then evaluated it once for a set of actual and predicted values. Here, I will compute the loss within another function called loss_function(), which first generates predicted values from the data and variables. The purpose of this is to construct a function of the trainable model variables that returns the loss. We can then repeatedly evaluate this function for different variable values until we find the minimum. In practice, we will pass this function to an optimizer in tensorflow. Note that features and targets have been defined and are available. Additionally, Variable, float32, and keras are available.

[17]:
```
features = constant([1., 2., 3., 4., 5.,])
```

[18]:
```
targets = constant([2., 4., 6., 8., 10.,])
```

[19]:
```
from tensorflow import float32
```

[20]:
```
# Initializing a variable named scalar
scalar = Variable(1.0, float32)

# Defining the model
def model(scalar, features = features):
    return scalar * features

# Defining a loss function
def loss_function(scalar, features = features, targets = targets):
    # Computing the predicted values
    predictions = model(scalar, features)

    # Returning the mean absolute error loss
    return keras.losses.mae(targets, predictions)

# Evaluating the loss function and print the loss
print(loss_function(scalar).numpy())
```

```
3.0
```

As we will see below, this exercise was the equivalent of evaluating the loss function for a linear regression where the intercept is 0.

Setting up a linear regression

A univariate linear regression identifies the relationship between a single feature and the target tensor. Here, I will use a property's lot size and price. I will take the natural logarithms of both tensors, which are available as price_log and size_log.

Here, I will define the model and the loss function. We will then evaluate the loss function for two different values of intercept and slope. Remember that the predicted values are given by intercept + features*slope. Additionally, note that keras.losses.mse() is available. Furthermore, slope and intercept have been defined as variables.

[21]:
```
price_log = np.array([12.309982 , 13.195614 , 12.100712 , 12.904459 , 12.
↪8992195, 12.691581])
size_log = np.array([8.639411 , 8.887652 , 9.2103405, 7.20786  , 7.7782116, 6.
↪9810057])
```

[22]:
```
# Defining a linear regression model
def linear_regression(intercept, slope, features = size_log):
    return intercept + features*slope

# Setting loss_function() to take the variables as arguments
def loss_function(intercept, slope, features = size_log, targets = price_log):
    # Setting the predicted values
    predictions = linear_regression(intercept, slope, features)

    # Returning the mean squared error loss
    return keras.losses.mse(targets, predictions)

# Computing the loss for different slope and intercept values
print(loss_function(0.1, 0.1).numpy())
print(loss_function(0.1, 0.5).numpy())
```

```
138.74718552748786
73.1122366897609
```

In the next exercise, I will actually run the regression and train intercept and slope

Training a linear model

Here I will pick up where the previous exercise ended. The intercept and slope, have been defined and initialized. Additionally, a function has been defined, loss_function(intercept, slope), which computes the loss using the data and model variables.

We will now define an optimization operation as opt. We will then train a univariate linear model by minimizing the loss to find the optimal values of intercept and slope. Note that the opt operation will try to move closer to the optimum with each step, but will require many steps to find it. Thus, we must repeatedly execute the operation.

[23]:
```
intercept = tf.Variable(0.1, np.float32)
slope = tf.Variable(0.1, np.float32)
```

[24]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Initializing an adam optimizer
opt = tf.keras.optimizers.Adam(0.5)

for j in range(100):
    # Applying minimize, pass the loss function, and supply the variables
    opt.minimize(lambda: loss_function(intercept, slope), var_list=[intercept,␣
 ↪slope])

    # Printing every 10th value of the loss
    if j % 10 == 0:
        print(loss_function(intercept, slope).numpy())
```

```
52.57076
1.924646
2.8864498
3.221184
2.7455146
1.9191192
1.6072747
1.5980115
1.5014572
1.441584
```

Notice that we printed loss_function(intercept, slope) every 10th execution for 100 executions. Each time, the loss got closer to the minimum as the optimizer moved the slope and intercept parameters closer to their optimal values.

Multiple linear regression

In most cases, performing a univariate linear regression will not yield a model that is useful for making accurate predictions. Below, I will perform a multiple regression, which uses more than one feature.

I will use price_log as our target and size_log and bedrooms as our features. Each of these tensors has been defined and is available. We will also switch from using the mean squared error loss to the mean absolute error loss: keras.losses.mae(). Finally, the predicted values are computed as follows: params[0] + feature1params[1] + feature2params[2]. Note that we've defined a vector of parameters, params, as a variable, rather than using three variables. Here, params[0] is the intercept and params[1] and params[2] are the slopes.

[25]:
```
params = tf.Variable(np.array([0.1, 0.05, 0.02], np.float32))
```

[26]:
```
params[1]
```

[26]:
```
<tf.Tensor: shape=(), dtype=float32, numpy=0.05>
```

[27]:
```
bedrooms = np.array([3, 3, 2, 2, 3, 2])
```

[28]:
```
# Defining the linear regression model
def linear_regression(params, feature1 = size_log, feature2 = bedrooms):
    return params[0] + feature1*params[1] + feature2*params[2]

# Defining the loss function
def loss_function(params, targets = price_log, feature1 = size_log, feature2 =␣
 ↪bedrooms):
    # Set the predicted values
    predictions = linear_regression(params, feature1, feature2)

    # Using the mean absolute error loss
    return keras.losses.mae(targets, predictions)

# Defining the optimize operation
opt = tf.keras.optimizers.Adam()

# Performing minimization and print trainable variables
for j in range(10):
    opt.minimize(lambda: loss_function(params), var_list=[params])
    # print_results(params)
    print(params)
    #print(loss_function(params, intercept))
```

```
<tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([0.101, 0.051,
0.021], dtype=float32)>
<tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([0.10199999,
0.05199999, 0.02199999], dtype=float32)>
<tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([0.10299999,
0.053     , 0.023     ], dtype=float32)>
<tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([0.10399999,
0.05399999, 0.02399999], dtype=float32)>
<tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([0.10499998,
0.05499999, 0.02499999], dtype=float32)>
<tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([0.10599998,
0.05599999, 0.02599999], dtype=float32)>
<tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([0.10699998,
0.05699999, 0.02699999], dtype=float32)>
<tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([0.10799997,
0.05799999, 0.02799999], dtype=float32)>
<tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([0.10899997,
0.05899999, 0.02899999], dtype=float32)>
<tf.Variable 'Variable:0' shape=(3,) dtype=float32, numpy=array([0.10999997,
0.05999999, 0.02999999], dtype=float32)>
```

Great job! Note that params[2] tells us how much the price will increase in percentage terms if we add one more bedroom. You could train params[2] and the other model parameters by increasing

the number of times we iterate over opt.

Preparing to batch train

Before we can train a linear model in batches, we must first define variables, a loss function, and an optimization operation. Here, we will prepare to train a model that will predict price_batch, a batch of house prices, using size_batch, a batch of lot sizes in square feet. In contrast to the above codes, we will do this by loading batches of data using pandas, converting it to numpy arrays, and then using it to minimize the loss function in steps.

Variable(), keras(), and float32 have been imported. Note that we should not set default argument values for either the model or loss function, since we will generate the data in batches during the training process.

[29]:
```
# Defining the intercept and slope
intercept = Variable(10.0, float32)
slope = Variable(0.5, float32)

# Defining the model
def linear_regression(intercept, slope, features):
    # Define the predicted values
    return intercept + features*slope

# Defining the loss function
def loss_function(intercept, slope, targets, features):
    # Defining the predicted values
    predictions = linear_regression(intercept, slope, features)

     # Defining the MSE loss
    return keras.losses.mse(targets, predictions)
```

Notice that we did not use default argument values for the input data, features and targets. This is because the input data has not been defined in advance. Instead, with batch training, we will load it during the training process.

Training a linear model in batches

Here I will train a linear regression model in batches, starting where I left off above. I will do this by stepping through the dataset in batches and updating the model's variables, intercept and slope, after each step. This approach will allow us to train with datasets that are otherwise too large to hold in memory.

Note that the loss function,loss_function(intercept, slope, targets, features), has been defined. The trainable variables should be entered into var_list in the order in which they appear as loss function arguments.

[30]:
```
# Computing predicted values and return loss function
def loss_function(intercept, slope, targets, features):
    predictions = linear_regression(intercept, slope, features)
    return tf.keras.losses.mse(targets, predictions)
```

[31]:
```
# Initializing adam optimizer
opt = tf.keras.optimizers.Adam()

# Loading data in batches
for batch in pd.read_csv('kc_house_data.csv', chunksize=100):
    size_batch = np.array(batch['sqft_lot'], np.float32)

    # Extracting the price values for the current batch
    price_batch = np.array(batch['price'], np.float32)

    # Completing the loss, fill in the variable list, and minimize
    opt.minimize(lambda: loss_function(intercept, slope, price_batch,␣
 ↪size_batch), var_list=[intercept, slope])

# Printing trained parameters
print(intercept.numpy(), slope.numpy())
```

```
10.217888 0.7016
```

The linear algebra of dense layers

There are two ways to define a dense layer in tensorflow. The first involves the use of low-level, linear algebraic operations. The second makes use of high-level keras operations.

The input layer contains 3 features – education, marital status, and age – which are available as borrower_features. The hidden layer contains 2 nodes and the output layer contains a single node.

For each layer, I will take the previous layer as an input, initialize a set of weights, compute the product of the inputs and weights, and then apply an activation function. Note that Variable(), ones(), matmul(), and keras() have been imported from tensorflow.

[32]:
```
credit = pd.read_csv('uci_credit_card.csv')
```

[33]:
```
borrower_features = credit[['EDUCATION', 'MARRIAGE', 'AGE']].values
```

[34]:
```
borrower_features = constant([[ 2.,  2., 43.]])
```

[35]:
```
# Initializing bias1
bias1 = Variable(1.0)

# Initializing weights1 as 3x2 variable of ones
weights1 = Variable(ones((3, 2)))

# Performing matrix multiplication of borrower_features and weights1
product1 = matmul(borrower_features, weights1)

# Applying sigmoid activation function to product1 + bias1
dense1 = keras.activations.sigmoid(product1 + bias1)
```

```
# Printing shape of dense1
print("\n dense1's output shape: {}".format(dense1.shape))
```

```
 dense1's output shape: (1, 2)
```

Initialize weights2 as a variable using a 2x1 tensor of ones.

Compute the product of dense1 by weights2 using matrix multiplication.

Use a sigmoid activation function to transform product2 + bias2.

[41]:
```
# From previous step
bias1 = Variable(1.0)
weights1 = Variable(ones((3, 2)))
product1 = matmul(borrower_features, weights1)
dense1 = keras.activations.sigmoid(product1 + bias1)

# Initializing bias2 and weights2
bias2 = Variable(1.0)
weights2 = Variable(ones((2, 1)))

# Performing matrix multiplication of dense1 and weights2
product2 = matmul(dense1, weights2)

# Applying activation to product2 + bias2 and print the prediction
prediction = keras.activations.sigmoid(product2 + bias2)
print('\n prediction: {}'.format(prediction.numpy()[0,0]))
print('\n actual: 1')
```

```
 prediction: 0.9525741338729858

 actual: 1
```

Our model produces predicted values in the interval between 0 and 1. For the example we considered, the actual value was 1 and the predicted value was a probability between 0 and 1. This, of course, is not meaningful, since we have not yet trained our model's parameters.

The low-level approach with multiple examples

Here, we'll build further intuition for the low-level approach by constructing the first dense hidden layer for the case where we have multiple examples. We'll assume the model is trained and the first layer weights, weights1, and bias, bias1, are available. We'll then perform matrix multiplication of the borrower_features tensor by the weights1 variable. Recall that the borrower_features tensor includes education, marital status, and age. Finally, we'll apply the sigmoid function to the elements of products1 + bias1, yielding dense1.

[46]:
```
weights1 = np.array([[-0.6 ,  0.6 ],
       [ 0.8 , -0.3 ],
       [-0.09, -0.08]],np.float32)
```

[47]:
```
borrower_features = constant([[ 3.,  3., 23.],
       [ 2.,  1., 24.],
       [ 1.,  1., 49.],
       [ 1.,  1., 49.],
       [ 2.,  1., 29.]], np.float32)
```

[48]:
```
bias1 =  tf.Variable([0.1,])
```

[49]:
```
# Computing the product of borrower_features and weights1
products1 = matmul(borrower_features, weights1)

# Applying a sigmoid activation function to products1 + bias1
dense1 = keras.activations.sigmoid(products1+bias1)

# Printing the shapes of borrower_features, weights1, bias1, and dense1
print('\n shape of borrower_features: ', borrower_features.shape)
print('\n shape of weights1: ', weights1.shape)
print('\n shape of bias1: ', bias1.shape)
print('\n shape of dense1: ', dense1.shape)
```

```
 shape of borrower_features:  (5, 3)

 shape of weights1:  (3, 2)

 shape of bias1:  (1,)

 shape of dense1:  (5, 2)
```

Note that our input data, borrower_features, is 5x3 because it consists of 5 examples for 3 features. The shape of weights1 is 3x2, as it was above, since it does not depend on the number of examples. Additionally, bias1 is a scalar. Finally, dense1 is 5x2, which means that we can multiply it by the following set of weights, weights2, which we defined to be 2x1 above.

Using the dense layer operation

We've now seen how to define dense layers in tensorflow using linear algebra. Here, we'll skip the linear algebra and let keras work out the details. This will allow us to construct the network below, which has 2 hidden layers and 10 features, using less code than we needed for the network with 1 hidden layer and 3 features.

To construct this network, we'll need to define three dense layers, each of which takes the previous layer as an input, multiplies it by weights, and applies an activation function. Note that input data has been defined and is available as a 100x10 tensor: borrower_features. Additionally, the keras.layers module is available.

[50]:
```
borrower_features = constant([[6.96469188e-01, 2.86139339e-01, 2.26851448e-01,␣
↪5.51314771e-01,
       7.19468951e-01, 4.23106462e-01, 9.80764210e-01, 6.84829712e-01,
```

```
 4.80931908e-01, 3.92117530e-01],
[3.43178004e-01, 7.29049683e-01, 4.38572258e-01, 5.96778952e-02,
 3.98044258e-01, 7.37995386e-01, 1.82491735e-01, 1.75451756e-01,
 5.31551361e-01, 5.31827569e-01],
[6.34400964e-01, 8.49431813e-01, 7.24455297e-01, 6.11023486e-01,
 7.22443402e-01, 3.22958916e-01, 3.61788660e-01, 2.28263229e-01,
 2.93714046e-01, 6.30976140e-01],
[9.21049416e-02, 4.33701187e-01, 4.30862755e-01, 4.93685097e-01,
 4.25830305e-01, 3.12261224e-01, 4.26351309e-01, 8.93389165e-01,
 9.44160044e-01, 5.01836658e-01],
[6.23952925e-01, 1.15618393e-01, 3.17285478e-01, 4.14826214e-01,
 8.66309166e-01, 2.50455379e-01, 4.83034253e-01, 9.85559762e-01,
 5.19485116e-01, 6.12894535e-01],
[1.20628662e-01, 8.26340795e-01, 6.03060126e-01, 5.45068026e-01,
 3.42763841e-01, 3.04120779e-01, 4.17022198e-01, 6.81300759e-01,
 8.75456870e-01, 5.10422349e-01],
[6.69313788e-01, 5.85936546e-01, 6.24903500e-01, 6.74689054e-01,
 8.42342436e-01, 8.31949860e-02, 7.63682842e-01, 2.43666381e-01,
 1.94222957e-01, 5.72456956e-01],
[9.57125202e-02, 8.85326803e-01, 6.27248943e-01, 7.23416328e-01,
 1.61292069e-02, 5.94431877e-01, 5.56785166e-01, 1.58959642e-01,
 1.53070509e-01, 6.95529521e-01],
[3.18766415e-01, 6.91970289e-01, 5.54383278e-01, 3.88950586e-01,
 9.25132513e-01, 8.41669977e-01, 3.57397556e-01, 4.35914621e-02,
 3.04768085e-01, 3.98185670e-01],
[7.04958856e-01, 9.95358467e-01, 3.55914861e-01, 7.62547791e-01,
 5.93176901e-01, 6.91701770e-01, 1.51127458e-01, 3.98876280e-01,
 2.40855902e-01, 3.43456000e-01],
[5.13128161e-01, 6.66624546e-01, 1.05908483e-01, 1.30894944e-01,
 3.21980596e-01, 6.61564350e-01, 8.46506238e-01, 5.53257346e-01,
 8.54452491e-01, 3.84837806e-01],
[3.16787899e-01, 3.54264677e-01, 1.71081826e-01, 8.29112649e-01,
 3.38670850e-01, 5.52370071e-01, 5.78551471e-01, 5.21533072e-01,
 2.68806447e-03, 9.88345444e-01],
[9.05341566e-01, 2.07635865e-01, 2.92489409e-01, 5.20010173e-01,
 9.01911378e-01, 9.83630896e-01, 2.57542074e-01, 5.64359069e-01,
 8.06968689e-01, 3.94370049e-01],
[7.31073022e-01, 1.61069021e-01, 6.00698590e-01, 8.65864456e-01,
 9.83521581e-01, 7.93657899e-02, 4.28347290e-01, 2.04542860e-01,
 4.50636476e-01, 5.47763586e-01],
[9.33267102e-02, 2.96860784e-01, 9.27584231e-01, 5.69003761e-01,
 4.57412004e-01, 7.53525972e-01, 7.41862178e-01, 4.85790335e-02,
 7.08697379e-01, 8.39243352e-01],
[1.65937886e-01, 7.80997932e-01, 2.86536604e-01, 3.06469738e-01,
 6.65261447e-01, 1.11392170e-01, 6.64872468e-01, 8.87856781e-01,
 6.96311295e-01, 4.40327883e-01],
[4.38214391e-01, 7.65096068e-01, 5.65641999e-01, 8.49041641e-02,
```

```
 5.82671106e-01, 8.14843714e-01, 3.37066382e-01, 9.27576602e-01,
 7.50716984e-01, 5.74063838e-01],
[7.51644015e-01, 7.91489631e-02, 8.59389067e-01, 8.21504116e-01,
 9.09871638e-01, 1.28631204e-01, 8.17800835e-02, 1.38415575e-01,
 3.99378717e-01, 4.24306870e-01],
[5.62218368e-01, 1.22243546e-01, 2.01399505e-01, 8.11644375e-01,
 4.67987567e-01, 8.07938218e-01, 7.42637832e-03, 5.51592708e-01,
 9.31932151e-01, 5.82175434e-01],
[2.06095725e-01, 7.17757583e-01, 3.78985852e-01, 6.68383956e-01,
 2.93197222e-02, 6.35900378e-01, 3.21979336e-02, 7.44780660e-01,
 4.72912997e-01, 1.21754356e-01],
[5.42635918e-01, 6.67744428e-02, 6.53364897e-01, 9.96086299e-01,
 7.69397318e-01, 5.73774099e-01, 1.02635257e-01, 6.99834049e-01,
 6.61167860e-01, 4.90971319e-02],
[7.92299330e-01, 5.18716574e-01, 4.25867707e-01, 7.88187146e-01,
 4.11569238e-01, 4.81026262e-01, 1.81628838e-01, 3.21318895e-01,
 8.45533013e-01, 1.86903745e-01],
[4.17291075e-01, 9.89034534e-01, 2.36599818e-01, 9.16832328e-01,
 9.18397486e-01, 9.12963450e-02, 4.63652730e-01, 5.02216339e-01,
 3.13668936e-01, 4.73395362e-02],
[2.41685644e-01, 9.55296382e-02, 2.38249913e-01, 8.07791114e-01,
 8.94978285e-01, 4.32228930e-02, 3.01946849e-01, 9.80582178e-01,
 5.39504826e-01, 6.26309335e-01],
[5.54540846e-03, 4.84909445e-01, 9.88328516e-01, 3.75185519e-01,
 9.70381573e-02, 4.61908758e-01, 9.63004470e-01, 3.41830611e-01,
 7.98922718e-01, 7.98846304e-01],
[2.08248302e-01, 4.43367690e-01, 7.15601265e-01, 4.10519779e-01,
 1.91006958e-01, 9.67494309e-01, 6.50750339e-01, 8.65459859e-01,
 2.52423584e-02, 2.66905814e-01],
[5.02071083e-01, 6.74486384e-02, 9.93033290e-01, 2.36462399e-01,
 3.74292195e-01, 2.14011908e-01, 1.05445869e-01, 2.32479781e-01,
 3.00610125e-01, 6.34442270e-01],
[2.81234771e-01, 3.62276763e-01, 5.94284385e-03, 3.65719140e-01,
 5.33885956e-01, 1.62015840e-01, 5.97433090e-01, 2.93152481e-01,
 6.32050514e-01, 2.61966046e-02],
[8.87593448e-01, 1.61186308e-02, 1.26958027e-01, 7.77162433e-01,
 4.58952338e-02, 7.10998714e-01, 9.71046150e-01, 8.71682942e-01,
 7.10161626e-01, 9.58509743e-01],
[4.29813325e-01, 8.72878909e-01, 3.55957657e-01, 9.29763675e-01,
 1.48777649e-01, 9.40029025e-01, 8.32716227e-01, 8.46054852e-01,
 1.23923011e-01, 5.96486926e-01],
[1.63924806e-02, 7.21184373e-01, 7.73751410e-03, 8.48222747e-02,
 2.25498408e-01, 8.75124514e-01, 3.63576323e-01, 5.39959908e-01,
 5.68103194e-01, 2.25463361e-01],
[5.72146773e-01, 6.60951793e-01, 2.98245400e-01, 4.18626845e-01,
 4.53088939e-01, 9.32350636e-01, 5.87493777e-01, 9.48252380e-01,
 5.56034744e-01, 5.00561416e-01],
```

```
      [3.53221106e-03, 4.80889052e-01, 9.27455008e-01, 1.98365688e-01,
       5.20911328e-02, 4.06778902e-01, 3.72396469e-01, 8.57153058e-01,
       2.66111158e-02, 9.20149207e-01],
      [6.80903018e-01, 9.04226005e-01, 6.07529044e-01, 8.11953306e-01,
       3.35543871e-01, 3.49566221e-01, 3.89874220e-01, 7.54797101e-01,
       3.69291186e-01, 2.42219806e-01],
      [9.37668383e-01, 9.08011079e-01, 3.48797321e-01, 6.34638071e-01,
       2.73842216e-01, 2.06115127e-01, 3.36339533e-01, 3.27099890e-01,
       8.82276118e-01, 8.22303832e-01],
      [7.09623218e-01, 9.59345222e-01, 4.22543347e-01, 2.45033041e-01,
       1.17398441e-01, 3.01053345e-01, 1.45263731e-01, 9.21861008e-02,
       6.02932215e-01, 3.64187449e-01],
      [5.64570367e-01, 1.91335723e-01, 6.76905870e-01, 2.15505451e-01,
       2.78023601e-01, 7.41760433e-01, 5.59737921e-01, 3.34836423e-01,
       5.42988777e-01, 6.93984687e-01],
      [9.12132144e-01, 5.80713212e-01, 2.32686386e-01, 7.46697605e-01,
       7.77769029e-01, 2.00401321e-01, 8.20574224e-01, 4.64934856e-01,
       7.79766679e-01, 2.37478226e-01],
      [3.32580268e-01, 9.53697145e-01, 6.57815099e-01, 7.72877812e-01,
       6.88374341e-01, 2.04304114e-01, 4.70688760e-01, 8.08963895e-01,
       6.75035119e-01, 6.02788571e-03],
      [8.74077454e-02, 3.46794724e-01, 9.44365561e-01, 4.91190493e-01,
       2.70176262e-01, 3.60423714e-01, 2.10652635e-01, 4.21200067e-01,
       2.18035445e-01, 8.45752478e-01],
      [4.56270605e-01, 2.79802024e-01, 9.32891667e-01, 3.14351350e-01,
       9.09714639e-01, 4.34180908e-02, 7.07115054e-01, 4.83889043e-01,
       4.44221050e-01, 3.63233462e-02],
      [4.06831913e-02, 3.32753628e-01, 9.47119534e-01, 6.17659986e-01,
       3.68874848e-01, 6.11977041e-01, 2.06131533e-01, 1.65066436e-01,
       3.61817271e-01, 8.63353372e-01],
      [5.09401739e-01, 2.96901524e-01, 9.50251639e-01, 8.15966070e-01,
       3.22973937e-01, 9.72098231e-01, 9.87351120e-01, 4.08660144e-01,
       6.55923128e-01, 4.05653208e-01],
      [2.57348120e-01, 8.26526731e-02, 2.63610333e-01, 2.71479845e-01,
       3.98639083e-01, 1.84886038e-01, 9.53818381e-01, 1.02879882e-01,
       6.25208557e-01, 4.41697389e-01],
      [4.23518062e-01, 3.71991783e-01, 8.68314683e-01, 2.80476987e-01,
       2.05761567e-02, 9.18097019e-01, 8.64480257e-01, 2.76901782e-01,
       5.23487568e-01, 1.09088197e-01],
      [9.34270695e-02, 8.37466121e-01, 4.10265714e-01, 6.61716521e-01,
       9.43200588e-01, 2.45130599e-01, 1.31598311e-02, 2.41484065e-02,
       7.09385693e-01, 9.24551904e-01],
      [4.67330277e-01, 3.75109136e-01, 5.42860448e-01, 8.58916819e-01,
       6.52153850e-01, 2.32979894e-01, 7.74580181e-01, 1.34613499e-01,
       1.65559977e-01, 6.12682283e-01],
      [2.38783404e-01, 7.04778552e-01, 3.49518538e-01, 2.77423948e-01,
       9.98918414e-01, 4.06161249e-02, 6.45822525e-01, 3.86995859e-02,
```

```
       7.60210276e-01, 2.30089962e-01],
      [8.98318663e-02, 6.48449719e-01, 7.32601225e-01, 6.78095341e-01,
       5.19009456e-02, 2.94306934e-01, 4.51088339e-01, 2.87103295e-01,
       8.10513437e-01, 1.31115109e-01],
      [6.12179339e-01, 9.88214970e-01, 9.02556539e-01, 2.22157061e-01,
       8.18876142e-05, 9.80597317e-01, 8.82712960e-01, 9.19472456e-01,
       4.15503561e-01, 7.44615436e-01],
      [2.12831497e-01, 3.92304063e-01, 8.51548076e-01, 1.27612218e-01,
       8.93865347e-01, 4.96507972e-01, 4.26095665e-01, 3.05646390e-01,
       9.16848779e-01, 5.17623484e-01],
      [8.04026365e-01, 8.57651770e-01, 9.22382355e-01, 3.03380728e-01,
       3.39810848e-01, 5.95073879e-01, 4.41324145e-01, 9.32842553e-01,
       3.97564054e-01, 4.77778047e-01],
      [6.17186069e-01, 4.04739499e-01, 9.92478430e-01, 9.88512859e-02,
       2.20603317e-01, 3.22655141e-01, 1.47722840e-01, 2.84219235e-01,
       7.79245317e-01, 5.22891998e-01],
      [3.39536369e-02, 9.82622564e-01, 6.16006494e-01, 5.89394793e-02,
       6.61168754e-01, 3.78369361e-01, 1.35673299e-01, 5.63664615e-01,
       7.27079928e-01, 6.71126604e-01],
      [2.47513160e-01, 5.24866223e-01, 5.37663460e-01, 7.16803372e-01,
       3.59867334e-01, 7.97732592e-01, 6.27921820e-01, 3.83316055e-02,
       5.46479046e-01, 8.61912072e-01],
      [5.67574143e-01, 1.75828263e-01, 5.10376394e-01, 7.56945848e-01,
       1.10105194e-01, 8.17099094e-01, 1.67481646e-01, 5.34076512e-01,
       3.85743469e-01, 2.48623773e-01],
      [6.47432506e-01, 3.73921096e-02, 7.60045826e-01, 5.26940644e-01,
       8.75771224e-01, 5.20718336e-01, 3.50331701e-02, 1.43600971e-01,
       7.95604587e-01, 4.91976053e-01],
      [4.41879272e-01, 3.18434775e-01, 2.84549206e-01, 9.65886295e-01,
       4.32969332e-01, 8.84003043e-01, 6.48163140e-01, 8.58427644e-01,
       8.52449536e-01, 9.56312001e-01],
      [6.97942257e-01, 8.05396914e-01, 7.33127892e-01, 6.05226815e-01,
       7.17354119e-01, 7.15750396e-01, 4.09077927e-02, 5.16110837e-01,
       7.92651355e-01, 2.42962182e-01],
      [4.65147972e-01, 4.34985697e-01, 4.02787179e-01, 1.21839531e-01,
       5.25711536e-01, 4.46248353e-01, 6.63392782e-01, 5.49413085e-01,
       2.75429301e-02, 3.19179893e-02],
      [7.01359808e-01, 7.07581103e-01, 9.59939122e-01, 8.76704693e-01,
       4.68059659e-01, 6.25906527e-01, 4.57181722e-01, 2.22946241e-01,
       3.76677006e-01, 1.03884235e-01],
      [6.66527092e-01, 1.92030147e-01, 4.75467801e-01, 9.67436612e-01,
       3.16689312e-02, 1.51729956e-01, 2.98579186e-01, 9.41806972e-01,
       9.08841789e-01, 1.62000835e-01],
      [9.81117785e-01, 7.50747502e-01, 5.39977074e-01, 9.31702912e-01,
       8.80607128e-01, 3.91316503e-01, 6.56343222e-01, 6.47385120e-01,
       3.26968193e-01, 1.79390177e-01],
      [4.66809869e-01, 2.63281047e-01, 3.55065137e-01, 9.54143941e-01,
```

```
 4.61137861e-01, 6.84891462e-01, 3.36229891e-01, 9.95861053e-01,
 6.58767581e-01, 1.96009472e-01],
[9.81839970e-02, 9.43180561e-01, 9.44777846e-01, 6.21328354e-01,
 1.69914998e-02, 2.25534886e-01, 8.01276803e-01, 8.75459850e-01,
 4.53989804e-01, 3.65520626e-01],
[2.74224997e-01, 1.16970517e-01, 1.15744539e-01, 9.52602684e-01,
 8.08626115e-01, 1.64779365e-01, 2.07050055e-01, 6.55551553e-01,
 7.64664233e-01, 8.10314834e-01],
[1.63337693e-01, 9.84128296e-01, 2.27802068e-01, 5.89415431e-01,
 5.87615728e-01, 9.67361867e-01, 6.57667458e-01, 5.84904253e-01,
 5.18772602e-01, 7.64657557e-01],
[1.06055260e-01, 2.09190114e-03, 9.52488840e-01, 4.98657674e-01,
 3.28335375e-01, 3.68053257e-01, 8.03843319e-01, 3.82370204e-01,
 7.70169199e-01, 4.40461993e-01],
[8.44077468e-01, 7.62040615e-02, 4.81128335e-01, 4.66849715e-01,
 2.64327973e-01, 9.43614721e-01, 9.05028462e-01, 4.43596303e-01,
 9.71596092e-02, 2.06783146e-01],
[2.71491826e-01, 4.84219760e-01, 3.38377118e-01, 7.74136066e-01,
 4.76026595e-01, 8.70370507e-01, 9.95781779e-01, 2.19835952e-01,
 6.11671388e-01, 8.47502291e-01],
[9.45236623e-01, 2.90086418e-01, 7.27042735e-01, 1.50161488e-02,
 8.79142463e-01, 6.39385507e-02, 7.33395398e-01, 9.94610369e-01,
 5.01189768e-01, 2.09333986e-01],
[5.94643593e-01, 6.24149978e-01, 6.68072760e-01, 1.72611743e-01,
 8.98712695e-01, 6.20991349e-01, 4.35687043e-02, 6.84041083e-01,
 1.96084052e-01, 2.73407809e-02],
[5.50953269e-01, 8.13313663e-01, 8.59941125e-01, 1.03520922e-01,
 6.63042784e-01, 7.10075200e-01, 2.94516981e-01, 9.71364021e-01,
 2.78687477e-01, 6.99821860e-02],
[5.19280374e-01, 6.94314897e-01, 2.44659781e-01, 3.38582188e-01,
 5.63627958e-01, 8.86678159e-01, 7.47325897e-01, 2.09591955e-01,
 2.51777083e-01, 5.23880661e-01],
[7.68958688e-01, 6.18761778e-01, 5.01324296e-01, 5.97125351e-01,
 7.56060004e-01, 5.37079811e-01, 8.97752762e-01, 9.47067499e-01,
 9.15354490e-01, 7.54518330e-01],
[2.46321008e-01, 3.85271460e-01, 2.79999942e-01, 6.57660246e-01,
 3.24221611e-01, 7.54391611e-01, 1.13509081e-01, 7.75364757e-01,
 5.85901976e-01, 8.35388660e-01],
[4.30875659e-01, 6.24964476e-01, 5.54412127e-01, 9.75671291e-01,
 7.55474389e-01, 5.44813275e-01, 1.74032092e-01, 9.04114246e-01,
 2.05837786e-01, 6.50043249e-01],
[9.36471879e-01, 2.23579630e-01, 2.25923538e-01, 8.51818919e-01,
 8.27655017e-01, 3.51703346e-01, 2.65096277e-01, 1.27388477e-01,
 9.87936080e-01, 8.35343122e-01],
[8.99391592e-01, 5.13679326e-01, 1.14384830e-01, 5.25803380e-02,
 3.30582112e-01, 9.20330405e-01, 9.47581828e-01, 8.41163874e-01,
 1.58679143e-01, 4.19923156e-01],
```

```
       [2.46242926e-01, 2.05349773e-01, 6.84825838e-01, 4.86111671e-01,
        3.24909657e-01, 1.00214459e-01, 5.44763386e-01, 3.47025156e-01,
        3.91095817e-01, 3.10508728e-01],
       [3.87195200e-01, 5.55859566e-01, 1.41438060e-02, 8.47647011e-01,
        9.21919882e-01, 5.50529718e-01, 2.68021107e-01, 9.90239024e-01,
        3.83194029e-01, 6.93655372e-01],
       [6.89952552e-01, 4.34309065e-01, 1.99158162e-01, 9.66579378e-01,
        6.36908561e-02, 4.85149384e-01, 2.20730707e-01, 2.93974131e-01,
        8.28527331e-01, 3.67265552e-01],
       [8.33482668e-02, 1.96309000e-01, 8.60373437e-01, 9.77028847e-01,
        2.67982155e-01, 6.75408959e-01, 8.11989978e-02, 7.23465621e-01,
        4.16436613e-01, 9.18159902e-01],
       [3.11536163e-01, 9.41466987e-01, 5.03247440e-01, 3.48892927e-01,
        6.47019625e-01, 2.49746203e-01, 2.29763597e-01, 1.96346447e-01,
        9.59899545e-01, 4.92913723e-01],
       [7.51614988e-01, 4.73991871e-01, 5.87540150e-01, 5.84138989e-01,
        9.79886293e-01, 6.68433130e-01, 2.39769474e-01, 1.51976589e-02,
        2.18682140e-01, 4.55519646e-01],
       [3.93420339e-01, 8.12326252e-01, 7.85556734e-01, 8.90959650e-02,
        9.52010751e-01, 5.27456701e-01, 5.96403956e-01, 4.05056775e-01,
        6.49500966e-01, 8.71326327e-01],
       [6.73935950e-01, 9.70098555e-01, 7.01122224e-01, 8.21720719e-01,
        4.50395830e-02, 6.72698498e-01, 6.54752672e-01, 1.01746053e-01,
        8.42387497e-01, 6.14172399e-01],
       [9.83280912e-02, 5.94467103e-01, 4.78415847e-01, 2.33293563e-01,
        1.97560899e-02, 3.65567267e-01, 6.19851053e-01, 3.29279125e-01,
        3.07254642e-01, 7.51121223e-01],
       [7.58624673e-01, 7.18765855e-01, 1.01181954e-01, 5.16165972e-01,
        5.57798684e-01, 7.44804502e-01, 9.03177738e-01, 3.69038880e-01,
        4.28663462e-01, 7.32767463e-01],
       [6.62636399e-01, 5.57869911e-01, 3.50139618e-01, 1.95352346e-01,
        1.83807373e-01, 8.15832913e-02, 8.12008530e-02, 8.45798194e-01,
        3.83672744e-01, 6.07396215e-02],
       [8.96425664e-01, 2.23270476e-01, 2.68124431e-01, 1.94497839e-01,
        9.67501044e-01, 1.12540089e-01, 7.22163260e-01, 9.32088733e-01,
        6.68001294e-01, 8.58726621e-01],
       [2.42447108e-01, 6.73927963e-01, 7.00871348e-01, 4.58332509e-01,
        8.70545626e-01, 6.94386125e-01, 8.94877791e-01, 7.53204346e-01,
        5.20290434e-01, 4.98688221e-01],
       [4.53727633e-01, 2.16468628e-02, 5.35141408e-01, 4.22973245e-01,
        1.57533601e-01, 1.19069695e-01, 4.49351877e-01, 3.99130546e-02,
        9.86579895e-01, 3.78120929e-01],
       [3.82109195e-01, 5.11263013e-02, 4.26672339e-01, 1.57454368e-02,
        3.00936326e-02, 3.39099228e-01, 8.20968926e-01, 4.58821088e-01,
        1.48405796e-02, 1.63220033e-01],
       [7.39922702e-01, 7.38293707e-01, 7.54522920e-01, 3.51669371e-01,
        3.52276951e-01, 8.02075684e-01, 3.98137897e-01, 7.27191031e-01,
```

```
       5.81122994e-01, 3.64341676e-01],
      [8.00065175e-02, 1.16125375e-01, 8.89558733e-01, 4.52340513e-01,
       9.94004548e-01, 3.63896936e-01, 2.49954298e-01, 3.50539327e-01,
       3.43086094e-01, 6.37356758e-01],
      [1.27375638e-02, 7.63268650e-01, 4.16414618e-01, 4.32239205e-01,
       4.81115013e-01, 4.49212462e-01, 4.97470886e-01, 3.45904320e-01,
       4.53346133e-01, 4.04651344e-01],
      [5.18242717e-01, 6.23269081e-01, 2.41040602e-01, 5.08437157e-01,
       5.94621897e-01, 1.69483144e-02, 5.20493746e-01, 2.39293247e-01,
       4.04538542e-01, 8.26530159e-01],
      [3.26235592e-01, 4.83216912e-01, 2.47411542e-02, 3.08750868e-01,
       6.39721096e-01, 3.15161765e-01, 2.05797508e-01, 2.90655673e-01,
       9.54378307e-01, 8.68018195e-02],
      [4.63357776e-01, 5.83869033e-02, 5.38658261e-01, 1.46035731e-01,
       6.34084821e-01, 2.64397472e-01, 6.90915406e-01, 3.47146064e-01,
       4.16848855e-03, 2.94894695e-01]])
```

[51]:
```
# Defining the first dense layer
dense1 = keras.layers.Dense(7, activation='sigmoid')(borrower_features)

# Defining a dense layer with 3 output nodes
dense2 = keras.layers.Dense(3, activation='sigmoid')(dense1)

# Defining a dense layer with 1 output node
predictions = keras.layers.Dense(1, activation ='sigmoid')(dense2)

# Printing the shapes of dense1, dense2, and predictions
print('\n shape of dense1: ', dense1.shape)
print('\n shape of dense2: ', dense2.shape)
print('\n shape of predictions: ', predictions.shape)
```

```
 shape of dense1:  (100, 7)

 shape of dense2:  (100, 3)

 shape of predictions:  (100, 1)
```

With just 8 lines of code, we were able to define 2 dense hidden layers and an output layer. This is the advantage of using high-level operations in tensorflow. Note that each layer has 100 rows because the input data contains 100 examples.

Binary classification problems

Here we will again make use of credit card data. The target variable, default, indicates whether a credit card holder defaults on his or her payment in the following period. Since there are only two options–default or not–this is a binary classification problem. While the dataset has many features, we will focus on just three: the size of the three latest credit card bills. Finally, we will compute predictions from our untrained network, outputs, and compare those the target variable, default.

The tensor of features has been loaded and is available as bill_amounts. Additionally, the constant(), float32, and keras.layers.Dense() operations are available.

[52]:
```
bill_amounts = credit[['BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6']].values
```

[53]:
```
default = credit[['default.payment.next.month']].values
```

[54]:
```
# Constructing input layer from features
inputs = tf.constant(bill_amounts)

# Defining first dense layer
dense1 = tf.keras.layers.Dense(3, activation='relu')(inputs)

# Defining second dense layer
dense2 = tf.keras.layers.Dense(2, activation='relu')(dense1)

# Defining output layer
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(dense2)

# Printing error for first five examples
error = default[:5] - outputs.numpy()[:5]
print(error)
```

```
WARNING:tensorflow:Layer dense_3 is casting an input tensor from dtype float64
to the layer's dtype of float32, which is new behavior in TensorFlow 2.  The
layer has dtype float32 because it's dtype defaults to floatx.

If you intended to run this layer in float32, you can safely ignore this
warning. If in doubt, this warning is likely only an issue if you are porting a
TensorFlow 1.X model to TensorFlow 2.

To change all layers to have dtype float64 by default, call
`tf.keras.backend.set_floatx('float64')`. To change just this layer, pass
dtype='float64' to the layer constructor. If you are the author of this layer,
you can disable autocasting by passing autocast=False to the base Layer
constructor.

[[ 0.5       ]
 [ 0.        ]
 [-1.        ]
 [-1.        ]
 [-0.01589111]]
```

If you run the code several times, you'll notice that the errors change each time. This is because we're using an untrained model with randomly initialized parameters. Furthermore, the errors fall on the interval between -1 and 1 because default is a binary variable that takes on values of 0 and 1 and outputs is a probability between 0 and 1.

Multiclass classification problems

Here we expand beyond binary classification to cover multiclass problems. A multiclass problem has targets that can take on three or more values. In the credit card dataset, the education variable can take on 6 different values, each corresponding to a different level of education. We will use that as our target in this exercise and will also expand the feature set from 3 to 10 columns.

As in the previous problem, we will define an input layer, dense layers, and an output layer. We will also print the untrained model's predictions, which are probabilities assigned to the classes. The tensor of features has been loaded and is available as borrower_features. Additionally, the constant(), float32, and keras.layers.Dense() operations are available.

[55]:
```
borrower_featuers = credit[['LIMIT_BAL', 'BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3',␣
↪'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1', 'PAY_AMT2', 'PAY_AMT3']].
↪values
```

[56]:
```
# Constructing input layer from borrower features
inputs = tf.constant(borrower_features)

# Defining first dense layer
dense1 = tf.keras.layers.Dense(10, activation='sigmoid')(inputs)

# Defining second dense layer
dense2 = tf.keras.layers.Dense(8, activation='relu')(dense1)

# Defining output layer
outputs = tf.keras.layers.Dense(6, activation='softmax')(dense2)

# Printing first five predictions
print(outputs.numpy()[:5])
```

```
[[0.18208596 0.10689667 0.1270582  0.27925304 0.17880994 0.12589628]
 [0.17899178 0.10902441 0.12608396 0.27839375 0.17957082 0.12793528]
 [0.18416762 0.10528424 0.12741585 0.28098127 0.17839135 0.1237596 ]
 [0.17955816 0.10763546 0.12778527 0.278163   0.17960513 0.12725301]
 [0.1930414  0.11306502 0.12848683 0.26436675 0.17367715 0.12736288]]
```

Notice that each row of outputs sums to one. This is because a row contains the predicted class probabilities for one example. As above, our predictions are not yet informative, since we are using an untrained model with randomly initialized parameters. This is why the model tends to assign similar probabilities to each class.

Initialization in TensorFlow

A good initialization can reduce the amount of time needed to find the global minimum. I will initialize weights and biases for a neural network that will be used to predict credit card default decisions. To build intuition, we will use the low-level, linear algebraic approach, rather than making use of convenience functions and high-level keras operations. We will also expand the set of input features from 3 to 23. Several operations have been imported from tensorflow: Variable(), random(), and ones().

[57]:
```
# Defining the layer 1 weights
w1 = tf.Variable(tf.random.normal([23, 7]))

# Initializing the layer 1 bias
b1 = tf.Variable(ones([7]))

# Defining the layer 2 weights
w2 = tf.Variable(tf.random.normal([7, 1]))

# Defining the layer 2 bias
b2 = tf.Variable(0.0)
```

Next I will start where we've ended and will finish constructing the neural network.

Defining the model and loss function

I will train a neural network to predict whether a credit card holder will default. The features and targets I will use to train the network are available as borrower_features and default. I defined the weights and biases above.

Note that the predictions layer is defined as (layer1∗w2+b2), where is the sigmoid activation, layer1 is a tensor of nodes for the first hidden dense layer, w2 is a tensor of weights, and b2 is the bias tensor.

The trainable variables are w1, b1, w2, and b2. Additionally, the following operations have been imported: keras.activations.relu() and keras.layers.Dropout().

[58]:
```
borrower_features = credit[['LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE',␣
↪'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1',␣
↪'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',␣
↪'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6']].values
```

[59]:
```
# Defining the model
def model(w1, b1, w2, b2, features = borrower_features):
    # Applying relu activation functions to layer 1
    layer1 = tf.keras.activations.relu(matmul(features, w1) + b1)
    # Applying dropout
    dropout = tf.keras.layers.Dropout(0.25)(layer1)
    return keras.activations.sigmoid(matmul(dropout, w2) + b2)

# Defining the loss function
def loss_function(w1, b1, w2, b2, features = borrower_features, targets =␣
↪default):
    predictions = model(w1, b1, w2, b2)
    # Passing targets and predictions to the cross entropy loss
    return keras.losses.binary_crossentropy(targets, predictions)
```

One of the benefits of using tensorflow is that we have the option to customize models down to the linear algebraic-level, as I've shown above. If we print w1, we can see that the objects we're working with are simply tensors.

Training neural networks with TensorFlow

Above I defined a model, model(w1, b1, w2, b2, features), and a loss function, loss_function(w1, b1, w2, b2, features, targets), both of which are available. I will now train the model and then evaluate its performance by predicting default outcomes in a test set, which consists of test_features and test_targets and are available. The trainable variables are w1, b1, w2, and b2. Additionally, the following operations have been imported: keras.activations.relu() and keras.layers.Dropout().

[60]:
```
test_features = tf.Variable([[1.4000000e-01, 1.0000000e+00, 1.6666667e-01, 2.
↪3578344e-02,
        2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
        2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
        2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
       [2.9999999e-02, 5.0000000e-01, 1.6666667e-01, 0.0000000e+00,
        0.0000000e+00, 2.7980173e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
        2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
        2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
       [5.0000001e-02, 1.0000000e+00, 3.3333334e-01, 9.4313379e-03,
        6.6666668e-03, 7.4349442e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
        2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
        2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
       [7.0000000e-02, 1.0000000e+00, 3.3333334e-01, 7.5450698e-03,
        3.3333334e-03, 0.0000000e+00, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
        2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
        2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
       [5.0000001e-02, 1.0000000e+00, 5.0000000e-01, 1.7768640e-02,
        3.3333333e-06, 2.8277570e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
        2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
        2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
       [2.3999999e-01, 1.0000000e+00, 3.3333334e-01, 6.8801609e-03,
        0.0000000e+00, 3.0978934e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
```

```
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [1.4000000e-01, 1.0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [2.9999999e-02, 5.0000000e-01, 1.6666667e-01, 0.0000000e+00,
       0.0000000e+00, 2.7980173e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [5.0000001e-02, 1.0000000e+00, 3.3333334e-01, 9.4313379e-03,
       6.6666668e-03, 7.4349442e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [7.0000000e-02, 1.0000000e+00, 3.3333334e-01, 7.5450698e-03,
       3.3333334e-03, 0.0000000e+00, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [5.0000001e-02, 1.0000000e+00, 5.0000000e-01, 1.7768640e-02,
       3.3333333e-06, 2.8277570e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [2.3999999e-01, 1.0000000e+00, 3.3333334e-01, 6.8801609e-03,
       0.0000000e+00, 3.0978934e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
```

```
      [1.4000000e-01, 1.0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [2.9999999e-02, 5.0000000e-01, 1.6666667e-01, 0.0000000e+00,
       0.0000000e+00, 2.7980173e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [5.0000001e-02, 1.0000000e+00, 3.3333334e-01, 9.4313379e-03,
       6.6666668e-03, 7.4349442e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [7.0000000e-02, 1.0000000e+00, 3.3333334e-01, 7.5450698e-03,
       3.3333334e-03, 0.0000000e+00, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [5.0000001e-02, 1.0000000e+00, 5.0000000e-01, 1.7768640e-02,
       3.3333333e-06, 2.8277570e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [2.3999999e-01, 1.0000000e+00, 3.3333334e-01, 6.8801609e-03,
       0.0000000e+00, 3.0978934e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [1.4000000e-01, 1.0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
```

```
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [2.9999999e-02, 5.0000000e-01, 1.6666667e-01, 0.0000000e+00,
       0.0000000e+00, 2.7980173e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [5.0000001e-02, 1.0000000e+00, 3.3333334e-01, 9.4313379e-03,
       6.6666668e-03, 7.4349442e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [7.0000000e-02, 1.0000000e+00, 3.3333334e-01, 7.5450698e-03,
       3.3333334e-03, 0.0000000e+00, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02],
      [5.0000001e-02, 1.0000000e+00, 5.0000000e-01, 1.7768640e-02,
       3.3333333e-06, 2.8277570e-03, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.4000000e-01, 1.
↪0000000e+00, 1.6666667e-01, 2.3578344e-02,
       2.0000000e-02, 1.2391574e-02, 1.2391574e-02, 1.2391574e-02, 1.
↪2391574e-02]])
```

[61]:
```
test_targets = tf.Variable([[0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
```

```
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.]])
```

[63]:
```
b2 = tf.Variable([[-1.2979275]], dtype=tf.double)
```

[ ]:
```
# Training the model
for j in range(100):
    # Complete the optimizer
    opt.minimize(lambda: loss_function(w1, b1, w2, b2),
                 var_list=[w1, b1, w2, b2])

# Making predictions with model
model_predictions = model(w1, b1, w2, b2, test_features)

# Constructing the confusion matrix
confusion_matrix(test_targets, model_predictions)
```

One of the benefits of using tensorflow is that we have the option to customize models down to the linear algebraic-level. If you print w1, you can see that the objects we're working with are simply tensors.

The sequential model in Keras

I have used components of the keras API in tensorflow to define a neural network, but we stopped short of using its full capabilities to streamline model definition and training. Now, I will use the keras sequential model API to define a neural network that can be used to classify images of sign language letters. We will also use the .summary() method to print the model's architecture, including the shape and number of parameters associated with each layer.

Note that the images were reshaped from (28, 28) to (784,), so that they could be used as inputs to a dense layer. Additionally, note that keras has been imported from tensorflow.

[65]:
```
from tensorflow import keras
```

[66]:
```
# Defining a Keras sequential model
model = keras.Sequential()

# Defining the first dense layer
model.add(keras.layers.Dense(16, activation='relu', input_shape=(784,)))

# Defining the second dense layer
```

```
model.add(keras.layers.Dense(8, activation='relu'))

# Defining the output layer
model.add(keras.layers.Dense(4, activation='softmax'))

# Printing the model architecture
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 16)                12560
_________________________________________________________________
dense_10 (Dense)             (None, 8)                 136
_________________________________________________________________
dense_11 (Dense)             (None, 4)                 36
=================================================================
Total params: 12,732
Trainable params: 12,732
Non-trainable params: 0
_________________________________________________________________
None
```

Notice that we've defined a model, but we haven't compiled it. The compilation step in keras allows us to set the optimizer, loss function, and other useful training parameters in a single line of code. Furthermore, the .summary() method allows us to view the model's architecture.

Compiling a sequential model

Here I will work towards classifying letters from the Sign Language MNIST dataset; however, I will adopt a different network architecture than what I used above. There will be fewer layers, but more nodes. We will also apply dropout to prevent overfitting. Finally, we will compile the model to use the adam optimizer and the categorical_crossentropy loss. I will also use a method in keras to summarize the model's architecture. Note that keras has been imported from tensorflow and a sequential keras model has been defined as model.

[67]:
```
# Defining the first dense layer
model.add(tf.keras.layers.Dense(16, activation='sigmoid', input_shape=(784,)))

# Applying dropout to the first layer's output
model.add(tf.keras.layers.Dropout(0.25))


# Defining the output layer
model.add(tf.keras.layers.Dense(4, activation='softmax'))


# Compiling the model
```

```
model.compile('adam', loss='categorical_crossentropy')

# Printing a model summary
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 16)                12560
_________________________________________________________________
dense_10 (Dense)             (None, 8)                 136
_________________________________________________________________
dense_11 (Dense)             (None, 4)                 36
_________________________________________________________________
dense_12 (Dense)             (None, 16)                80
_________________________________________________________________
dropout (Dropout)            (None, 16)                0
_________________________________________________________________
dense_13 (Dense)             (None, 4)                 68
=================================================================
Total params: 12,880
Trainable params: 12,880
Non-trainable params: 0
_________________________________________________________________
None
```

We've now defined and compiled a neural network using the keras sequential model. Notice that printing the .summary() method shows the layer type, output shape, and number of parameters of each layer.

Defining a multiple input model

In some cases, the sequential API will not be sufficiently flexible to accommodate your desired model architecture and we will need to use the functional API instead. If, for instance, we want to train two models with different architectures jointly, we will need to use the functional API to do this. Below, we will see how to do this. We will also use the .summary() method to examine the joint model's architecture.

Note that keras has been imported from tensorflow for you. Additionally, the input layers of the first and second models have been defined as m1_inputs and m2_inputs, respectively. Note that the two models have the same architecture, but one of them uses a sigmoid activation in the first layer and the other uses a relu.

[68]:
```
# Defining model 1 input layer shape
m1_inputs = tf.keras.Input(shape=(28*28,))
# Defining model 2 input layer shape
m2_inputs = tf.keras.Input(shape=(10,))
```

[69]:
```
# For model 1, passing the input layer to layer 1 and layer 1 to layer 2
m1_layer1 = keras.layers.Dense(12, activation='sigmoid')(m1_inputs)
m1_layer2 = keras.layers.Dense(4, activation='softmax')(m1_layer1)

# For model 2, passing the input layer to layer 1 and layer 1 to layer 2
m2_layer1 = keras.layers.Dense(12, activation='relu')(m2_inputs)
m2_layer2 = keras.layers.Dense(4, activation='softmax')(m2_layer1)

# Merging model outputs and defining a functional model
merged = keras.layers.add([m1_layer2, m2_layer2])
model = keras.Model(inputs=[m1_inputs, m2_inputs], outputs=merged)

# Printing a model summary
print(model.summary())
```

```
Model: "model"
__________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
input_1 (InputLayer)            [(None, 784)]        0
__________________________________________________________________________________
__________________
input_2 (InputLayer)            [(None, 10)]         0
__________________________________________________________________________________
__________________
dense_14 (Dense)                (None, 12)           9420        input_1[0][0]
__________________________________________________________________________________
__________________
dense_16 (Dense)                (None, 12)           132         input_2[0][0]
__________________________________________________________________________________
__________________
dense_15 (Dense)                (None, 4)            52          dense_14[0][0]
__________________________________________________________________________________
__________________
dense_17 (Dense)                (None, 4)            52          dense_16[0][0]
__________________________________________________________________________________
__________________
add (Add)                       (None, 4)            0           dense_15[0][0]
                                                                 dense_17[0][0]
==================================================================================
==================
Total params: 9,656
Trainable params: 9,656
Non-trainable params: 0
__________________________________________________________________________________
```

```
------------------
None
```

Notice that the .summary() method yields a new column: connected to. This column tells us how layers connect to each other within the network. We can see that dense_20, for instance, is connected to the input_2 layer. We can also see that the add layer, which merged the two models, connected to both dense_21 and dense_23.

[70]:
```
image = pd.read_csv('slmnist.csv')
```

[71]:
```
sign_language_labels = image.iloc[:, :4]
```

[72]:
```
sign_language_labels.shape
```

[72]:
```
(1999, 4)
```

[73]:
```
sign_language_features = image.iloc[:, :-4]
```

[74]:
```
sign_language_features.shape
```

[74]:
```
(1999, 781)
```

[75]:
```
# Defining a sequential model
model = keras.Sequential()

# Defining a hidden layer
model.add(keras.layers.Dense(16, activation='relu', input_shape=(781,)))

# Defining the output layer
model.add(keras.layers.Dense(4, activation='softmax'))# Compile the

# Compiling the model
model.compile('SGD', loss='categorical_crossentropy')

# Completing the fitting operation
model.fit(sign_language_features, sign_language_labels, epochs=5)
```

```
Train on 1999 samples
Epoch 1/5
1999/1999 [==============================] - 1s 700us/sample - loss: nan
Epoch 2/5
1999/1999 [==============================] - 0s 116us/sample - loss: nan
Epoch 3/5
1999/1999 [==============================] - 0s 129us/sample - loss: nan
Epoch 4/5
1999/1999 [==============================] - 0s 93us/sample - loss: nan
Epoch 5/5
1999/1999 [==============================] - 0s 132us/sample - loss: nan
```

[75]: <tensorflow.python.keras.callbacks.History at 0x636a94510>

Notice that our only measure of performance improvement was the value of the loss function in the training sample, which is not particularly informative. We will improve on this in the next.

[76]:
```
# Defining sequential model
model = keras.Sequential()

# Defining the first layer
model.add(keras.layers.Dense(32, activation='sigmoid', input_shape=(781,)))

# Adding activation function to classifier
model.add(keras.layers.Dense(4, activation='softmax'))

# Setting the optimizer, loss function, and metrics
model.compile(optimizer='RMSprop', loss='categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# Adding the number of epochs and the validation split
model.fit(sign_language_features, sign_language_labels, epochs=10,␣
 ↪validation_split=0.10)
```

```
Train on 1799 samples, validate on 200 samples
Epoch 1/10
1799/1799 [==============================] - 1s 437us/sample - loss: 657.4326 -
accuracy: 0.1968 - val_loss: 669.3862 - val_accuracy: 0.1250
Epoch 2/10
1799/1799 [==============================] - 0s 92us/sample - loss: 670.2862 -
accuracy: 0.1384 - val_loss: 668.3160 - val_accuracy: 0.1250
Epoch 3/10
1799/1799 [==============================] - 0s 101us/sample - loss: 669.4290 -
accuracy: 0.1384 - val_loss: 668.1704 - val_accuracy: 0.1250
Epoch 4/10
1799/1799 [==============================] - 0s 100us/sample - loss: 668.8502 -
accuracy: 0.1384 - val_loss: 666.6482 - val_accuracy: 0.1250
Epoch 5/10
1799/1799 [==============================] - 0s 113us/sample - loss: 667.9564 -
accuracy: 0.1384 - val_loss: 666.2109 - val_accuracy: 0.1250
Epoch 6/10
1799/1799 [==============================] - 0s 96us/sample - loss: 667.2323 -
accuracy: 0.1384 - val_loss: 666.4086 - val_accuracy: 0.1250
Epoch 7/10
1799/1799 [==============================] - 0s 93us/sample - loss: 666.5401 -
accuracy: 0.1384 - val_loss: 664.8201 - val_accuracy: 0.1250
Epoch 8/10
1799/1799 [==============================] - 0s 230us/sample - loss: 665.8054 -
accuracy: 0.1384 - val_loss: 664.7102 - val_accuracy: 0.1250
Epoch 9/10
```

```
1799/1799 [==============================] - 0s 148us/sample - loss: 665.5052 -
accuracy: 0.1384 - val_loss: 663.4086 - val_accuracy: 0.1250
Epoch 10/10
1799/1799 [==============================] - 0s 128us/sample - loss: 664.6294 -
accuracy: 0.1384 - val_loss: 663.3066 - val_accuracy: 0.1250
```

[76]:
```
<tensorflow.python.keras.callbacks.History at 0x6369b8f90>
```

With the keras API, we only needed 14 lines of code to define, compile, train, and validate a model. It can be noticed that your model performed quite well. In just 10 epochs, we achieved a classification accuracy of over 70% in the validation sample!

Overfitting detection

I'll work with a small subset of the examples from the original sign language letters dataset. A small sample, coupled with a heavily-parameterized model, will generally lead to overfitting. This means that the model will simply memorize the class of each example, rather than identifying features that generalize to many examples.

I will detect overfitting by checking whether the validation sample loss is substantially higher than the training sample loss and whether it increases with further training. With a small sample and a high learning rate, the model will struggle to converge on an optimum. WE will set a low learning rate for the optimizer, which will make it easier to identify overfitting.

[77]:
```
# Defining sequential model
model = keras.Sequential()

# Defining the first layer
model.add(keras.layers.Dense(1024, activation='relu', input_shape=(781,)))

# Adding activation function to classifier
model.add(keras.layers.Dense(4, activation='softmax'))

# Finishing the model compilation
model.compile(optimizer=keras.optimizers.Adam(lr=0.001),
              loss='categorical_crossentropy', metrics=['accuracy'])

# Completing the model fit operation
model.fit(sign_language_features, sign_language_labels, epochs=50,␣
 ↪validation_split=0.5)
```

```
Train on 999 samples, validate on 1000 samples
Epoch 1/50
999/999 [==============================] - 1s 1ms/sample - loss: 1210250.5166 -
accuracy: 0.1421 - val_loss: 2667354.6440 - val_accuracy: 0.1610
Epoch 2/50
999/999 [==============================] - 1s 511us/sample - loss: 4115103.8008
- accuracy: 0.1421 - val_loss: 6067762.8360 - val_accuracy: 0.1610
Epoch 3/50
999/999 [==============================] - 1s 504us/sample - loss: 7587791.5926
```

```
- accuracy: 0.1421 - val_loss: 9826091.9280 - val_accuracy: 0.1610
Epoch 4/50
999/999 [==============================] - 1s 506us/sample - loss: 11354326.8198
- accuracy: 0.1421 - val_loss: 13774983.6560 - val_accuracy: 0.1610
Epoch 5/50
999/999 [==============================] - 0s 456us/sample - loss: 15243603.9269
- accuracy: 0.1421 - val_loss: 17838146.5760 - val_accuracy: 0.1610
Epoch 6/50
999/999 [==============================] - 0s 498us/sample - loss: 19184612.6567
- accuracy: 0.1421 - val_loss: 21918285.6000 - val_accuracy: 0.1610
Epoch 7/50
999/999 [==============================] - 0s 485us/sample - loss: 23405212.0801
- accuracy: 0.1421 - val_loss: 26220958.0160 - val_accuracy: 0.1610
Epoch 8/50
999/999 [==============================] - 0s 421us/sample - loss: 27315012.0941
- accuracy: 0.1421 - val_loss: 30156526.4960 - val_accuracy: 0.1610
Epoch 9/50
999/999 [==============================] - 0s 328us/sample - loss: 31526075.0110
- accuracy: 0.1421 - val_loss: 34674660.4800 - val_accuracy: 0.1610
Epoch 10/50
999/999 [==============================] - 0s 426us/sample - loss: 35748539.9359
- accuracy: 0.1421 - val_loss: 38868562.6560 - val_accuracy: 0.1610
Epoch 11/50
999/999 [==============================] - 0s 317us/sample - loss: 39846583.0230
- accuracy: 0.1421 - val_loss: 43065012.4800 - val_accuracy: 0.1610
Epoch 12/50
999/999 [==============================] - 0s 399us/sample - loss: 43667161.1411
- accuracy: 0.1421 - val_loss: 47096236.8000 - val_accuracy: 0.1610
Epoch 13/50
999/999 [==============================] - 0s 400us/sample - loss: 47761425.0210
- accuracy: 0.1421 - val_loss: 51279398.8480 - val_accuracy: 0.1610
Epoch 14/50
999/999 [==============================] - 0s 387us/sample - loss: 51953587.1191
- accuracy: 0.1421 - val_loss: 55546007.5520 - val_accuracy: 0.1610
Epoch 15/50
999/999 [==============================] - 0s 324us/sample - loss: 55781492.4084
- accuracy: 0.1421 - val_loss: 59484410.5600 - val_accuracy: 0.1610
Epoch 16/50
999/999 [==============================] - 0s 303us/sample - loss: 59985926.5265
- accuracy: 0.1421 - val_loss: 63719652.7040 - val_accuracy: 0.1610
Epoch 17/50
999/999 [==============================] - 0s 307us/sample - loss: 63780902.2543
- accuracy: 0.1421 - val_loss: 67559073.2480 - val_accuracy: 0.1610
Epoch 18/50
999/999 [==============================] - 0s 319us/sample - loss: 67754764.7888
- accuracy: 0.1421 - val_loss: 71688194.5280 - val_accuracy: 0.1610
Epoch 19/50
999/999 [==============================] - 0s 336us/sample - loss: 71768148.4925
```

```
- accuracy: 0.1421 - val_loss: 75655482.8800 - val_accuracy: 0.1610
Epoch 20/50
999/999 [==============================] - 0s 436us/sample - loss: 75731278.2783
- accuracy: 0.1421 - val_loss: 80107552.7040 - val_accuracy: 0.1610
Epoch 21/50
999/999 [==============================] - 0s 449us/sample - loss: 79837325.9580
- accuracy: 0.1421 - val_loss: 83879291.8400 - val_accuracy: 0.1610
Epoch 22/50
999/999 [==============================] - 0s 447us/sample - loss: 83545927.1111
- accuracy: 0.1421 - val_loss: 87916081.4720 - val_accuracy: 0.1610
Epoch 23/50
999/999 [==============================] - 1s 535us/sample - loss: 87740232.6567
- accuracy: 0.1421 - val_loss: 92296674.0480 - val_accuracy: 0.1610
Epoch 24/50
999/999 [==============================] - 0s 430us/sample - loss: 91848345.4895
- accuracy: 0.1421 - val_loss: 96376208.0000 - val_accuracy: 0.1610
Epoch 25/50
999/999 [==============================] - 0s 377us/sample - loss: 95813699.3473
- accuracy: 0.1421 - val_loss: 100318471.2960 - val_accuracy: 0.1610
Epoch 26/50
999/999 [==============================] - 1s 587us/sample - loss: 99635086.0861
- accuracy: 0.1421 - val_loss: 104326243.8400 - val_accuracy: 0.1610
Epoch 27/50
999/999 [==============================] - 0s 461us/sample - loss:
103766366.0701 - accuracy: 0.1421 - val_loss: 108690488.2560 - val_accuracy:
0.1610
Epoch 28/50
999/999 [==============================] - 0s 431us/sample - loss:
107198157.2853 - accuracy: 0.1421 - val_loss: 112078790.0800 - val_accuracy:
0.1610
Epoch 29/50
999/999 [==============================] - 0s 417us/sample - loss:
111456695.4234 - accuracy: 0.1421 - val_loss: 116399406.9760 - val_accuracy:
0.1610
Epoch 30/50
999/999 [==============================] - 0s 476us/sample - loss:
115473243.2593 - accuracy: 0.1421 - val_loss: 120521988.5440 - val_accuracy:
0.1610
Epoch 31/50
999/999 [==============================] - 0s 445us/sample - loss:
118752363.0991 - accuracy: 0.1421 - val_loss: 123908740.9280 - val_accuracy:
0.1610
Epoch 32/50
999/999 [==============================] - 0s 447us/sample - loss:
122764463.9439 - accuracy: 0.1421 - val_loss: 128212590.0160 - val_accuracy:
0.1610
Epoch 33/50
999/999 [==============================] - 0s 411us/sample - loss:
```

```
126061477.6537 - accuracy: 0.1421 - val_loss: 131400030.4000 - val_accuracy:
0.1610
Epoch 34/50
999/999 [==============================] - 0s 423us/sample - loss:
131160077.8859 - accuracy: 0.1421 - val_loss: 136860384.5760 - val_accuracy:
0.1610
Epoch 35/50
999/999 [==============================] - 1s 502us/sample - loss:
134375388.1161 - accuracy: 0.1421 - val_loss: 139965717.1200 - val_accuracy:
0.1610
Epoch 36/50
999/999 [==============================] - 0s 485us/sample - loss:
138406061.4695 - accuracy: 0.1421 - val_loss: 144080337.8560 - val_accuracy:
0.1610
Epoch 37/50
999/999 [==============================] - 0s 499us/sample - loss:
141406816.1121 - accuracy: 0.1421 - val_loss: 147119511.6800 - val_accuracy:
0.1610
Epoch 38/50
999/999 [==============================] - 0s 477us/sample - loss:
145804099.9560 - accuracy: 0.1421 - val_loss: 151907339.5200 - val_accuracy:
0.1610
Epoch 39/50
999/999 [==============================] - 1s 556us/sample - loss:
149747343.5516 - accuracy: 0.1421 - val_loss: 155832131.8400 - val_accuracy:
0.1610
Epoch 40/50
999/999 [==============================] - 1s 502us/sample - loss:
154311255.2873 - accuracy: 0.1421 - val_loss: 160474079.7440 - val_accuracy:
0.1610
Epoch 41/50
999/999 [==============================] - 0s 409us/sample - loss:
157301465.2573 - accuracy: 0.1421 - val_loss: 163269972.2240 - val_accuracy:
0.1610
Epoch 42/50
999/999 [==============================] - 0s 414us/sample - loss:
161950168.0080 - accuracy: 0.1421 - val_loss: 168723545.3440 - val_accuracy:
0.1610
Epoch 43/50
999/999 [==============================] - 0s 473us/sample - loss:
166263376.0480 - accuracy: 0.1421 - val_loss: 172592166.9120 - val_accuracy:
0.1610
Epoch 44/50
999/999 [==============================] - 0s 477us/sample - loss:
169767692.6847 - accuracy: 0.1421 - val_loss: 176203097.7280 - val_accuracy:
0.1610
Epoch 45/50
999/999 [==============================] - 0s 452us/sample - loss:
```

```
173327947.4194 - accuracy: 0.1421 - val_loss: 179720843.5200 - val_accuracy:
0.1610
Epoch 46/50
999/999 [==============================] - 0s 420us/sample - loss:
178057406.6707 - accuracy: 0.1421 - val_loss: 184697008.5120 - val_accuracy:
0.1610
Epoch 47/50
999/999 [==============================] - 0s 412us/sample - loss:
180780046.5746 - accuracy: 0.1421 - val_loss: 187641498.1120 - val_accuracy:
0.1610
Epoch 48/50
999/999 [==============================] - 0s 426us/sample - loss:
186545228.6366 - accuracy: 0.1421 - val_loss: 193897925.3760 - val_accuracy:
0.1610
Epoch 49/50
999/999 [==============================] - 0s 462us/sample - loss:
189916284.0601 - accuracy: 0.1421 - val_loss: 196737339.3920 - val_accuracy:
0.1610
Epoch 50/50
999/999 [==============================] - 0s 399us/sample - loss:
194534844.5085 - accuracy: 0.1421 - val_loss: 201680145.2800 - val_accuracy:
0.1610
```

[77]:
```
<tensorflow.python.keras.callbacks.History at 0x1a3810b9d0>
```

[78]:
```
sign_language_features.shape
```

[78]:
```
(1999, 781)
```

It can be noticed that the validation loss, val_loss, was substantially higher than the training loss, loss. Furthermore, if val_loss started to increase before the training process was terminated, then we may have overfitted. When this happens, we will want to try decreasing the number of epochs.

Evaluating models

Two models have been trained and are available: large_model, which has many parameters; and small_model, which has fewer parameters. Both models have been trained using train_features and train_labels. A separate test set, which consists of test_features and test_labels, is also available.

The goal is to evaluate relative model performance and also determine whether either model exhibits signs of overfitting. We will do this by evaluating large_model and small_model on both the train and test sets. For each model, we can do this by applying the .evaluate(x, y) method to compute the loss for features x and labels y. We will then compare the four losses generated.

[ ]:
```
# Evaluating the small model using the train data
small_train = small_model.evaluate(train_features, train_labels)

# Evaluating the small model using the test data
small_test = small_model.evaluate(test_features, test_labels)
```

```
# Evaluating the large model using the train data
large_train = large_model.evaluate(train_features, train_labels)

# Evaluating the large model using the test data
large_test = large_model.evaluate(test_features, test_labels)

# Print losses
print('\n Small - Train: {}, Test: {}'.format(small_train, small_test))
print('Large - Train: {}, Test: {}'.format(large_train, large_test))
```

Preparing to train with Estimators

I'll return to the King County housing transaction dataset. I will again develop and train a machine learning model to predict house prices; however, this time, I'll do it using the estimator API. Rather than completing everything in one step, I'll break this procedure down into parts. I'll begin by defining the feature columns and loading the data. Next I'll define and train a premade estimator. Note that feature_column has been imported from tensorflow. Additionally, numpy has been imported as np, and the Kings County housing dataset is available as a pandas DataFrame: housing.

[80]:
```
from tensorflow import feature_column
```

[ ]:
```
# Define feature columns for bedrooms and bathrooms
bedrooms = feature_column.numeric_column("bedrooms")
bathrooms = feature_column.numeric_column("bathrooms")

# Define the list of feature columns
feature_list = [bedrooms, bathrooms]

def input_fn():
    # Define the labels
    labels = np.array(housing['price'])
    # Define the features
    features = {'bedrooms':np.array(housing['bedrooms']),
                'bathrooms':np.array(housing['bathrooms'])}
    return features, labels
```

Next I'll use the feature columns and data input function to define and train an estimator.

Defining Estimators

Above I defined a list of feature columns, feature_list, and a data input function, input_fn(). Below I will build on that work by defining an estimator that makes use of input data.

Using a deep neural network regressor with 2 nodes in both the first and second hidden layers and 1 training step.

[82]:
```
from tensorflow import estimator
```

[83]:
```
tf.keras.backend.set_floatx('float64')
```

[84]:
```
# Defining the model and setting the number of steps
model = estimator.DNNRegressor(feature_columns=feature_list, hidden_units=[2,2])
model.train(input_fn, steps=1)
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory:
/var/folders/_0/4_8_5bqn44q4jyrxjy8x922c0000gn/T/tmp1tctyqvs
INFO:tensorflow:Using config: {'_model_dir':
'/var/folders/_0/4_8_5bqn44q4jyrxjy8x922c0000gn/T/tmp1tctyqvs',
'_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_steps':
None, '_save_checkpoints_secs': 600, '_session_config': allow_soft_placement:
true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None, '_device_fn': None,
'_protocol': None, '_eval_distribute': None, '_experimental_distribute': None,
'_experimental_max_worker_delay_secs': None, '_session_creation_timeout_secs':
7200, '_service': None, '_cluster_spec': ClusterSpec({}), '_task_type':
'worker', '_task_id': 0, '_global_id_in_cluster': 0, '_master': '',
'_evaluation_master': '', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
WARNING:tensorflow:From /opt/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/ops/resource_variable_ops.py:1635: calling
BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops)
with constraint is deprecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
WARNING:tensorflow:From /opt/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/training/training_util.py:236:
Variable.initialized_value (from tensorflow.python.ops.variables) is deprecated
and will be removed in a future version.
Instructions for updating:
Use Variable.read_value. Variables in 2.X are initialized automatically both in
eager and graph (inside tf.defun) contexts.
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:Layer hiddenlayer_0 is casting an input tensor from dtype
float32 to the layer's dtype of float64, which is new behavior in TensorFlow 2.
The layer has dtype float64 because it's dtype defaults to floatx.

If you intended to run this layer in float64, you can safely ignore this
warning. If in doubt, this warning is likely only an issue if you are porting a
```

```
TensorFlow 1.X model to TensorFlow 2.

To change all layers to have dtype float32 by default, call
`tf.keras.backend.set_floatx('float32')`. To change just this layer, pass
dtype='float32' to the layer constructor. If you are the author of this layer,
you can disable autocasting by passing autocast=False to the base Layer
constructor.

WARNING:tensorflow:From /opt/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/keras/optimizer_v2/adagrad.py:103: calling
Constant.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated
and will be removed in a future version.
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the
constructor
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 0 into
/var/folders/_0/4_8_5bqn44q4jyrxjy8x922c0000gn/T/tmp1tctyqvs/model.ckpt.
INFO:tensorflow:loss = 426470500000.0, step = 0
INFO:tensorflow:Saving checkpoints for 1 into
/var/folders/_0/4_8_5bqn44q4jyrxjy8x922c0000gn/T/tmp1tctyqvs/model.ckpt.
INFO:tensorflow:Loss for final step: 426470500000.0.
```

[84]:
```
<tensorflow_estimator.python.estimator.canned.dnn.DNNRegressorV2 at
0x1a38288f50>
```

Modifying the code to use a LinearRegressor(), removing the hidden_units, and setting the number of steps to 2.

[85]:
```
# Defining the model and setting the number of steps
model = estimator.LinearRegressor(feature_columns=feature_list)
model.train(input_fn, steps=2)
```

```
INFO:tensorflow:Using default config.
WARNING:tensorflow:Using temporary folder as model directory:
/var/folders/_0/4_8_5bqn44q4jyrxjy8x922c0000gn/T/tmp9sxy30s3
INFO:tensorflow:Using config: {'_model_dir':
'/var/folders/_0/4_8_5bqn44q4jyrxjy8x922c0000gn/T/tmp9sxy30s3',
'_tf_random_seed': None, '_save_summary_steps': 100, '_save_checkpoints_steps':
None, '_save_checkpoints_secs': 600, '_session_config': allow_soft_placement:
true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
```

```
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000,
'_log_step_count_steps': 100, '_train_distribute': None, '_device_fn': None,
'_protocol': None, '_eval_distribute': None, '_experimental_distribute': None,
'_experimental_max_worker_delay_secs': None, '_session_creation_timeout_secs':
7200, '_service': None, '_cluster_spec': ClusterSpec({}), '_task_type':
'worker', '_task_id': 0, '_global_id_in_cluster': 0, '_master': '',
'_evaluation_master': '', '_is_chief': True, '_num_ps_replicas': 0,
'_num_worker_replicas': 1}
INFO:tensorflow:Calling model_fn.
WARNING:tensorflow:From /opt/anaconda3/lib/python3.7/site-
packages/tensorflow_core/python/feature_column/feature_column_v2.py:518:
Layer.add_variable (from tensorflow.python.keras.engine.base_layer) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use `layer.add_weight` method instead.
INFO:tensorflow:Done calling model_fn.
INFO:tensorflow:Create CheckpointSaverHook.
INFO:tensorflow:Graph was finalized.
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Saving checkpoints for 0 into
/var/folders/_0/4_8_5bqn44q4jyrxjy8x922c0000gn/T/tmp9sxy30s3/model.ckpt.
INFO:tensorflow:loss = 426471500000.0, step = 0
INFO:tensorflow:Saving checkpoints for 2 into
/var/folders/_0/4_8_5bqn44q4jyrxjy8x922c0000gn/T/tmp9sxy30s3/model.ckpt.
INFO:tensorflow:Loss for final step: 426469920000.0.
```

[85]:
```
<tensorflow_estimator.python.estimator.canned.linear.LinearRegressorV2 at
0x1a3828b590>
```

Note that you have other premade estimator options, such as BoostedTreesRegressor(), and can also create your own custom estimators.

0.0.1 Thanks a lot for your attention.

[ ]: