Laboratory #1 Tensorflow

Table of Contents

Based on the definition that tensorflow website has provided: “TensorFlow is an open source software library for high performance numerical computation. Its flexible architecture allows easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. Originally developed by researchers and engineers from the Google Brain team within Google’s AI organization, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains.” Its first version was released in February 2017. One of the interesting properties of this application is that it is an interdisciplinary software whereas it uses several kernels, software, GPU,…. Figure 1 is from Google and is a good overview of its structure.

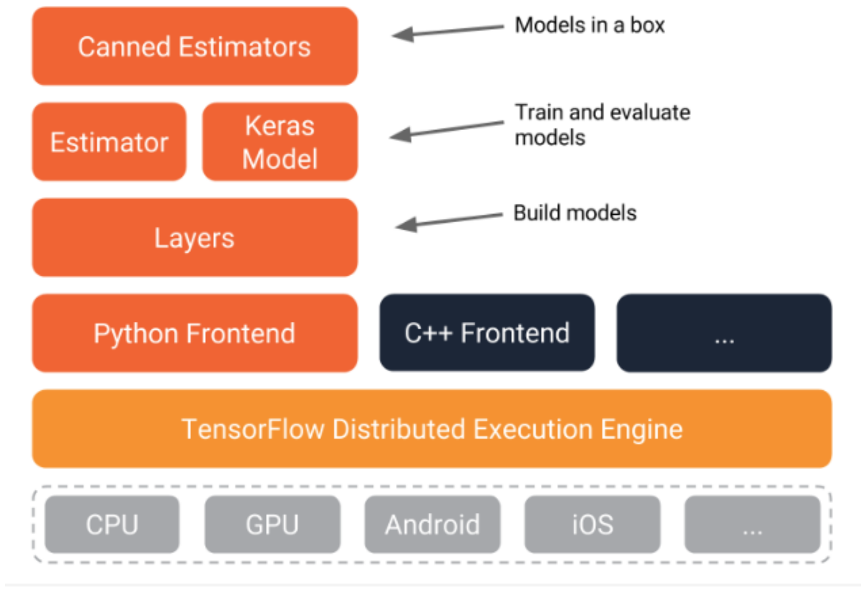


Figure 1- Tensorflow flowchart (from Google)

In this lab, we are going to see some examples on Tensorflow and its applications.

What is a Tensors?

A tensor is a multidimensional array. You can see some figures of tensors in figure 2.

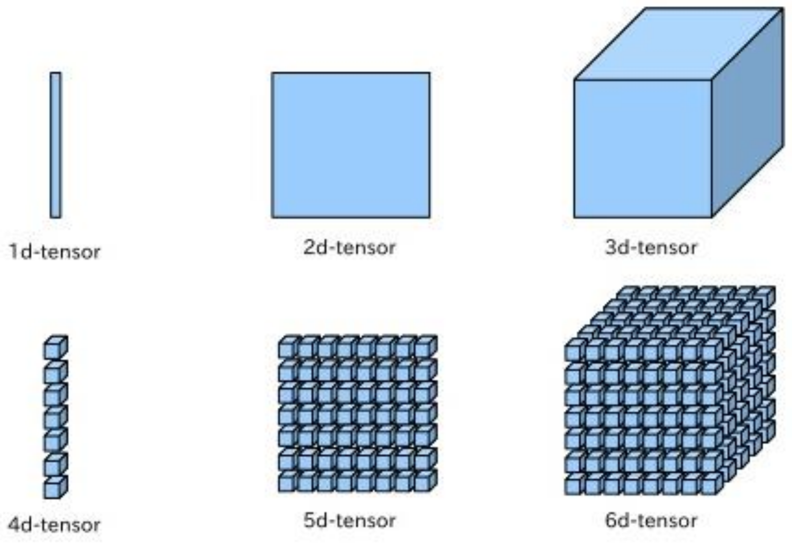


Figure 2- Different tensors

Please run through all the steps of the instruction and answer to the questions carefully. Finally write a report that contains your analysis plus the answer to questions that are indicated by Q.

Step1. Warm-up

Let us start with basic commands in Tensorflow. Run following code in Python and analyze the output:

```
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
print(hello)
```

Analyze the response and explain what *tf.constant()* command do.

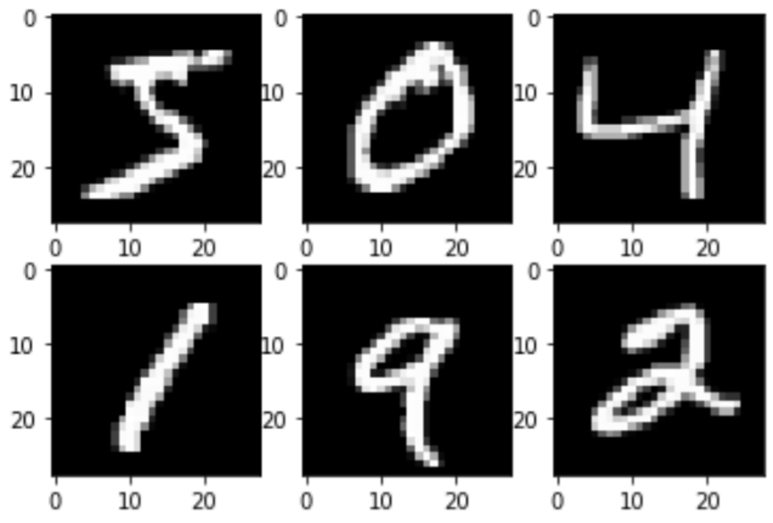
Step2. Implement OCR code in Tensorflow

We have already seen how to implement OCR (Optical Character Recognition) in python and Keras. The goal of this section is to implement same code in Tensorflow. We start with loading the data. We can use following code to load MNIST dataset in Python:

```
mnist = tf.keras.datasets.mnist
```

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Q1- Use same method that explain in lecture to show first 6 elements of the dataset. You should see:



Q2- Normalize the data by dividing the values by 255.

Next step is to design the model. For this part we use only one hidden layer with 128 nodes on it.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='sigmoid'),
  tf.keras.layers.Dense(10)
])
```

We can run this model on our training data and see the result of the network. For each example the model returns a vector of "logits" or "log-odds" scores, one for each class.

```
predictions= model(X_train[:1]).numpy()
predictions
```

Hint: In case this command does not work, there is a discrepancy on the version of Tensorflow. Please run this line of codes first:

```
tf.enable_eager_execution()
```

```
WARNING:tensorflow:Layer flatten_1 is casting an input tensor from dtype float6
4 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 warnin
g. If in doubt, this warning is likely only an issue if you are porting a Tenso
rFlow 1.X model to TensorFlow 2.

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


array([[1.1243229 , 0.02825141, 0.3847161 , 0.22021875, 0.4930552 ,
        0.254972  , 0.11298236, 1.350273  , 0.73544914, 0.34704828]],
      dtype=float32)
```

If we want to see the sigmoid result, we can add the desired activation function at last node:

```
tf.nn.sigmoid(predictions).numpy()
```

```
array([[0.7547897 , 0.5070624 , 0.59501004, 0.55483323, 0.6208259 ,
        0.5633999 , 0.5282156 , 0.7941743 , 0.67599994, 0.5859016 ]],
      dtype=float32)
```

Now is the time to define the loss function. We use a cross entropy as the loss function.

```
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
```

This loss is equal to the negative log probability of the true class: It is zero if the model is sure of the correct class. This untrained model gives probabilities close to random (1/10 for each class), so the initial loss should be close to -tf.log(1/10) ~= 2.3. We can see this by typing:

```
loss_fn(y_train[:1], predictions).numpy()
```

Next step is to design the model and start training:

```
model.compile(optimizer='SGD',
              loss=loss_fn,
              metrics=['accuracy'])
model.fit(X_train, y_train, epochs=5)
```

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 5s 83us/sample - loss: 1.4160 -
accuracy: 0.7046
Epoch 2/5
60000/60000 [==============================] - 4s 71us/sample - loss: 0.6961 -
accuracy: 0.8473
Epoch 3/5
60000/60000 [==============================] - 4s 74us/sample - loss: 0.5216 -
accuracy: 0.8724
Epoch 4/5
60000/60000 [==============================] - 5s 75us/sample - loss: 0.4471 -
accuracy: 0.8846
Epoch 5/5
60000/60000 [==============================] - 5s 80us/sample - loss: 0.4057 -
accuracy: 0.8917

<tensorflow.python.keras.callbacks.History at 0x2b129432a58>
```

You may see slightly different results because we did not initialize the weights and biases the same way.

Q3- Add another layer with 128 nodes, increase the epochs to 10 and report the result. How much is the accuracy? (*model.evaluate* can tell you the accuracy).

Use following command on your training section. This will consider 20% of your data for validation:

```
hist = model.fit(X_train, y_train, validation_split=0.2, epochs=10)
```

Now use following commands to draw the learning curves.

```
plt.subplot(2,1,1)
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')

plt.subplot(2,1,2)
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')

plt.show()
```

As we discussed, dropout can help to overcome the overfitting. In following code we add the dropout to the model.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='sigmoid'),
  tf.keras.layers.Dense(128, activation='sigmoid'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)])
```

Q4- Now train the model again with adding stochastic gradient descent with batch size of 200. What do you expect to be changed? Speed or accuracy?

Step3. Structured data

For this part, we will use a small dataset. In this dataset, the diagnostic, binary-valued variable investigated is whether the patient shows signs of diabetes according to World Health Organization criteria (i.e., if the 2 hour post-load plasma glucose was at least 200 mg/dl at any survey examination or if found during routine medical care). The population lives near Phoenix, Arizona, USA.

Attribute Information:

0. Patient ID
1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1)

You can load the dataset using:

```
file = 'address/to/file/pima-indians-diabetes.csv'
dataframe = pd.read_csv(file)
dataframe.head()
```

	ID	preg_no	plasma	diastolic	triceps	serum	mass	pedigree	age	diabete
0	1	6	148	72	35	0	33.6	0.627	50	1
1	2	1	85	66	29	0	26.6	0.351	31	0
2	3	8	183	64	0	0	23.3	0.672	32	1
3	4	1	89	66	23	94	28.1	0.167	21	0
4	5	0	137	40	35	168	43.1	2.288	33	1

Use *describe()* to see more information about dataset.

	ID	preg_no	plasma	diastolic	triceps	serum	mass	pedi
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.00
mean	384.500000	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.47
std	221.846794	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.33
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.07
25%	192.750000	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.24
50%	384.500000	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.37
75%	576.250000	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.62
max	768.000000	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.42

Let's identify input and outputs:

```
y = dataframe['diabete']
X = dataframe.drop(['ID','diabete'], axis=1)
```

Now use your knowledge and what we have learnt in class, to design a good neural netwrk that makes prediction for this dataset. Please consider 80% of data for training and 20% for testing using:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20,
random_state=1234)
```

Q5- Design a neural network with 2 hidden layers (hidden layer 1 with 12 nodes and hidden layer 2 with 8 nodes), report the accuracy and draw the learning curves. Apply a dropout to see if you could get any better result. Use *'adam'* as optimizer.

We don't expect a good accuracy from this example. That is one of the issues with NN. We need more data to create an accurate model.