TENSORFLOW VS KERAS: INTRODUCTION TO MACHINE LEARNING

In this Guide, we're exploring machine learning through two popular frameworks: TensorFlow and Keras. We have argued before that Keras should be used instead of TensorFlow in most situations as it's simpler and less prone to error, and for the other reasons cited in the above article. Though other libraries can work in tandem, many data scientists toggle between TensorFlow and Keras. There's also a healthy debate about when to use which one, which is a bit of a trick question!

This entire Guide (see the navigation on the right) will offer real examples of machine learning in both TensorFlow and Keras. In this article, we'll start by looking at and comparing TensorFlow and Keras, and then we'll code the same neural network using both frameworks—it's a multi-class classification problem.

Machine learning libraries

Machine learning uses a variety of math models and calculations to answer specific questions about data. Examples of machine learning in action include detecting spam emails, determining certain objects using computer vision, recognizing speech, recommending products, and even predicting commodities values years in the future.

The calculations implicit in machine learning and deep learning are very complicated to set up to ensure correct output (answers). A variety of machine learning libraries have emerged to help navigate these complexities. With these options, new folks can start getting into data science easily. Some of the most popular machine learning libraries include:

- TensorFlow
- Keras
- sciKit learn
- Theano
- Microsoft Cognitive Toolkit (CNTK)

So, how do you know when to use which? It's a bit of a trick question—using Keras wrapped around TensorFlow means you're using both. But we'll show the benefits of each, and why we think Keras should be used in most circumstances.

What is TensorFlow?

TensorFlow is an open-source ML library that uses symbolic math for dataflow and differentiable programming. Developed by Google and released in 2015, TensorFlow is an ML newcomer that's earned worldwide popularity because of its easy-to-use APIs and simplicity compared to its predecessors. TensorFlow's most used ML application is neural networks, which can analyze handwriting and recognize faces. Though TF is written in Python, a JavaScript port is available thanks to JavaScript's recent popularity.

TensorFlow is useful because it can scale problems with no limit—nodes in a graph can run across a distributed network. The logic in TF is unique, relying on both a machine's CPU and its GPU. Adding in the graphical processor unit gives TF a lot more power per machine.

The benefits of TensorFlow include:

- **Increased functionality.** While Keras has many general functions for ML and deep learning, TF's is more advanced, particularly in high-level operations like threading and queues and debugging.
- **Increased control.** You don't always need a lot of control, but some neural networks may require it so you have better understanding and insight, particularly when working with operations like weights or gradients.

Many users and data scientists recognize that TensorFlow can be difficult to use because of this complexity—it's not the most welcoming, particularly for new users.

Benefits of using Keras

Like TensorFlow, Keras is an open-source, ML library that's written in Python. The biggest difference, however, is that Keras wraps around the functionalities of other ML and DL libraries, including TensorFlow, Theano, and CNTK. Because of TF's popularity, Keras is closely tied to that library.

Many users and data scientists, us included, like using Keras because it makes TensorFlow much easier to navigate—which means you're far less prone to make models that offer the wrong conclusions.

Keras builds and trains neural networks, but it is user friendly and modular, so you can experiment more easily with deep neural networks. Keras is a great option for anything from fast prototyping to state-of-the-art research to production. The key advantages of using Keras, particularly over TensorFlow, include:

- **Ease of use.** The simple, consistent UX in Keras is optimized for use cases, so you get clear,

actionable feedback for most errors.
- **Modular composition.** Keras models connect configurable building blocks, with few restrictions.
- **Highly flexible and extendable.** You can write custom blocks for new research and create new layers, loss functions, metrics, and whole models.

When to use Keras

Keras offers something unique in machine learning: a single API that works across several ML frameworks to make that work easier. We recommend using Keras for most, if not all, of your machine learning projects.

Some say that a good rule of thumb is to use Keras unless you are building a very special neural network or you want the control and ability to watch how your network changes over time. Because Keras is so integrated with TensorFlow, you can start and build on Keras and then insert anything using TF.

Still not convinced to use Keras over TensorFlow? Consider:

- **Pandas makes life easy.** A big reason for this is that Keras works with Pandas datasets, creating Tensors for you. TensorFlow requires you to write all the code to create Tensors yourself.
- **Less NumPys.** NumPy is complicated. As you'll see in the TensorFlow code below, much effort is spent working with NumPy arrays. The Keras code uses it only once (to make a one-hot vector). Keras lets you work with dataframes or NumPy arrays interchangeably.
- **Keras avoids low-level details.** Keras figures out low-level details for you, like whether a Pandas column is categorical or a number. In TensorFlow, you have to tell it explicitly:

```
Casino = tf.feature_column.numeric_column("Casino")
```

- **Advanced TensorFlow functions.** Keras code still imports TensorFlow, so you can program TensorFlow functions directly.
- **GPU Support.** Keras imports TensorFlow, so you can opt for CPU-only support or add in GPU support. It's up to you.
- **Keras supports other frameworks, too.** You're not locked into TensorFlow when you use Keras; you can work with additional ML frameworks and libraries.

Neural networks coded in Keras and TensorFlow

Let's look at code for both, first Keras, then TensorFlow. As you'll see, the TensorFlow code is about double the length of Keras: 45 lines versus 25 lines. (We've left off the prediction and evaluation parts of the code to focus just on the model setup and training.)

Keras

```
1.      import tensorflow as tf
2.      from keras.models import Sequential
3.      import pandas as pd
4.      from keras.layers import Dense
5.
6.      cols =
```

```
10.     data = pd.read_csv('/home/ubuntu/Downloads/tripAdvisorFL.csv',
delimiter=',',names=cols)
11.     import numpy as np
12.     from keras.utils import np_utils
13.     from sklearn.model_selection import train_test_split
14.     labels = data
15.     features = data.drop(, axis=1)
16.     X=features
17.     y=np_utils.to_categorical(labels)
18.     X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)
19.     from keras.models import Sequential
20.     from keras.layers import Dense
21.     model = Sequential()
22.     model.add(Dense(8,  input_dim=19, activation='relu'))
23.     model.add(Dense(6, activation='softmax'))
24.     model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=)
25.     model.fit(X_train, y_train,epochs=4, batch_size=1, verbose=1)
```

TensorFlow

```
1.      import tensorflow as tf
2.      import numpy as np
3.      feature_names =
4.      FIELD_DEFAULTS = , , , , ,
                  , , , , ,
                  , , , , ,
                  , , , , ]
5.      def parse_line(line):
6.          parsed_line = tf.decode_csv(line, FIELD_DEFAULTS)
7.          features = parsed_line
8.          d = dict(zip(feature_names, features))
9.          print ("dictionary", d, " label = ", label)
10.         return d, label
11.     def csv_input_fn(csv_path, batch_size):
12.         dataset = tf.data.TextLineDataset(csv_path)
13.         dataset = dataset.map(parse_line)
14.         dataset = dataset.shuffle(1000).repeat().batch(batch_size)
15.         return dataset

16.     Usercountry = tf.feature_column.numeric_column("Usercountry")
17.     Nrreviews = tf.feature_column.numeric_column("Nrreviews")
18.     Nrhotelreviews = tf.feature_column.numeric_column("Nrhotelreviews")
19.     Helpfulvotes = tf.feature_column.numeric_column("Helpfulvotes")
20.     Periodofstay = tf.feature_column.numeric_column("Periodofstay")
21.     Travelertype = tf.feature_column.numeric_column("Travelertype")
22.     Pool = tf.feature_column.numeric_column("Pool")
```

```
23.     Gym = tf.feature_column.numeric_column("Gym")
24.     Tenniscourt = tf.feature_column.numeric_column("Tenniscourt")
25.     Spa = tf.feature_column.numeric_column("Spa")
26.     Casino = tf.feature_column.numeric_column("Casino")
27.     Freeinternet = tf.feature_column.numeric_column("Freeinternet")
28.     Hotelname = tf.feature_column.numeric_column("Hotelname")
29.     Hotelstars = tf.feature_column.numeric_column("Hotelstars")
30.     Nrrooms = tf.feature_column.numeric_column("Nrrooms")
31.     Usercontinent = tf.feature_column.numeric_column("Usercontinent")
32.     Memberyears = tf.feature_column.numeric_column("Memberyears")
33.     Reviewmonth = tf.feature_column.numeric_column("Reviewmonth")
34.     Reviewweekday = tf.feature_column.numeric_column("Reviewweekday")

35.     feature_columns =

37.     classifier=tf.estimator.DNNClassifier(
38.         feature_columns=feature_columns,
39.         hidden_units=,
40.         n_classes=6,
41.         model_dir="/tmp")

42.     batch_size = 100

43.     classifier.train(
44.         steps=100,
45.         input_fn=lambda :
csv_input_fn("/home/ubuntu/Downloads/tripAdvisorFL.csv", batch_size))
```