HOW KERAS MACHINE LANGUAGE API MAKES TENSORFLOW EASIER

Keras is a Python framework designed to make working with Tensorflow (also written in Python) easier. It builds neural networks, which, of course, are used for classification problems. The example problem below is binary classification. You can find the code here. The binary classification problem here is to determine whether a customer will buy something given 14 different features. You can see the data here.

Keras can run on top of:

- TensorFlow
- cuDNN
- CNTK

Here we use Tensorflow. So install Aconda and then run these commands to install the rest.

```
conda install theano
conda install tensorflow
conda install keras
```

(This tutorial is part of our Guide to Machine Learning with TensorFlow & Keras. Use the right-hand menu to navigate.)

The Code

In the code below, we have a dataframe of shape (673,14), meaning 673 rows and 14 feature columns. We take the columns called **Buy** and use that for labels. You can use the Keras methods with dataframes, numpy arrays, or Tensors.

We declare our model to be **Sequential**. These are a stack of **layers**.

We tell Keras to return the accuracy metric **metrics=**.

```
import tensorflow as tf
from keras.models import Sequential
import pandas as pd
from keras.layers import Dense

url = 'https://raw.githubusercontent.com/werowe/
logisticRegressionBestModel/master/KidCreative.csv'

data = pd.read_csv(url, delimiter=',')

labels=data
features = data.iloc

model = Sequential()

model.add(Dense(units=64, activation='relu', input_dim=1))
model.add(Dense(units=14, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=)

model.fit(labels, features,
          batch_size=12,
          epochs=10,
          verbose=1,
          validation_data=(labels, features))
model.evaluate(labels, features, verbose=0)

model.summary()
```

The output looks like this. As you can see it ran the **sgd** (standard gradient descent) optimizer and **categorical_crossentropy** 10 times, since we set the **epochs** to 10. Since we used the same data for training and evaluating we get a 0.9866 **accuracy**. In actual use we would split the input data into training and test data, following the standard convention. The **loss** is 362.1225. We could have used mse (mean squared error), but we used categorical_crossentropy. The goal of the model (in this case sgd) is to minimize the loss function, meaning the difference between the actual and predicted values.

```
rain on 673 samples, validate on 673 samples
Epoch 1/10
2018-07-26 08:43:32.122722: I
tensorflow/core/platform/cpu_feature_guard.cc:140] Your CPU supports
instructions that this TensorFlow binary was not compiled to use: AVX2 FMA
673/673  - 0s 494us/step - loss: 1679.5777 - acc: 0.9851 - val_loss: 362.1225
- val_acc: 0.9866
Epoch 2/10
673/673  - 0s 233us/step - loss: 362.1225 - acc: 0.9866 - val_loss: 362.1225
- val_acc: 0.9866
Epoch 3/10
673/673  - 0s 218us/step - loss: 362.1225 - acc: 0.9866 - val_loss: 362.1225
- val_acc: 0.9866
Epoch 4/10
673/673  - 0s 208us/step - loss: 362.1225 - acc: 0.9866 - val_loss: 362.1225
- val_acc: 0.9866
Epoch 5/10
673/673  - 0s 213us/step - loss: 362.1225 - acc: 0.9866 - val_loss: 362.1225
- val_acc: 0.9866
Epoch 6/10
673/673  - 0s 212us/step - loss: 362.1225 - acc: 0.9866 - val_loss: 362.1225
- val_acc: 0.9866
Epoch 7/10
673/673  - 0s 216us/step - loss: 362.1225 - acc: 0.9866 - val_loss: 362.1225
- val_acc: 0.9866
Epoch 8/10
673/673  - 0s 218us/step - loss: 362.1225 - acc: 0.9866 - val_loss: 362.1225
- val_acc: 0.9866
Epoch 9/10
673/673  - 0s 228us/step - loss: 362.1225 - acc: 0.9866 - val_loss: 362.1225
- val_acc: 0.9866
Epoch 10/10
673/673  - 0s 239us/step - loss: 362.1225 - acc: 0.9866 - val_loss: 362.1225
- val_acc: 0.9866
<keras.callbacks.History object at 0x7fa48f3ccac8>
>>>
... model.evaluate(labels, features, verbose=0)

>>>
>>> model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                128
_________________________________________________________________
dense_2 (Dense)              (None, 14)                910
=================================================================
```

```
Total params: 1,038
Trainable params: 1,038
Non-trainable params: 0
```

Now you can use the predict() method to make some prediction on whether a person is likely to buy this product or not.