Artificial Neural Network: Keras and Tensorflow

AAA-Python Edition

Plan

- 1- Keras and Tensorflow
- 2- MLP with Tensorflow: High Level API
- 3- MLP with Keras
- 4- More about tensorflow
- 5- Tensorboard

1- Keras and Tensorflow

Introduction

- **Keras**: a high level **API** to **build** & **train** a **deep learning model.**

Application Programming Interface: it defines how to interact and use built-in Keras modules.

- It is written in **python** and runs on the top of **Tensorflow**

We will talk about in the next lesson

- Implementation:
  - **tf.keras** : tensorflow implementation of keras
  - **keras**: Keras library (apart)

We will use the second implementation

[By Amina Delali]

1- Keras and Tensorflow

Keras

```python
# tensorflow implementation of keras
import tensorflow as tf
from tensorflow.keras import layers

print("Tensorflow version:",tf.VERSION)
print("Keras version:",tf.keras.__version__)
```

```
Tensorflow version: 1.13.0-rc2
Keras version: 2.2.4-tf
```

- Tensorflow keras version may no be up to date (right now, this is not the case)

```python
# keras library
!pip install keras
```

```python
import keras
print("Keras version:",keras.__version__)
```

```
Keras version: 2.2.4
```

- The default saving formats of the model's weights are different

[By Amina Delali]

Tensorflow

1- Keras and Tensorflow

- **Tensorflow:** an **open source library** that enables you **develop** and **train ML models.**

- There is **2** important releases of **Tensorflow:**
  - Versions 1.*.* defined by the **APIs r1.***
  - Versions 2.*.* defined by the **APIs r2.***

```
# you have to restart runtime after installation
!pip install tensorflow==2.0.0-alpha0
```

- Right now there is only one version: **TensorFlow 2.0 Alpha** defined by the **API r2.0**
- Unlike the previous release, you have to manually install it on google colab.

```
# tensorflow implementation of keras
import tensorflow as tf
from tensorflow.keras import layers

print("Tensorflow version:",tf.version.VERSION)
print("Keras version:",tf.keras.__version__)
```

```
Tensorflow version: 2.0.0-alpha0
Keras version: 2.2.4-tf
```

In the previous example, we wrote: tf.VERSION

[By Amina Delali]

Using tensorflow

- There are **2** ways to implement artificial neural networks in tensorflow:
  - Using the **high level API**
  - Using the **low level API**
- For example, building and training an **MLP** Using the high level API is simple and trivial.
- The low level **API**, permits **more flexibility** in defining the architecture of your model, but will require more code.
- In our first example we will use the high level API. In other world, we will use its **premade estimators**
- Just a reminder, we will implement the same MLP we defined in the previous lesson.

Steps

- Elements to consider when using a pre-made estimator of tensorflow:

  - Define at least **one input function**: it will be used by the estimator to create a structured data that it will use later for training and/or predicting. For our example, the function will return a **tuple** of:
    - **Features:** a dictionary with the features names and their corresponding values.
    - The corresponding labels

The number of values will determine the **batch** size

  - Define the **features columns**: used to build the estimator. In our case, they will be an array of the **numeric feature columns** constructed using tensorflow. They indicate the features to use from the data returned from the previous defined **input function.**

  - Build the estimator and use it for training, predicting … etc

2- MLP with Tensorflow: High Level API

Build the MLP

```python
# the input function
def myInputFunction(x,y=None):

    myFeat = dict({"sepal_length":x[:,0],
                   "sepal_width":x[:,1],
                   "petal_length":x[:,2],
                   "petal_width":x[:,3]})

    myLab = y
    return myFeat, myLab
```

To be able to use the function for making prediction

We will use sklearn to use iris dataset

```python
# define the function that construct the features columns
def constructFeat(keys):
  myFC= []
  for k in keys:
    myFC.append(tf.feature_column.numeric_column(key=k))
  return myFC
```

The features values will be numeric

The keys correspond to those used in the input function

Tanh formula:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e - x}$$

```python
# in our example the features columns are simply columns
# of numeric values

myKeys = ["sepal_length","sepal_width","petal_length","petal_width"]
myFeatureColumns= constructFeat(myKeys)

# we will build the same MLP we used in the previous lesson
## the estimator will be a DNNClassifier
## we have to specify one activation function for all
## the hidden layer. The output layer will have, by default
## the softmax activation function
## optimisation using Stochastic gradient descent
myMLP = tf.estimator.DNNClassifier(hidden_units=[2],
                                   n_classes=3,feature_columns=myFeatureColumns,
                                   activation_fn= tf.nn.tanh,
                                   optimizer = "SGD")
```

Approximately the same as tansig function

[By Amina Delali]

2- MLP with Tensorflow: High Level API

Training and evaluating

```
# training the MLP
myMLP.train(input_fn=lambda:myInputFunction(x_train,y_train), max_steps=50000)
```

- Our input function dosen’t return fixed data values;the data must be passed as parameters. This way, we can use the same input function for both training and evaluating our model
- The train and evaluate methods require a callable function. So, to use our defined function, we have to define **another one** that **calls** our function with the desired data parameter==> will lead to have two sepearate input functions: one for training and one for testing.
- To avoid defining 2 functions, we will use the **python high order function lambda:** we give it a function’s **definition**, and it returns a **function** (without a name).

Loss is calculated using softmax cross entropy.

```
import numpy as np
# evaluating the MLP

evalRes = myMLP.evaluate(input_fn=lambda:myInputFunction(x_test,y_test),steps=1)
print("\nTest set accuracy: ", np.round(evalRes["accuracy"],3))
print("All evaluation values:\n",evalRes)
```

```
Test set accuracy:  1.0
All evaluation values:
 {'accuracy': 1.0, 'average_loss': 0.052664462, 'loss': 0.052664462, 'global_step': 50000}
```

Predicting

```
# we will use a portion of our test data (3 samples)
# to predict the classification

yPred = myMLP.predict(input_fn= lambda: myInputFunction(x_test[:3]))
```

```
# labels for the corresponding classes

myClass = ["Iris-Setosa","Iris-Versicolour","Iris-Virginica"]
for pred,trueL in zip(yPred,y_test[:3]):
  print("predicted class:",myClass[pred["class_ids"][0]],"\n the true class is:",
        myClass[trueL],"\n\n" )
```

Concerning the labels array in the training and evaluating, we didn't have to convert it to a multidimensional array.

```
predicted class: Iris-Virginica
 the true class is: Iris-Virginica


predicted class: Iris-Versicolour
 the true class is: Iris-Versicolour


predicted class: Iris-Setosa
 the true class is: Iris-Setosa
```

We expected to have the right predictions since we used the test set that scored **1.0 accuracy**

We didn't have to define a function that returns the corresponding class for each prediction (the predicted class is in the output of the prediction).

[By Amina Delali]

Sequential model

- There is **2** ways to build a **ANN** with keras:
  - Using a **sequential** model: to build a sequential stack of layers. Ideal for building simple, fully-connected networks.
  - Using a **functional** model: ideal to build complex model topologies.

- To build our MLP with the Sequential model, we have to:
  - Define our layers:
    - Specifying the number of neurons
    - Selecting the activation function
    - Define how the neuron's weights (and the bias term) will be initialized
    - Define the optimization method: it defines how the learning is performed.
    - Define the loss function: the function to be minimized during the learning.

[By Amina Delali]

3- MLP with Keras

Building our model

- The **input** array will have the shape : **(*,4)**

```python
import keras
from keras import layers

myMLP2 = keras.Sequential()
# we sepecified the activation functions
# and the initialization of the weights
myMLP2.add(layers.Dense(2, activation='tanh', kernel_initializer="TruncatedNormal",input_shape=(4,))
myMLP2.add(layers.Dense(3, activation='softmax',kernel_initializer="TruncatedNormal"))
```

It specifies a regular densely-connected NN layer: applies the activation function on a weighted sum.

- It specifies the function that will be used to initialize the weights
- The **truncated normal distribution:** generates the same values as the **normal distribution** except that values more than two standard deviations from the mean are discarded and redrawn
- The default values are:
  - Mean: **0.0**
  - Standard deviation: **0.05**

```
!pip install tensorflow==1.13.1
```

To be compatible with our keras example

[By Amina Delali]

Training

3- MLP with Keras

```
# before starting the training we have to configure our model
# with some additional parameters

# the optimizer used is the Stochastic gradient descent

myMLP2.compile(loss="mean_squared_error",
               optimizer="sgd",
               metrics=["mae", "acc"])
```

The loss function will be the mean square error

The learning algorithm will be the stochastic gradient descent

The metrics that will be returned by the evaluation method in addition to the loss function value.

```
#training
from keras.utils import to_categorical

# converting the labels array into a multidimensional array (*,3)
# with 0, 1 values as we did in the previous lesson
Ylabels = to_categorical(y_train, num_classes=3)
myMLP2.fit(x_train,Ylabels,epochs=1200,verbose=0)
```

For example:
The label **2** will be converted into **0. 0. 1.**

[By Amina Delali]

Evaluating and predicting

3- MLP with Keras

```python
# evaluating
import numpy as np

Ytests = to_categorical(y_test, num_classes=3)
eval2= myMLP2.evaluate(x_test, Ytests )
print("Accuracy = ",np.round(eval2[2],3))
```

```
Accuracy =  0.933
```

```python
# predicting
# we will predict for the 3 first samples from the test data
# and compare our predictions with the true labels
yPred = myMLP2.predict(x_test[:5])
```

We defined the prediction class to extract the class corresponding to the highest probability

```python
# our prediction function
def predict(pred):
  myClass =  ["Iris-Setosa","Iris-Versicolour","Iris-Virginica"]
  return myClass[np.argmax(pred)]
```

```python
for i in range(5):
  print("The predicted class is: ", predict(yPred[i,:]), "\n The true class is: ",
        predict(Ytests[i,:]))
```

```
The predicted class is:  Iris-Setosa
 The true class is:  Iris-Setosa
The predicted class is:  Iris-Virginica
 The true class is:  Iris-Virginica
The predicted class is:  Iris-Versicolour
 The true class is:  Iris-Versicolour
The predicted class is:  Iris-Virginica
 The true class is:  Iris-Versicolour
The predicted class is:  Iris-Setosa
 The true class is:  Iris-Setosa
```

A correct prediction

A wrong prediction

[By Amina Delali]

3- MLP with Keras

Cross entropy

- For a multi-class classification, using the **softmax** activation function, the cross entropy loss function is a better choice to compute the cost to minimize.

- It takes into consideration the values of the **probabilities** returned by the **output neurons** instead of just taking into account correct or the wrong classification:

$$J(w) = \frac{-1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

Number of Classes

The weights to be updated

Number of instances

The **true** label value, == **1** if the class of the instance (**i**) is **k**, otherwise it equals to **0**

Predicted probability for the instance **i** for the class **k**

[By Amina Delali]

3- MLP with Keras

Applying the cross entropy

```
myMLP2.compile(loss="categorical_crossentropy",
               optimizer="sgd",
               metrics=["mae", "acc"])
```

We changed only the loss function

```
import numpy as np
Ytests = to_categorical(y_test, num_classes=3)

eval3= myMLP2.evaluate(x_test, Ytests )
print("Accuracy = ",np.round(eval3[2],3))
```

```
Accuracy =  1.0
```

We have same accuracy result as we had with tensorflow (because of the use of the cross entropy loss function)

```
myMLP2.compile(loss="sparse_categorical_crossentropy",
               optimizer="sgd",
               metrics=["mae", "acc"])
```

We don't have to convert our labels array

```
myMLP2.fit(x_train,y_train,epochs=1200,verbose=0)
eval4= myMLP2.evaluate(x_test, y_test )
```

```
Accuracy =  1.0
```

```
yPred3 = myMLP2.predict(x_test[:5])
print(yPred3[0])
```

```
[9.973870e-01 2.590685e-03 2.240326e-05]
```

But we still need a function to extract the predicted class

[By Amina Delali]

4- More about tensorflow

Graph, tensor, operation

- With **tensorflow** it is possible to define your model as a **graph.**

- The concept is simple:
  - You define your **graph**: the steps of the computation(the tensorflow program)
  - You run your graph

- Your graph may contain:
  - **Tensors**: the central unit of data. Arrays of any number of dimension (a scalar is a tensor with dimension **(rank)** 0). They also represent the **Edges** of the graph.
  - **Operations**: the nodes of the graph. They describe calculations with tensors . We can use constructor for operations as follow:
    - tensorflow.constant(3.5): creates an operation that will produce the value **3.5** and add it to the **default graph** (TensorFlow provides a default graph that is an implicit argument to all API functions in the same context.)

Tensorboard

4- More about tensorflow

- **Tenorboard** is a suite of **visualization tools** that can be utilised to visualize TensorFlow **graph**, plot **quantitative metrics** about the **execution of your graph.**

- To use Tensorboard with **google colab**, you can use the library **tensorboardcolab:**

```
from tensorboardcolab import TensorBoardColab, TensorBoardColabCallback

tbc=TensorBoardColab()
```

```
Using TensorFlow backend.
Wait for 8 seconds...
TensorBoard link:
https://[redacted].ngrok.io
```

By clicking on this link you can access to tensorboard

5- Tensorboard

A simple graph in tensorboard

```python
import tensorflow as tf

#delete all previous graphs
tf.reset_default_graph()

x = tf.constant(7.0)
y = tf.constant(8.0)
result = x + y

with tf.Session() as sess:
  writer =tf.summary.FileWriter("./Graph",sess.graph)
  print(sess.run(result))
  writer.close()
```

Tensor produced by the operation tf.constant

```
Tensor("Const:0", shape=(), dtype=float32)
```

```
Tensor("add:0", shape=(), dtype=float32)
```

Log file generated in "./Graph/"

```
events.out.tfevents.1552591694.1099d6afc364
```

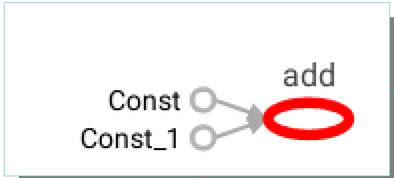
Visualization generated in tensorboard

Main GraphAuxiliary Nodes

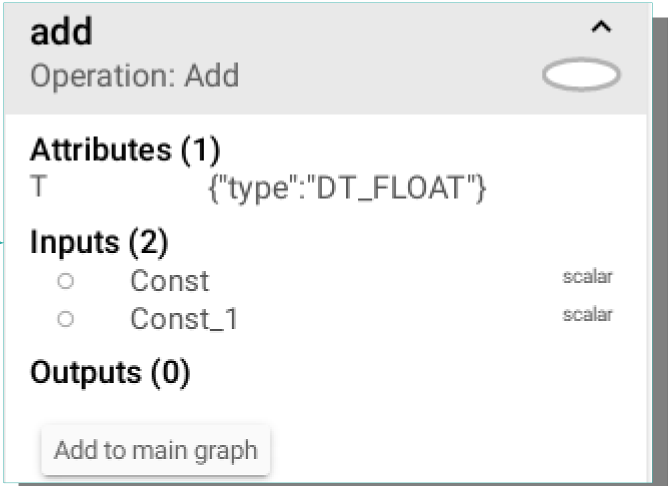
[By Amina Delali]

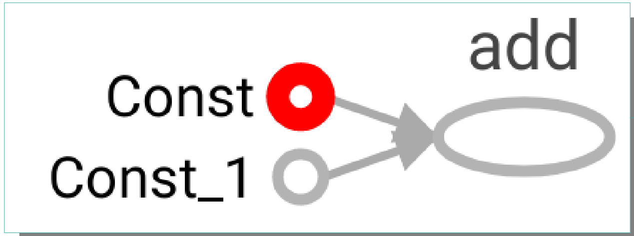
A simple graph in tensorboard (suite)

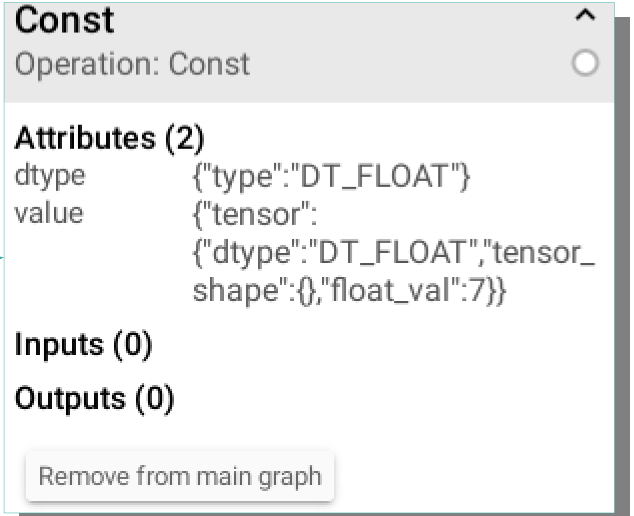
5- Tensorboard



If you click on the node “add”







[By Amina Delali]

5- Tensorboard

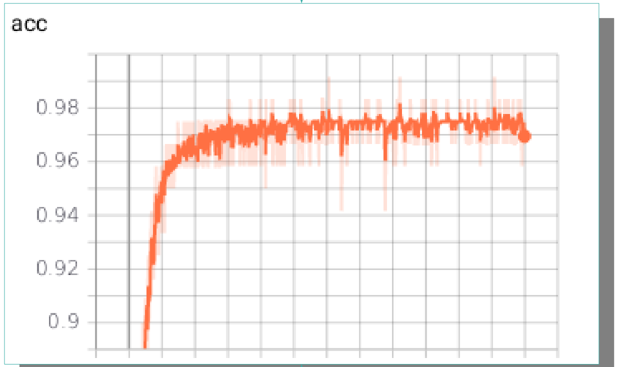
Tensorboard with Keras

```python
from keras.utils import to_categorical
import tensorflow as tf




# use the model compiled with the cross entropy loss function
myMLP2.compile(loss="categorical_crossentropy",
               optimizer="sgd",
               metrics=["mae", "acc"])

Ylabels = to_categorical(y_train, num_classes=3)
myMLP2.fit(x_train,Ylabels,epochs=1200,verbose=0,callbacks=[TensorBoardColabCallback(tbc)])
```



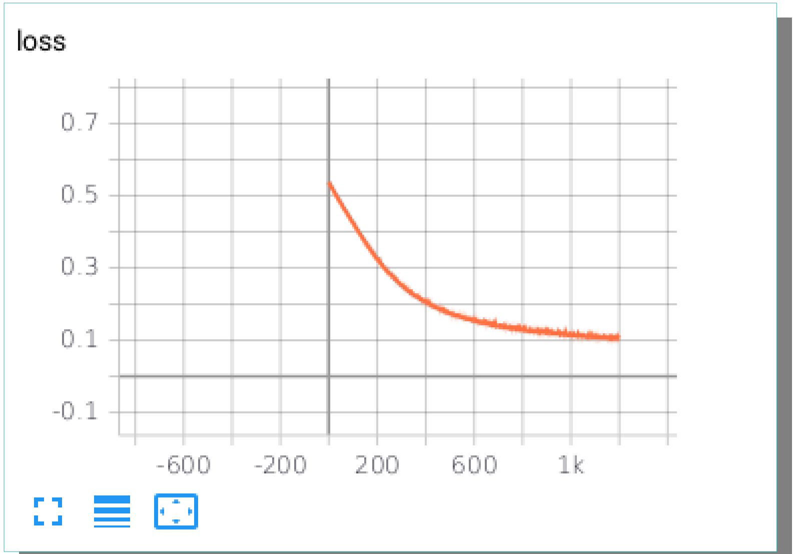
We use the variable we already defined by TensorBoardColab() call

Since we specified “accuracy” and “mae” as metrics, their progression graphs will be generated in tenorboard

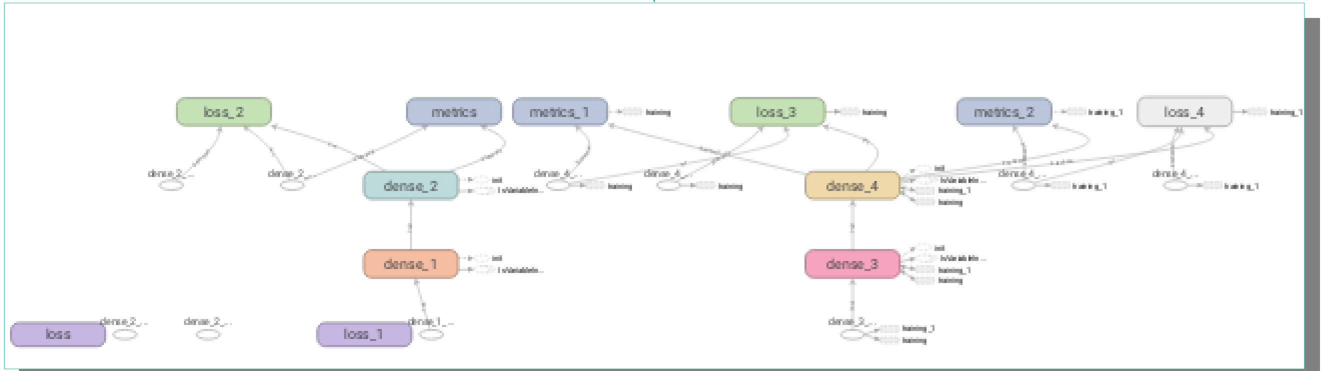
[By Amina Delali]

Tensorboard with Keras

5- Tensorboard



The generated graph



[By Amina Delali]

References

- Keras, https://www.tensorflow.org/guide/keras
- TensorBoard: Visualizing Learning, https://www.tensorflow.org/guide/summaries_and_tensorboard
- Keras 2.2.4, https://pypi.org/project/Keras/
- Premade Estimators https://www.tensorflow.org/guide/premade_estimators
- Feature Columns, https://www.tensorflow.org/guide/feature_columns
- tansig , https://edoras.sdsu.edu/doc/matlab/toolbox/nnet/tansig.html
- Hyperbolic functions https://www.math10.com/en/algebra/hyperbolic-functions/hyperbolic-functions.html
- Tensorflow, Introduction, https://www.tensorflow.org/guide/low_level_intro
- Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. O’Reilly Media, Inc.

Thank you!

FOR ALL YOUR TIME