

# Keras Introduction

January 10, 2021

## 1 Keras Introduction

### 1.0.1 What is Keras?

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research.

See <https://keras.io/about/> for detailed Keras guideline.

### 1.0.2 Environment

Our suggested coding environment setting is:

- Python: 3
- Keras: 2.4

### 1.0.3 How can I install Keras?

Keras/TensorFlow are compatible with:

- Python 3.5–3.8
- Ubuntu 16.04 or later
- Windows 7 or later
- macOS 10.12.6 (Sierra) or later.

Keras comes packaged with TensorFlow 2.0 as `tensorflow.keras`. To start using Keras, simply install **TensorFlow 2**. Tensorflow is an open-source programming language developed by Google that is specifically designed to make programming deep-learning programs easy, or at least easier.

You can install **Anaconda** as environment manager.

### 1.0.4 Code Examples

Our textbook use Tensorflow 1. However, Tensorflow 2 and Keras is becoming more and more popular. We will use the Keras in Tensorflow 2 for project implementations of this course. However, a brief understanding of tensorflow 2 is necessary before you use Keras.

In the following, we give some examples in both Tensorflow 2 and Keras.

**Hello World Example from Section 2.1** The example code in the textbook is in Tensorflow 1. We migrate it to Tensorflow 2.

```
[1]: import tensorflow as tf
import tensorflow.compat.v1 as tf1
import numpy

print(tf.version)

# code in tf 2.0
x = tf.constant("Hello World") #x is a constant tensor
tf.print(x) #will print out "Hello World"
```

```
<module 'tensorflow._api.v2.version' from
'C:\\Users\\amber\\Anaconda3\\lib\\site-
packages\\tensorflow\\_api\\v2\\version\\__init__.py'>
Hello World
```

```
[2]: print(x) #x is a constant tensor. empty shape (0 dimensionality), type string
```

```
tf.Tensor(b'Hello World', shape=(), dtype=string)
```

- Placeholders in **TensorFlow 1** are similar to variables and you can declare it using `tf.placeholder`. You don't have to provide an initial value and you can specify it at run-time with `feed_dict` argument inside `Session.run`, whereas in `tf.Variable` you can provide initial value when you declare it. There is no usage of placeholder in **Tensorflow 2** anymore.

```
[3]: import tensorflow as tf

bt = tf.random.normal([10], stddev=.1)
b = tf.Variable(bt, name = 'W')
W = tf.Variable(tf.random.normal([784,10],stddev=.1), name='b')

#
@tf.function
def forward(x):
    return W * x + b

out = forward(b)

print(out)
print(W.shape, b.shape)
print(W)
```

```
tf.Tensor(
[[ 0.02249808  0.17590831 -0.03387045 ... -0.10475103  0.1332572
 -0.12232003]
 [ 0.01836991  0.13974863 -0.03627536 ... -0.11784038  0.13608111
 -0.11757181]
```

```
[ 0.01545492  0.17088744 -0.03310285 ... -0.1105068  0.14276789
 -0.10371671]
...
[ 0.01622064  0.17426997 -0.03177525 ... -0.09873184  0.14218308
 -0.13001856]
[ 0.0220406  0.1670794 -0.03630526 ... -0.11157584  0.15035588
 -0.10959108]
[ 0.02341765  0.15367316 -0.03077138 ... -0.10992643  0.12650637
 -0.12167677]], shape=(784, 10), dtype=float32)
(784, 10) (10,)
<tf.Variable 'b:0' shape=(784, 10) dtype=float32, numpy=
array([[ 0.09250043,  0.11505096,  0.02849841, ..., -0.00947891,
         0.04249997,  0.05829099],
       [-0.10796215, -0.11415862,  0.1015252 , ...,  0.11429337,
         0.06459207,  0.01721029],
       [-0.24951349,  0.0832246 ,  0.00518974, ...,  0.04494744,
         0.11690428, -0.10266156],
       ...,
       [-0.21233018,  0.10466587, -0.03512369, ..., -0.06639615,
         0.1123291 ,  0.12489736],
       [ 0.07028524,  0.05908615,  0.10243287, ...,  0.05505619,
         0.17626671, -0.05183755],
       [ 0.13715439, -0.02589366, -0.06560692, ...,  0.03945943,
        -0.0103132 ,  0.05272567]], dtype=float32)>
```

**A Simple Feedforward NN in Tensorflow 2.0 & Keras** Figure 2.2 in textbook: a simple feedforward NN on MNIST handwriting dataset.

Tensorflow 2 uses Keras layers and models to manage variables. The layers and models are two core data structures of Keras. The simplest type of model is the Sequential model, a linear stack of layers. For more complex architectures, you should use the Keras functional API, which allows to build arbitrary graphs of layers, or write models entirely from scratch via subclassing.

Here is the Sequential model:

```
[4]: import tensorflow as tf
      from tensorflow.keras.models import Sequential

      model = Sequential()
```

Stacking layers is as easy as `.add()`:

```
[5]: from tensorflow.keras.layers import Dense, Flatten

      model.add(Flatten(input_shape=(28, 28)))
      model.add(Dense(64, activation='relu'))
      model.add(Dense(10, activation='softmax'))
```

Prints a string summary of the network

```
[6]: model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 64)	50240
dense_1 (Dense)	(None, 10)	650

Total params: 50,890

Trainable params: 50,890

Non-trainable params: 0

Once your model looks good, configure its learning process with `.compile()`:

```
[7]: model.compile(optimizer='adam',  
                  loss='sparse_categorical_crossentropy',  
                  metrics=['accuracy'])
```

Load MNIST dataset as an example

```
[8]: # x_train and y_train are Numpy arrays --just like in the Scikit-Learn API.  
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()  
  
x_train, x_test = x_train / 255.0, x_test / 255.0
```

You can now iterate on your training data in batches:

```
[9]: history = model.fit(x_train, y_train, epochs=20)
```

Epoch 1/20

1875/1875 [=====] - 3s 2ms/step - loss: 0.4924 -  
accuracy: 0.8627

Epoch 2/20

1875/1875 [=====] - 3s 2ms/step - loss: 0.1610 -  
accuracy: 0.9539: 0s - loss: 0.1617 - ac

Epoch 3/20

1875/1875 [=====] - 3s 1ms/step - loss: 0.1156 -  
accuracy: 0.9664

Epoch 4/20

1875/1875 [=====] - 3s 2ms/step - loss: 0.0886 -  
accuracy: 0.9745

Epoch 5/20

1875/1875 [=====] - 3s 2ms/step - loss: 0.0711 -  
accuracy: 0.9785

Epoch 6/20

```

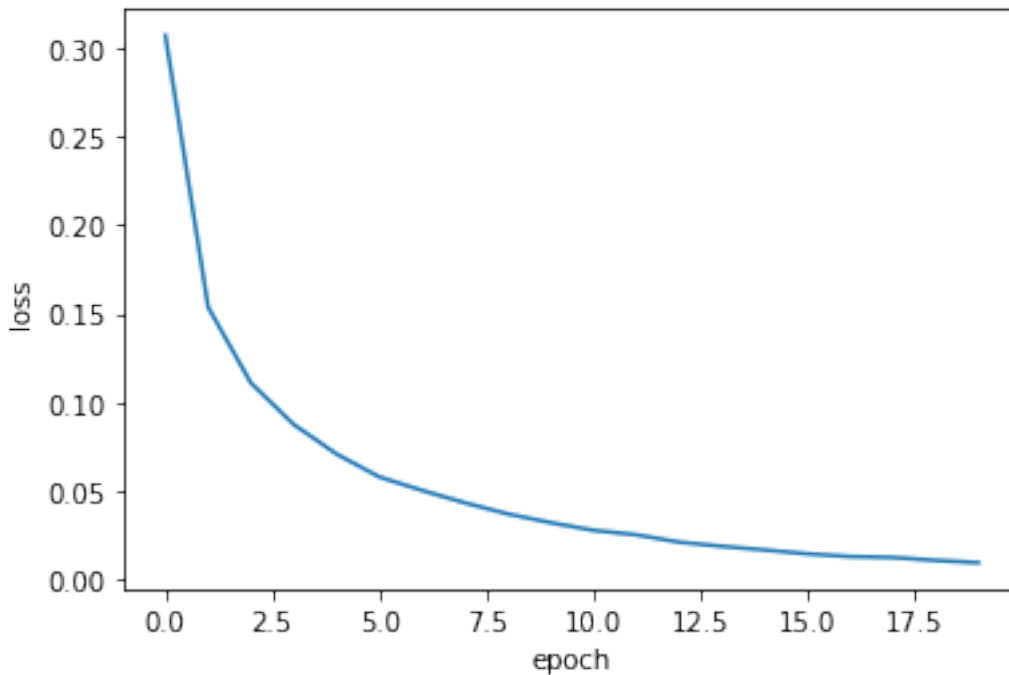
1875/1875 [=====] - 7s 4ms/step - loss: 0.0557 -
accuracy: 0.9835
Epoch 7/20
1875/1875 [=====] - 5s 2ms/step - loss: 0.0469 -
accuracy: 0.9866
Epoch 8/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.0410 -
accuracy: 0.9879
Epoch 9/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.0361 -
accuracy: 0.9896
Epoch 10/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.0314 -
accuracy: 0.9906
Epoch 11/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.0266 -
accuracy: 0.9926
Epoch 12/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.0242 -
accuracy: 0.9927
Epoch 13/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.0178 -
accuracy: 0.9953
Epoch 14/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.0169 -
accuracy: 0.9955
Epoch 15/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.0150 -
accuracy: 0.9955
Epoch 16/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.0133 -
accuracy: 0.9958
Epoch 17/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.0110 -
accuracy: 0.9967
Epoch 18/20
1875/1875 [=====] - 4s 2ms/step - loss: 0.0108 -
accuracy: 0.9967
Epoch 19/20
1875/1875 [=====] - 5s 2ms/step - loss: 0.0108 -
accuracy: 0.9969
Epoch 20/20
1875/1875 [=====] - 3s 2ms/step - loss: 0.0091 -
accuracy: 0.9974

```

Visualize the loss value during training

```
[10]: from matplotlib import pyplot as plt

plt.plot(history.history['loss'],label="loss")
plt.xlabel("epoch")
plt.ylabel("loss")
plt.show()
```



Evaluate your test loss and metrics in one line:

```
[11]: model.evaluate(x_test, y_test)
```

```
313/313 [=====] - 1s 1ms/step - loss: 0.1099 -
accuracy: 0.9747
```

```
[11]: [0.10994642227888107, 0.9746999740600586]
```

Or generate predictions on new data:

```
[12]: model.predict(x_test)
```

```
[12]: array([[1.6028704e-12, 4.9453214e-16, 3.7315817e-07, ..., 9.9999654e-01,
1.2544329e-10, 2.6817379e-09],
[4.5180605e-16, 2.6933292e-11, 1.0000000e+00, ..., 1.6457892e-27,
1.2101538e-09, 3.5370632e-25],
[6.1276850e-10, 9.9743003e-01, 1.3615943e-04, ..., 8.2414881e-06,
2.4249963e-03, 7.1421186e-10],
```

```
...,
[5.0217689e-20, 7.2626926e-21, 2.0075069e-19, ..., 9.8839414e-10,
 1.5221675e-08, 8.2123544e-09],
[4.0450516e-16, 9.7965543e-24, 1.9620505e-18, ..., 1.7043765e-14,
 4.2884884e-07, 1.2076735e-18],
[2.5741761e-12, 2.2340474e-28, 4.0519548e-17, ..., 1.5809322e-24,
 2.3095423e-19, 5.2315833e-20]], dtype=float32)
```

[ ]: