02_how_to_use_keras

September 29, 2021

1 How to use Keras

Keras was designed as a high-level or meta API to accelerate the iterative workflow when designing and training deep neural networks with computational backends like TensorFlow, Theano, or CNTK. It has been integrated into TensorFlow in 2017 and is set to become the principal TensorFlow interface with the 2.0 release. You can also combine code from both libraries to leverage Keras' high-level abstractions as well as customized TensorFlow graph operations.

Please follow the installations instructions in `Installation Guide.md` in the root folder.

1.1 Imports & Settings

[1]:
```
import warnings
warnings.filterwarnings('ignore')
```

[2]:
```
%matplotlib inline
from copy import deepcopy
import numpy as np
import pandas as pd
import sklearn
from sklearn.datasets import make_circles # To generate the dataset
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras import optimizers
from keras.callbacks import TensorBoard

import matplotlib
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from mpl_toolkits.mplot3d import Axes3D  # 3D plots

import seaborn as sns
```

```
Using Theano backend.
```

[3]:
```
# plotting style
sns.set_style('darkgrid')
# for reproducibility
```

```
np.random.seed(seed=42)
```

1.2 Input Data

1.2.1 Generate random data

The target y represents two classes generated by two circular distribution that are not linearly separable because class 0 surrounds class 1.

[4]:
```
# dataset params
N = 50000
factor = 0.1
noise = 0.1
```

[5]:
```
# generate data
X, y = make_circles(
    n_samples=N,
    shuffle=True,
    factor=factor,
    noise=noise)
```

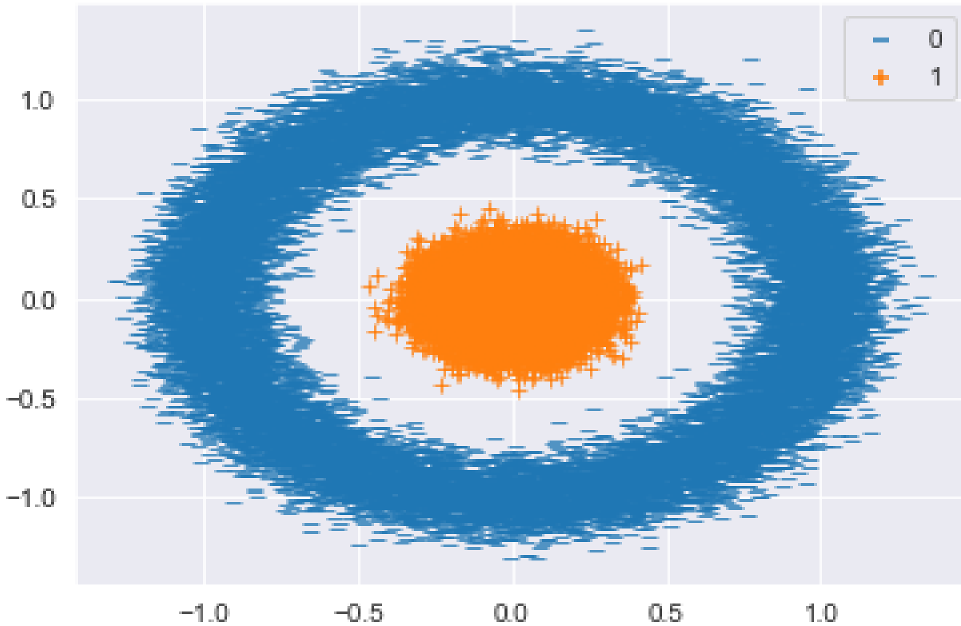
[6]:
```
# define outcome matrix
Y = np.zeros((N, 2))
for c in [0, 1]:
    Y[y == c, c] = 1
```

[7]:
```
f'Shape of: X: {X.shape} | Y: {Y.shape} | y: {y.shape}'
```

[7]:
```
'Shape of: X: (50000, 2) | Y: (50000, 2) | y: (50000,)'
```

1.2.2 Visualize Data

[8]:
```
sns.scatterplot(x=X[:, 0],
                y=X[:, 1],
                hue=y,
               style=y,
               markers=['_', '+']);
```



1.3 Build Keras Model

Keras supports both a slightly simpler Sequential and more flexible Functional API. We will introduce the former at this point and use the Functional API in more complex examples in the following chapters.

To create a model, we just need to instantiate a Sequential object and provide a list with the sequence of standard layers and their configurations, including the number of units, type of activation function, or name.

1.3.1 Define Architecture

[9]:
```
model = Sequential([
    Dense(units=3, input_shape=(2,), name='hidden'),
    Activation('sigmoid', name='logistic'),
    Dense(2, name='output'),
    Activation('softmax', name='softmax'),
])
```

The first hidden layer needs information about the number of features in the matrix it receives from the input layer via the input_shape argument. In our simple case, these are just two. Keras infers the number of rows it needs to process during training, through the batch_size argument that we will pass to the fit method below.

Keras infers the sizes of the inputs received by other layers from the previous layer's units argument.

Keras provides numerous standard building blocks, including recurrent and convolutional layers, various options for regularization, a range of loss functions and optimizers, and also preprocessing, visualization and logging (see documentation on GitHub for reference). It is also extensible.

The model's summary method produces a concise description of the network architecture, including a list of the layer types and shapes, and the number of parameters:

[10]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
hidden (Dense)               (None, 3)                 9
_________________________________________________________________
logistic (Activation)        (None, 3)                 0
_________________________________________________________________
output (Dense)               (None, 2)                 8
_________________________________________________________________
softmax (Activation)         (None, 2)                 0
=================================================================
Total params: 17
Trainable params: 17
Non-trainable params: 0
_________________________________________________________________
```

1.4 Compile Model

Next, we compile the Sequential model to configure the learning process. To this end, we define the optimizer, the loss function, and one or several performance metrics to monitor during training:

[11]:
```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

1.5 Tensorboard Callback

Keras uses callbacks to enable certain functionality during training, such as logging information for interactive display in TensorBoard (see next section):

[12]:
```
tb_callback = TensorBoard(log_dir='./tensorboard',
                          histogram_freq=1,
                          write_graph=True,
                          write_images=True)
```

1.6 Train Model

To train the model, we call its fit method and pass several parameters in addition to the training data:

[13]:
```
model.fit(X,
          Y,
          epochs=25,
          validation_split=.2,
          batch_size=128,
          verbose=1,
          callbacks=[tb_callback])
```

```
Train on 40000 samples, validate on 10000 samples
Epoch 1/25
40000/40000 [==============================] - 0s 5us/step - loss: 0.7062 - acc:
0.4973 - val_loss: 0.6921 - val_acc: 0.5785
Epoch 2/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.6915 - acc:
0.6147 - val_loss: 0.6900 - val_acc: 0.3484
Epoch 3/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.6886 - acc:
0.5819 - val_loss: 0.6858 - val_acc: 0.7033
Epoch 4/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.6815 - acc:
0.6847 - val_loss: 0.6757 - val_acc: 0.8074
Epoch 5/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.6660 - acc:
0.7901 - val_loss: 0.6544 - val_acc: 0.8581
Epoch 6/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.6385 - acc:
0.8732 - val_loss: 0.6207 - val_acc: 0.8806
Epoch 7/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.5992 - acc:
0.8884 - val_loss: 0.5772 - val_acc: 0.8871
Epoch 8/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.5516 - acc:
0.8950 - val_loss: 0.5277 - val_acc: 0.8896
Epoch 9/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.5004 - acc:
0.8985 - val_loss: 0.4773 - val_acc: 0.8929
Epoch 10/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.4503 - acc:
0.9007 - val_loss: 0.4286 - val_acc: 0.8950
Epoch 11/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.4025 - acc:
0.9032 - val_loss: 0.3831 - val_acc: 0.8967
Epoch 12/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.3582 - acc:
0.9068 - val_loss: 0.3408 - val_acc: 0.9014
Epoch 13/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.3163 - acc:
```

```
0.9106 - val_loss: 0.2996 - val_acc: 0.9088
Epoch 14/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.2751 - acc:
0.9177 - val_loss: 0.2583 - val_acc: 0.9178
Epoch 15/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.2340 - acc:
0.9284 - val_loss: 0.2174 - val_acc: 0.9311
Epoch 16/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.1949 - acc:
0.9488 - val_loss: 0.1798 - val_acc: 0.9546
Epoch 17/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.1597 - acc:
0.9685 - val_loss: 0.1462 - val_acc: 0.9724
Epoch 18/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.1286 - acc:
0.9893 - val_loss: 0.1170 - val_acc: 0.9972
Epoch 19/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.1024 - acc:
0.9990 - val_loss: 0.0928 - val_acc: 0.9995
Epoch 20/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.0803 - acc:
0.9997 - val_loss: 0.0720 - val_acc: 0.9997
Epoch 21/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.0620 - acc:
0.9998 - val_loss: 0.0556 - val_acc: 0.9998
Epoch 22/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.0474 - acc:
0.9999 - val_loss: 0.0421 - val_acc: 0.9998
Epoch 23/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.0358 - acc:
0.9999 - val_loss: 0.0318 - val_acc: 1.0000
Epoch 24/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.0270 - acc:
0.9999 - val_loss: 0.0240 - val_acc: 1.0000
Epoch 25/25
40000/40000 [==============================] - 0s 4us/step - loss: 0.0204 - acc:
0.9999 - val_loss: 0.0182 - val_acc: 1.0000
```

[13]:
```
<keras.callbacks.History at 0x7f1613583fd0>
```

1.7 Get Weights

[14]:
```python
hidden = model.get_layer('hidden').get_weights()
```

[15]:
```python
[t.shape for t in hidden]
```

[15]:
```
[(2, 3), (3,)]
```

1.8 Plot Decision Boundary

The visualization of the decision boundary resembles the result from the manual network implementation. The training with Keras runs a multiple faster, though.

[16]:
```
n_vals = 200
x1 = np.linspace(-1.5, 1.5, num=n_vals)
x2 = np.linspace(-1.5, 1.5, num=n_vals)
xx, yy = np.meshgrid(x1, x2)  # create the grid
```

[17]:
```
X_ = np.array([xx.ravel(), yy.ravel()]).T
```

[18]:
```
y_hat = np.argmax(model.predict(X_), axis=1)
```

[19]:
```
# Create a color map to show the classification colors of each grid point
cmap = ListedColormap([sns.xkcd_rgb["pale red"],
                       sns.xkcd_rgb["denim blue"]])

# Plot the classification plane with decision boundary and input samples
plt.contourf(xx, yy, y_hat.reshape(n_vals, -1), cmap=cmap, alpha=.25)

# Plot both classes on the x1, x2 plane
data = pd.DataFrame(X, columns=['$x_1$', '$x_2$']).assign(Class=pd.Series(y).
 ↪map({0:'negative', 1:'positive'}))
sns.scatterplot(x='$x_1$', y='$x_2$', hue='Class', data=data, style=y,␣
 ↪markers=['_', '+'], legend=False)
plt.title('Decision Boundary');
```