TP05-IntroductionToKeras

September 18, 2019

1 Introduction to Keras

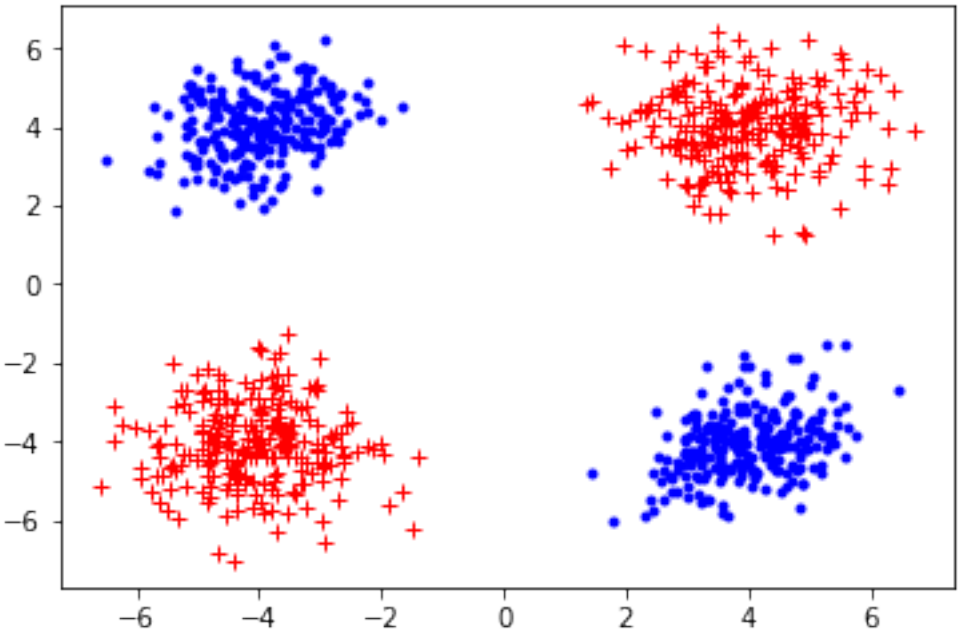
Keras is a High Level API above Tensorflow to design Neural Network. In the next release of Tensorflow, it will be the default way to build such models (General Introduction Guide).

Look at the following code. It produce the same network as the precedent notebook. You can inspect the learning by lauching tensorboard on the "logs" folder:

```
tensorboard --logdir=logs
```

[1]:
```
import numpy as np
import tensorflow as tf
from tensorflow import keras

import matplotlib.pyplot as plt
%matplotlib inline
import datetime as dt


def generate_all_dataset():
    # --- Fake dataset ---

    np.random.seed(0)

    ntrain = 1000
    nvalid = 100
    ntest = 100

    mupos = np.array([4., 4.])
    sigmapos = np.array([[1., 0.], [0., 1.]])
    muneg = np.array([4., -4.])
    sigmaneg = np.array([[.7, .2], [.2, .7]])

    def generate_a_dataset(mupos, sigmapos, muneg, sigmaneg, n):
        nelem = int(n/4)
        npos1 = n-nelem*3
        npos2, nneg1, nneg2 = nelem, nelem, nelem
```

```
        Xpos1 = np.random.multivariate_normal(mupos, sigmapos, npos1)
        Ypos1 = np.stack((np.ones((npos1,)), np.zeros((npos1,))), axis=1)
        Xpos2 = np.random.multivariate_normal(-mupos, sigmapos, npos2)
        Ypos2 = np.stack((np.ones((npos2,)), np.zeros((npos2,))), axis=1)

        Xneg1 = np.random.multivariate_normal(muneg, sigmaneg, nneg1)
        Yneg1 = np.stack((np.zeros((nneg1,)), np.ones((nneg1,))), axis=1)
        Xneg2 = np.random.multivariate_normal(-muneg, sigmaneg, nneg2)
        Yneg2 = np.stack((np.zeros((nneg2,)), np.ones((nneg2,))), axis=1)

        X = np.concatenate((Xpos1, Xpos2, Xneg1, Xneg2))
        Y = np.concatenate((Ypos1, Ypos2, Yneg1, Yneg2))

        idx = np.arange(n)
        np.random.shuffle(idx)

        X, Y = X[idx], Y[idx]

        return np.array(X, dtype='float32'), np.array(Y, dtype='float32')

    Xtrain, Ytrain = generate_a_dataset(
        mupos, sigmapos, muneg, sigmaneg, ntrain)
    Xvalid, Yvalid = generate_a_dataset(
        mupos, sigmapos, muneg, sigmaneg, nvalid)
    Xtest, Ytest = generate_a_dataset(mupos, sigmapos, muneg, sigmaneg, ntest)

    return (Xtrain, Ytrain), (Xvalid, Yvalid), (Xtest, Ytest)


def plot_dataset(X, Y):
    plt.figure()
    idpos, = np.nonzero(Y[:, 0] == 1.)
    idneg, = np.nonzero(Y[:, 1] == 1.)
    plt.plot(X[idpos, 0], X[idpos, 1], 'r+')
    plt.plot(X[idneg, 0], X[idneg, 1], 'b.')
    plt.show()


def demo(trainset, validset, testset):
    plot_dataset(*trainset)


demo(*generate_all_dataset())
```



[2]:
```
def mlpkerasv1(trainset, validset, testset):

    # Reset keras
    keras.backend.clear_session()

    Xtrain, Ytrain = trainset
    _, ninputs = Xtrain.shape
    _, noutputs = Ytrain.shape
    Xvalid, Yvalid = validset
    Xtest, Ytest = testset

    nhiddens = 10

    # Build the model
    model = keras.Sequential()
    model.add(keras.layers.Dense(nhiddens,  # indicates the number of outputs␣
↪for that layer,
                                 activation='sigmoid',  # indicates the␣
↪activation function of the layer
                                 kernel_initializer=keras.initializers.Zeros(),␣
↪ # optional weight initializer
                                 bias_initializer=keras.initializers.Zeros(), ␣
↪# optional bias initializer
                                 input_shape=Xtrain[0].shape, # optional the␣
↪shape of the input
```

```
                                name='Layer1'  # optional layer name
                                ))
    model.add(keras.layers.Dense(noutputs,
                                activation='softmax',
                                kernel_initializer='zeros',  # equivalent␣
↪notation as precedent layer
                                bias_initializer='zeros',
                                name='Layer2'
                                ))

    # Compile the model with optimizer, loss and metrics
    model.compile(
        optimizer=keras.optimizers.SGD(lr=1e-1),  # indicates the optimizer
        loss=keras.losses.categorical_crossentropy,  # indicates the training␣
↪loss,
        # loss='categorical_crossentropy' # equivalent notation
        metrics=[keras.metrics.categorical_accuracy]  # indicates the metrics
        # metrics=['categorical_accuracy'] # equivalent notation
    )

    # print out a summup of the model
    model.build(Xtrain[0].shape) #need to be called before summary
    model.summary()

    # Optional : tensorboard logfiles
    logdir = "logs/mlp/" + dt.datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback = keras.callbacks.TensorBoard(log_dir=logdir,
                                                       write_grads=True,
                                                       histogram_freq=1)

    # The actual training
    model.fit(Xtrain, Ytrain,  # training dataset, you can also give a␣
↪tensorflow dataset here
              epochs=100, batch_size=5,
              validation_data=(Xvalid, Yvalid),
              # optional for tensorboard logging
              callbacks=[tensorboard_callback],
              )

    # Evaluation of the model on the tet set
    # Compute outputs on the test set
    Ytestpred = model.predict(Xtest) > .5

    plot_dataset(Xtest, Ytestpred)

    # That's all !
```

```
mlpkerasv1(*generate_all_dataset())
```

```
WARNING:tensorflow:From /home/rherault/.local/venvs/spyder/lib/python3.7/site-
packages/tensorflow/python/ops/resource_variable_ops.py:435: colocate_with (from
tensorflow.python.framework.ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Colocations handled automatically by placer.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Layer1 (Dense)               (None, 10)                30
_________________________________________________________________
Layer2 (Dense)               (None, 2)                 22
=================================================================
Total params: 52
Trainable params: 52
Non-trainable params: 0
_________________________________________________________________
Train on 1000 samples, validate on 100 samples
WARNING:tensorflow:From /home/rherault/.local/venvs/spyder/lib/python3.7/site-
packages/tensorflow/python/ops/math_ops.py:3066: to_int32 (from
tensorflow.python.ops.math_ops) is deprecated and will be removed in a future
version.
Instructions for updating:
Use tf.cast instead.
Epoch 1/100
1000/1000 [==============================] - 0s 285us/sample - loss: 0.7035 -
categorical_accuracy: 0.4990 - val_loss: 0.7350 - val_categorical_accuracy:
0.5000
Epoch 2/100
1000/1000 [==============================] - 0s 133us/sample - loss: 0.7030 -
categorical_accuracy: 0.5010 - val_loss: 0.6936 - val_categorical_accuracy:
0.5000
Epoch 3/100
1000/1000 [==============================] - 0s 130us/sample - loss: 0.7032 -
categorical_accuracy: 0.4890 - val_loss: 0.6939 - val_categorical_accuracy:
0.5000
Epoch 4/100
1000/1000 [==============================] - 0s 131us/sample - loss: 0.7036 -
categorical_accuracy: 0.4960 - val_loss: 0.7011 - val_categorical_accuracy:
0.5000
Epoch 5/100
1000/1000 [==============================] - 0s 127us/sample - loss: 0.7049 -
categorical_accuracy: 0.4830 - val_loss: 0.7139 - val_categorical_accuracy:
0.5000
```

```
Epoch 6/100
1000/1000 [==============================] - 0s 139us/sample - loss: 0.6987 -
categorical_accuracy: 0.5090 - val_loss: 0.7062 - val_categorical_accuracy:
0.5000
Epoch 7/100
1000/1000 [==============================] - 0s 159us/sample - loss: 0.7059 -
categorical_accuracy: 0.4610 - val_loss: 0.7075 - val_categorical_accuracy:
0.5000
Epoch 8/100
1000/1000 [==============================] - 0s 148us/sample - loss: 0.7044 -
categorical_accuracy: 0.4770 - val_loss: 0.7065 - val_categorical_accuracy:
0.5000
Epoch 9/100
1000/1000 [==============================] - 0s 139us/sample - loss: 0.7014 -
categorical_accuracy: 0.5010 - val_loss: 0.6943 - val_categorical_accuracy:
0.5000
Epoch 10/100
1000/1000 [==============================] - 0s 132us/sample - loss: 0.7057 -
categorical_accuracy: 0.4860 - val_loss: 0.7042 - val_categorical_accuracy:
0.5000
Epoch 11/100
1000/1000 [==============================] - 0s 131us/sample - loss: 0.7020 -
categorical_accuracy: 0.5170 - val_loss: 0.6971 - val_categorical_accuracy:
0.5000
Epoch 12/100
1000/1000 [==============================] - 0s 132us/sample - loss: 0.7051 -
categorical_accuracy: 0.4790 - val_loss: 0.6946 - val_categorical_accuracy:
0.5000
Epoch 13/100
1000/1000 [==============================] - 0s 137us/sample - loss: 0.7024 -
categorical_accuracy: 0.4870 - val_loss: 0.6998 - val_categorical_accuracy:
0.5000
Epoch 14/100
1000/1000 [==============================] - 0s 135us/sample - loss: 0.7027 -
categorical_accuracy: 0.5040 - val_loss: 0.6961 - val_categorical_accuracy:
0.5000
Epoch 15/100
1000/1000 [==============================] - 0s 124us/sample - loss: 0.7027 -
categorical_accuracy: 0.4890 - val_loss: 0.6944 - val_categorical_accuracy:
0.5000
Epoch 16/100
1000/1000 [==============================] - 0s 122us/sample - loss: 0.7002 -
categorical_accuracy: 0.5080 - val_loss: 0.7089 - val_categorical_accuracy:
0.5000
Epoch 17/100
1000/1000 [==============================] - 0s 163us/sample - loss: 0.6964 -
categorical_accuracy: 0.5120 - val_loss: 0.6955 - val_categorical_accuracy:
0.5000
```

```
Epoch 18/100
1000/1000 [==============================] - 0s 243us/sample - loss: 0.7015 -
categorical_accuracy: 0.4950 - val_loss: 0.6978 - val_categorical_accuracy:
0.5000
Epoch 19/100
1000/1000 [==============================] - 0s 178us/sample - loss: 0.7041 -
categorical_accuracy: 0.4830 - val_loss: 0.6943 - val_categorical_accuracy:
0.5000
Epoch 20/100
1000/1000 [==============================] - 0s 163us/sample - loss: 0.6990 -
categorical_accuracy: 0.5080 - val_loss: 0.6932 - val_categorical_accuracy:
0.5000
Epoch 21/100
1000/1000 [==============================] - 0s 214us/sample - loss: 0.7007 -
categorical_accuracy: 0.5100 - val_loss: 0.6957 - val_categorical_accuracy:
0.5000
Epoch 22/100
1000/1000 [==============================] - 0s 199us/sample - loss: 0.6981 -
categorical_accuracy: 0.5140 - val_loss: 0.7014 - val_categorical_accuracy:
0.5000
Epoch 23/100
1000/1000 [==============================] - 0s 186us/sample - loss: 0.7040 -
categorical_accuracy: 0.4900 - val_loss: 0.6963 - val_categorical_accuracy:
0.5000
Epoch 24/100
1000/1000 [==============================] - 0s 226us/sample - loss: 0.7004 -
categorical_accuracy: 0.5090 - val_loss: 0.6931 - val_categorical_accuracy:
0.4700
Epoch 25/100
1000/1000 [==============================] - 0s 148us/sample - loss: 0.7031 -
categorical_accuracy: 0.4900 - val_loss: 0.6993 - val_categorical_accuracy:
0.5000
Epoch 26/100
1000/1000 [==============================] - 0s 139us/sample - loss: 0.6987 -
categorical_accuracy: 0.5210 - val_loss: 0.6992 - val_categorical_accuracy:
0.5000
Epoch 27/100
1000/1000 [==============================] - 0s 244us/sample - loss: 0.7000 -
categorical_accuracy: 0.4900 - val_loss: 0.6937 - val_categorical_accuracy:
0.5000
Epoch 28/100
1000/1000 [==============================] - 0s 158us/sample - loss: 0.6985 -
categorical_accuracy: 0.5050 - val_loss: 0.6966 - val_categorical_accuracy:
0.5000
Epoch 29/100
1000/1000 [==============================] - 0s 160us/sample - loss: 0.7012 -
categorical_accuracy: 0.4890 - val_loss: 0.6935 - val_categorical_accuracy:
0.5000
```

```
Epoch 30/100
1000/1000 [==============================] - 0s 162us/sample - loss: 0.7032 -
categorical_accuracy: 0.4570 - val_loss: 0.6962 - val_categorical_accuracy:
0.5000
Epoch 31/100
1000/1000 [==============================] - 0s 158us/sample - loss: 0.7018 -
categorical_accuracy: 0.5050 - val_loss: 0.6930 - val_categorical_accuracy:
0.2700
Epoch 32/100
1000/1000 [==============================] - 0s 159us/sample - loss: 0.7008 -
categorical_accuracy: 0.4730 - val_loss: 0.6953 - val_categorical_accuracy:
0.5000
Epoch 33/100
1000/1000 [==============================] - 0s 160us/sample - loss: 0.6970 -
categorical_accuracy: 0.5220 - val_loss: 0.7067 - val_categorical_accuracy:
0.5000
Epoch 34/100
1000/1000 [==============================] - 0s 151us/sample - loss: 0.7019 -
categorical_accuracy: 0.4870 - val_loss: 0.7250 - val_categorical_accuracy:
0.5000
Epoch 35/100
1000/1000 [==============================] - 0s 135us/sample - loss: 0.6997 -
categorical_accuracy: 0.5030 - val_loss: 0.6949 - val_categorical_accuracy:
0.5000
Epoch 36/100
1000/1000 [==============================] - 0s 123us/sample - loss: 0.6981 -
categorical_accuracy: 0.5120 - val_loss: 0.6928 - val_categorical_accuracy:
0.2500
Epoch 37/100
1000/1000 [==============================] - 0s 123us/sample - loss: 0.7044 -
categorical_accuracy: 0.4580 - val_loss: 0.6934 - val_categorical_accuracy:
0.5000
Epoch 38/100
1000/1000 [==============================] - 0s 123us/sample - loss: 0.6950 -
categorical_accuracy: 0.5310 - val_loss: 0.6960 - val_categorical_accuracy:
0.5000
Epoch 39/100
1000/1000 [==============================] - 0s 125us/sample - loss: 0.6980 -
categorical_accuracy: 0.4780 - val_loss: 0.7006 - val_categorical_accuracy:
0.5000
Epoch 40/100
1000/1000 [==============================] - 0s 124us/sample - loss: 0.7007 -
categorical_accuracy: 0.4900 - val_loss: 0.6982 - val_categorical_accuracy:
0.5000
Epoch 41/100
1000/1000 [==============================] - 0s 126us/sample - loss: 0.7012 -
categorical_accuracy: 0.4680 - val_loss: 0.6916 - val_categorical_accuracy:
0.7300
```

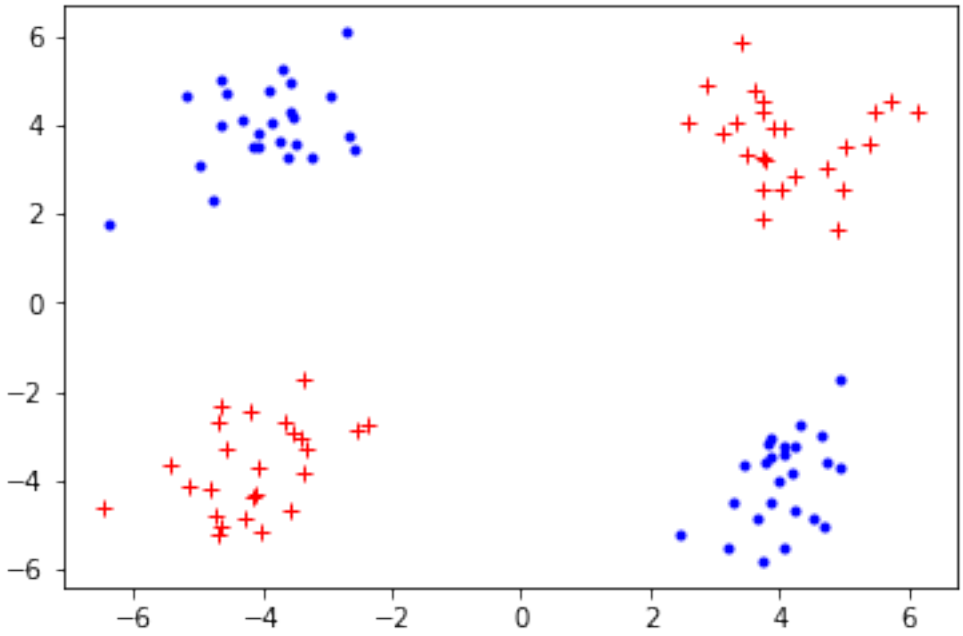
```
Epoch 42/100
1000/1000 [==============================] - 0s 140us/sample - loss: 0.7005 -
categorical_accuracy: 0.4510 - val_loss: 0.6941 - val_categorical_accuracy:
0.3100
Epoch 43/100
1000/1000 [==============================] - 0s 124us/sample - loss: 0.6961 -
categorical_accuracy: 0.4680 - val_loss: 0.6944 - val_categorical_accuracy:
0.2500
Epoch 44/100
1000/1000 [==============================] - 0s 136us/sample - loss: 0.6963 -
categorical_accuracy: 0.4540 - val_loss: 0.6901 - val_categorical_accuracy:
0.2500
Epoch 45/100
1000/1000 [==============================] - 0s 148us/sample - loss: 0.6879 -
categorical_accuracy: 0.4890 - val_loss: 0.6825 - val_categorical_accuracy:
0.3400
Epoch 46/100
1000/1000 [==============================] - 0s 150us/sample - loss: 0.6853 -
categorical_accuracy: 0.5260 - val_loss: 0.6732 - val_categorical_accuracy:
0.6300
Epoch 47/100
1000/1000 [==============================] - 0s 136us/sample - loss: 0.6774 -
categorical_accuracy: 0.5720 - val_loss: 0.6642 - val_categorical_accuracy:
0.7500
Epoch 48/100
1000/1000 [==============================] - 0s 128us/sample - loss: 0.6493 -
categorical_accuracy: 0.6150 - val_loss: 0.6179 - val_categorical_accuracy:
0.6800
Epoch 49/100
1000/1000 [==============================] - 0s 127us/sample - loss: 0.5588 -
categorical_accuracy: 0.6870 - val_loss: 0.4373 - val_categorical_accuracy:
0.7500
Epoch 50/100
1000/1000 [==============================] - 0s 126us/sample - loss: 0.3006 -
categorical_accuracy: 0.9570 - val_loss: 0.1929 - val_categorical_accuracy:
1.0000
Epoch 51/100
1000/1000 [==============================] - 0s 128us/sample - loss: 0.1414 -
categorical_accuracy: 0.9970 - val_loss: 0.1017 - val_categorical_accuracy:
1.0000
Epoch 52/100
1000/1000 [==============================] - 0s 127us/sample - loss: 0.0850 -
categorical_accuracy: 0.9960 - val_loss: 0.0668 - val_categorical_accuracy:
1.0000
Epoch 53/100
1000/1000 [==============================] - 0s 130us/sample - loss: 0.0603 -
categorical_accuracy: 0.9990 - val_loss: 0.0490 - val_categorical_accuracy:
1.0000
```

```
Epoch 54/100
1000/1000 [==============================] - 0s 132us/sample - loss: 0.0464 -
categorical_accuracy: 0.9980 - val_loss: 0.0365 - val_categorical_accuracy:
1.0000
Epoch 55/100
1000/1000 [==============================] - 0s 141us/sample - loss: 0.0379 -
categorical_accuracy: 0.9990 - val_loss: 0.0299 - val_categorical_accuracy:
1.0000
Epoch 56/100
1000/1000 [==============================] - 0s 152us/sample - loss: 0.0318 -
categorical_accuracy: 0.9990 - val_loss: 0.0253 - val_categorical_accuracy:
1.0000
Epoch 57/100
1000/1000 [==============================] - 0s 159us/sample - loss: 0.0282 -
categorical_accuracy: 0.9990 - val_loss: 0.0209 - val_categorical_accuracy:
1.0000
Epoch 58/100
1000/1000 [==============================] - 0s 160us/sample - loss: 0.0248 -
categorical_accuracy: 0.9990 - val_loss: 0.0185 - val_categorical_accuracy:
1.0000
Epoch 59/100
1000/1000 [==============================] - 0s 160us/sample - loss: 0.0220 -
categorical_accuracy: 0.9990 - val_loss: 0.0180 - val_categorical_accuracy:
1.0000
Epoch 60/100
1000/1000 [==============================] - 0s 141us/sample - loss: 0.0202 -
categorical_accuracy: 0.9990 - val_loss: 0.0154 - val_categorical_accuracy:
1.0000
Epoch 61/100
1000/1000 [==============================] - 0s 131us/sample - loss: 0.0183 -
categorical_accuracy: 0.9990 - val_loss: 0.0132 - val_categorical_accuracy:
1.0000
Epoch 62/100
1000/1000 [==============================] - 0s 124us/sample - loss: 0.0174 -
categorical_accuracy: 0.9990 - val_loss: 0.0126 - val_categorical_accuracy:
1.0000
Epoch 63/100
1000/1000 [==============================] - 0s 126us/sample - loss: 0.0159 -
categorical_accuracy: 0.9990 - val_loss: 0.0123 - val_categorical_accuracy:
1.0000
Epoch 64/100
1000/1000 [==============================] - 0s 126us/sample - loss: 0.0150 -
categorical_accuracy: 0.9990 - val_loss: 0.0108 - val_categorical_accuracy:
1.0000
Epoch 65/100
1000/1000 [==============================] - 0s 126us/sample - loss: 0.0135 -
categorical_accuracy: 1.0000 - val_loss: 0.0097 - val_categorical_accuracy:
1.0000
```

```
Epoch 66/100
1000/1000 [==============================] - 0s 126us/sample - loss: 0.0132 -
categorical_accuracy: 0.9990 - val_loss: 0.0096 - val_categorical_accuracy:
1.0000
Epoch 67/100
1000/1000 [==============================] - 0s 124us/sample - loss: 0.0121 -
categorical_accuracy: 0.9990 - val_loss: 0.0083 - val_categorical_accuracy:
1.0000
Epoch 68/100
1000/1000 [==============================] - 0s 125us/sample - loss: 0.0117 -
categorical_accuracy: 0.9990 - val_loss: 0.0083 - val_categorical_accuracy:
1.0000
Epoch 69/100
1000/1000 [==============================] - 0s 124us/sample - loss: 0.0113 -
categorical_accuracy: 0.9990 - val_loss: 0.0083 - val_categorical_accuracy:
1.0000
Epoch 70/100
1000/1000 [==============================] - 0s 124us/sample - loss: 0.0108 -
categorical_accuracy: 0.9990 - val_loss: 0.0076 - val_categorical_accuracy:
1.0000
Epoch 71/100
1000/1000 [==============================] - 0s 132us/sample - loss: 0.0101 -
categorical_accuracy: 0.9990 - val_loss: 0.0069 - val_categorical_accuracy:
1.0000
Epoch 72/100
1000/1000 [==============================] - 0s 123us/sample - loss: 0.0100 -
categorical_accuracy: 0.9990 - val_loss: 0.0071 - val_categorical_accuracy:
1.0000
Epoch 73/100
1000/1000 [==============================] - 0s 123us/sample - loss: 0.0093 -
categorical_accuracy: 0.9990 - val_loss: 0.0073 - val_categorical_accuracy:
1.0000
Epoch 74/100
1000/1000 [==============================] - 0s 125us/sample - loss: 0.0094 -
categorical_accuracy: 0.9990 - val_loss: 0.0066 - val_categorical_accuracy:
1.0000
Epoch 75/100
1000/1000 [==============================] - 0s 126us/sample - loss: 0.0089 -
categorical_accuracy: 0.9990 - val_loss: 0.0059 - val_categorical_accuracy:
1.0000
Epoch 76/100
1000/1000 [==============================] - 0s 126us/sample - loss: 0.0086 -
categorical_accuracy: 0.9990 - val_loss: 0.0057 - val_categorical_accuracy:
1.0000
Epoch 77/100
1000/1000 [==============================] - 0s 125us/sample - loss: 0.0086 -
categorical_accuracy: 0.9990 - val_loss: 0.0056 - val_categorical_accuracy:
1.0000
```

```
Epoch 78/100
1000/1000 [==============================] - 0s 128us/sample - loss: 0.0083 -
categorical_accuracy: 0.9990 - val_loss: 0.0057 - val_categorical_accuracy:
1.0000
Epoch 79/100
1000/1000 [==============================] - 0s 125us/sample - loss: 0.0077 -
categorical_accuracy: 0.9990 - val_loss: 0.0050 - val_categorical_accuracy:
1.0000
Epoch 80/100
1000/1000 [==============================] - 0s 125us/sample - loss: 0.0077 -
categorical_accuracy: 0.9990 - val_loss: 0.0052 - val_categorical_accuracy:
1.0000
Epoch 81/100
1000/1000 [==============================] - 0s 125us/sample - loss: 0.0074 -
categorical_accuracy: 0.9990 - val_loss: 0.0053 - val_categorical_accuracy:
1.0000
Epoch 82/100
1000/1000 [==============================] - 0s 125us/sample - loss: 0.0073 -
categorical_accuracy: 0.9990 - val_loss: 0.0050 - val_categorical_accuracy:
1.0000
Epoch 83/100
1000/1000 [==============================] - 0s 125us/sample - loss: 0.0069 -
categorical_accuracy: 0.9990 - val_loss: 0.0046 - val_categorical_accuracy:
1.0000
Epoch 84/100
1000/1000 [==============================] - 0s 125us/sample - loss: 0.0067 -
categorical_accuracy: 0.9990 - val_loss: 0.0045 - val_categorical_accuracy:
1.0000
Epoch 85/100
1000/1000 [==============================] - 0s 136us/sample - loss: 0.0066 -
categorical_accuracy: 0.9990 - val_loss: 0.0047 - val_categorical_accuracy:
1.0000
Epoch 86/100
1000/1000 [==============================] - 0s 133us/sample - loss: 0.0064 -
categorical_accuracy: 1.0000 - val_loss: 0.0042 - val_categorical_accuracy:
1.0000
Epoch 87/100
1000/1000 [==============================] - 0s 126us/sample - loss: 0.0062 -
categorical_accuracy: 0.9990 - val_loss: 0.0047 - val_categorical_accuracy:
1.0000
Epoch 88/100
1000/1000 [==============================] - 0s 134us/sample - loss: 0.0062 -
categorical_accuracy: 1.0000 - val_loss: 0.0047 - val_categorical_accuracy:
1.0000
Epoch 89/100
1000/1000 [==============================] - 0s 151us/sample - loss: 0.0057 -
categorical_accuracy: 0.9990 - val_loss: 0.0046 - val_categorical_accuracy:
1.0000
```

```
Epoch 90/100
1000/1000 [==============================] - 0s 130us/sample - loss: 0.0057 -
categorical_accuracy: 1.0000 - val_loss: 0.0037 - val_categorical_accuracy:
1.0000
Epoch 91/100
1000/1000 [==============================] - 0s 134us/sample - loss: 0.0058 -
categorical_accuracy: 0.9990 - val_loss: 0.0043 - val_categorical_accuracy:
1.0000
Epoch 92/100
1000/1000 [==============================] - 0s 206us/sample - loss: 0.0054 -
categorical_accuracy: 1.0000 - val_loss: 0.0036 - val_categorical_accuracy:
1.0000
Epoch 93/100
1000/1000 [==============================] - 0s 220us/sample - loss: 0.0054 -
categorical_accuracy: 0.9990 - val_loss: 0.0040 - val_categorical_accuracy:
1.0000
Epoch 94/100
1000/1000 [==============================] - 0s 175us/sample - loss: 0.0050 -
categorical_accuracy: 1.0000 - val_loss: 0.0034 - val_categorical_accuracy:
1.0000
Epoch 95/100
1000/1000 [==============================] - 0s 155us/sample - loss: 0.0054 -
categorical_accuracy: 0.9990 - val_loss: 0.0036 - val_categorical_accuracy:
1.0000
Epoch 96/100
1000/1000 [==============================] - 0s 161us/sample - loss: 0.0050 -
categorical_accuracy: 1.0000 - val_loss: 0.0033 - val_categorical_accuracy:
1.0000
Epoch 97/100
1000/1000 [==============================] - 0s 210us/sample - loss: 0.0049 -
categorical_accuracy: 1.0000 - val_loss: 0.0033 - val_categorical_accuracy:
1.0000
Epoch 98/100
1000/1000 [==============================] - 0s 195us/sample - loss: 0.0048 -
categorical_accuracy: 1.0000 - val_loss: 0.0031 - val_categorical_accuracy:
1.0000
Epoch 99/100
1000/1000 [==============================] - 0s 155us/sample - loss: 0.0047 -
categorical_accuracy: 1.0000 - val_loss: 0.0032 - val_categorical_accuracy:
1.0000
Epoch 100/100
1000/1000 [==============================] - 0s 156us/sample - loss: 0.0048 -
categorical_accuracy: 0.9990 - val_loss: 0.0031 - val_categorical_accuracy:
1.0000
```



1.1 Exercise

In fact that model behave very badly on that very simple dataset...

Improve the model training by modifying the following aspects:

- the weight initialization available initializers
- the optimizer available optimizers
- the number of hidden units (reduce them)
- the batch size (increase it)
- the learning rate (reduce it)

Actually you can converge in few (<20) iterations on that dataset

1.2 Sequential API

Here we have used the sequential API of Keras to instantiate our model. It is made for Feed Forward Neural Network where the information always flow from one layer to the other.

A simple two layer perceptron:

```
model = keras.Sequential()
model.add(keras.layers.Dense(60,activation='sigmoid'))
model.add(keras.layers.Dense(10,activation='softmax'))
```

1.3 Functional API

If you want more complex model, such a resnet where information can jump above a layer you need to used the functional API.

Here is a network with a skip connection:

```
inputs = tf.keras.Input(shape=(32,))  # Returns a placeholder tensor
h1 = keras.layers.Dense(60,activation='sigmoid')(inputs) # you call the layer on the input
h2  = keras.layers.Dense(60,activation='sigmoid')(h1)
h1plush2 = keras.layers.Add()([h1, h2]) # skip connection
outputs = keras.layers.Dense(10,activation='softmax')(h1plush2)

model = keras.Model(inputs=inputs, outputs=predictions) # Actual creation of the model
```

1.4 Using Keras models in Tensorflow

You can use keras models inside tensorflow as an operation on Tensor

```
x = tf.placeholder(tf.float32, (1, 5))
y = keras_model(x)
```

1.5 Saving and loading a model

You can save a model into a hdf5 file by using the `save`method. Both configuration and parameters are recorded.

```
my_keras_model.save('myfile.h5')
```

Recreate the exact same model, including weights and optimizer.

```
my_loaded_keras_model = keras.models.load_model('myfile.h5')
```