HW4 Deep Learning

2015129053 김형철

1.

(a**) tf.keras.layers.dense**: (1) Basically, it takes in certain input arrays and then uses its matrix(weight parameters) to give certain output arrays. Thus, tf.keras.layers.dense gives out an output which is computed using a specified activation function with its weights layer.

(2) The function has various arguments. “units" is a dimension for the output array. “activation" is the type of activation function it will use. “use_bias" is a Boolean factor to choose whether to have bias vector. “kernel_initializer" is a initializer for the kernel matrix. “bias_initializer" is initializer for the bias vector. There are also other options such as “kernel_regularizer", “activity_regularizer", “bias_constraint", etc.

tf.keras.layers.Dropout: (1) This is a code that implements “Dropout" procedure in Neural Network construction. While performing propagation, the Neural Network drops out some of the units to avoid overfitting issue. This procedure is implemented using this code. During the training process, this code will make the dropout layer to drop some of the units in the Neural Network by some chosen probability(rate).

(2) The arguments of this function are “rate", “noise_shape", and “seed". “rate" is a float variable from 0 to 1 that determines the probability of the frequency of the dropout. “noise_shape" is 1D integer tensor that is multiplied to the input. In a way, it is analogous of the size (or shape) of the dropout mask. “seed" is argument which is used for random seed.

(b) I used following codes to build my own neural network. I chose ReLu functions for the activation functions (except for the last layer for binary classification. I used sigmoid there).

```
# Importing the libraries
from numpy import loadtxt
import numpy as np
import tensorflow as tf
```

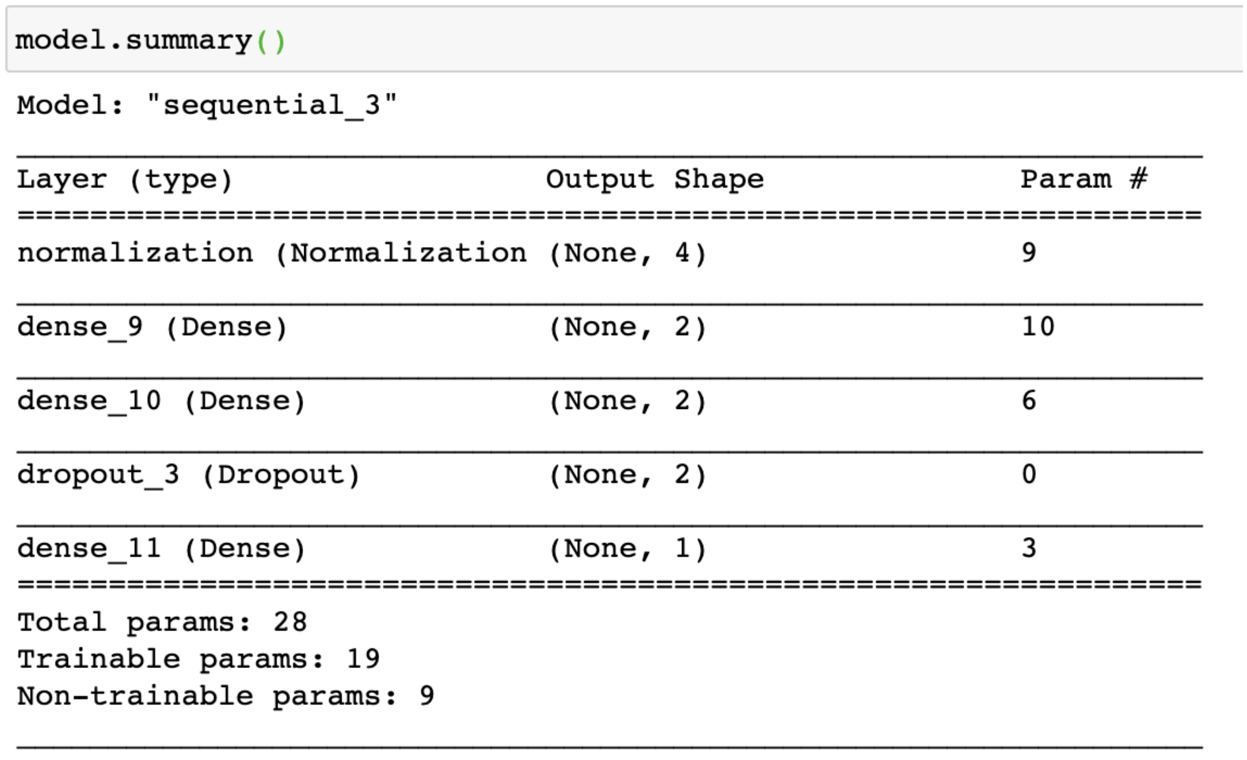
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from tensorflow.keras.layers.experimental import preprocessing

# load the dataset
dataset_train = loadtxt('healthTrain.csv', delimiter=',')
# split into input (X) and output (y) variables
x_train = dataset_train[:,0:4]
y_train = dataset_train[:,4]
print(x_train.shape)
print(y_train.shape)

# normalization
normalizer = preprocessing.Normalization()
normalizer.adapt(np.array(x_train))
print(normalizer.mean.numpy())

# construct the model using the following codes
model = tf.keras.models.Sequential([
  normalizer,
  tf.keras.layers.Dense(2, activation='relu'),
  tf.keras.layers.Dense(2, activation='relu'),
  tf.keras.layers.Dropout(.2),
  tf.keras.layers.Dense(1, activation='sigmoid')
])
```

I used model.summary function to check the structure of my Neural Network:



```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
normalization (Normalization (None, 4)                 9
_________________________________________________________________
dense_9 (Dense)              (None, 2)                 10
_________________________________________________________________
dense_10 (Dense)             (None, 2)                 6
_________________________________________________________________
dropout_3 (Dropout)          (None, 2)                 0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 3
=================================================================
Total params: 28
Trainable params: 19
Non-trainable params: 9
_________________________________________________________________
```

(c) I compiled my model using model.compile function. This is the code I used:

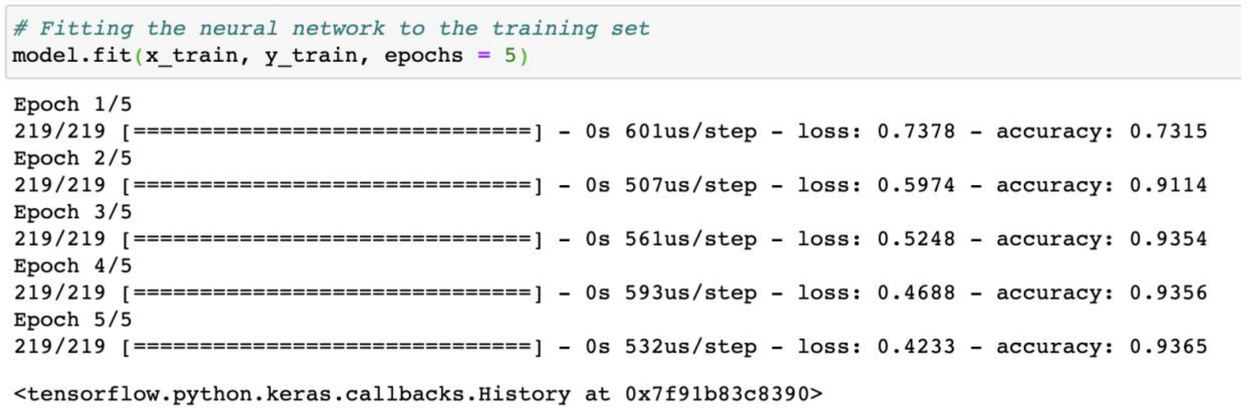
```
# Compiling the model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

(1) The reason we are using binary crossentropy loss function for this problem is because we are trying to perform binary classification of the data. In the case of binary classification, the binary crossentropy loss function is used for the classification model which gives a probability that can be used as an indicator for binary classification (0 or 1).

(2) I fit the neural network model using model.fit function as follows:

```
# Fitting the neural network to the training set
model.fit(x_train, y_train, epochs = 5)
```

(3) The accuracy for the model is as follows: (after 5 epochs, it was around 94%.)



```
# Fitting the neural network to the training set
model.fit(x_train, y_train, epochs = 5)
```

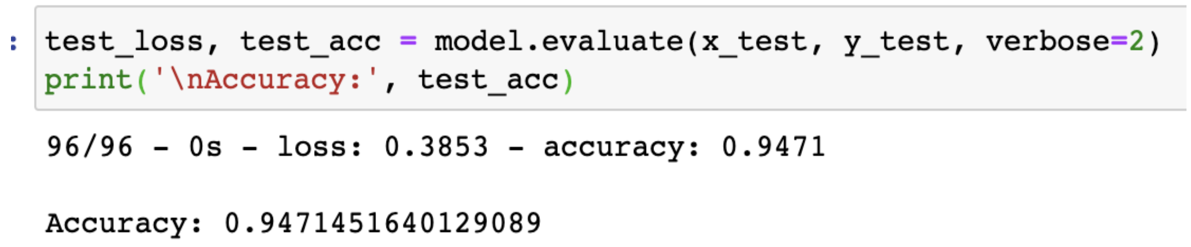
```
Epoch 1/5
219/219 [==============================] - 0s 601us/step - loss: 0.7378 - accuracy: 0.7315
Epoch 2/5
219/219 [==============================] - 0s 507us/step - loss: 0.5974 - accuracy: 0.9114
Epoch 3/5
219/219 [==============================] - 0s 561us/step - loss: 0.5248 - accuracy: 0.9354
Epoch 4/5
219/219 [==============================] - 0s 593us/step - loss: 0.4688 - accuracy: 0.9356
Epoch 5/5
219/219 [==============================] - 0s 532us/step - loss: 0.4233 - accuracy: 0.9365

<tensorflow.python.keras.callbacks.History at 0x7f91b83c8390>
```

(d) After loading the test data set, I checked accuracy of the model using model.evaluate function. It was around 95%.

```
# load the test dataset
dataset_test = loadtxt('healthTest.csv', delimiter=',')
# split into input (X) and output (y) variables
x_test = dataset_test[:,0:4]
y_test = dataset_test[:,4]

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print('\nAccuracy:', test_acc)
```



```
: test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
  print('\nAccuracy:', test_acc)
```

```
96/96 - 0s - loss: 0.3853 - accuracy: 0.9471

Accuracy: 0.9471451640129089
```