**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 that specifies whether the layer will apply bias vector or not. “kernel\_initializer" is basically an initializer for the kernel matrix. “bias\_initializer" is initializer for the bias vector. “kernel\_regularizer" is function that performs regularization, which is used in the kernel matrix mentioned above. “Activity\_regularizer" another function which performs regularization. This is a function used in the layer’s output. “Bias\_constraint" and “Kernel\_constraint” are both constraint function that is used in bias vector and kernel matrix respectively.

**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 to 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:

Table

Description automatically generated

(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%.)

Text

Description automatically generated

(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)*

Graphical user interface, text, application

Description automatically generated