MULTIVARIATE GAUSSIAN DISTRIBUTED DATA ONTO A SPHERE

Wenbo Feng

fengwenb@myvuw.ac.nz

1. INTRODUCTION

This report includes the concept and results of implementing projecting standard normal distributed data on a sphere by using Tensorflow and Keras. The implementation of this assignment can be found on github and Colaboratory
https://github.com/yahoo0742/COMP421/tree/master/ass1
https://colab.research.google.com/drive/1_cg613Q1TI6GumALAr3C7UTdhC-JrySI

2. CONCEPT/IMPLEMENTATION

According to the requirements of the assignment, we first need to generate zero-mean, unit variance normal-distributed data and project the data on a sphere, the 3D normal distributed data can be generated by calling a tensorflow API *tensorflow.random.normal*.
As we are going to project the 3D data on a unit-radius sphere, we should normalize the data generated from the previous step. This can be simply achieved by dividing each element of the data with the Euclidean norm of the element.
The implementation of the two steps is in the function "gendata" in the source code.

For visualizing the training data generated, I use the API scprep.plot.rotate_scatter3d from scprep library to scatter the instances of the data. Figure 1 shows the effect.

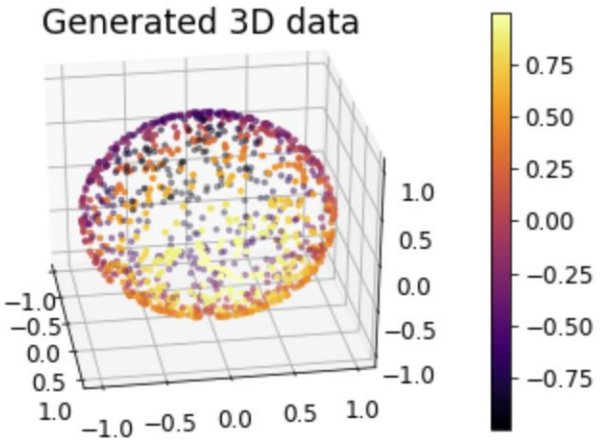

Fig. 1. Gaussian distributed data on a sphere

As required, except the input layer, we need a 3 layers fully-connected neural network model. The network can be created with a *keras.Sequential*, by adding 2 hidden layers and 1 output layer. The activation function of the 2 hidden layers can be "relu". The "relu" function is cheap to compute, and converges fast, and importantly it is not a linear function, so that the layers with non-linear activation functions can generalize or adapt with a variety of data and to differentiate between the output. If every layer has a linear transformation, the final output of the layers is still a linear transformation of the input. The activation function of the output layer is "Linear".

I choose keras built-in function "mean_squared_error" as the loss function. Since this is a regression problem, I considered the choice between "mean squared error", "mean absolute error" loss functions. MAE is a L1 measure, which is more robust to outliers, but the problem is even the loss is tiny, the parameters of the model are still updated in the same scale. Compared to MAE, although with a fixed learning rate, the model is punished for making larger mistakes, so that the values propagated on parameters of the model can be adjusted based on the error. The larger error results in a more aggressive update to the parameters. Consider the data not having any outliers theoretically, MSE is preferred apparently. Adam optimizer is chosen for updating the models with the gradients of the loss to minimize the loss. With the loss function and the optimizer, the model can be compiled by the API *compile*.

The next step is training the model with training data and evaluating the model with test data. At this step, I select 30% of the training data for validation.

3. RESULTS

The visualized network is shown in Figure 2. Figure 3 and Figure 4 illustrate that the model is properly trained at about 200th epoch. At last, with 500 epochs, the loss is 0.0002558, and the accuracy is 0.9867. The weights of the model are also exported to a file model.index under folder logs/normaldistributed3ddata/

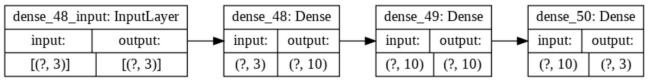

Fig. 2. The network

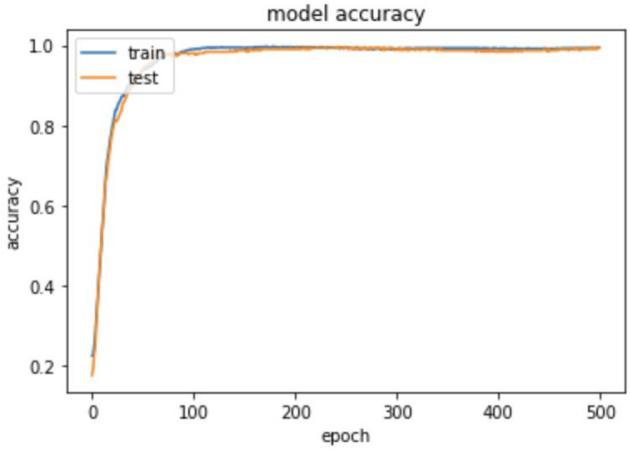

Fig. 3. Model accuracy

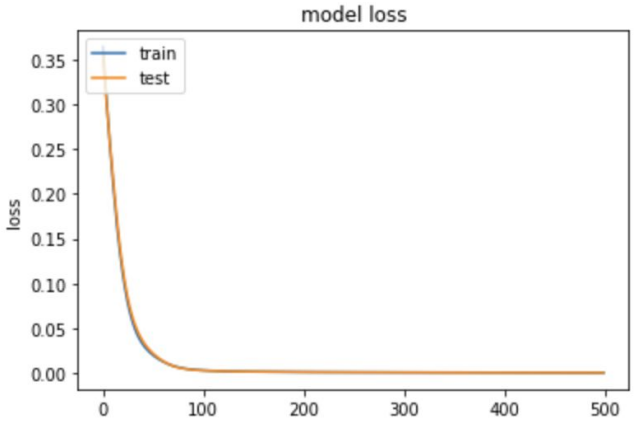

Fig. 4. Model loss