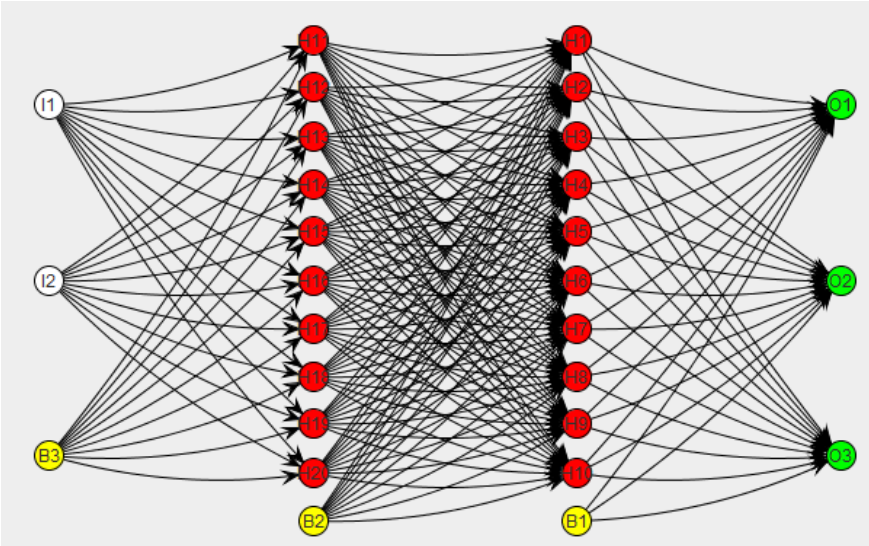
Anuradha Niroshan Follow
Computer Science and Engineering, University of Moratuwa. Lives in Srilanka. "Blogging is a conversation, not a code."
Aug 29, 2017 · 4 min read



Step By Step Guide To Run Your Trained Neural Network Model On Android (Part II)

> *Hope you have enjoyed the part I of this guide.In part II we'll build our neural network, import it and run it on our android project. Complete android studio project is available on my git hub repository.*

Creating a Neural Network using Keras Deep learning library.

Our training data set has 11 features. It contain the details of files which were accessed within 30 days. all the attributes are in numeric format. our target is predicting successor file given set of previously accessed files. So this is a Regression problem. You can use any data set you want and continue. The only change you'll have to do is changing the input and output dimensions of the NN as per your data set.

Our NN will have one hidden layer with 11 nodes and one output node. We select sequential model for creating our NN. The sequential model is a linear stack of layers. Add one hidden layer and output layer to the model. you can find the complete python script in here.

```
# define the model
def larger_model():
    # create model
    model = Sequential()
    model.add(Dense(11, input_dim=11, kernel_initializer='normal', activation="relu"))
    model.add(Dense(1, kernel_initializer='normal', activation="relu"))
    # Compile model
    model.compile(loss='mean_squared_error', optimizer='adam', metrics=['accuracy'])
    return model
```

I'm not going to explain all the parameters one by one. Lets save that for another tutorial. Input dimension is 11 for our data set as we have 11 features. It is enough only to specify the input dimension in the first hidden layer even if you have several hidden layers.

How to save your trained Model as protobuf(.pb) file.

Creating the model and training the model is intuitive. Now lets look at the function where we freeze our model.

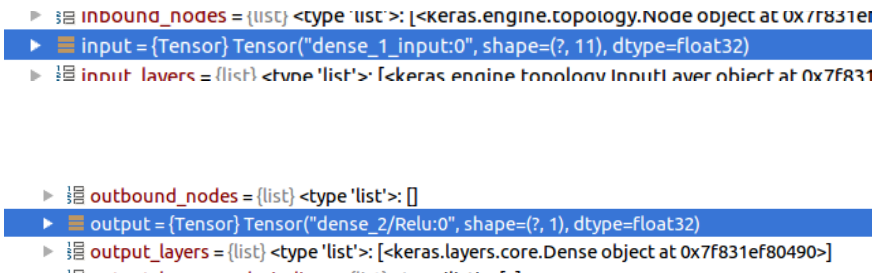
```
def export_model(saver, model, input_node_names, output_node_name):
    tf.train.write_graph(K.get_session().graph_def, 'out', \
                         MODEL_NAME + '_graph.pbtxt')

    saver.save(K.get_session(), 'out/' + MODEL_NAME + '.chkp')

    freeze_graph.freeze_graph('out/' + MODEL_NAME + '_graph.pbtxt', None, \
                              False, 'out/' + MODEL_NAME + '.chkp', output_node_name, \
                              "save/restore_all", "save/Const:0", \
                              'out/frozen_' + MODEL_NAME + '.pb', True, "")

    input_graph_def = tf.GraphDef()
    with tf.gfile.Open('out/frozen_' + MODEL_NAME + '.pb', "rb") as f:
        input_graph_def.ParseFromString(f.read())

    output_graph_def = optimize_for_inference_lib.optimize_for_inference(
        input_graph_def, input_node_names, [output_node_name],
        tf.float32.as_datatype_enum)

    with tf.gfile.FastGFile('out/opt_' + MODEL_NAME + '.pb', "wb") as f:
        f.write(output_graph_def.SerializeToString())

    print("graph saved!")

    return
```

saving our model as pb file

Next most important part is that you should pass the correct input and output node names. Here node names means the nodes in computational graph in tensor-flow back end.

```
export_model(tf.train.Saver(), estimator, ["dense_1_input"], "dense_2/Relu")
```

There are several ways to find these node names. The easiest way is to set a debug pointer after creating the model.



After we run this code the protobuf file will be saved in the out folder. Good job. Now we are almost done.

Importing trained model to your android project and make predictions

Now that we have created the .pb file of our trained model, we can copy paste it in the assets folder in our android app. Before moving into the next part it is necessary that you have installed the android NDK.

The main reason why we need NDK is because we have to inference from tensorflow which is written in c++. Your Java code can then call functions in your native library through the Java Native Interface (JNI) framework.

The following piece of code will call the feed function which output the predicted result. Note that we have used the class TensorFlowInferenceInterface which is written by google to use our pre-trained model.

```
tfHelper.feed(inputName, features,1,inputSize);

tfHelper.run(predictedValue);

tfHelper.fetch(outputName, output);
```

I have just given an array of features and it will print the predicted value in the console. Great!!! we have used our pre-trained model to get the prediction in real time from our android application. Now you can create your own ML model and use it in your android app.

References

[1]https://github.com/llSourcell
/A_Guide_to_Running_Tensorflow_Models_on_Android/tree/master
/mnistandroid

[2]https://www.programcreek.com/2017/01/how-to-select-the-right-tool-for-deep-learning/

[3]https://www.tensorflow.org/

[4]https://keras.io/