Day 7 - Building Complex Models Using the Functional API - Part 2

Tags: Functional API | Wide and Deep Neural Network Architecture

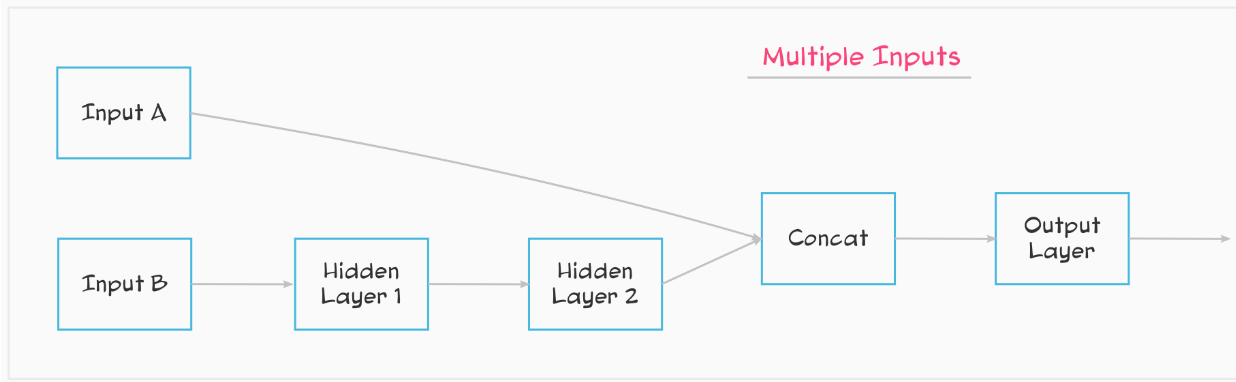
Building Complex Models Using the Functional API - Part 2

Author: **Chandan Kumar**

`enchandan.com`

Multiple Inputs

- It's a good idea to separate these features out and let them escape the deep route.



Implementation

Google Colaboratory

https://colab.research.google.com/drive/1pxDatchdvQtoT5NcYtDkcLyxDhlv3OMz?usp=sharing

- Create the Network

```
input_A = keras.layers.Input(shape=[5], name='wide_input')
input_B = keras.layers.Input(shape=[6], name='deep_input')

hidden1 = keras.layers.Dense(30, activation='relu', name='hidden_1')(input_B)
hidden2 = keras.layers.Dense(30, activation='relu', name='hidden_2')(hidden1)

concat = keras.layers.concatenate([input_A, hidden2])
output = keras.layers.Dense(1, name='output')(concat)

model = keras.Model(inputs=[input_A, input_B], outputs=[output])
```

```
model.summary()

>>>
Model: "functional_3"
_________________________________________________________________________________________________
_
Layer (type)                    Output Shape         Param #     Connected to
=================================================================================================
=
deep_input (InputLayer)         [(None, 6)]          0
```

```
_______________________________________________________________________________________________
_
hidden_1 (Dense)                (None, 30)          210         deep_input[0][0]
_______________________________________________________________________________________________
_
wide_input (InputLayer)         [(None, 5)]         0
_______________________________________________________________________________________________
_
hidden_2 (Dense)                (None, 30)          930         hidden_1[0][0]
_______________________________________________________________________________________________
_
concatenate_1 (Concatenate)     (None, 35)          0           wide_input[0][0]
                                                                hidden_2[0][0]
_______________________________________________________________________________________________
_
output (Dense)                  (None, 1)           36          concatenate_1[0][0]
===============================================================================================
=
Total params: 1,176
Trainable params: 1,176
Non-trainable params: 0
```

- Compile and Train the network

```
model.compile(optimizer='adam', loss='huber_loss')

history = model.fit(x=[X_train[:, :5], X_train[:, 2:]], y=y_train, epochs=30, validation_data=([X
_valid[:, :5], X_valid[:, 2:]], y_valid))
# history = model.fit(X_train, y_train, epochs=30, validation_data=(X_valid, y_valid))

>>>
.
.
Epoch 30/30
363/363 [==============================] - 1s 1ms/step - loss: 0.1238 - val_loss: 0.1273
```

Split Input features

- In a network with multiple inputs, make sure that while calling `model.fit()`
    - Training data `X_train` is split accordingly
    - Validation data `X_valid` is split accordingly
- Also while calling `model.evaluate()`
    - Test data `X_test` is split accordingly

- Evaluate the network

```
hloss_test = model.evaluate([X_test[:, :5], X_test[:, 2:]], y_test)
hloss_test

>>>
0.13067755103111267
```

Experiments

- Take inputs with lower correlation via deep route
- Take inputs with higher correlation via simple route

Implementation

Google Colaboratory

https://colab.research.google.com/drive/18RP16Bq4Zw6qzRaMIfcFP0_H
79DBuM_p?usp=sharing

```
MedInc         0.688075
AveRooms       0.151948
HouseAge       0.105623
AveOccup      -0.023737
Population    -0.024650
Longitude     -0.045967
AveBedrms     -0.046701
Latitude      -0.144160


Higher Correlation -> ['MedInc', 'AveRooms']
Lower Correlation -> ['HouseAge', 'AveOccup', 'Population', 'Longitude', 'AveBedrms', 'Latitude']
```

```
X_train_A, X_train_B = X_train[:, :2], X_train[:, 2:]
X_valid_A, X_valid_B = X_valid[:, :2], X_valid[:, 2:]
X_test_A, X_test_B = X_test[:, :2], X_test[:, 2:]
X_new_A, X_new_B = X_test_A[:3], X_test_B[:3]
```

Training

```
Epoch 30/30
363/363 [==============================] - 1s 1ms/step - loss: 0.1393 - val_loss: 0.1374
```

Evaluate

```
hloss_test = model.evaluate(x=[X_test_A, X_test_B], y=y_test)
hloss_test

>>>
162/162 [==============================] - 0s 843us/step - loss: 0.1425
0.14253628253936768
```

Predict

```
X_new_A, X_new_B = X_test_A[:3], X_test_B[:3]

y_predict = model.predict([X_new_A, X_new_B])
y_predict

>>>
array([[0.44521797],
       [1.041192  ],
       [3.7085667 ]], dtype=float32)
```

Multiple Outputs

Why we need it?

Think about a task where you have to locate and classify the objects in an image.

- Locate
  - Finding the coordinates of the object's center, width & height
  - It looks like a regression problem
- Classify

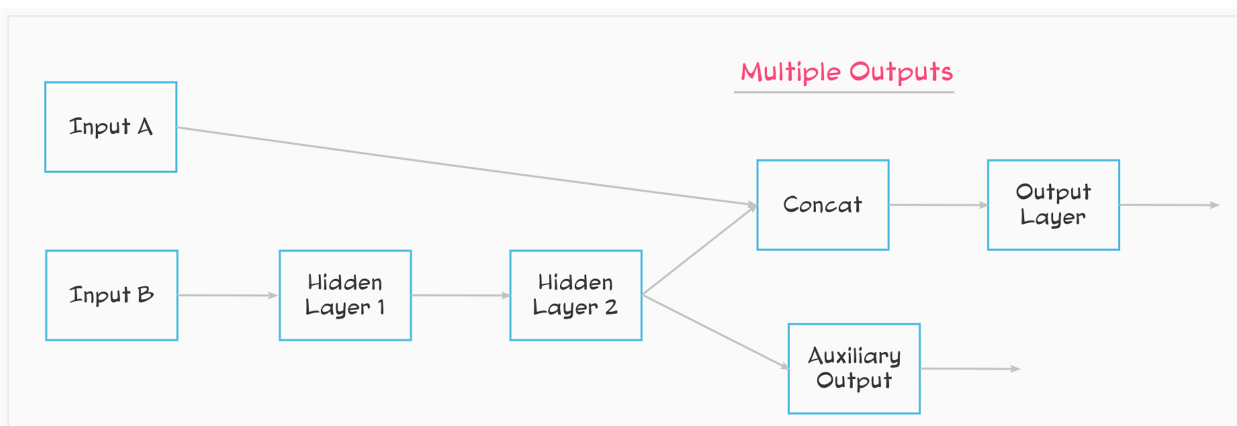
- To classify the object
- which is of course a classification task

Another example, a multitask classification problem →

- In a picture, classify person's facial expression (Smiling, Surprised, etc) with one Output
- Another Output, to identify if they are wearing glasses or not

- You might argue that we can always train separate neural network for each of these different tasks. And yes we can, train one neural network per task.
- Though we get better results on all tasks if we train a single neural network with one output per task. The neural network learns features which are useful across tasks.

- One more use case could be use the extra output (called auxiliary output) as a Regularization check
- This auxiliary output can be analyzed if model is learning on its own or depending on the whole framework



Implementation

Google Colaboratory

https://colab.research.google.com/drive/1nJmgbfFWviy15laH45HTaX7Cib26EqOa?usp=sharing