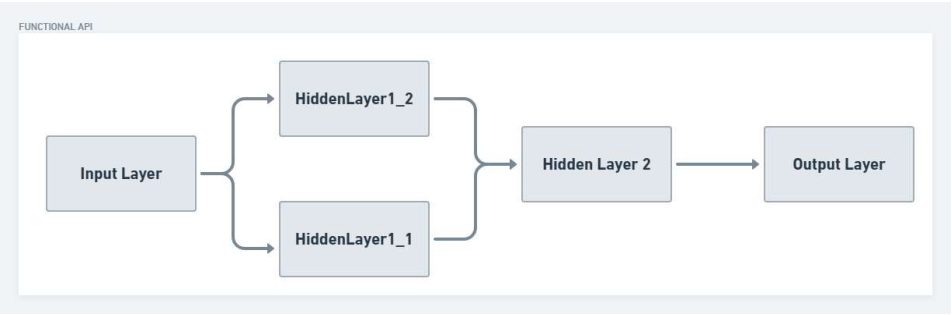
FUNCTIONAL API

- In Functional Model we are connecting layers in parallel
- In Fuctional Model, the input is being copied to two different layers. Output of both the layers is added to the output layer.



NN ARCH USING FUNCTIONAL API

```
from keras.layers import *
from keras.layers import Dense
from keras.models import Model

Inp = Input(shape = (2))
# h1_1 = Dense(unit = neuron_h1_1, activation = 'relu')(x) # x as the input
# h1_2 = Dense(unit = neuron_h1_2, activation = 'relu')(x) # x as the input

h1_1 = Dense(units = 2, activation = 'relu')(Inp) # x as the input
h1_2 = Dense(units = 2, activation = 'relu')(Inp) # x as the input

concat = concatenate([h1_1, h1_2])

# h2 = Dense(units = neuron_h2, activation = 'relu')(concat)
h2 = Dense(units = 2, activation = 'relu')(concat)
Output = Dense(units = 1, activation='sigmoid')(h2)
Functional_model = Model(Inp, Output)
Functional_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 2)]               0         
                                                                 
 dense (Dense)               (None, 2)                 6         
                                                                 
 dense_1 (Dense)             (None, 2)                 6         
                                                                 
 concatenate (Concatenate)   (None, 4)                 0         
                                                                 
 dense_2 (Dense)             (None, 2)                 10        
                                                                 
 dense_3 (Dense)             (None, 1)                 3         
                                                                 
=================================================================
Total params: 25
Trainable params: 25
Non-trainable params: 0
_________________________________________________________________
```

- **After concatination the value will be 2 in the hidden layer 2 while the value feeded will be 2 in both of the lower layers**
- **Params of h2 are 10: 6 usual and then 2 weight and bias from hidden layer1_1 and then 2 from hiddenlayer1_2**

OR GATE TRAINING

```
import numpy as np
from keras.layers import Dense
from keras.models import Model

x_train = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_train = np.array([[0], [1], [1], [1]])

Inp = Input(shape = (2))
h1_1 = Dense(units = 2, activation = 'relu')(Inp)
h1_2 = Dense(units = 2, activation = 'relu')(Inp)
c = concatenate([h1_1, h1_2])
h2 = Dense(units = 2, activation = 'relu')(c)
Output = Dense(units = 1, activation='sigmoid')(h2)
functional_model = Model(Inp, Output)

functional_model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])
functional_model.fit(x_train, y_train, epochs=3)

x_test = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_pred = functional_model.predict(x_test)

y_pred = np.round(y_pred)
print("Prediction: ", y_pred)
```

```
Epoch 1/3
1/1 [==============================] - 1s 1s/step - loss: 0.6931 - accuracy: 0.2500
Epoch 2/3
1/1 [==============================] - 0s 12ms/step - loss: 0.6925 - accuracy: 0.7500
Epoch 3/3
1/1 [==============================] - 0s 7ms/step - loss: 0.6919 - accuracy: 0.7500
1/1 [==============================] - 0s 85ms/step
Prediction:  [[1.]
 [1.]
 [1.]
 [1.]]
```

Setting a seed value to get the same output.

```
import tensorflow as tf
tf.random.set_seed(34)

import numpy as np
from keras.layers import Dense
from keras.models import Model

x_train = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_train = np.array([[0], [1], [1], [1]])

Inp = Input(shape = (2))
h1_1 = Dense(units = 2, activation = 'relu')(Inp)
h1_2 = Dense(units = 2, activation = 'relu')(Inp)
c = concatenate([h1_1, h1_2])
h2 = Dense(units = 2, activation = 'relu')(c)
Output = Dense(units = 1, activation='sigmoid')(h2)
functional_model = Model(Inp, Output)

functional_model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])
functional_model.fit(x_train, y_train, epochs=10)

x_test = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_pred = functional_model.predict(x_test)

y_pred = np.round(y_pred)
print("Prediction: ", y_pred)

pred = functional_model.predict(x_train)
pred
```

```
Epoch 1/10
1/1 [==============================] - 1s 706ms/step - loss: 0.2494 - accuracy: 1.0000
Epoch 2/10
1/1 [==============================] - 0s 8ms/step - loss: 0.2491 - accuracy: 0.7500
Epoch 3/10
1/1 [==============================] - 0s 10ms/step - loss: 0.2485 - accuracy: 1.0000
Epoch 4/10
1/1 [==============================] - 0s 8ms/step - loss: 0.2482 - accuracy: 1.0000
Epoch 5/10
1/1 [==============================] - 0s 8ms/step - loss: 0.2477 - accuracy: 1.0000
Epoch 6/10
1/1 [==============================] - 0s 8ms/step - loss: 0.2472 - accuracy: 1.0000
Epoch 7/10
1/1 [==============================] - 0s 6ms/step - loss: 0.2468 - accuracy: 1.0000
Epoch 8/10
1/1 [==============================] - 0s 7ms/step - loss: 0.2463 - accuracy: 1.0000
Epoch 9/10
1/1 [==============================] - 0s 8ms/step - loss: 0.2460 - accuracy: 1.0000
Epoch 10/10
1/1 [==============================] - 0s 6ms/step - loss: 0.2455 - accuracy: 1.0000
1/1 [==============================] - 0s 62ms/step
Prediction:  [[0.]
 [1.]
 [1.]
 [1.]]
1/1 [==============================] - 0s 16ms/step
array([[0.49900916],
       [0.95066494],
       [0.8052047 ],
       [0.9781396 ]], dtype=float32)
```

Printing all the layers

```
functional_model.layers
```

```
[<keras.engine.input_layer.InputLayer at 0x7f3c894df280>,
 <keras.layers.core.dense.Dense at 0x7f3c894df190>,
 <keras.layers.core.dense.Dense at 0x7f3c894df070>,
 <keras.layers.merging.concatenate.Concatenate at 0x7f3c894df250>,
 <keras.layers.core.dense.Dense at 0x7f3c894df3d0>,
 <keras.layers.core.dense.Dense at 0x7f3c89378430>]
```

```
hidden1 = functional_model.layers[1]
print(hidden1)
```

```
<keras.layers.core.dense.Dense object at 0x7f3c894df190>
```

Checking the weights and biases of this layer.

Weights and biases are updated in each epoch.

```
weights, biases = hidden1.get_weights()
print("Weights:\n ", weights, "\nBiases:", biases)
```

```
Weights:
  [[-0.06775985 -0.5860491 ]
 [ 1.1021895   0.46462747]]
Biases: [-4.7680642e-04  3.3294389e-05]
```

Output Layer

```
hiddenOut = functional_model.layers[-1]
weightOut, biasOut = hiddenOut.get_weights()
print("Weights: \n", weightOut, "\nBias:", biasOut)
```

```
Weights:
 [[-0.29235289]
 [ 1.4237397 ]]
Bias: [-0.00565884]
```

Printing the weights of all the layers.

```
for i in functional_model.get_weights():
  print("\n", i)
```

```


 [[-0.06775985 -0.5860491 ]
 [ 1.1021895   0.46462747]]

 [-4.7680642e-04  3.3294389e-05]

 [[-1.0973021   1.1864463 ]
 [ 0.02295705  1.0725151 ]]
```

```
[3.0703057e-05 6.9284003e-04]

[[ 0.5355368   0.854027  ]
 [ 0.2573739   0.78471994]
 [-0.46672544  0.6611636 ]
 [ 0.8246855   1.011863  ]]

[0.00130396 0.0008273 ]

[[-0.29235289]
 [ 1.4237397 ]]

[-0.00565884]
```

Alterante:

```
functional_model.get_weights()
```

Colab paid products - Cancel contracts here