Advanced Deep Learning with Keras

Import Libraries

In [96]:
```
#Data Manipulation
#============================================================
import pandas as pd
import numpy as np
from numpy import unique
#Data visualization
#============================================================
import matplotlib.pyplot as plt

#Keras
#============================================================
from keras.layers import Input , Dense,Embedding,Flatten,Add,Concatenate
from keras.utils import plot_model
from keras.models import Model
from keras.optimizers import Adam

#Data preperation
#============================================================
from sklearn.model_selection import train_test_split
```

Dataset 1: Regular season

Column Name	Explanation
Team ID 1	Unique identifier for the first team participating in the game.
Team ID 2	Unique identifier for the second team participating in the game.
Home vs Away	Indicates whether the game was played at home ('Home') or away ('Away') for Team 1.
Score Difference	The difference in score between Team 1 and Team 2 (Team 1 Score - Team 2 Score).
Team 1 Score	The total score achieved by Team 1 during the game.
Team 2 Score	The total score achieved by Team 2 during the game.
Won vs Lost	Indicates whether Team 1 won ('Won') or lost ('Lost') the game.

Dataset 2: Tournament games

Column Name	Explanation
Team ID 1	Unique identifier for the first team participating in the game.
Team ID 2	Unique identifier for the second team participating in the game.
Home vs Away	Indicates whether the game was played at home ('Home') or away ('Away') for Team 1.
Score Difference	The difference in score between Team 1 and Team 2 (Team 1 Score - Team 2 Score).
Team 1 Score	The total score achieved by Team 1 during the game.
Team 2 Score	The total score achieved by Team 2 during the game.
Won vs Lost	Indicates whether Team 1 won ('Won') or lost ('Lost') the game.
Difference in Seed	The difference in seed between Team 1 and Team 2. A positive value indicates Team 1's higher seed. A negative value indicates Team 2's higher seed.

load_Dataset

In [2]:
```
games_season=pd.read_csv('games_season.csv')
games_season.head()
```

Out[2]:

	season	team_1	team_2	home	score_diff	score_1	score_2	won
0	1985	3745	6664	0	17	81	64	1
1	1985	126	7493	1	7	77	70	1
2	1985	288	3593	1	7	63	56	1
3	1985	1846	9881	1	16	70	54	1
4	1985	2675	10298	1	12	86	74	1

In [3]:
```
games_tourney=pd.read_csv('games_tourney.csv')
games_tourney.head()
```

Out[3]:

	season	team_1	team_2	home	seed_diff	score_diff	score_1	score_2	won
0	1985	288	73	0	-3	-9	41	50	0
1	1985	5929	73	0	4	6	61	55	1
2	1985	9884	73	0	5	-4	59	63	0
3	1985	73	288	0	3	9	50	41	1
4	1985	3920	410	0	1	-9	54	63	0

Keras Refresher

Keras input and Dense Layer

In [4]:
```
input_tensor=Input(shape=(1,))
output_layar=Dense(1)
output_tensor=output_layar(input_tensor)
```

```
WARNING:tensorflow:From C:\Users\HP\anaconda3\lib\site-packages\keras\src\backend.py:1398: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions instead.
```

In [5]:
```
print(output_tensor)
```

```
KerasTensor(type_spec=TensorSpec(shape=(None, 1), dtype=tf.float32, name=None), name='dense/BiasAdd:0', description="created by layer 'dense'")
```

Model

In [6]:
```
model=Model(input_tensor,output_tensor)
model.compile(optimizer='adam',loss='mae')
```

```
WARNING:tensorflow:From C:\Users\HP\anaconda3\lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.
```

In [7]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1)]               0

 dense (Dense)               (None, 1)                 2

=================================================================
Total params: 2 (8.00 Byte)
Trainable params: 2 (8.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

pip install --upgrade pydot

In [8]:
```
# plot_model(model,to_file='model.png')
# img=plt.imread('model.png')
# plt.imshow(img)
# plt.show()
```

we Are going to use the games_tourney data

goal : predict tournament outcomes

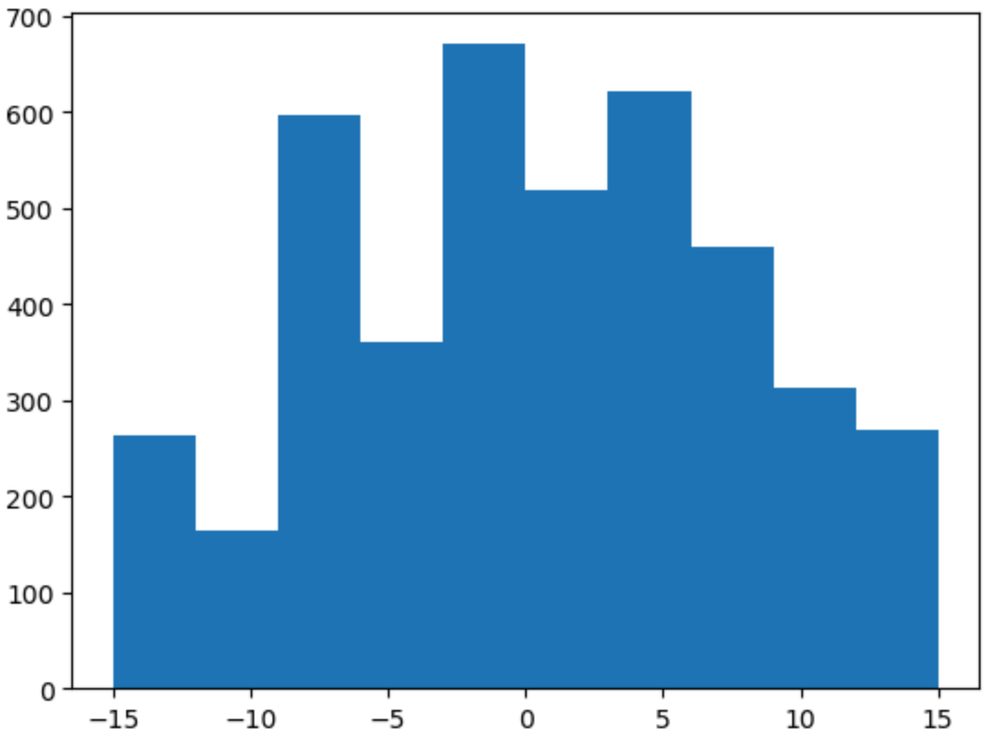
In [9]:
```
games_tourney.head()
```

Out[9]:

	season	team_1	team_2	home	seed_diff	score_diff	score_1	score_2	won
0	1985	288	73	0	-3	-9	41	50	0
1	1985	5929	73	0	4	6	61	55	1
2	1985	9884	73	0	5	-4	59	63	0
3	1985	73	288	0	3	9	50	41	1
4	1985	3920	410	0	1	-9	54	63	0

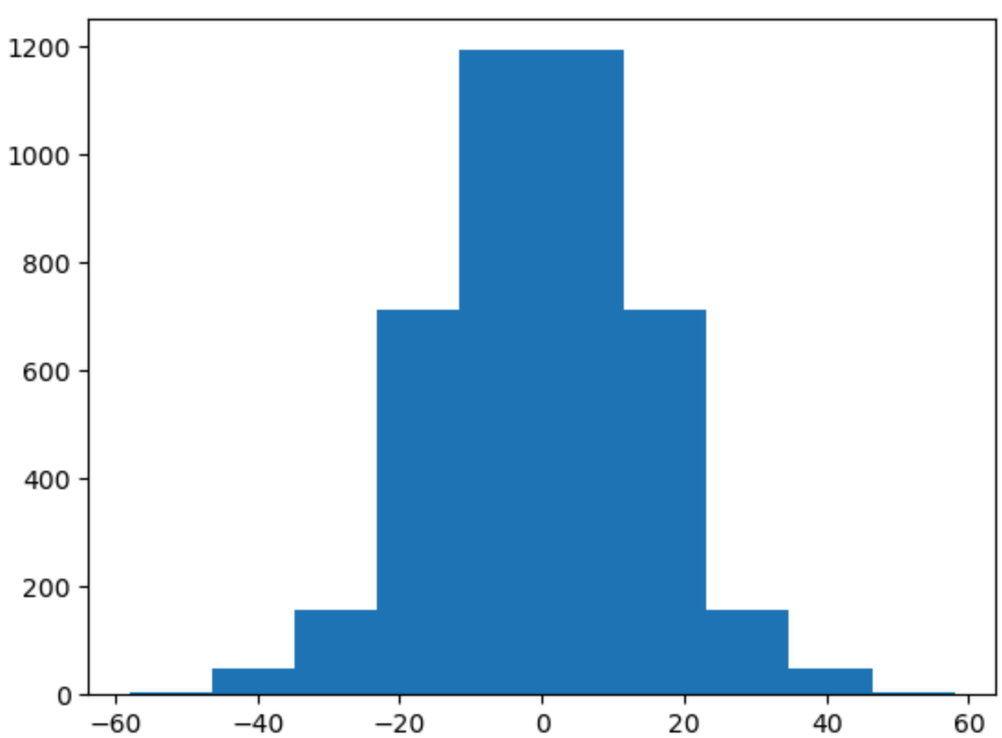
In [10]:
```
plt.hist(games_tourney['seed_diff'])
```

Out[10]:
```
(array([264., 164., 596., 360., 670., 518., 621., 460., 312., 269.]),
 array([-15., -12.,  -9.,  -6.,  -3.,   0.,   3.,   6.,   9.,  12.,  15.]),
 <BarContainer object of 10 artists>)
```



In [11]:
```
plt.hist(games_tourney['score_diff'])
```

Out[11]:
```
(array([   5.,   49.,  157.,  713., 1193., 1193.,  713.,  157.,   49.,
           5.]),
 array([-58. , -46.4, -34.8, -23.2, -11.6,   0. ,  11.6,  23.2,  34.8,
        46.4,  58. ]),
 <BarContainer object of 10 artists>)
```



In [12]:
```
X_tarin,X_test,y_train,y_test=train_test_split(games_tourney['seed_diff'],games
```

In [13]:
```
X_tarin.shape
```

Out[13]:
```
(3387,)
```

In [14]:
```
input_tensor=Input(shape=(1,))
output_tensor=Dense(1)(input_tensor)
model=Model(input_tensor,output_tensor)
model.compile(optimizer='adam',loss='mae',metrics=['accuracy'])
```

In [15]:
```
model.fit(X_tarin,y_train,batch_size=64,validation_split=0.2,verbose=True)
```

```
WARNING:tensorflow:From C:\Users\HP\anaconda3\lib\site-packages\keras\src\uti
ls\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Pleas
e use tf.compat.v1.ragged.RaggedTensorValue instead.

WARNING:tensorflow:From C:\Users\HP\anaconda3\lib\site-packages\keras\src\eng
ine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions
is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions in
stead.

43/43 [==============================] - 1s 8ms/step - loss: 11.7277 - accura
cy: 7.3828e-04 - val_loss: 10.8999 - val_accuracy: 0.0015
```

Out[15]:
```
<keras.src.callbacks.History at 0x1dde31b9ab0>
```

In [16]:
```
model.evaluate(X_test,y_test)
```

```
27/27 [==============================] - 0s 2ms/step - loss: 11.6684 - accura
cy: 0.0000e+00
```

Out[16]:
```
[11.668437957763672, 0.0]
```

Two Input Networks Using Categorical Embeddings, Shared Layers, and Merge Layers

Category Embeddings

we will use the games_season Dataset

-In the context of an embedding layer in machine learning or deep learning, "high cardinality" refers to a situation where a categorical variable has a large number of unique values.

-For example, consider a dataset containing a categorical variable like "country" which can take on values like "USA", "Canada", "UK", "France", etc. If there are many different countries represented in the dataset, the cardinality of the "country" variable is high.

-In an embedding layer, each unique value of a categorical variable is typically represented by a dense vector (an embedding) of fixed size. When dealing with high cardinality variables, this can pose challenges because the embedding layer needs to learn representations for a large number of unique values, potentially leading to overfitting or inefficient use of resources.

-Strategies for handling high cardinality in embedding layers include dimensionality reduction techniques, regularization methods, or using more advanced embedding techniques like entity embeddings. These approaches aim to capture meaningful representations of the categorical variable while mitigating the risk of overfitting or resource inefficiency.

In [17]:
```
games_season.head()
```

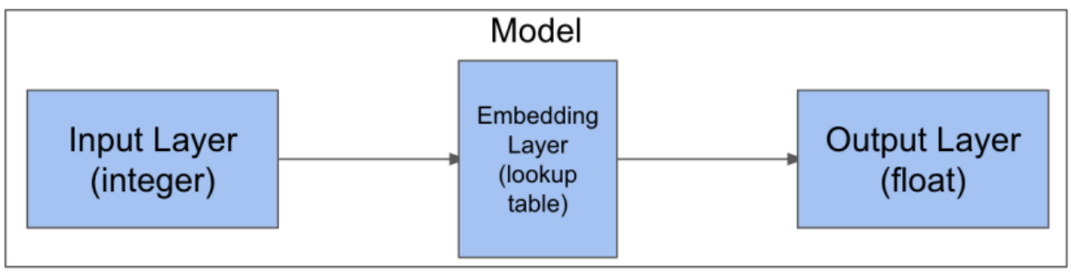
Out[17]:

	season	team_1	team_2	home	score_diff	score_1	score_2	won
0	1985	3745	6664	0	17	81	64	1
1	1985	126	7493	1	7	77	70	1
2	1985	288	3593	1	7	63	56	1
3	1985	1846	9881	1	16	70	54	1
4	1985	2675	10298	1	12	86	74	1

In [18]:
```
n_teams=unique(games_season['team_1']).shape[0]
n_teams
```

Out[18]:
```
10888
```

In [19]:
```
input_tensor=Input(shape=(1,))
embed_layer=Embedding(input_dim=n_teams,input_length=1,output_dim=1,name='Team
embed_tensor=embed_layer(input_tensor)
## Because we have A dimention increases we have to use flatten layer to conver
Flatten_layer=Flatten()(embed_tensor)
model=Model(input_tensor,embed_tensor)
```



Shared Layers

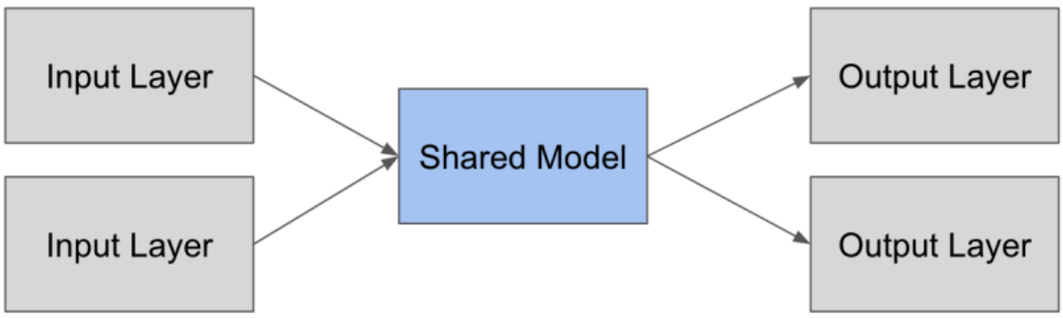
Model with 2 input one For Each Team 2 output

```
Input_tensor_1
      |
      |
shared_layer
      |
      |
output_tensor_1

Input_tensor_2
      |
      |
shared_layer
      |
      |
output_tensor_2
```

In [20]:
```
Input_tensor_1=Input((1,))
Input_tensor_2=Input((1,))
shared_layer=Dense(1)
output_tensor_1=shared_layer(Input_tensor_1)
output_tensor_2=shared_layer(Input_tensor_2)
```

Full Code

In [21]:
```
input_tensor=Input(shape=(1,))
embed_layer=Embedding(input_dim=n_teams,input_length=1,output_dim=1,name='Team
embed_tensor=embed_layer(input_tensor)
## Because we have A dimention increases we have to use flatten layer to conver
Flatten_layer=Flatten()(embed_tensor)
model=Model(input_tensor,embed_tensor)



Input_tensor_1=Input((1,))
Input_tensor_2=Input((1,))

output_tensor_1=model(Input_tensor_1)
output_tensor_2=model(Input_tensor_2)
```



Merge Layers

Add , Subtract ,Multiply (same shape) , Concatenate(layer with diffrence column)

2 input 1 output

In [22]:
```
Input_tensor_1=Input((1,))
Input_tensor_2=Input((1,))

output_tensor=Add()([Input_tensor_1,Input_tensor_2])

#in_tensor_3=Input((1,))
#output_tensor=Add()([Input_tensor_1,Input_tensor_2,in_tensor_3])
```

In [23]:
```
model=Model([Input_tensor_1,Input_tensor_2],output_tensor)
```

In [24]:
```
model.compile(optimizer='adam',loss='mean_absolute_error')
```

Fitting and predicting with multiple inputs

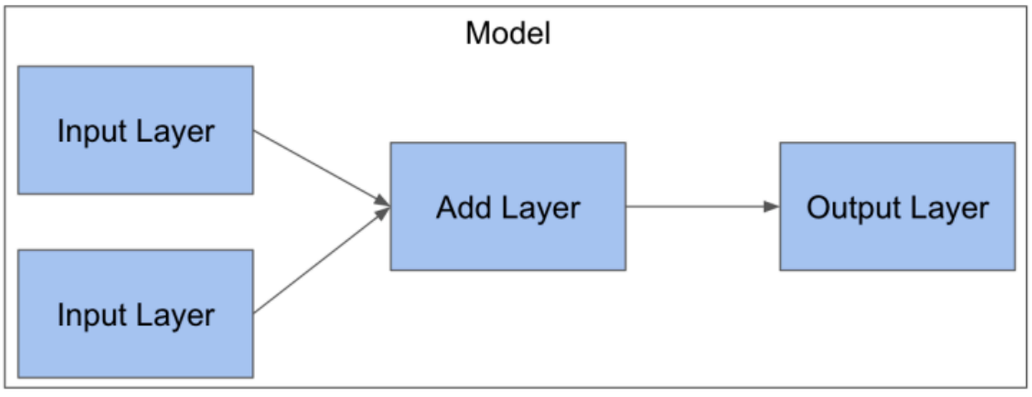
In [25]:
```
input_1 = games_season['team_1']
input_2 = games_season['team_2']


model.fit([input_1,input_2],games_season['score_diff'],epochs=1,batch_size=204
```

```
138/138 [==============================] - 1s 3ms/step - loss: 10900.2002 - v
al_loss: 10913.7383
```

Out[25]:
```
<keras.src.callbacks.History at 0x1dde33eb670>
```

In [26]:
```
input_1 = games_tourney['team_1']
input_2 = games_tourney['team_2']

print(model.evaluate([input_1, input_2], games_tourney['score_diff'], verbose=
```

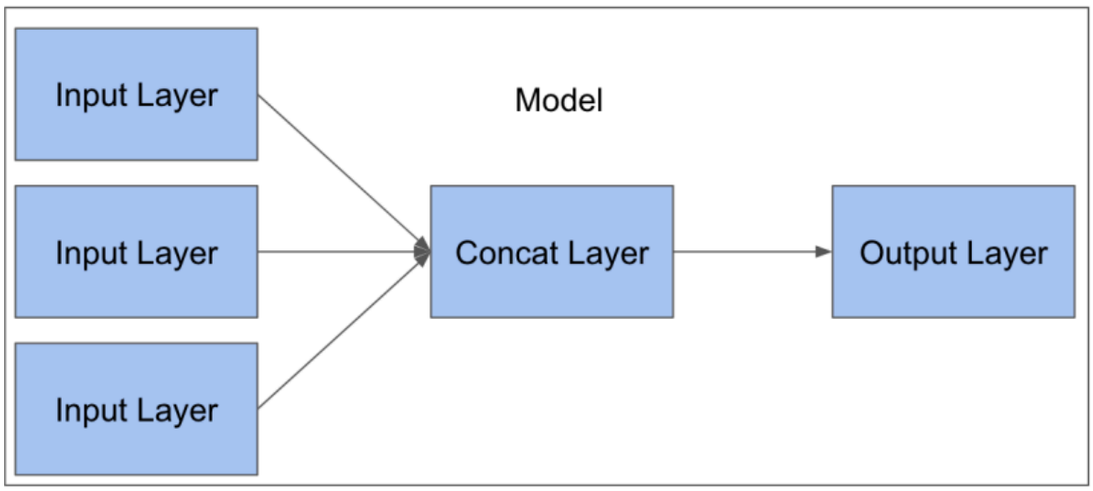
```
11178.2919921875
```



Multiple Inputs: 3 Inputs (and Beyond!)

In [27]:
```
in_tensor_1=Input(shape=(1,))
in_tensor_2=Input(shape=(1,))
in_tensor_3=Input(shape=(1,))
output_tensor=Concatenate()([in_tensor_1,in_tensor_2,in_tensor_3])
output_tensor=Dense(1)(output_tensor)
```

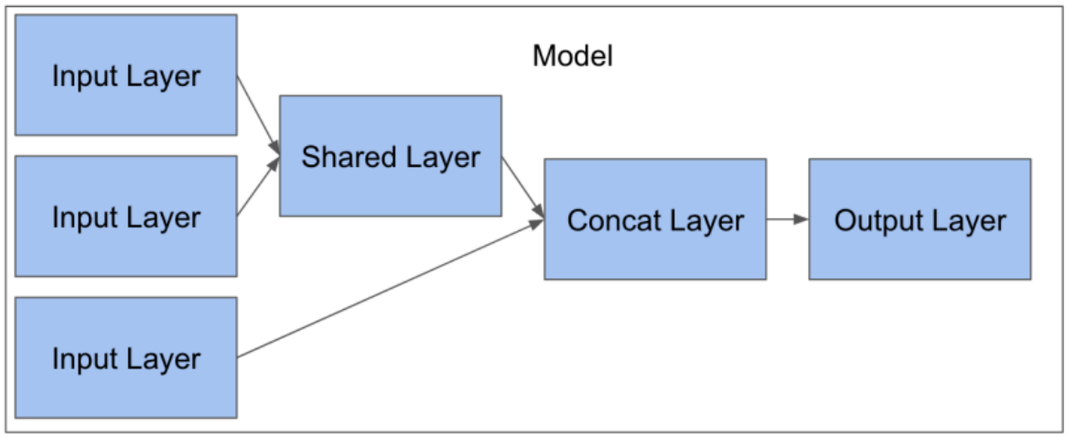
In [28]:
```
model=Model([in_tensor_1,in_tensor_2,in_tensor_3],output_tensor)
```



Shared Layer with 3 inputs

In [29]:
```
shared_layer=Dense(1)
shared_tensor_1=shared_layer(in_tensor_1)
shared_tensor_2=shared_layer(in_tensor_2)

out_tensor=Concatenate()([shared_tensor_1,shared_tensor_2,in_tensor_3])
out_tensor=Dense(1)(out_tensor)
```

In [30]:
```
model=Model([in_tensor_1,in_tensor_2,in_tensor_3],out_tensor)
```



In [31]:
```
model.compile(loss='mae', optimizer='adam')
```

Team strength model

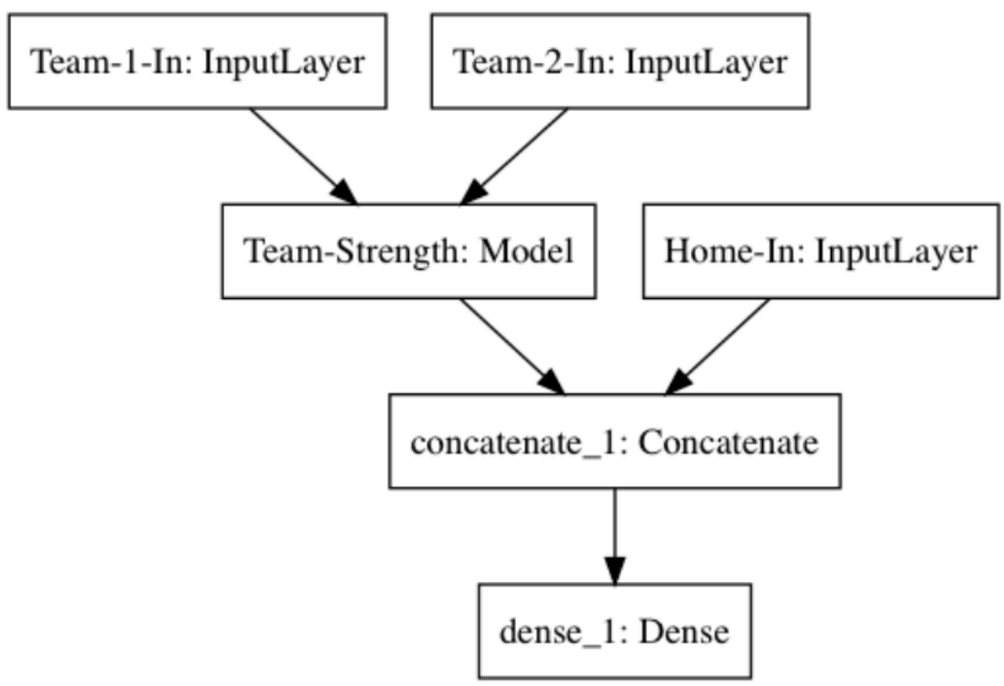
In [32]:
```
def team_strength_model(team_input):
    # Example implementation
    team_strength = Dense(64, activation='relu')(team_input)
    return team_strength

# Assuming the rest of your code follows
team_in_1 = Input(shape=(1,), name='Team-1-In')
team_in_2 = Input(shape=(1,), name='Team-2-In')

home_in = Input(shape=(1,), name='Home-In')

team_1_strength = team_strength_model(team_in_1)
team_2_strength = team_strength_model(team_in_2)

out = Concatenate()([team_1_strength, team_2_strength, home_in])
out = Dense(1)(out)
```

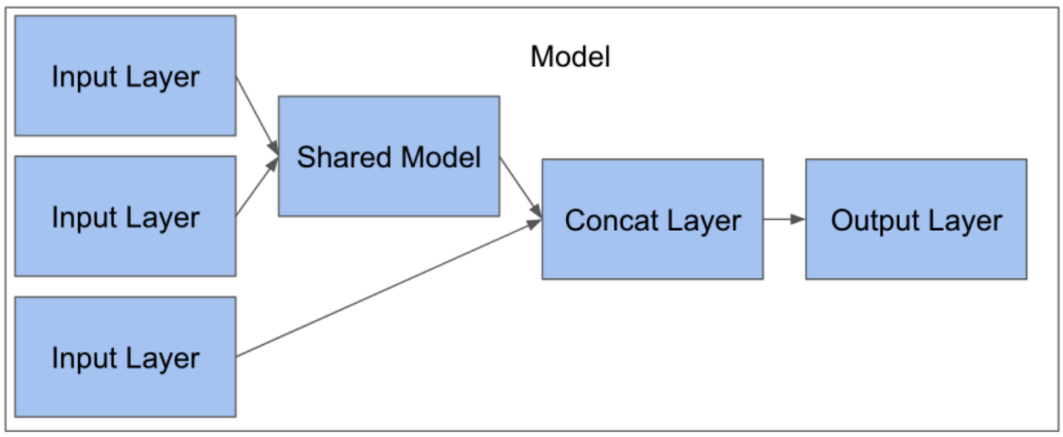
In [33]:
```
# Make a Model
model = Model([team_in_1, team_in_2, home_in], out)

# Compile the model
model.compile(optimizer='adam', loss='mean_absolute_error')
```

In [34]:
```
# Fit the model to the games_season dataset
model.fit([games_season['team_1'], games_season['team_2'], games_season['home']
          games_season['score_diff'],
          epochs=1,
          verbose=True,
          validation_split=0.10,
          batch_size=2048)

# Evaluate the model on the games_tourney dataset
print(model.evaluate([games_tourney['team_1'], games_tourney['team_2'], games_t
```

```
138/138 [==============================] - 1s 5ms/step - loss: 52.1369 - val_
loss: 12.1709
11.703423500061035
```

Summarizing and plotting models





In [35]:
```
model.summary()
```

```
Model: "model_7"
__________________________________________________________________________________
_____________________
 Layer (type)                Output Shape              Param #   Connected
to
==================================================================================
=====================
 Team-1-In (InputLayer)      [(None, 1)]               0         []

 Team-2-In (InputLayer)      [(None, 1)]               0         []

 dense_6 (Dense)             (None, 64)                128       ['Team-1-
In[0][0]']

 dense_7 (Dense)             (None, 64)                128       ['Team-2-
In[0][0]']

 Home-In (InputLayer)        [(None, 1)]               0         []

 concatenate_2 (Concatenate  (None, 129)               0         ['dense_6
[0][0]',
 )                                                               'dense_7
[0][0]',
                                                                 'Home-In
[0][0]']

 dense_8 (Dense)             (None, 1)                 130       ['concate
nate_2[0][0]']

==================================================================================
=====================
Total params: 386 (1.51 KB)
Trainable params: 386 (1.51 KB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________
_____________________
```

Stacking Model : Using prediction of one model as input to another model

One of the best to win Kaggel Competition

Advice on Stacking in Deep Learning

Stacking, also known as stacked generalization, is a powerful technique in machine learning and deep learning where the predictions of multiple models are combined to improve overall performance. Here are some key points to consider when using stacking in deep learning:

1. Diverse Model Selection:

- Choose diverse base models that capture different aspects of the data. This diversity helps to reduce correlation among the models and can lead to better performance when combined.

2. Heterogeneous Architectures:

- Experiment with using different architectures for your base models. This could include combinations of convolutional neural networks (CNNs), recurrent neural networks (RNNs), and other types of neural networks depending on your data and problem domain.

3. Ensemble Methods:

- Employ ensemble methods such as bagging and boosting in conjunction with stacking. These methods can further enhance the diversity of the base models and improve the overall robustness of the stacked model.

4. Cross-Validation:

- Utilize cross-validation during the stacking process to ensure robustness and avoid overfitting. Cross-validation helps to estimate the performance of the stacked model on unseen data and allows for better model selection and hyperparameter tuning.

5. Model Blending:

- Consider blending predictions from different layers of the stacked model. Instead of directly using the final predictions, you can blend predictions from intermediate layers or combine them using weighted averages to further improve performance.

6. Interpretability:

- Maintain interpretability by understanding the contributions of individual base models to the final prediction. Techniques such as model inspection, feature importance analysis, and model explainability methods can help in interpreting the stacked model's predictions.

7. Computational Resources:

- Be mindful of computational resources when stacking deep learning models, especially if you're dealing with large datasets or complex architectures. Consider techniques like model parallelism, distributed training, or model distillation to efficiently utilize resources.

8. Regularization:

- Apply regularization techniques such as dropout, L1/L2 regularization, and early stopping to prevent overfitting in individual base models and the stacked model.

In [36]:
```
games_season.head()
```

Out[36]:

	season	team_1	team_2	home	score_diff	score_1	score_2	won
0	1985	3745	6664	0	17	81	64	1
1	1985	126	7493	1	7	77	70	1
2	1985	288	3593	1	7	63	56	1
3	1985	1846	9881	1	16	70	54	1
4	1985	2675	10298	1	12	86	74	1

In [37]:
```
games_tourney.head()
```

Out[37]:

	season	team_1	team_2	home	seed_diff	score_diff	score_1	score_2	won
0	1985	288	73	0	-3	-9	41	50	0
1	1985	5929	73	0	4	6	61	55	1
2	1985	9884	73	0	5	-4	59	63	0
3	1985	73	288	0	3	9	50	41	1
4	1985	3920	410	0	1	-9	54	63	0

In [42]:
```
games_season['team_1'].shape
```

Out[42]:
```
(312178,)
```

In [43]:
```
in_data_1=games_tourney['team_1']
in_data_2=games_tourney['team_2']
in_data_3=games_tourney['home']

games_tourney['pred'] = model.predict([games_tourney['team_1'],games_tourney['
```

```
133/133 [==============================] - 0s 2ms/step
```

In [44]:
```
# Create an input layer with 3 columns
input_tensor = Input(shape=(3,))

# Pass it to a Dense layer with 1 unit
output_tensor = Dense(1)(input_tensor)

# Create a model
model = Model(input_tensor,output_tensor)

# Compile the model
model.compile(optimizer='adam', loss='mean_absolute_error')
```

In [47]:
```
X = games_tourney[['home', 'seed_diff', 'pred']]
y = games_tourney['score_diff']
```

In [48]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
```

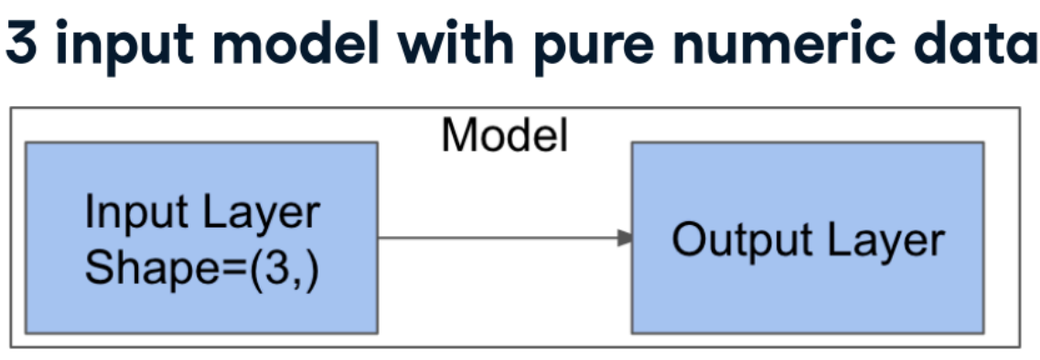
In [49]:
```
# Fit the model
model.fit(X_train,
          y_train,
          epochs=1,
          verbose=True)
```

```
106/106 [==============================] - 1s 2ms/step - loss: 10.4470
```

Out[49]:
```
<keras.src.callbacks.History at 0x1ddec3e4b80>
```

In [50]:
```
# Evaluate the model on the games_tourney_test dataset
print(model.evaluate(X_test,y_test, verbose=False))
```

```
10.576225280761719
```



Multiple Outputs

Two-output models

In [77]:
```
games_tourney.head()
```

Out[77]:

	season	team_1	team_2	home	seed_diff	score_diff	score_1	score_2	won	pred
0	1985	288	73	0	-3	-9	41	50	0	0.024063
1	1985	5929	73	0	4	6	61	55	1	0.614121
2	1985	9884	73	0	5	-4	59	63	0	1.027878
3	1985	73	288	0	3	9	50	41	1	-0.011671
4	1985	3920	410	0	1	-9	54	63	0	0.383255

In [78]:
```
X=games_tourney[['seed_diff','pred']]
y=games_tourney[['score_1','score_2']]
```

In [79]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
```

In [80]:
```
X_train.shape
```

Out[80]:
```
(3387, 2)
```

In [81]:
```
y_train.shape
```

Out[81]:
```
(3387, 2)
```

In [82]:
```
Input_tensor=Input(shape=(2,))
output_tensor=Dense(2)(Input_tensor)## This is the Multiple output Dense = 2
```

In [83]:
```
model=Model(Input_tensor,output_tensor)
model.compile(optimizer='adam',loss='mean_absolute_error')
```

In [84]:
```
model.fit(X_train,y_train,verbose=True,epochs=100,batch_size=100)
```

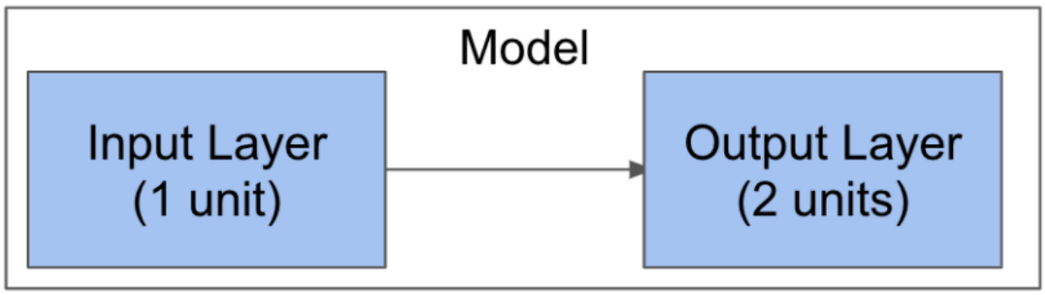
```
Epoch 1/100
34/34 [==============================] - 0s 2ms/step - loss: 71.1433
Epoch 2/100
34/34 [==============================] - 0s 2ms/step - loss: 71.1015
Epoch 3/100
34/34 [==============================] - 0s 2ms/step - loss: 71.0591
Epoch 4/100
34/34 [==============================] - 0s 2ms/step - loss: 71.0172
Epoch 5/100
34/34 [==============================] - 0s 2ms/step - loss: 70.9756
Epoch 6/100
34/34 [==============================] - 0s 2ms/step - loss: 70.9329
Epoch 7/100
34/34 [==============================] - 0s 2ms/step - loss: 70.8909
Epoch 8/100
34/34 [==============================] - 0s 2ms/step - loss: 70.8488
Epoch 9/100
34/34 [==============================] - 0s 2ms/step - loss: 70.8066
Epoch 10/100
```

In [86]:
```
# Print the model's weights
print(model.get_weights())

# Print the column means of the training data
print(X_tarin.mean())
```

```
[array([[-0.45656827,  0.46888477],
       [ 2.2964952 ,  2.6562371 ]], dtype=float32), array([3.399933, 3.39993
3], dtype=float32)]
0.04428697962798937
```

In [87]:
```
print(model.evaluate(X_test,y_test , verbose=False))
```

```
67.8629379272461
```



Single model for Classification and Regression

In [88]:
```
games_tourney.head()
```

Out[88]:

	season	team_1	team_2	home	seed_diff	score_diff	score_1	score_2	won	pred
0	1985	288	73	0	-3	-9	41	50	0	0.024063
1	1985	5929	73	0	4	6	61	55	1	0.614121
2	1985	9884	73	0	5	-4	59	63	0	1.027878
3	1985	73	288	0	3	9	50	41	1	-0.011671
4	1985	3920	410	0	1	-9	54	63	0	0.383255

In [90]:
```
X=games_tourney[['seed_diff','pred']]
y_regression=games_tourney['score_diff']
y_class=games_tourney['won']
```

In [92]:
```
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X, y_regres
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(X, y_class
```

In [93]:
```
X_train_reg.shape
```

Out[93]:
```
(3387, 2)
```

In [94]:
```
X_train_cls.shape
```

Out[94]:
```
(3387, 2)
```

In [95]:
```
Input_layer=Input(shape=(2,))
output_reg=Dense(1, activation='linear', use_bias=False)(Input_layer)
output_class=Dense(1, activation='sigmoid', use_bias=False)(output_tensor_1)
```

In [ ]:
```
model = Model(input_tensor, [output_tensor_1, output_tensor_2])
```

In [ ]:
```
model.compile(loss=['mean_absolute_error', 'binary_crossentropy'], optimizer=A
```

In [97]:
```
model.fit(X_train_cls ,# you can put X_train_reg its the  same X
          [y_train_cls, y_train_reg],
          epochs=10,
          verbose=True,
          batch_size=16384)
```

```
Epoch 1/10
1/1 [==============================] - 0s 238ms/step - loss: 4.1825
Epoch 2/10
1/1 [==============================] - 0s 8ms/step - loss: 4.1809
Epoch 3/10
1/1 [==============================] - 0s 6ms/step - loss: 4.1773
Epoch 4/10
1/1 [==============================] - 0s 6ms/step - loss: 4.1719
Epoch 5/10
1/1 [==============================] - 0s 6ms/step - loss: 4.1650
Epoch 6/10
1/1 [==============================] - 0s 6ms/step - loss: 4.1569
Epoch 7/10
1/1 [==============================] - 0s 6ms/step - loss: 4.1476
Epoch 8/10
1/1 [==============================] - 0s 7ms/step - loss: 4.1373
Epoch 9/10
1/1 [==============================] - 0s 5ms/step - loss: 4.1263
Epoch 10/10
1/1 [==============================] - 0s 5ms/step - loss: 4.1145
```

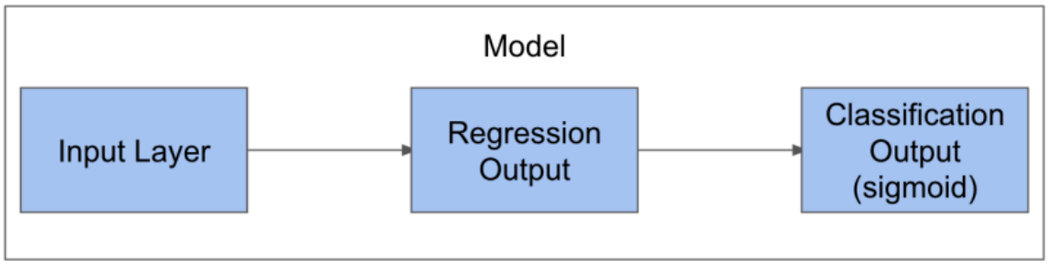
Out[97]:
```
<keras.src.callbacks.History at 0x1ddf3b6acb0>
```

In [98]:
```
# Print the model weights
print(model.get_weights())

# Print the training data means
print(X_train_reg.mean())
```

```
[array([[-0.43322337,  0.45126298],
       [ 2.2986887 ,  2.6582477 ]], dtype=float32), array([3.4034598, 3.40308
], dtype=float32)]
seed_diff    0.044287
pred         0.236663
dtype: float64
```

In [100]:
```
print(model.evaluate(X_test_cls,[y_test_reg, y_test_cls], verbose=False))
```

```
12.33867073059082
```



In [ ]:
```
```