Predicting Player Performance Using Feed-Forward Neural Networks

Objective:

Welcome to this tutorial where we'll explore the power of Feed-Forward Neural Networks (FNN) and apply them to predict whether a basketball player will win an award based on historical data. This process is a fascinating example of how machine learning can be used to gain insights into sports performance.

In this tutorial, we'll go step by step through everything you need to know to use FNNs for predicting player awards. Don't worry if you're new to this—by the end of this tutorial, you'll have a solid understanding of how to build and evaluate a neural network model using Google Colab, and you'll be able to apply this technique to your own sports analytics projects!

What is a Feed-Forward Neural Network?

Before we dive into the code, let's briefly explain what a Feed-Forward Neural Network is. Imagine a network where information flows in one direction—from the input, through one or more hidden layers, and finally to the output. This type of neural network is simple but powerful for various predictive tasks, including:

Classification (e.g., predicting whether a player will win an award)

Regression (e.g., predicting a player's score based on their stats)

In our case, we're going to use a binary classification model, where the goal is to predict whether a player will win an award or not.

Dataset Overview:

The data we'll use is from the Player Awards dataset in the Men's Professional Basketball collection on Kaggle. This dataset provides:

playerID: A unique ID for each player.

award: The award won by the player.

year: The year the award was won.

lgID: The league the player was a part of.

note: Notes related to the award.

pos: The position of the player.

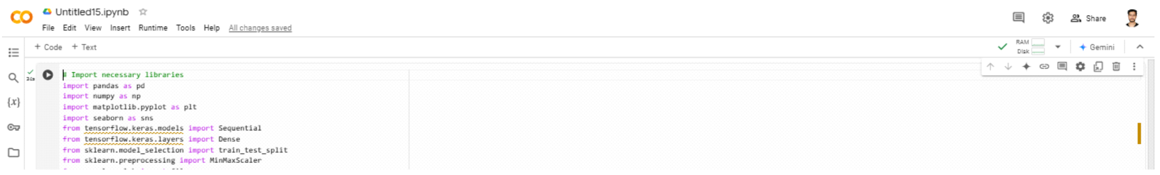
The dataset contains historical award data for basketball players, and we will focus on predicting whether a player wins an award based on their past performance.

code:

Import Necessary Libraries:

let's import the libraries we'll use. These will help us load and manipulate the data, as well as build and train the neural network.

Importing necessary libraries



```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

Here's a quick breakdown of the libraries:

pandas: Helps us work with data (like loading CSV files and data cleaning).

numpy: Used for numerical operations.

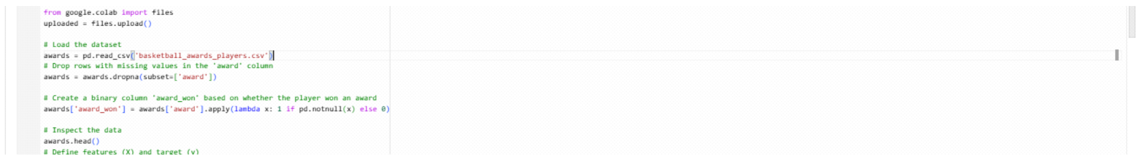
matplotlib/seaborn: Used for visualizing data.

tensorflow.keras: This is where we build and train the neural network.

sklearn: Used for preprocessing and splitting the data

Load and Inspect the Dataset

Now, let's upload the Player Awards dataset to Google Colab. You can do this by clicking the file icon on the left-hand side of the Colab window, then selecting "Upload". Once the file is uploaded, we can read it into a pandas DataFrame.



```
from google.colab import files
uploaded = files.upload()

# Load the dataset
awards = pd.read_csv('basketball_awards_players.csv')
# Drop rows with missing values in the 'award' column
awards = awards.dropna(subset=['award'])

# Create a binary column 'award_won' based on whether the player won an award
awards['award_won'] = awards['award'].apply(lambda x: 1 if pd.notnull(x) else 0)

# Inspect the data
awards.head()
# Define features (X) and target (y)
```

The first few rows will give us a preview of what the data looks like. We should see columns like playerID, award, year, etc.]

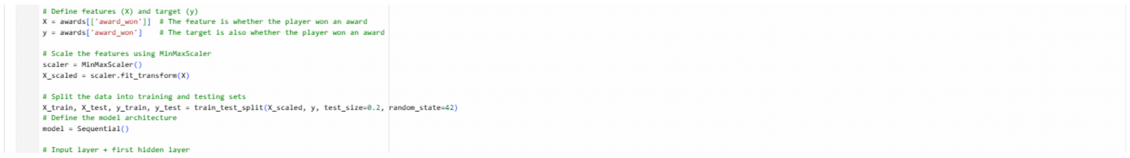
Data Preprocessing

Next, we need to clean up and preprocess the data so that it’s ready for our model. In this case, we’ll:Drop any rows where the award column is missing (as we’re interested in whether a player won an award).Create a binary target variable (award_won) that indicates if a player won an award or not.

Now we have a clean dataset with the award_won column as our target variable.

Prepare Features and Labels

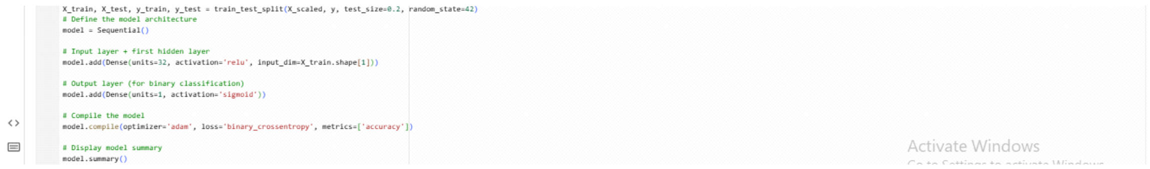
Now, we'll separate the features (input data) from the labels (output data), and scale the features to improve model performance.



```python
# Define features (X) and target (y)
X = awards[['award_won']]  # The feature is whether the player won an award
y = awards['award_won']    # The target is also whether the player won an award

# Scale the features using MinMaxScaler
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Define the model architecture
model = Sequential()

# Input layer + first hidden layer
```

Build the Feed-Forward Neural Network

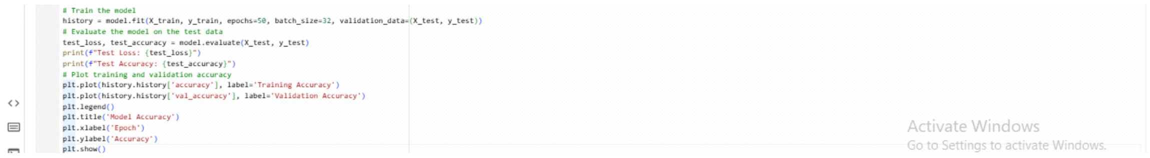
Here’s where we build the FNN using Keras. We'll add one hidden layer with 32 neurons and an output layer with a sigmoid activation function (for binary classification).



```python
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
# Define the model architecture
model = Sequential()

# Input layer + first hidden layer
model.add(Dense(units=32, activation='relu', input_dim=X_train.shape[1]))

# Output layer (for binary classification)
model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Display model summary
model.summary()
```

Train and Evaluate the Model

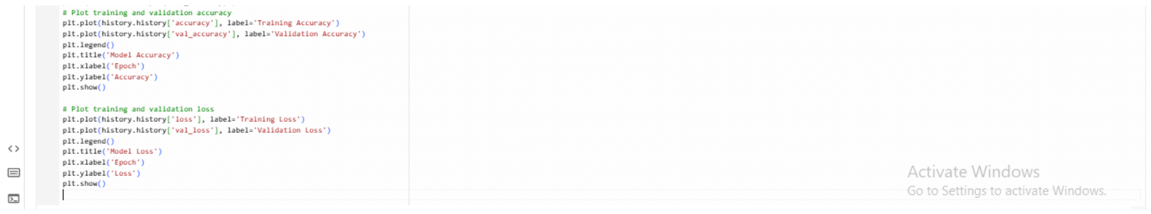
Train the neural network on the dataset for 50 epochs to learn the relationship between the features and the target.After training, we’ll evaluate the model on the test data to see how well it performs.



```python
# Train the model
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {test_loss}")
print(f"Test Accuracy: {test_accuracy}")
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show()
```

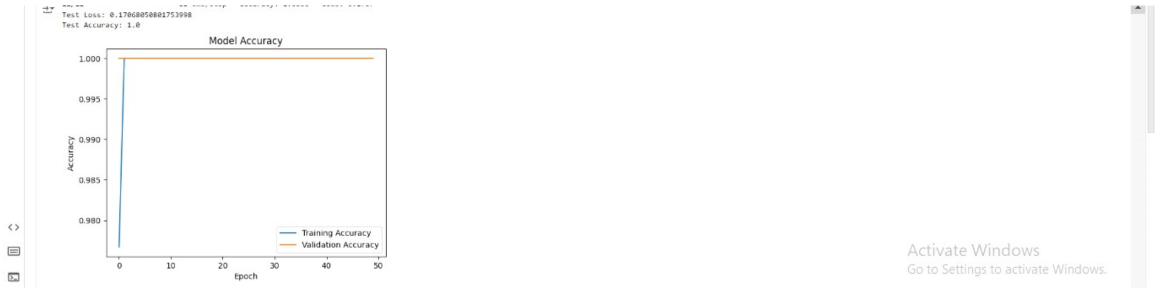
Visualize the Results

Let’s visualize the model’s performance by plotting the accuracy and loss over the training epochs.



```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.show()

# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.legend()
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.show()
```

Training and Validation Accuracy Graph



```
Test Loss: 0.17068050801753998
Test Accuracy: 1.0
```

This graph shows how well the model is learning from the data. It plots the accuracy (how many correct predictions the model makes) for both the training set and the validation set over time as the model trains.

Training Accuracy (Blue Line): This represents how well the model is performing on the training data. As the training progresses, this line usually increases because the model is exposed to the same data multiple times and gets better at predicting it.

Validation Accuracy (Orange Line): This shows how well the model performs on data it hasn't seen before (validation data). Ideally, this line should also increase and stay close to the training accuracy line, indicating that the model is not just memorizing the data but generalizing well to new data.

Explanation of Graph:

If both lines are rising: This indicates the model is learning correctly from both the training data and new data. The model is improving over time.

If training accuracy goes up but validation accuracy plateaus or drops: This suggests that the model is overfitting — it's memorizing the training data too well and not generalizing well to new data. You may need to adjust the model to prevent overfitting.

Training and Validation Loss Graph

Loss is a measure of how "off" the model's predictions are. A lower loss means the model is making fewer errors in its predictions. This graph tracks the loss for both the training data and the validation data over time.

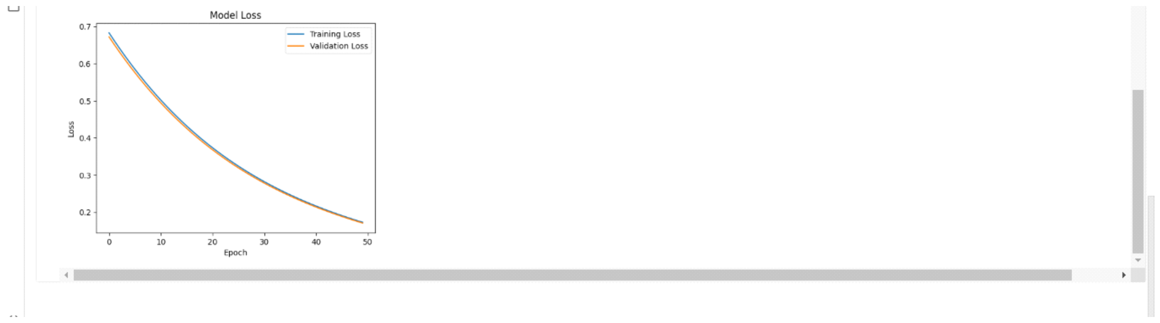
Training Loss (Blue Line): Shows how much error the model is making on the training data. A lower value is better because it means the model is learning and getting better at predicting the correct outcomes.

Validation Loss (Orange Line): Shows how much error the model is making on the unseen validation data. You want this line to go down as well, because it suggests the model is generalizing well to new data.

Explanation of Graph:

If both lines are decreasing: This is a good sign because it means the model is learning effectively and reducing its errors on both the training and validation data.

If the training loss is decreasing but the validation loss starts to increase: This is a classic sign of overfitting. It means that while the model is becoming better at predicting the training data (reducing training loss), it's not doing well on the validation data (increasing validation loss). This could be due to the model learning noise or irrelevant details in the training data that don't generalize well to new data



Summary of What to Look for in the Graphs:

Good Model Behavior:

Both accuracy should increase over time (for training and validation).

Both loss should decrease over time (for training and validation).

Signs of Overfitting:

Training accuracy increases, but validation accuracy decreases or plateaus.

Training loss decreases, but validation loss increases.

Conclusion

You've just built a Feed-Forward Neural Network to predict basketball player awards. Here's a recap of what we did:

Data Preprocessing: We cleaned the dataset and created a binary target variable (award_won).

Model Building: We created an FNN using Keras with one hidden layer.

Model Training: We trained the model for 50 epochs to learn the patterns in the data.

Evaluation: We tested the model on a separate dataset to see how well it performed.

Visualization: We plotted training and validation accuracy and loss to understand the model's learning process.

References:

Kaggle Dataset: https://www.kaggle.com/datasets/open-source-sports/mens-professional-basketball?resource=download&select=basketball_awards_players.csv

TensorFlow Keras Documentation: Keras Sequential API

"Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.