ANN Model

By Josh Houlding

ANN models strive to learn relationships through different sets of data. With this information, ANN models can be used to predict outcomes.

In this assignment, students will build an ANN, then write a comprehensive technical report as a Python Jupyter notebook (to include all code, code comments, all outputs, plots, and analysis). Make sure the project documentation contains a) problem statement, b) algorithm of the solution, c) analysis of the findings, and d) references.

Part 1

Download the "NFL Players Dataset," located in the topic Resources. The dataset contains football players' characteristics.

1. Select from a pool of 200 "Active Players" and 200 "Retired Players."
2. Define "optimal team" based on your decision of the player characteristics necessary to build a team:
    - Starting offense (11 players): passing, rushing, and receiving.
    - Starting special teams (11 players): punting, punt returns, and kick returns.
    - Starting defense (11 players): tackles, sacks, safeties, and fumbles.
3. Your task is to identify the optimal team from each pool comprised of 33 "Active Player" and "Retired Player" members.
4. Examine the multilayer neural network MLP architecture depicted in the "DSC-550 An Artificial Neural Network Model Image."
5. Build a deep artificial neural network MLP to include the following: a) 1 input layer, b) as many hidden layers as you deem necessary, and c) an output layer fully connected to the hidden layers.
6. Explain your architecture and how the NFL player characteristics are used as inputs.

Part 2

Activate the MLP by performing the following steps:

1. Starting at the input layer, forward propagate the patterns of the training data through the network to generate an output.
2. Based on the network's output, calculate the error that we want to minimize using a cost function that we will describe later.
3. Back propagate the error, find its derivative with respect to each weight in the network, and update the model.
4. Repeat steps 1 through 3 for multiple epochs and learn the weights of the MLP.
5. Use forward propagation to calculate the network output and apply a threshold function to obtain the predicted class labels in the one-hot representation.
6. Interpret the output of your MLP in the context of selecting an optimal football team.

Part 1

Download the "NFL Players Dataset," located in the topic Resources. The dataset contains football players' characteristics.

In [96]:
```
import pandas as pd

# Load all relevant files
combine_df = pd.read_csv("nfl/combine.csv")
passer_df = pd.read_csv("nfl/passer.csv")
receiver_df = pd.read_csv("nfl/receiver.csv")
rusher_df = pd.read_csv("nfl/rusher.csv")
tackles_df = pd.read_csv("nfl/tackles.csv")
sacks_df = pd.read_csv("nfl/sacks.csv")
interceptions_df = pd.read_csv("nfl/interceptions.csv")
fumbles_df = pd.read_csv("nfl/fumbles.csv")
game_participation_df = pd.read_csv("nfl/gameParticipation.csv")
```

`combine.csv` has all the data from `players.csv` plus more, so we will use that file as a starting point and merge additional data. The following data files will need to be merged (and possibly aggregated beforehand): `passer.csv`, `receiver.csv`, `rusher.csv`, `tackles.csv`, `sacks.csv`, `interceptions.csv`, `fumbles.csv`, and `gameParticipation.csv`. All of these provide valuable information about the performance metrics for each player that will help our model determine which player is best for each role.

Aggregating performance-related dataframes

This includes `passer_df`, `receiver_df`, `rusher_df`, `tackles_df`, `sacks_df`, `interceptions_df`, and `fumbles_df`. The data in these dataframes needs to be aggregated because each contains more than one entry per player, and we need exactly one entry per player so they can all be merged together into the final dataframe used for modeling.

In [97]:

```
# Aggregate passer data
passer_agg = passer_df.groupby("playerId").agg({
    "passAtt": "sum",
    "passComp": "sum",
    "passLength": "sum",
    "passTd": "sum",
    "passInt": "sum",
    "passSack": "sum",
    "passSackYds": "sum",
    "passHit": "sum",
    "passDef": "sum"
}).reset_index()

# Aggregate receiver data
receiver_agg = receiver_df.groupby("playerId").agg({
    "recYards": "sum",
    'recYards': "sum",
    'rec': "sum",
    'recYac': "sum",
    'rec1down': "sum",
    'recFumble': "sum",
    'recPassDef': "sum",
    'recPassInt': "sum"
}).reset_index()

# Aggregate rusher data
rusher_agg = rusher_df.groupby("playerId").agg({
    "rushYards": "sum",
    "rushPrimary": "sum",
    "rushTd": "sum"
}).reset_index()

# Aggregate tackle data
tackles_agg = tackles_df.groupby("playerId").agg({
    "tackleYdsScrim": "sum"
}).reset_index()

# Aggregate sack data
sacks_agg = sacks_df.groupby("playerId").agg({
    "sackYards": "sum"
}).reset_index()

# Aggregate interception data
interceptions_agg = interceptions_df.groupby("playerId").agg({
```

```
    "int": "sum",
    "intYards": "sum",
    "intTd": "sum"
}).reset_index()

# Aggregate fumble data
fumbles_agg = fumbles_df.groupby("playerId").agg({
    "fumOOB": "sum",
    "fumTurnover": "sum"
})
```

Aggregating `game_participation_df`

This dataframe contains several useful pieces of information not contained in the other dataframes, including `gamePartUnit` (whether the player participated in offense, defense or special teams) and `gamePartSnapCount` (the number of snap the player participated in during that game). Thus, it is in our best interest to aggregate it so it can be merged with `combine_df` into an overall dataframe.

In [98]:
```
# Aggregate game participation data
game_participation_agg = game_participation_df.groupby("playerId").agg({
    "gamePartSnapCount": "sum",
    "gameId": "nunique",
    "gamePartUnit": lambda x: ",".join(x.unique())
}).reset_index()
```

Merging all relevant dataframes into one overall dataframe for modeling

In [99]:
```
# Create copy of combine_df for overall dataframe
df = combine_df.copy()

# Select dataframes that need to be merged to combine_df
dfs_to_merge = [passer_agg, receiver_agg, rusher_agg, tackles_agg, sacks_agg, interceptions_agg,
                fumbles_agg, game_participation_agg]

# Merge dataframes
for dataframe in dfs_to_merge:
    df = pd.merge(df, dataframe, on="playerId", how="left")
```

In [100…
```
# Show final merged dataframe
print(f"Dataframe shape: {df.shape}")
```

```
print()
print(f"Dataframe sample: \n {df.sample(5, random_state=42)}")
```

```
Dataframe shape: (10080, 64)

Dataframe sample:
       combineId  playerId  combineYear combinePosition  combineHeight  \
9823       20243  20190163         2019               P           73.0
33         10033  19870053         1987              CB           70.6
1369       11369  19910851         1991             OLB           72.8
3620       13620  19970183         1997              WR           70.4
3272       13272  19960855         1996              QB           73.8

      combineWeight  combineHand nameFirst nameLast         nameFull  ...  \
9823            200          NaN      Jake   Bailey      Jake Bailey  ...
33              163         9.50       Lou    Brock        Lou Brock  ...
1369            223         9.63   Spencer  Hammond  Spencer Hammond  ...
3620            187         9.25    Robert     Tate      Robert Tate  ...
3272            211         9.63     James   Richey     James Richey  ...

     tackleYdsScrim  sackYards  int intYards  intTd  fumOOB fumTurnover  \
9823            0.0        NaN  NaN      NaN    NaN     NaN         NaN
33              NaN        NaN  NaN      NaN    NaN     NaN         NaN
1369            NaN        NaN  NaN      NaN    NaN     NaN         NaN
3620          508.0      -10.0  3.0     71.0    1.0     NaN         NaN
3272            NaN        NaN  NaN      NaN    NaN     NaN         NaN

      gamePartSnapCount gameId   gamePartUnit
9823              548.0   21.0  special teams
33                  NaN    NaN            NaN
1369                NaN    NaN            NaN
3620                NaN    NaN            NaN
3272                NaN    NaN            NaN

[5 rows x 64 columns]
```

This merged dataframe should contain all the data we need for the artificial neural network (ANN) to determine the optimal NFL team, but there are quite a few missing values that will need to be explored further and dealt with.

Handling duplicate players

Let's check how many duplicate players there are:

In [101…
```
# Get total entry count and count of unique players
print(f"Total entries: {len(df)}")
unique_player_count = len(df["playerId"].unique())
print(f"Unique players: {unique_player_count}")
```

```
Total entries: 10080
Unique players: 10078
```

There are only a couple of duplicate players. Let's remove these:

In [102…
```
# Drop duplicate players
df.drop_duplicates(subset="playerId", keep="first", inplace=True)
```

In [103…
```
# Check for duplicates again
print(f"Total entries: {len(df)}")
unique_player_count = len(df["playerId"].unique())
print(f"Unique players: {unique_player_count}")
```

```
Total entries: 10078
Unique players: 10078
```

Handling missing values

In [104…
```
# Check null counts
print(f"Total entries: {len(df)}")

# Find missing value count by column
missing_values = df.isnull().sum().to_frame().reset_index()
missing_values.rename(columns={"index": "column", 0: "missing_count"}, inplace=True)
missing_values.sort_values(by="missing_count", ascending=False).transpose()
```

```
Total entries: 10078
```

Out[104]:

	34	36	40	43	42	41	37	39	38	35	...	11	
column	combineWonderlic	passComp	passSack	passDef	passHit	passSackYds	passLength	passInt	passTd	passAtt	...	collegeId	nameFu
missing_count	9713	9604	9604	9604	9604	9604	9604	9604	9604	9604	...	2	

2 rows × 64 columns

Some of the columns have almost 10,000 missing values, which seems problematic when there are only 10,080 entries in the entire dataframe. However, it is important to note that not every player performs every action in football. Quarterbacks have passing stats, running backs have rushing stats, and wide receivers have receiving stats, to name 3 examples, but each of these types of players is likely to be missing data for an action usually performed by another position, eg. the quarterback will not have data available on receiving.

Task 1.1

Select from a pool of 200 "Active Players" and 200 "Retired Players."

There is no field in any of the dataset's CSV files that indicates whether a player is active or not, nor is there any information about the dates of each game in `gameParticipation.csv`. However, DOB is available for each player, so the following criterion will be used to mark whether a player is active or retired:

- **At most 40 years old:** Active.
- **Over 40 years old:** Retired.

The dataset was uploaded to Kaggle in 2020 and the tagline says the data is from 2004-present, so we will use 2020 as the current year for this assignment. Thus, any player born before 1980 will be considered retired.

Determining if players are active or retired

First, we need to add a column for each player's age based on their DOB. Then, we will use their age to determine if they are active or not.

In [105…

```
# Define cutoff age
cutoff_age = 40

# Convert date of birth column to datetime
df["dob"] = pd.to_datetime(df["dob"])

# Add column for player age
df["age"] = 2020 - df["dob"].dt.year

# Mark players over 40 as retired and the rest as active
df["status"] = df["age"].apply(lambda x: "retired" if x > cutoff_age else "active")
```

```
# Show sample data
df[["dob", "age", "status"]].sample(3, random_state=42)
```

Out[105]:

	dob	age	status
9823	1997-06-18	23.0	active
33	1964-05-08	56.0	retired
1369	NaT	NaN	active

In [106…

```
# Show null count for player ages
print(f"Total players: {len(df)}")
null_dob_count = df["age"].isnull().sum()
print(f"Players without available DOB: {null_dob_count}")
```

```
Total players: 10078
Players without available DOB: 1589
```

There are quite a few players without DOB listed, and thus their ages are unknown and their statuses cannot be determined. For simplicity, we will drop these players.

In [107…

```
# Remove players with missing birthdays
df.dropna(subset=["dob"], inplace=True)

# Show updated counts
print(f"Total players: {len(df)}")
null_dob_count = df["age"].isnull().sum()
print(f"Players without available DOB: {null_dob_count}")
```

```
Total players: 8489
Players without available DOB: 0
```

Selecting 200 active and 200 retired players

In [108…

```
# Filter data down to active and retired players
active_players = df[df["status"] == "active"]
retired_players = df[df["status"] == "retired"]

# Take 200 random samples of each player status
active_sample = active_players.sample(200, random_state=42)
retired_sample = retired_players.sample(200, random_state=42)
```

In [109…
```
# Show sub-sample of active players
active_sample.sample(3, random_state=42)
```

Out[109]:

	combineId	playerId	combineYear	combinePosition	combineHeight	combineWeight	combineHand	nameFirst	nameLast	nameFull
9387	19807	20170726	2017	OL	78.75	332	NaN	Javarius	Leamon	Javarius Leamon
9355	19775	20170798	2017	SS	70.50	204	NaN	Lorenzo	Jerome	Lorenzo Jerome
6172	16173	20060141	2006	OT	78.38	315	NaN	Jonathan	Scott	Jonathan Scott

3 rows × 66 columns

In [110…
```
# Show sub-sample of retired players
retired_sample.sample(3, random_state=42)
```

Out[110]:

	combineId	playerId	combineYear	combinePosition	combineHeight	combineWeight	combineHand	nameFirst	nameLast	nameFull
4644	14644	20010162	2001	WR	71.6	180	NaN	Jonathan	Carter	Jonathan Carter
771	10771	19890165	1989	RB	71.4	205	NaN	Eric	Mitchel	Eric Mitchel
4045	14045	19990070	1999	DT	75.8	279	NaN	Jared	DeVries	Jared DeVries

3 rows × 66 columns

Task 1.2

1. *Define "optimal team" based on your decision of the player characteristics necessary to build a team:*
   - Starting offense (11 players): passing, rushing, and receiving.
   - Starting special teams (11 players): punting, punt returns, and kick returns.
   - Starting defense (11 players): tackles, sacks, safeties, and fumbles.

Each part of the team contains several types of positions whose performances are judged by different metrics:

Starting Offense (11 players)

1. Quarterback (1):
   - Passing Yards (`passLength`)
   - Touchdowns (`passTd`)
   - Completion Percentage (calculated from `passComp` and `passAtt`)
   - Interceptions (lower is better) (`passInt`)
2. Running Backs (2):
   - Rushing Yards (`rushYards`)
   - Touchdowns (`rushTd`)
   - Yards per Carry (calculated from `rushYards` and `rushPrimary`)
   - Fumbles (lower is better) (`recFumble`)
3. Wide Receivers (3):
   - Receiving Yards (`recYards`)
   - Touchdowns (`recYards` can be used as a proxy)
   - Receptions (`rec`)
   - Yards per Reception (calculated from `recYards` and `rec`)
4. Tight End (1):
   - Receiving Yards (`recYards`)
   - Touchdowns (`recYards` can be used as a proxy)
   - Receptions (`rec`)
   - Blocking Efficiency (if available) (not available in our data)
5. Offensive Line (4):
   - Sacks Allowed (lower is better) (not available in our data)
   - Run Blocking Grades (if available) (not available in our data)
   - Pass Blocking Grades (if available) (not available in our data)

Starting Special Teams (11 players)

1. Punter (1):
   - Punt Average (not available in our data)

- Inside 20 Yards (not available in our data)
- Touchbacks (lower is better) (not available in our data)

2. Kick Returner (1):
    - Return Yards (not available in our data)
    - Touchdowns (not available in our data)
    - Average Yards per Return (not available in our data)
3. Punt Returner (1):
    - Return Yards (not available in our data)
    - Touchdowns (not available in our data)
    - Average Yards per Return (not available in our data)
4. Special Teams Players (8):
    - Tackles on Special Teams (not available in our data)
    - Blocks(not available in our data)
    - Overall Special Teams Grades (if available) (not available in our data)

Starting Defense (11 players)

1. Defensive Line (4):
    - Sacks (`sackYards` can be used)
    - Tackles for Loss (not available in our data)
    - QB Hits (not available in our data)
    - Fumble Recoveries (not available in our data)
2. Linebackers (3):
    - Tackles (not available in our data)
    - Sacks (`sackYards` can be used)
    - Interceptions (`int`)
    - Forced Fumbles (not available in our data)
3. Defensive Backs (4):
    - Interceptions (`int`)
    - Pass Deflections (`passDef`)
    - Tackles (not available in our data)
    - Forced Fumbles (not available in our data)

The ideal team will be comprised of players who each maximize the positive metrics associated with their positions while minimizing the negative ones.

Some of these features need to be calculated, so let's handle that now.

In [111…
```
# Calculate completion percentage (quarterbacks)
df["completion_percentage"] = df["passComp"] / df["passAtt"]

# Calculate yards per carry (running backs)
df["yards_per_carry"] = df["rushYards"] / df["rushPrimary"]

# Calculate yards per reception (wide receivers and tight ends)
df["yards_per_reception"] = df["recYards"] / df["rec"]
```

In [112…
```
# Show sample values
df[["playerId", "completion_percentage", "yards_per_carry", "yards_per_reception"]].sample(3, random_state=42)
```

Out[112]:

	playerId	completion_percentage	yards_per_carry	yards_per_reception
2660	19940081	NaN	NaN	NaN
7099	20100239	0.66129	-1.0	NaN
7194	20100232	NaN	NaN	NaN

Again, many of these calculated features will be null because the features that went into them were null, which is because not every feature is relevant to every position.

Tasks 1.3 and 1.4

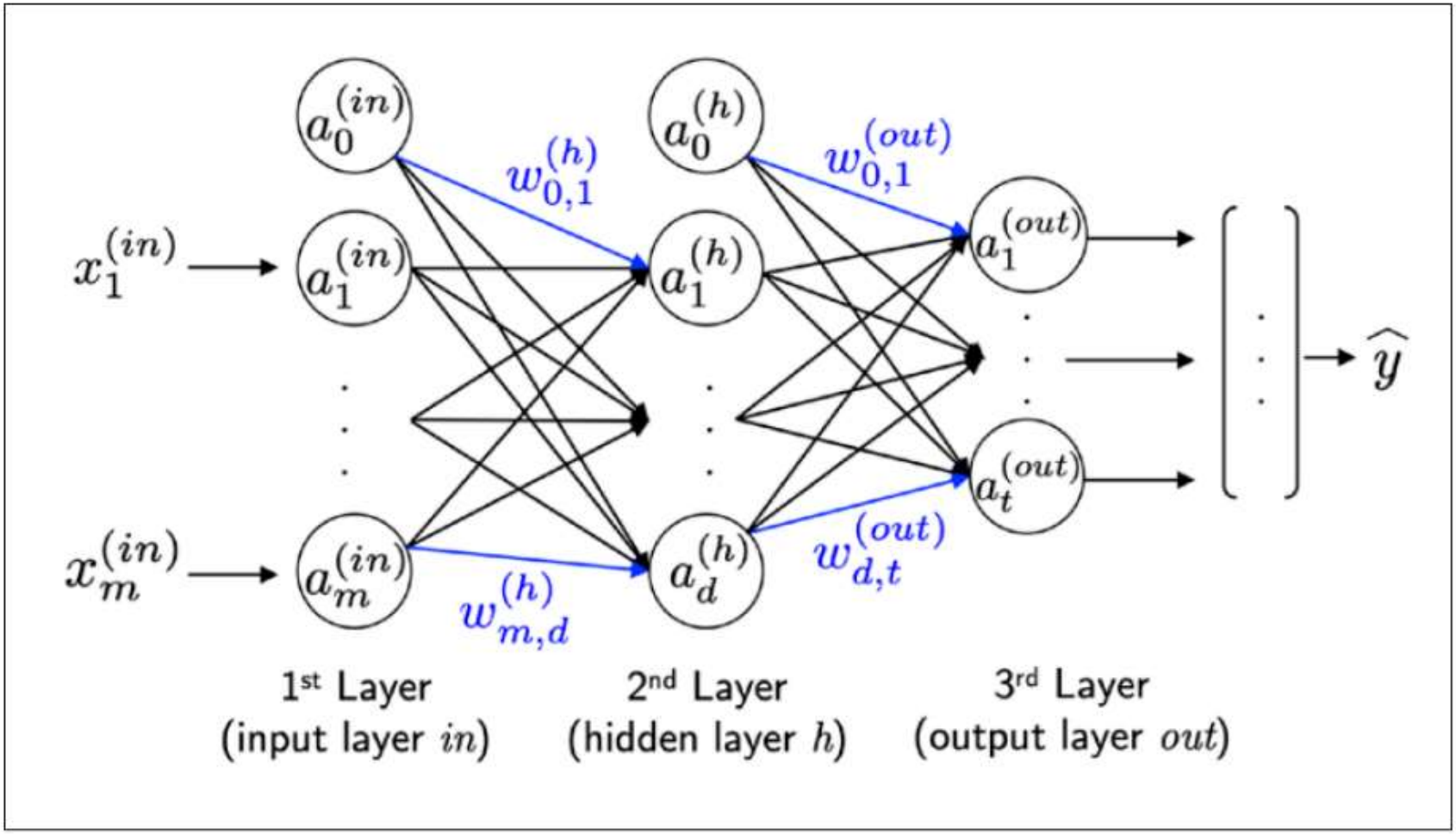
1.3: *Your task is to identify the optimal team from each pool comprised of 33 "Active Player" and "Retired Player" members.*

1.4: *Examine the multilayer neural network MLP architecture depicted in the "DSC-550 An Artificial Neural Network Model Image."*

In [113…
```
from IPython.display import Image

# Display ensemble method diagram
Image("ANN Model Image.png")
```

Out[113]:



Our inputs will be the players, our hidden layers will be decided later when the neural network is constructed, and the outputs will be the two optimal 33-man NFL teams, one comprised of only active players and the other only of retired players.

Tasks 1.5, 1.6 and 2.1

1.5: *Build a deep artificial neural network MLP to include the following: a) 1 input layer, b) as many hidden layers as you deem necessary, and c) an output layer fully connected to the hidden layers.*

1.6: *Explain your architecture and how the NFL player characteristics are used as inputs.*

2.1: *Starting at the input layer, forward propagate the patterns of the training data through the network to generate an output.*

Now it is time to put our data to use in developing an ANN. We will execute the following procedure:

1. Prepare features by selecting desired features, filtering the dataframe down to these features, and encoding features requiring it.
2. Split the data into training and testing sets.
3. Standardize the data to ensure all features are on the same scale.
4. Build, train and evaluate the neural network.
5. Use the outputs of the neural network as our optimal football teams.

Preparing relevant features

In [114…

```
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
import pandas as pd

# Define relevant feature columns
feature_columns = ["age", "position", "passAtt", "passComp", "passLength", "passTd", "passInt", "completion_percentage"
                   "rushYards", "rushPrimary", "rushTd", "yards_per_carry", "recYards", "rec", "recYac", "rec1down",
                   "recFumble", "yards_per_reception", "tackleYdsScrim", "sackYards", "int", "intYards", "intTd", "fumO
                   "fumTurnover", "combine40yd", "combineVert", "combineBench", "combineShuttle", "combineBroad",
                   "combine3cone", "combine60ydShuttle"]

# Filter dataframe down to desired features
features = df[feature_columns]

# Define model target, a general performance score
# ...
```

We will need a general performance score for each player in order to determine how effective they are.

In [115…

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np

# Convert features to dataframe
features = pd.DataFrame(features, columns=feature_columns)

# Fill infinite values with NaN and then NaN values with 0
features.replace([np.inf, -np.inf], np.nan, inplace=True)
```

```
features = features.fillna(0)

# Store position column before normalization
position = features["position"]

# Normalize columns
scaler = MinMaxScaler()
features = features.drop(columns=["position"])
features = scaler.fit_transform(features)

# Reassemble feature dataframe
features = pd.DataFrame(features, columns=[col for col in feature_columns if col != "position"])

# Define weights for different performance metrics (metrics where lower values are better have negative weights)
weights = {"age": 1, "passAtt": 1, "passComp": 1, "passLength": 1, "passTd": 1, "passInt": -1, "completion_percentage":
           "rushYards": 1, "rushPrimary": 1, "rushTd": 1, "yards_per_carry": 1, "recYards": 1, "rec": 1, "recYac": 1,
           "rec1down": 1, "recFumble": -1, "yards_per_reception": 1, "tackleYdsScrim": 1, "sackYards": 1, "int": 1,
           "intYards": 1, "intTd": 1, "fumOOB": -1, "fumTurnover": -1, "combine40yd": 1, "combineVert": 1,
           "combineBench": 1, "combineShuttle": 1, "combineBroad": 1, "combine3cone": 1, "combine60ydShuttle": 1}

# Calculate performance scores
features["performance_score"] = features.apply(lambda row: sum(row[metric] * weights.get(metric, 1) for metric in featu

# Add position column back in
features = pd.concat([features, position.reset_index(drop=True)], axis=1)

# Set target to performance score
target = features["performance_score"]
```

In [116…

```
# Show newly calculated performance scores
features[["position", "performance_score"]].head(5).transpose()
```

Out[116]:

	0	1	2	3	4
position	DB	C	C	WR	DT
performance_score	6.794919	6.018686	5.970604	6.66263	5.915415

Now that we have our performance scores for each player, we can finish preparing the data, then construct, train and test the ANN model.

Building the ANN

We will now build the ANN model with the following architecture:

- A dense input layer with 64 nodes, employing the ReLU (Rectified Linear Unit) activation function.
  - There are 64 features in the data, so having 64 nodes ensures every feature is utilized.
- Two dense hidden layers, one with 32 nodes and one with 16, both employing ReLU.
  - Each hidden layer has less nodes than the layer before it to capture more complex and abstract patterns among the input features.
- A dense output layer with a single node and linear activation function.
  - Only one node is needed because we are dealing with a regression task involving predicting a single value for each player.

In [117…

```
# Apply one-hot encoding to position column
encoder = OneHotEncoder(sparse_output=False)
features["position"] = features["position"].astype(str)
encoded_positions = encoder.fit_transform(features[["position"]])
encoded_positions_df = pd.DataFrame(encoded_positions, columns=encoder.get_feature_names_out(["position"]))

# Concatenate encoded positions with other features
features = features.drop("position", axis=1)
features = pd.concat([features, encoded_positions_df], axis=1)

# Apply train-test split (80% training, 20% testing)
x_train, x_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)

# Construct the neural network
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation="relu", input_shape=(x_train.shape[1],)),
    tf.keras.layers.Dense(32, activation="relu"),
    tf.keras.layers.Dense(16, activation="relu"),
    tf.keras.layers.Dense(1)
])

# Compile the model
model.compile(optimizer="adam", loss="mse")

# Train the model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=0)

# Evaluate the model
mse = model.evaluate(x_test, y_test, verbose=0)
print(f"Mean Squared Error: {mse}")
```

```
C:\Users\jdh10\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an `input_shape
`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first lay
er in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Mean Squared Error: 0.00016113516176119447
```

The model displays an extremely low MSE, so it should be highly effective at selecting the best players. Let's calculate predicted performance for each player and use that to find the optimal team.

In [118…
```
# Predict performance scores
df["predicted_performance"] = model.predict(features)
```

```
266/266 ━━━━━━━━━━━━━━━━━━━━ 0s 790us/step
```

In [119…
```
# Show performance scores for players
df[["playerId", "position", "status", "predicted_performance"]].sample(5, random_state=42)
```

Out[119]:

	playerId	position	status	predicted_performance
2660	19940081	LB	retired	6.637643
7099	20100239	QB	active	6.301752
7194	20100232	DE	active	6.439315
10032	20190119	DB	active	4.991574
440	19880070	C	retired	6.163869

Now it is time to select the 33 best players for each team. This can be done by filtering the data down to active players only, sorting them by `predicted_performance` in descending order, and selecting the appropriate number of players for each position on the team, then repeating this process for the retired players.

Each team needs the following positions in these exact quantities:

- Quarterback (1)
- Running back (2)
- Wide receiver (3)
- Tight end (1)
- Offensive line (4)
- Punter (1)

- Kick returner (1)
- Punt returner (1)
- Special teams player (8)
- Defensive line (4)
- Linebacker (3)
- Defensive back (4)

These are the position labels present in `df["position"]`:

In [120…
```
# Show values of df["position"] to determine which string corresponds to each position
print("Position labels:", df["position"].unique())
```

```
Position labels: ['DB' 'C' 'WR' 'DT' 'RB' 'OT' 'TE' 'DE' 'LB' 'QB' 'OG' 'K' 'S' 'FB' 'OLB'
 'OL' 'P' 'DL' 'LS' nan]
```

We are assuming that quarterback is "QB", running back is "RB", wide receiver is "WR", tight end is "TE", offensive line is "OL", punter is "P", kick returner is "K", special teams player is "S", defensive line is "DE", linebacker is "LB", and defensive back is "DB". There is no label for punt returner, so we will take the second-best punter for this position instead.

In [121…
```
# Filter the dataframe down to active and retired players respectively
active_players = df[df["status"] == "active"]
retired_players = df[df["status"] == "retired"]
```

Finding the optimal active team

In [122…
```
# Sort active players by predicted performance in descending order
active_players = active_players.sort_values(by="predicted_performance", ascending=False)

# Filter data down to relevant features
relevant_columns = ["playerId", "nameFull", "position", "predicted_performance"]
active_players = active_players[relevant_columns]

# Function to find the top n players for each position
def get_top_position_players(df, position, n):
    return df[df["position"] == position].head(n)

# Create dataframe to store the optimal active team in
optimal_team_active = pd.DataFrame(columns=relevant_columns)

# Find optimal starting offense players
```

```
optimal_team_active = pd.concat([optimal_team_active, get_top_position_players(active_players, "QB", 1)])
optimal_team_active = pd.concat([optimal_team_active, get_top_position_players(active_players, "RB", 2)])
optimal_team_active = pd.concat([optimal_team_active, get_top_position_players(active_players, "WR", 3)])
optimal_team_active = pd.concat([optimal_team_active, get_top_position_players(active_players, "TE", 1)])
optimal_team_active = pd.concat([optimal_team_active, get_top_position_players(active_players, "OL", 4)])

# Find optimal starting special teams players
punter_and_punt_returner = get_top_position_players(active_players, "P", 2)
optimal_team_active = pd.concat([optimal_team_active, punter_and_punt_returner.iloc[0:1]])
optimal_team_active = pd.concat([optimal_team_active, get_top_position_players(active_players, "K", 1)])
optimal_team_active = pd.concat([optimal_team_active, punter_and_punt_returner.iloc[1:2]])
optimal_team_active = pd.concat([optimal_team_active, get_top_position_players(active_players, "S", 8)])

# Find optimal starting defense players
optimal_team_active = pd.concat([optimal_team_active, get_top_position_players(active_players, "DE", 4)])
optimal_team_active = pd.concat([optimal_team_active, get_top_position_players(active_players, "LB", 3)])
optimal_team_active = pd.concat([optimal_team_active, get_top_position_players(active_players, "DB", 4)])

# Show optimal active team
print(f"Team size: {len(optimal_team_active)}")
print(f"Duplicate players: {optimal_team_active.duplicated().sum()}")

optimal_team_active
```

```
Team size: 33
Duplicate players: 0
```

Out[122]:

	playerId	nameFull	position	predicted_performance
7961	20120075	Russell Wilson	QB	8.227610
7997	20130048	Le'Veon Bell	RB	10.082479
6370	20070012	Marshawn Lynch	RB	9.431379
7274	20100082	Emmanuel Sanders	WR	8.873080
7296	20100060	Golden Tate	WR	8.701510
7084	20100195	Antonio Brown	WR	8.414613
5645	20040032	Benjamin Watson	TE	7.913313
7384	20110138	Marcus Cannon	OL	6.624431
8831	20150059	Ty Sambrailo	OL	6.528787
8784	20150049	Mitch Morse	OL	6.480525
7097	20100106	Bruce Campbell	OL	6.463354
9767	20180172	JK Scott	P	5.529027
8081	20130177	Dustin Hopkins	K	4.174629
9595	20180751	Trevor Daniel	P	4.574535
5975	20060207	Antoine Bethea	S	8.816082
6487	20070037	Eric Weddle	S	8.611670
7919	20120029	Harrison Smith	S	7.918158
6900	20090014	Malcolm Jenkins	S	7.847250
8290	20140128	Tre Boston	S	7.817276
7189	20100163	Reshad Jones	S	7.733195
6911	20090095	Rashad Johnson	S	7.623786
5891	20050123	Kerry Rhodes	S	7.591487
5743	20050128	Chauncey Davis	DE	6.904581
5557	20040035	Igor Olshansky	DE	6.842937
5503	20040253	Isaac Hilton	DE	6.806894

	playerId	nameFull	position	predicted_performance
5728	20050706	Eric Coleman	DE	6.657728
8161	20130030	Alec Ogletree	LB	7.865907
5925	20050045	Lofa Tatupu	LB	7.340002
5148	20030029	Nick Barnett	LB	7.137477
5284	20030039	Rashean Mathis	DB	8.874640
7218	20100027	Devin McCourty	DB	8.778269
6076	20060024	Johnathan Joseph	DB	8.770149
6744	20080020	Aqib Talib	DB	8.641890

This is the optimal NFL team comprised of active players.

Finding the optimal retired team

In [123…

```
# Filter data down to relevant features
retired_players = retired_players[relevant_columns]

# Create dataframe to store the optimal active team in
optimal_team_retired = pd.DataFrame(columns=relevant_columns)

# Find optimal starting offense players
optimal_team_retired = pd.concat([optimal_team_retired, get_top_position_players(retired_players, "QB", 1)])
optimal_team_retired = pd.concat([optimal_team_retired, get_top_position_players(retired_players, "RB", 2)])
optimal_team_retired = pd.concat([optimal_team_retired, get_top_position_players(retired_players, "WR", 3)])
optimal_team_retired = pd.concat([optimal_team_retired, get_top_position_players(retired_players, "TE", 1)])
optimal_team_retired = pd.concat([optimal_team_retired, get_top_position_players(retired_players, "OL", 4)])

# Find optimal starting special teams players
punter_and_punt_returner = get_top_position_players(retired_players, "P", 2)
optimal_team_retired = pd.concat([optimal_team_retired, punter_and_punt_returner.iloc[0:1]])
optimal_team_retired = pd.concat([optimal_team_retired, get_top_position_players(retired_players, "K", 1)])
optimal_team_retired = pd.concat([optimal_team_retired, punter_and_punt_returner.iloc[1:2]])
optimal_team_retired = pd.concat([optimal_team_retired, get_top_position_players(retired_players, "S", 8)])

# Find optimal starting defense players
optimal_team_retired = pd.concat([optimal_team_retired, get_top_position_players(retired_players, "DE", 4)])
optimal_team_retired = pd.concat([optimal_team_retired, get_top_position_players(retired_players, "LB", 3)])
```

```python
optimal_team_retired = pd.concat([optimal_team_retired, get_top_position_players(retired_players, "DB", 4)])

# Show optimal retired team
print(f"Team size: {len(optimal_team_retired)}")
print(f"Duplicate players: {optimal_team_retired.duplicated().sum()}")

optimal_team_retired
```

```
Team size: 33
Duplicate players: 0
```

Out[123]:

	playerId	nameFull	position	predicted_performance
25	19870110	Steve Beuerlein	QB	5.406932
7	19870224	Joe Armentrout	RB	6.796946
16	19870143	Steve Bartalo	RB	6.628869
4	19870801	Lyneal Alston	WR	6.654113
11	19870083	Stephen Baker	WR	6.480316
14	19870114	Roy Banks	WR	6.459214
10	19870062	Robert Awalt	TE	6.423102
382	19880264	Brian Habib	OL	6.329853
571	19880094	Dave Widell	OL	5.848831
1881	19920244	Jay Leeuwenburg	OL	5.778121
1893	19920068	Siupeli Malamala	OL	4.523255
552	19880068	Tom Tupa	P	5.686451
50	19870246	Greg Davis	K	5.704953
980	19900190	Kent Elmore	P	2.502450
63	19870810	Tony Elliott	S	6.452024
78	19870812	Charles Glaze	S	6.397377
147	19870034	Tim McDonald	S	6.635945
262	19870010	Rod Woodson	S	6.622608
290	19880257	John Booty	S	6.822235
296	19880808	Selwyn Brown	S	6.851101
323	19880816	Greg Cox	S	6.699334
472	19880035	Anthony Newman	S	6.841083
13	19870176	Robert Banks	DE	3.796422
28	19870016	John Bosa	DE	6.007041
36	19870017	Jason Buck	DE	2.998559

	playerId	nameFull	position	predicted_performance
66	19870047	Donald Evans	DE	6.203267
24	19870044	Ray Berry	LB	6.889822
30	19870060	David Brandon	LB	6.531764
40	19870159	Toby Caston	LB	6.779288
0	19870067	Michael Adams	DB	6.782707
6	19870256	Anthony Anderson	DB	6.490819
9	19870179	Gene Atkins	DB	6.822636
21	19870076	Len Bell	DB	6.729558

Thus, we have found the optimal retired team as well.

Task 2.2

Based on the network's output, calculate the error that we want to minimize using a cost function that we will describe later.

The error we want to minimize from the ANN is the mean squared error (MSE), calculated from the differences between the predicted performance and actual performance of each player. MSE is calculated using the following formula:

$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$, where y_i is the ith player's actual performance rating, and $\hat{y}_i$ is the ith player's predicted performance rating by the ANN.

The MSE calculated earlier after 50 epochs was very small:

In [124…
```
print(f"Mean Squared Error: {mse}")
```

```
Mean Squared Error: 0.00016113516176119447
```

This means the model was highly accurate at predicting how well a player would perform in a certain position.

Task 2.3

Back propagate the error, find its derivative with respect to each weight in the network, and update the model.

The ANN was built and trained using the following code:

In [125…

```
"""
# Construct the neural network
model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation="relu", input_shape=(x_train.shape[1],)),
    tf.keras.layers.Dense(32, activation="relu"),
    tf.keras.layers.Dense(16, activation="relu"),
    tf.keras.layers.Dense(1)
])

# Compile the model
model.compile(optimizer="adam", loss="mse")

# Train the model
history = model.fit(x_train, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=0)
""";
```

Backpropagation is automatically performed by TensorFlow through the `fit` method. The following procedure is performed during each epoch:

1. **Forward pass:** The inputs are pushed through each layer, and weights and biases are applied. Each layer applies activation functions to the outputs, and eventually the final output is produced.
2. **Loss calculation:** The loss function is used to calculate the difference between the prediction and actual target value.
3. **Backward pass (backpropagation):** The model calculates the gradient of the loss with respect to the output and propagates this backward through the network, starting at the output layer and ending at the input layer.
4. **Updating weights:** The weights and biases are updated according to the computed gradient. These updates are affected by the learning rate and specific algorithm employed by the optimizer.

Task 2.4

Repeat steps 1 through 3 for multiple epochs and learn the weights of the MLP.

The following code prints the weights of the neural network.

In [126…

```python
# Get weights for each layer
weights = model.get_weights()

# Print the weights and biases for each layer
for i, weight in enumerate(weights):
    print(f"Layer {i+1} weights and biases:")
    print(weight)
```

```
Layer 1 weights and biases:
[[ 0.10791263 -0.11166793  0.19243194 ... -0.08394465 -0.05833316
  -0.06283175]
 [ 0.10957924 -0.22078486 -0.16940694 ...  0.13474932  0.18129638
  -0.14329545]
 [-0.00483246  0.15151969 -0.1259583  ... -0.19238164 -0.22446622
  -0.16241279]
 ...
 [ 0.08490603  0.13478017  0.13555744 ...  0.1612862  -0.21720973
  -0.0499775 ]
 [ 0.01805583 -0.16807885 -0.1298865  ...  0.05937767 -0.19714668
   0.29229847]
 [-0.13878994  0.00777497  0.16908844 ...  0.22034204  0.1078299
  -0.0759768 ]]
Layer 2 weights and biases:
[ 0.0328433  -0.03153051  0.03247614  0.          0.2637064   0.03052419
  0.00184031 -0.03538732  0.02819627  0.01316529  0.02766686 -0.01993771
 -0.02157724 -0.01912044 -0.00422852  0.03047007 -0.00907813  0.01966785
 -0.00771281  0.02350657 -0.01385856 -0.03075346  0.         -0.01191116
 -0.00775678  0.          0.01880149  0.01700885  0.03294697  0.02249399
  0.03386337  0.          0.01530033  0.          0.15680367 -0.02382887
  0.          0.          0.03205633  0.02891006  0.03057027 -0.03224433
  0.01890181 -0.01808345 -0.0267111  -0.02041529 -0.03008166  0.03136162
 -0.00751695  0.03636418  0.0354649   0.02463092  0.03256302  0.
  0.01190501  0.          0.03614042 -0.03084602  0.03505427  0.0210706
  0.00906854  0.          0.         -0.00435697]
Layer 3 weights and biases:
[[ 0.17262913  0.17346889  0.08352719 ... -0.01027527  0.05138338
  -0.09418618]
 [-0.04023653 -0.19917443 -0.12477641 ...  0.16590707  0.08218954
  -0.08322424]
 [ 0.14116523 -0.1739076   0.0785571  ... -0.09235166  0.19572376
   0.13221858]
 ...
 [-0.2124005   0.10720146 -0.1394363  ... -0.21953619  0.06381285
  -0.19567144]
 [ 0.09612423  0.19843978 -0.014817   ...  0.00306779 -0.24152464
   0.08606851]
 [-0.05679864  0.18349795  0.2117424  ...  0.03782072 -0.08530036
   0.01365934]]
Layer 4 weights and biases:
[ 0.03084091  0.03441382  0.02823644  0.02088876 -0.02793427 -0.02230735
  0.         -0.01774844  0.00586179  0.          0.03107589 -0.01492645
 -0.01700318 -0.01619694 -0.00994726  0.0309065  -0.00600243  0.02473598
  0.          0.03632221  0.         -0.02348351  0.          0.03675796
```

```
 -0.01008778 -0.01138195  0.21551238  0.0387839   0.01887994 -0.03530304
  0.03758569 -0.01613939]
Layer 5 weights and biases:
[[-2.93013752e-01  3.09478968e-01  2.13907212e-01 -3.61828394e-02
  -2.89421141e-01  1.50582269e-01 -3.01334292e-01  1.42175734e-01
  -1.44718587e-01  1.82988077e-01 -1.42609447e-01  3.47678453e-01
   1.77213520e-01  1.59592216e-03  3.69260728e-01 -2.96213210e-01]
 [-2.61513233e-01 -2.11773980e-02 -1.96274668e-01 -1.77758351e-01
   1.89524263e-01  2.39204556e-01 -6.78252487e-04 -4.16223949e-04
  -2.56797463e-01 -3.48519504e-01  2.98951149e-01 -2.46793821e-01
  -3.66551071e-01  5.12617491e-02  3.13409477e-01 -3.37688774e-01]
 [ 2.10734099e-01 -3.33901703e-01 -1.48499012e-03  6.02979325e-02
  -8.21617246e-02 -1.34993792e-01  6.35475805e-03  1.89794134e-02
  -2.91660100e-01  5.27916476e-02 -8.63392949e-02  3.04021627e-01
  -2.55966902e-01  3.72680068e-01 -6.53119683e-02  1.95743337e-01]
 [-1.25769496e-01  2.12532237e-01 -4.25423682e-02 -1.43879429e-01
  -2.84849375e-01 -1.10721461e-01  3.14086080e-01 -2.82493293e-01
   2.11731985e-01 -1.35715470e-01 -1.60790950e-01  2.14734197e-01
  -3.19174111e-01  1.16772115e-01  1.09203383e-01 -2.07611546e-01]
 [-1.76491700e-02  8.39313492e-02 -1.54521897e-01 -2.64482141e-01
  -1.03528768e-01  2.07913175e-01  1.29721940e-01  1.80359378e-01
   6.80703670e-02 -3.53133351e-01 -1.51498556e-01 -2.96443909e-01
   2.42452875e-01  3.72233987e-01  3.13663147e-02  2.82844692e-01]
 [-2.43506387e-01  3.42565179e-02  6.65036738e-02  2.25123167e-01
   3.44285280e-01 -2.24100128e-01  2.68181056e-01 -2.21233353e-01
   2.15809986e-01  3.04386944e-01  8.73010606e-02 -5.58275841e-02
  -1.00271210e-01  8.46147388e-02  7.70747662e-02  1.22104876e-01]
 [-1.09895036e-01  3.33657235e-01 -2.26564601e-01  5.13521433e-02
  -2.24022388e-01 -3.33083928e-01  3.01675409e-01  3.01645488e-01
  -2.84156740e-01  3.04345399e-01 -1.09748036e-01 -8.01161826e-02
   2.06697285e-02 -1.19581312e-01  1.63788408e-01  1.03786588e-02]
 [ 1.95998490e-01  5.77220693e-02  7.44680762e-02 -1.73916399e-01
   1.91355735e-01  1.00016780e-01 -1.42158136e-01 -3.34168136e-01
   1.53993620e-02 -3.58462147e-02 -9.14223343e-02 -1.36618763e-01
   3.18915427e-01  9.62858871e-02  2.08935961e-02  1.74396783e-01]
 [ 2.71036059e-01  2.27884606e-01 -7.78165460e-03  2.93972641e-01
  -1.44627243e-01 -3.15488167e-02  2.51946688e-01 -1.14814565e-01
   4.85263951e-02  1.82985336e-01 -3.72545645e-02  2.28252634e-01
   3.21227700e-01 -1.25280231e-01  1.62177697e-01 -2.64357984e-01]
 [ 1.98483765e-02 -2.21999079e-01  1.64792925e-01 -1.80151388e-01
   3.28481495e-02 -2.02372730e-01  2.04259485e-01 -2.05814689e-01
   3.07402581e-01  1.73351020e-01 -5.18164337e-02 -2.29867727e-01
  -1.85834765e-02  1.13843530e-01 -2.58003354e-01 -5.26495874e-02]
 [-2.91895151e-01  2.92223785e-02  2.39479154e-01 -1.15887620e-01
  -2.07714140e-02  3.93014699e-01 -1.59994084e-02 -2.08379164e-01
```

```
 -1.86431602e-01 -3.04465592e-02  2.71684706e-01 -1.64084122e-01
 -1.53929472e-01  3.15489829e-01 -2.14641273e-01 -3.47032428e-01]
[-2.72777498e-01 -2.88394779e-01  1.12045288e-01 -7.21867085e-02
  3.17843288e-01 -1.10605367e-01  1.78542405e-01  2.98428208e-01
  8.40248317e-02 -2.33310983e-01 -3.09433371e-01 -1.50908232e-01
  1.09800018e-01 -3.24465364e-01  7.65072405e-02  2.32154042e-01]
[ 3.11436683e-01 -2.28212595e-01 -1.76660120e-01  2.94498682e-01
  3.53280753e-01 -2.04030141e-01 -3.10042769e-01 -5.06885834e-02
 -5.85198142e-02  3.37367654e-01 -4.78814542e-03 -1.68382823e-01
 -2.30171233e-01 -1.33759528e-01  7.23672882e-02 -3.08653325e-01]
[-3.12963836e-02  2.98623592e-01  3.06049138e-01 -6.38206527e-02
 -1.99898958e-01  1.99404463e-01  1.96331143e-02  4.35031159e-03
  2.04399213e-01  2.63921320e-01 -2.68096775e-01 -2.72573143e-01
  1.80660561e-01  2.19093427e-01  1.17665395e-01  2.63446748e-01]
[ 3.16684514e-01 -3.90950777e-02  3.50358218e-01  3.24633196e-02
 -1.64528728e-01  4.36416380e-02  2.71016985e-01 -6.87437281e-02
  9.77662206e-03  8.87898654e-02  3.20553541e-01 -3.34018081e-01
  2.77867168e-01  1.30941803e-02  1.92332342e-02 -2.22455695e-01]
[-3.07619512e-01  8.35064501e-02 -3.04318696e-01  3.37716714e-02
  3.44617635e-01  2.32075289e-01 -2.15138584e-01  9.81440395e-02
  1.26092061e-02 -3.64436269e-01  1.85948182e-02  3.21404576e-01
  2.36571580e-01 -5.96887395e-02  2.44170517e-01 -4.62407544e-02]
[-1.43001927e-02 -1.97784454e-01  1.76907092e-01  2.92370260e-01
  2.44808763e-01 -1.69546902e-01 -2.41454035e-01 -2.30137646e-01
 -3.58806282e-01  3.01682144e-01  3.31760257e-01 -6.55752048e-02
  2.75131077e-01  2.98607111e-01  3.82492840e-02  2.91996121e-01]
[-2.42502943e-01 -3.04442614e-01 -1.24580681e-01 -3.69907498e-01
 -1.58411711e-01  1.15574868e-02 -2.91670799e-01  5.73193505e-02
  4.82818708e-02 -2.66749591e-01  1.69127718e-01  3.31323206e-01
  2.13983193e-01  2.47435927e-01 -1.43059015e-01  3.05218637e-01]
[ 1.10928148e-01  2.93977529e-01  1.25459790e-01 -2.32042670e-01
  9.81957614e-02 -2.36853987e-01 -1.86343774e-01  1.36431873e-01
 -6.20198250e-03  8.80784690e-02 -3.38941246e-01  3.42198461e-01
 -2.12478355e-01  7.38493800e-02 -1.90537572e-02  8.91640782e-03]
[ 1.13428153e-01 -6.30505458e-02 -3.41567159e-01  8.95717219e-02
 -2.63157576e-01  2.08719417e-01  1.39658138e-01 -9.98012871e-02
  1.11850359e-01 -6.95186779e-02  3.38371485e-01  5.13267666e-02
 -2.73552895e-01 -7.14894980e-02  1.54137358e-01 -2.03298405e-01]
[ 1.25224173e-01 -1.79849371e-01  2.79340953e-01 -3.17630023e-01
  2.40360707e-01 -1.70492768e-01  3.09013277e-01  2.84752518e-01
 -2.38974392e-02  2.49810070e-01  1.48675710e-01  2.38887817e-01
 -1.26019329e-01  1.65358096e-01  1.99228853e-01  3.03507358e-01]
[ 2.16220871e-01  3.39328378e-01 -1.52623594e-01 -3.29229534e-01
 -1.74798235e-01  9.06019509e-02  2.53075272e-01 -2.08809629e-01
 -2.52936095e-01  2.67244250e-01 -2.19869614e-01 -1.25838295e-01
```

```
  -1.92990035e-01  3.57569396e-01 -2.74188042e-01 -4.32764702e-02]
 [-2.48013049e-01  3.01872492e-02 -2.28538081e-01  3.38609725e-01
   1.16524071e-01  1.86976165e-01 -1.10669196e-01 -1.65172070e-01
   1.76981390e-02 -1.71378255e-03  1.17922992e-01 -3.39832067e-01
   3.42442185e-01  5.68485260e-02  2.41652817e-01 -2.44407713e-01]
 [-5.01982309e-02 -1.93837676e-02  2.57379323e-01  1.51596054e-01
  -2.22634971e-02  1.48652330e-01  1.38219625e-01 -1.98567554e-01
  -2.55557835e-01  9.17476602e-03  2.10798487e-01  3.72168809e-01
   1.20396197e-01 -1.37776136e-01  4.81695347e-02  2.66361069e-02]
 [-2.51346529e-02 -1.12679843e-02 -8.45021605e-02  3.35412771e-01
   5.49289882e-02 -6.48388118e-02  2.83835810e-02 -1.93178192e-01
   4.17288877e-02 -3.22110772e-01  2.69661367e-01 -3.18686575e-01
   1.72352046e-01 -9.60644782e-02  3.41488212e-01  2.65729338e-01]
 [ 4.87695634e-02 -9.45803300e-02  2.18894631e-01  3.95046026e-02
  -6.98117912e-02  3.07438195e-01  1.70053959e-01  1.31296277e-01
  -3.01131487e-01 -1.13477096e-01  2.68512040e-01  2.81299818e-02
   1.01975381e-01  8.96685869e-02 -2.37065163e-02 -1.80051327e-02]
 [ 8.60083103e-03 -3.40399712e-01 -5.18137217e-02  2.91859329e-01
   6.86008930e-02  1.76607817e-01  4.29535687e-01 -2.33476579e-01
  -3.05475384e-01  1.61170959e-04 -4.07850504e-01 -1.08046040e-01
   2.34260783e-01 -1.58647522e-01  7.71677122e-02  2.45357126e-01]
 [ 8.57239068e-02  2.67678499e-01 -2.81672597e-01  4.16239724e-02
  -1.88447744e-01 -2.48554274e-01  2.40203589e-01  1.50275782e-01
  -2.11584076e-01  1.09469265e-01  3.18831533e-01  2.00200781e-01
  -3.11708212e-01  6.44548684e-02 -1.51553690e-01 -1.95787355e-01]
 [-1.85518991e-02  3.07651430e-01 -2.33502805e-01  1.72585726e-01
  -1.02498770e-01  6.67735487e-02  8.80447701e-02 -8.67535248e-02
  -9.47418734e-02 -2.49499194e-02 -1.38345554e-01  1.22798108e-01
  -3.04694235e-01  2.81375557e-01  3.02033871e-01 -6.11555874e-02]
 [-3.25581469e-02  1.15314387e-01  1.65643781e-01  4.98110428e-02
  -2.45595068e-01  3.05805635e-02 -1.99197754e-01  4.73471060e-02
  -1.90453246e-01  2.64456689e-01  3.74795228e-01 -3.16980332e-01
   2.72610486e-01  2.15596020e-01 -3.13891441e-01 -1.64568707e-01]
 [ 2.77825117e-01  1.11636661e-01 -7.32054412e-02 -9.06846896e-02
   1.24986470e-01 -2.48698771e-01  2.71742254e-01 -3.50624770e-01
   6.25775903e-02 -3.38321716e-01  3.89245898e-01  3.59340280e-01
   7.37117454e-02 -2.02855423e-01  4.44419496e-02  2.53600538e-01]
 [ 1.63829744e-01 -2.50718623e-01  2.84164160e-01  1.42426074e-01
   1.50462896e-01  1.46756008e-01 -1.94030613e-01  1.10825896e-01
   2.45329410e-01  1.31723091e-01  3.10489416e-01  9.31808576e-02
  -1.23834401e-01  8.05945881e-03 -3.08245450e-01 -2.33667627e-01]]
Layer 6 weights and biases:
[-0.0136522   0.00557379  0.         -0.0326172   0.          0.03407002
 -0.00864603 -0.0165364  -0.01083859 -0.01426214  0.03311809  0.03306242
 -0.01949466  0.03409554  0.03638664 -0.01057555]
```

```
Layer 7 weights and biases:
[[-0.06545139]
 [ 0.13250503]
 [ 0.3443883 ]
 [-0.17966524]
 [ 0.3221181 ]
 [ 0.06062834]
 [-0.4298065 ]
 [-0.4072117 ]
 [-0.5062634 ]
 [-0.06432533]
 [ 0.24307938]
 [ 0.437585  ]
 [-0.45870033]
 [ 0.05966457]
 [ 0.5665586 ]
 [-0.5671701 ]]
Layer 8 weights and biases:
[0.03293692]
```

The original model architecture only includes 4 layers (2 hidden layers), but this output contains 6 hidden layers because TensorFlow automatically adds layers for tasks like batch normalization and dropout regularization.

Task 2.5

Use forward propagation to calculate the network output and apply a threshold function to obtain the predicted class labels in the one-hot representation.

Our model has applied forward propagation by passing data through the input layer, processing it in the hidden layers with activation functions, then receiving an output from the output layer. The threshold function is then applied to this output, and in our case, it selected the top n players for each position on the team based on their performance scores, where n is the number of players needed for that position on a 33-man team.

Task 2.6

Interpret the output of your MLP in the context of selecting an optimal football team.

The original objective of the model was to select two optimal football teams, one made up of active players and the other made up of retired players. This was done by selecting relevant metrics to judge player performance on, applying weights to each metric according to its importance and sign (metrics where lower values are better must have a negative weight applied), calculating performance scores for each player using forward propagation through the MLP, grouping players by their positions, and then choosing the best players for each position based on their performance scores.

Thus, the two 33-man teams I have constructed are what I believe to be the optimal active and retired teams. These are the teams again:

In [127…

```
# Display optimal active team
optimal_active_players = optimal_team_active["nameFull"].unique()
print(f"Optimal active team: {optimal_active_players}\n")

# Display optimal retired team
optimal_retired_players = optimal_team_retired["nameFull"].unique()
print(f"Optimal retired team: {optimal_retired_players}")
```

```
Optimal active team: ['Russell Wilson' "Le'Veon Bell" 'Marshawn Lynch' 'Emmanuel Sanders'
 'Golden Tate' 'Antonio Brown' 'Benjamin Watson' 'Marcus Cannon'
 'Ty Sambrailo' 'Mitch Morse' 'Bruce Campbell' 'JK Scott' 'Dustin Hopkins'
 'Trevor Daniel' 'Antoine Bethea' 'Eric Weddle' 'Harrison Smith'
 'Malcolm Jenkins' 'Tre Boston' 'Reshad Jones' 'Rashad Johnson'
 'Kerry Rhodes' 'Chauncey Davis' 'Igor Olshansky' 'Isaac Hilton'
 'Eric Coleman' 'Alec Ogletree' 'Lofa Tatupu' 'Nick Barnett'
 'Rashean Mathis' 'Devin McCourty' 'Johnathan Joseph' 'Aqib Talib']

Optimal retired team: ['Steve Beuerlein' 'Joe Armentrout' 'Steve Bartalo' 'Lyneal Alston'
 'Stephen Baker' 'Roy Banks' 'Robert Awalt' 'Brian Habib' 'Dave Widell'
 'Jay Leeuwenburg' 'Siupeli Malamala' 'Tom Tupa' 'Greg Davis'
 'Kent Elmore' 'Tony Elliott' 'Charles Glaze' 'Tim McDonald' 'Rod Woodson'
 'John Booty' 'Selwyn Brown' 'Greg Cox' 'Anthony Newman' 'Robert Banks'
 'John Bosa' 'Jason Buck' 'Donald Evans' 'Ray Berry' 'David Brandon'
 'Toby Caston' 'Michael Adams' 'Anthony Anderson' 'Gene Atkins' 'Len Bell']
```

References

- GeeksforGeeks. (2023b, June 2). Artificial Neural Networks and its Applications. GeeksforGeeks. https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/
- OpenAI. (2024). ChatGPT [Large language model]. https://chatgpt.com/

- Microsoft. (2024). Copilot [Large language model]. https://www.bing.com/?FORM=Z9FD1