Enhancing Fantasy Football Score Prediction Using Statistical and Textual Features

***Abstract*—We aim to improve fantasy football score prediction. Progress in this task has direct financial relevance in the form of gambling. This is a prediction problem that is quite challenging due to highly complex data that depends on many unmodeled features, as well as a general scarcity of valid data. For this problem, our overarching objective is to maximize prediction accuracy in determining the relative ordering of player performance (within various position groups) for a given week of NFL games. In doing so, we consider varied sources of data to determine which data and features are most successful in predicting week-to-week performance. We plan to investigate the efficacy of using embeddings of text-based sports articles as a feature. The idea behind this method is that articles may capture additional context and better inform previous statistics. If successful, a similar methodology may be used to inform trading decisions by analyzing the analysis and articles published by financial analysts. We intend to compare our prediction performance against the analysis and rankings from fantasy-focused sports media outlets.**

***Index Terms*—Fantasy football, neural networks, prediction, sports analytics, embeddings, text mining.**

I. Train–Test Split

Two distinct data partitioning strategies are adopted in this study to evaluate model performance and generalization. The dataset spans the 2023 and 2024 NFL seasons, and a sliding window of 15 weeks is employed to construct training samples. Each data point has the statistics from the previous 15 weeks performance (passing yards, interceptions, etc.) as the features, which are used to predict the fantasy points for the current week. Because of this construction, the samples in the dataset are not independent, as the results from a previous game directly affect the features of of the data points for future games. Changing the width of the sliding window results in different models.

a) (1) Identically Distributed Split for Quarterbacks: The feature-extracted dataset containing quarterback performance records was randomly divided into two disjoint subsets: 80% for training and 20% for testing. This partition approximately satisfies the identically distributed assumption and provides a representative estimate of the model's predictive capability on quarterback data. However, because player statistics across consecutive weeks are not strictly independent, the samples do not fully satisfy the i.i.d. assumption.

b) (2) Non-IID Split for All Players: To assess cross-role generalization, a non-i.i.d. partition was constructed as follows: the same 80–20 split used for quarterbacks (prior to feature extraction) was retained, and all non-quarterback data were added exclusively to the training set. This configuration deliberately introduces a distributional shift between the training and testing data, enabling the model to exploit heterogeneous player information during training while being evaluated solely on quarterback performance.

II. Optimization from Scratch

To evaluate optimization performance, a perceptron model was trained using a stochastic gradient descent (SGD) algorithm implemented from scratch in `PyTorch`. The optimizer minimized the mean squared error (MSE) loss computed over randomly sampled mini-batches.

The training process exhibited a consistent reduction in both training and validation losses across epochs, indicating stable convergence and effective gradient updates. Over 200 epochs, the MSE steadily decreased before reaching a plateau, suggesting convergence toward a local minimum with diminishing improvement beyond approximately 150 epochs. The close alignment between the training and validation loss curves further demonstrates stable generalization and limited overfitting. Moreover, the identically distributed split yielded superior convergence behavior and overall predictive performance compared to the non-i.i.d. setting.

Listing 1. SGD training loop implemented from scratch in `PyTorch`.

```
w = torch.randn(feature_dim, 1)
b = torch.zeros(1)

for epoch in range(1, EPOCHS + 1):
    running_loss, count = 0.0, 0
    for xb, yb in train_loader:
        # Forward pass
        y_hat = xb @ w + b

        # MSE loss
        loss = ((y_hat - yb) ** 2).mean()

        # Calculate gradient
        r = y_hat - yb
        m = xb.size(0)

        grad_w = (2.0 / m) * (xb.T @ r)
        grad_b = (2.0 / m) * r.sum(0)

        # SGD update
        with torch.no_grad():
            w -= lr * grad_w
            b -= lr * grad_b
```

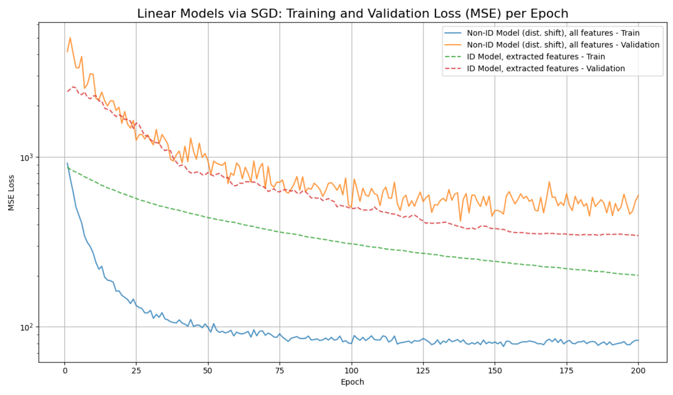

Fig. 1. Training and validation MSE for the perceptron model optimized with custom SGD.

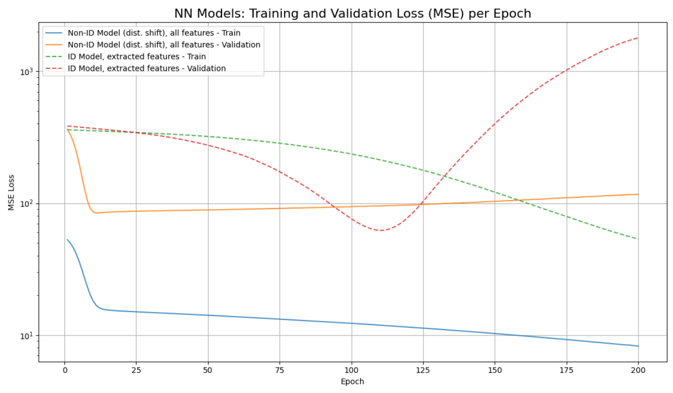

Fig. 2. Training and validation MSE for the feedforward neural network.

```
running_loss += loss.item() * m
count += m
```

III. Model Comparison

The neural network consists of two hidden layers with dimensions $256 \rightarrow 128$, employing ReLU activations, dropout regularization, and the SGD optimizer. The network achieved rapid convergence, reducing validation MSE from approximately 179 to below 70 within the first five epochs. Subsequent epochs yielded marginal improvements with mild oscillations due to stochastic noise. The best generalization occurred between Epochs 5 and 10, where validation MSE stabilized around 68, surpassing the perceptron baseline. Notably, the non-i.i.d. dataset resulted in better overall performance, suggesting that exposure to heterogeneous player data improved model generalization.