Neural Network Full Report

Megan Joseph

May 6, 2025

Abstract

Business owners need to come up with exceedingly novel ways to attract customers. In order to do so, analyzing how customers behave is essential for driving smarter marketing decisions and long-term engagement. This prompted the decision to create a binary classifier that predicts a customer purchase based on factors about the customer. Using a simulated dataset from Kaggle that includes attributes like annual income, number of purchases, and loyalty membership, I built and evaluated multiple neural network architectures using TensorFlow and Keras.

I began by normalizing and exploring the data to understand feature distributions and ensure class balance. I first developed a logistic regression baseline model and iteratively expanded to deep neural networks, overfitting initially to understand the upper bound of performance. I then trained a range of smaller models to find the best validation accuracy, ultimately selecting a neural network with a 512-64-8-1 architecture. Further analysis included feature importance testing by training single-feature models, removing the least important features, and computing Shapley values. These revealed that behavioral features such as time spent on the website and loyalty program status were more informative than demographic ones like gender. This finding may be particularly relevant when advising the startup I support on where to focus data collection efforts. Finally, I evaluated model performance using metrics like precision, recall, F1 score, and ROC-AUC, achieving a maximum AUC of around 0.85.

While the simulated nature of the dataset limits real-world applicability, this project provides a strong foundation for future work using actual client data, with the goal of enabling personalized, data-driven marketing strategies.

Introduction

Customer behavior can often be unpredictable, but as the leader of the analytics team at a startup, it's important to analyze nonetheless. In order to help our clients, we need to assess what influences a purchase and how to better entice the customers to do so. This drew me to the Customer Purchase Behavior Dataset[1] on Kaggle. This dataset contains simulated data on different characteristics of users, like annual income, time spent on the website, and loyalty membership. This is similar to the data we hope to collect on our client's customers.

With this data, I aim to create a binary classifier to predict whether a user will make a purchase or not based on all or a subset of the given features. This will be a stepping stone towards our goal of individualizing customer rewards by predicting what variables will prompt them to make a purchase. I will also be able to determine whether other factors, such as socioeconomic ones, may come into play.

While I had hoped to find data from an actual vendor, it was difficult to find, especially with the constraints. Many of the insights may not be representative of real life and how people actually make purchases, but understanding and testing the coding and math behind classification and neural networks is essential before production. As a result, this analysis will be valid in the knowledge gained.

[1] Rabie El Kharoua. (2024). Predict Customer Purchase Behavior Dataset [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/8725150

Phase 1

In this phase, I loaded the dataset and min-max normalized the data by subtracting each feature by its minimum value and dividing it by the difference between its maximum and minimum value. I chose to do a min-max normalization because the distribution of the data has few outliers and many of the features are roughly uniform. Then, I visualized the distributions of the normalized data.

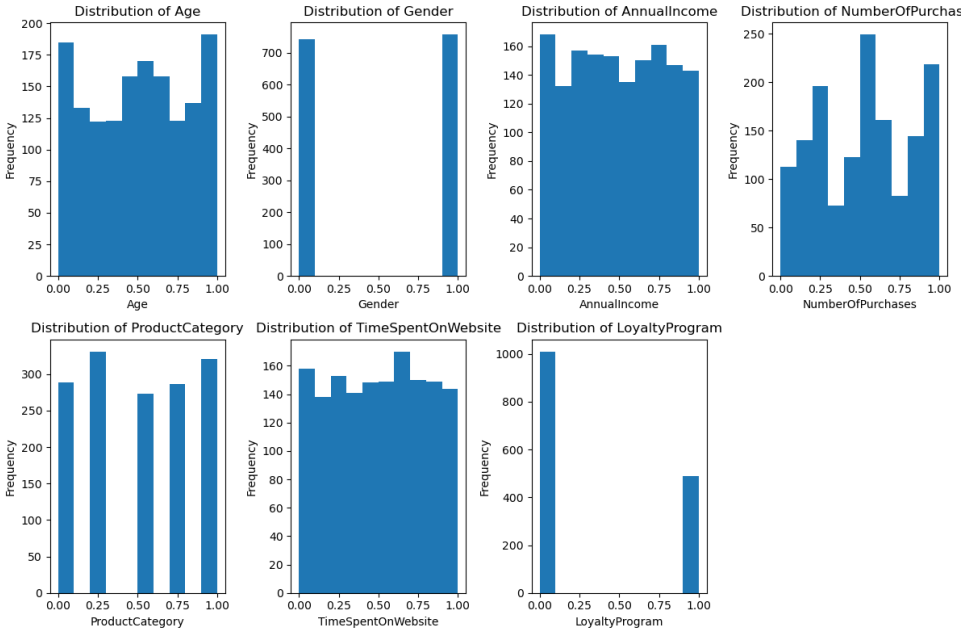


Figure 1: Distribution of features after normalization

After looking at the distributions, I looked at the mean, median, minimum, and maximum for each feature. Some features were binary while the rest were continuous. Something interesting is that the maximum time spent of the website in 59.99 minutes. Finally, I checked the distribution of the target variable to make sure it's not imbalanced. It's about 57% to 43% so it's not imbalanced.

Phase 2

In this phase, I used Tensorflow and Keras to build a neural network. In the beginning, I started off with a logistic regression model. This meant creating a neural network with a single neuron and a single layer with sigmoid as the activation function. The loss function I used was binary cross-entropy which is a standard loss for binary classification. I also used RMSprop as the optimizer which is also a standard. I trained with 128, 256, 512, and 1024 epochs, which are the number of times a model sees the entire dataset, all of which did not increase accuracy much at 0.7608 with 1024 epochs.

Since the goal of this phase is to overfit and find the maximum size of the neural network, I added layers and increased the number of neurons. Each layer, from last to first, has 1, 4, 16, 64, 256, and 1024 neurons respectively. This made a model with 288,153 total parameters. Using 1024 epochs and the same loss function and optimizer, I achieved an accuracy of 0.9997.

Model: "sequential_11"

Layer (type)	Output Shape	Param #
dense_41 (Dense)	(None, 1024)	8,192
dense_42 (Dense)	(None, 256)	262,400
dense_43 (Dense)	(None, 64)	16,448
dense_44 (Dense)	(None, 16)	1,040
dense_45 (Dense)	(None, 4)	68
dense_46 (Dense)	(None, 1)	5

Total params: 288,153 (1.10 MB)

Trainable params: 288,153 (1.10 MB)

Non-trainable params: 0 (0.00 B)

Figure 2: Structure of Overfit Neural Network

Phase 3

The goal of this phase is to train neural networks smaller than the one that overfits and pick the one that has the greatest accuracy on the validation set.

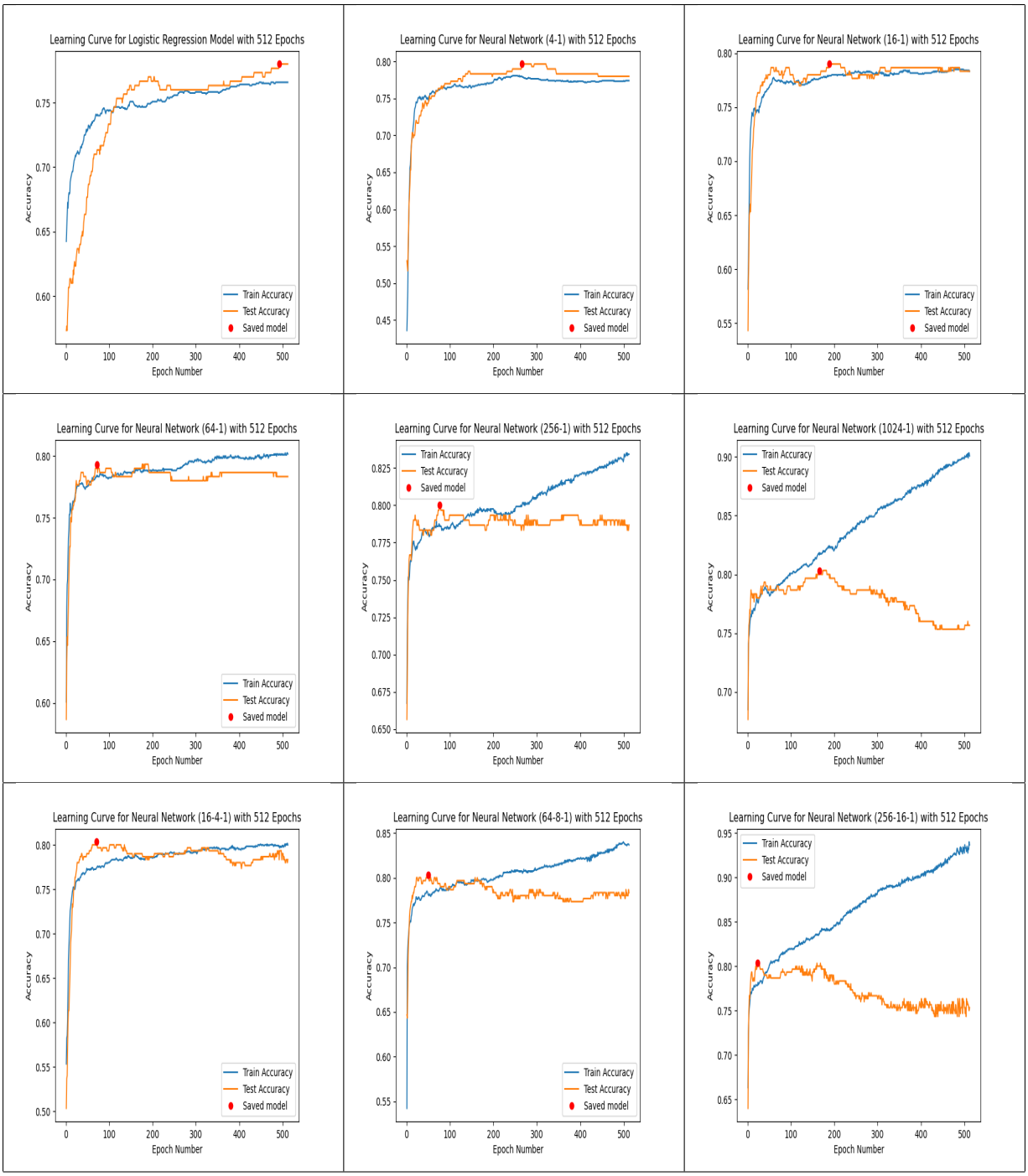
First, I did an 80/20 train test split. This means that 80% of the data was used for training the model and the remaining 20% was used to evaluate the model. I shuffled the indices first before splitting the dataset. Then, I min-max normalized the inputs. After, I created lists for the columns of the table to show the parameters and metrics of each model. The columns are the type of model, the number of epochs, train and test accuracy, train and test loss, precision, recall, F1 score, and the total number of parameters. Loss measures the difference between the predicted and actual values. Precision describes the accuracy of the model over all positive values. It's calculated by using the formula $\frac{TP}{TP+FP}$ where TP is the number of true positive observations and FP is the number of observations classified as positive by the model but are actually negative. Recall describes the model's ability to predict true when given true examples. It's calculated by using the formula $\frac{TP}{TP+FN}$ where FN is the number of observations that the model predicted as negative but are actually positive. F1 score is the harmonic mean of precision and recall and is used mainly when precision and recall are equally important. It's calculated by using the formula $2 \cdot \frac{precision \cdot recall}{precision+recall}$

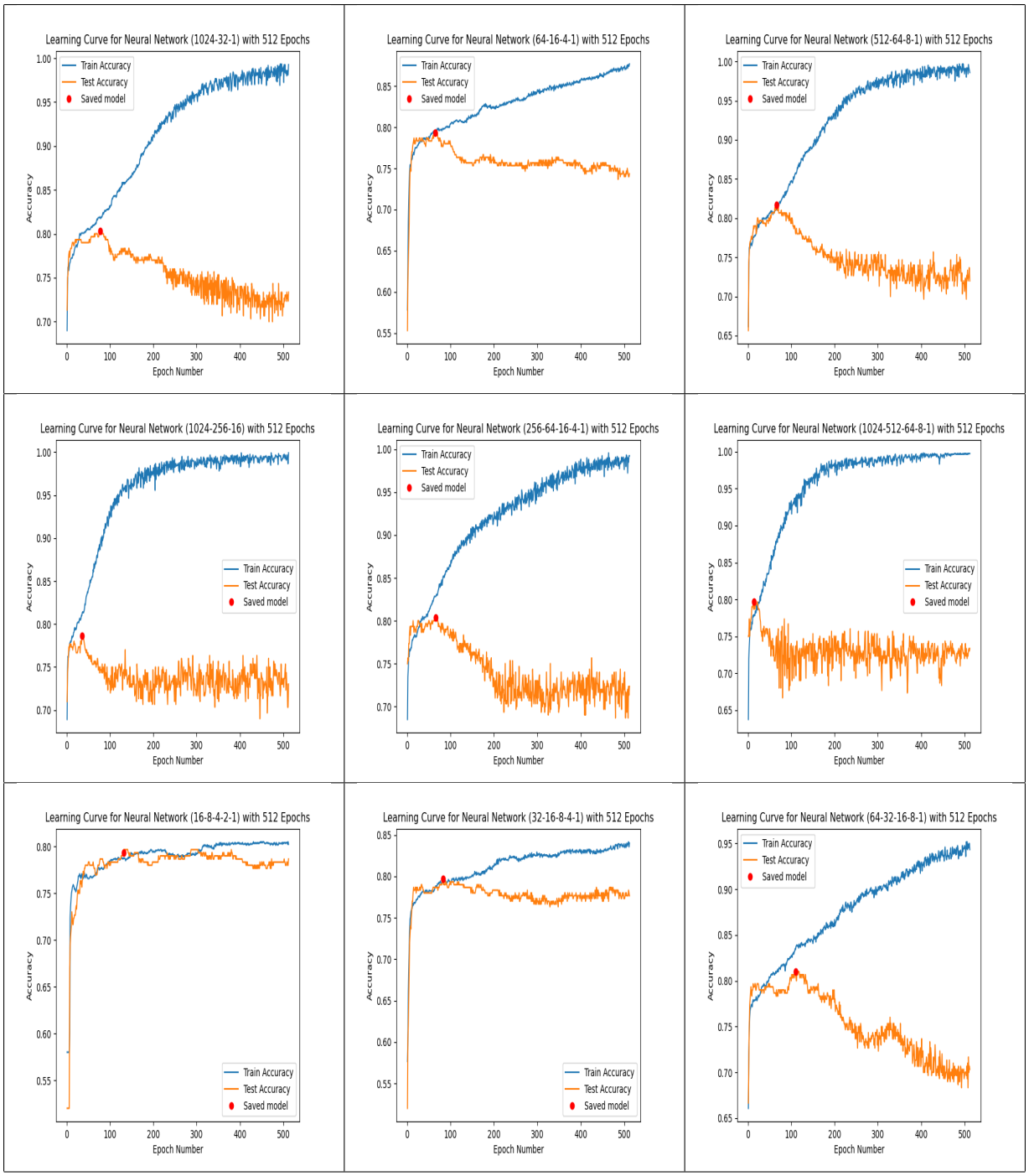
I started out with a random baseline classifier which classifies 50/50 1 or 0. Then, I made functions to make the neural networks part more clean. One function, create_model() creates the neural network model. It incorporated model checkpointing by saving the model with the best test accuracy. The next function draws the learning curve with the x-axis as the number of epochs and the y-axis as the test accuracy since that is what we're optimizing. The last function adds the remaining metrics to the lists. The last two functions required me to check my files for the epoch of the best model and the file path.

The neural network models I made had between 1 and 5 layers and neurons up to 1024. The following table shows the metrics for each model. The learning curves for each model are also shown below.

	Model	Number of Epochs	Train Accuracy	Train Loss	Test Accuracy	Test Loss	Precision	Recall	F1 Score	Total Parameters
0	random baseline classifier	--	0.500000	17.845270	0.500000	16.693915	0.50	0.44	0.47	--
1	Logistic Regression Model	512	0.765833	0.497768	0.780000	0.494166	0.81	0.71	0.76	8
2	Neural Network (4-1)	512	0.779167	0.477286	0.793333	0.481558	0.82	0.74	0.78	37
3	Neural Network (16-1)	512	0.779167	0.467139	0.790000	0.476660	0.80	0.76	0.78	145
4	Neural Network (64-1)	512	0.784167	0.464641	0.786667	0.477044	0.82	0.74	0.78	577
5	Neural Network (256-1)	512	0.786667	0.447799	0.800000	0.476504	0.81	0.76	0.78	2305
6	Neural Network (1024-1)	512	0.818333	0.394840	0.803333	0.494854	0.82	0.75	0.78	9217
7	Neural Network (16-4-1)	512	0.775000	0.483025	0.800000	0.484524	0.82	0.75	0.78	201
8	Neural Network (64-8-1)	512	0.780833	0.465341	0.800000	0.478991	0.81	0.76	0.78	1041
9	Neural Network (256-16-1)	512	0.781667	0.459659	0.803333	0.478630	0.81	0.76	0.78	6177
10	Neural Network (1024-32-1)	512	0.819167	0.388758	0.800000	0.501092	0.81	0.77	0.79	41025
11	Neural Network (64-16-4-1)	512	0.795833	0.440569	0.793333	0.482842	0.81	0.74	0.77	1625
12	Neural Network (512-64-8-1)	512	0.810833	0.394495	0.816667	0.501285	0.82	0.78	0.80	37457
13	Neural Network (1024-256-16)	512	0.814167	0.398530	0.783333	0.524125	0.81	0.73	0.77	274721
14	Neural Network (256-64-16-4-1)	512	0.829167	0.383657	0.800000	0.506182	0.80	0.79	0.79	19609
15	Neural Network (1024-512-64-8-1)	512	0.778333	0.466138	0.790000	0.491132	0.82	0.74	0.78	566353
16	Neural Network (16-8-4-2-1)	512	0.787500	0.459712	0.796667	0.475381	0.81	0.75	0.78	313
17	Neural Network (64-32-16-8-1)	512	0.837500	0.384278	0.803333	0.505955	0.82	0.78	0.80	3265
18	Neural Network (128-64-32-16-1)	512	0.812500	0.416376	0.796667	0.494844	0.81	0.76	0.78	11905
19	Neural Network (256-128-64-32-1)	512	0.777500	0.468310	0.796667	0.486753	0.81	0.76	0.78	45313
20	Neural Network (32-16-8-4-1)	512	0.793333	0.445318	0.790000	0.481553	0.79	0.78	0.78	961

Figure 3: Table of metrics for each model


Learning Curve for Logistic Regression Model with 512 Epochs
Learning Curve for Neural Network (4-1) with 512 Epochs
Learning Curve for Neural Network (16-1) with 512 Epochs
Learning Curve for Neural Network (64-1) with 512 Epochs
Learning Curve for Neural Network (256-1) with 512 Epochs
Learning Curve for Neural Network (1024-1) with 512 Epochs
Learning Curve for Neural Network (16-4-1) with 512 Epochs
Learning Curve for Neural Network (64-8-1) with 512 Epochs
Learning Curve for Neural Network (256-16-1) with 512 Epochs
Train Accuracy
Test Accuracy
Saved model
Accuracy
Epoch Number


Learning Curve for Neural Network (1024-32-1) with 512 Epochs
Learning Curve for Neural Network (64-16-4-1) with 512 Epochs
Learning Curve for Neural Network (512-64-8-1) with 512 Epochs
Learning Curve for Neural Network (1024-256-16) with 512 Epochs
Learning Curve for Neural Network (256-64-16-4-1) with 512 Epochs
Learning Curve for Neural Network (1024-512-64-8-1) with 512 Epochs
Learning Curve for Neural Network (16-8-4-2-1) with 512 Epochs
Learning Curve for Neural Network (32-16-8-4-1) with 512 Epochs
Learning Curve for Neural Network (64-32-16-8-1) with 512 Epochs
Train Accuracy
Test Accuracy
Saved model
Accuracy
Epoch Number

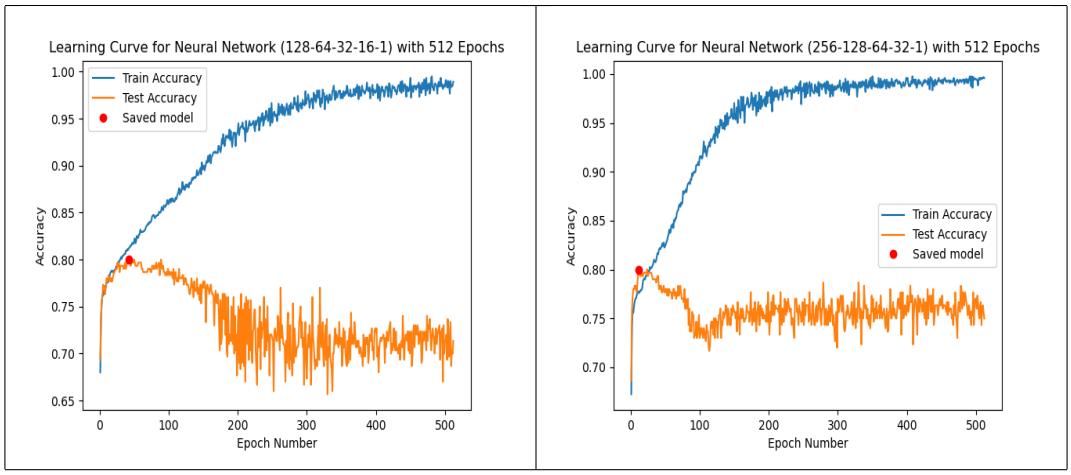


Table 1: Learning curves for all models trained.

The learning curves plotted both train and test accuracy on the y-axis and epoch number on the x-axis. A red dot shows the model saved from checkpointing. The purpose is to track how the model learns with each training iteration. In most architectures, particularly the larger ones, the accuracies diverged early on. Train accuracy continued to increase while test accuracy either plateaued or even decreased. This gap in accuracies is a sign of model overfitting where it starts to memorize the train set rather than learning the patterns and generalizing. Model checkpointing was crucial here since it did not take the overfitted model at the end of training, but the one with the best validation accuracy. On the other hand, a possible sign of underfitting would be an increasing training accuracy that may end with a plateau with a consistently low test accuracy. This shows that the model may be too simple or not training sufficiently enough.

I noticed that no matter what I changed, the test accuracy would stay at around 0.8. I believe this may be due to the data being simulated and not from a real source. As a result, I ended up picking the Neural Network (512-64-8-1) as the best model since it scored the highest on the validation set at around 0.82. It had 37,457 total parameters which is much smaller than the overfit model. For all of the models, precision was higher than recall and the F1 scores are comparable.

The architecture required to overfit when the output is also a feature is a simple logistic regression model. It does not require too many epochs to overfit.

The functions to represent the model and serve as the prediction are below:

```
# represent model
def model(file_path):
    return keras.models.load_model(file_path)

# prediction model
def predictions(model, X_test):
    num_layers = 4
    curr_input = X_test
    for i in range(num_layers):
        weights, bias = model.layers[i].get_weights()
        if i == (num_layers - 1):
            output = 1 / (1 + np.exp(-((curr_input @ weights) + bias)))
            curr_input = output
        else:
            output = np.maximum(0, (curr_input @ weights) + bias)
            curr_input = output

    curr_input[curr_input >= 0.5] = 1
    curr_input[curr_input < 0.5] = 0

    return curr_input
```

Listing 1: Function to represent the model and function to serve as the prediction model.

The prediction function manually computes the forward pass using the weights and biases from the best trained model. For each layer, it extracts the corresponding weights and biases, computes the output using either the ReLU or Sigmoid activation function, and maps the predictions to 1 or 0 based on the 0.5 threshold. The goal was to better understand the inner workings of the prediction process and verify that the Keras predict() method behaves as expected.

To validate the correctness of the custom predictions() function, I compared its outputs to those generated by the built-in predict() function given the same input. After applying the same thresholding procedure (values $\geq$ 0.5 mapped to class 1, and ¡ 0.5 to class 0), I confirmed that the two sets of predictions were identical:

```
pred = predictions(best_model, X_test_norm)
trained_pred = best_model.predict(X_test_norm).flatten()
trained_pred[trained_pred >= 0.5] = 1
trained_pred[trained_pred < 0.5] = 0

(pred[0] == trained_pred).unique() # Output: array([ True])
```

Listing 2: Validating correctness of custom predictions function.

pred[0] is a Pandas Series containing the predictions from the custom function.

Phase 4

The goal of this phase is to study feature importance, specifically which features give the model the most information. Reducing the number of features simplifies the model, improves interpretability, reduces risk of overfitting, and is cheaper computationally. Simpler models may sacrifice accuracy but are usually easier to understand and debug while also generalizing better to unseen data. Increased interpretability makes it easier to see how the model makes decisions which is useful when presenting to a nontechnical audience. Fewer inputs can lead to faster training which is useful if resources are constrained.

The features of my chosen dataset are Age, Gender, AnnualIncome, NumberOfPurchases, ProductCategory, TimeSpentOnWebsite, and LoyaltyProgram. First, I created seven models trained on a single feature and plotted the validation accuracies of each. The time spent on the website had the highest validation accuracy while gender and product category had the smallest.

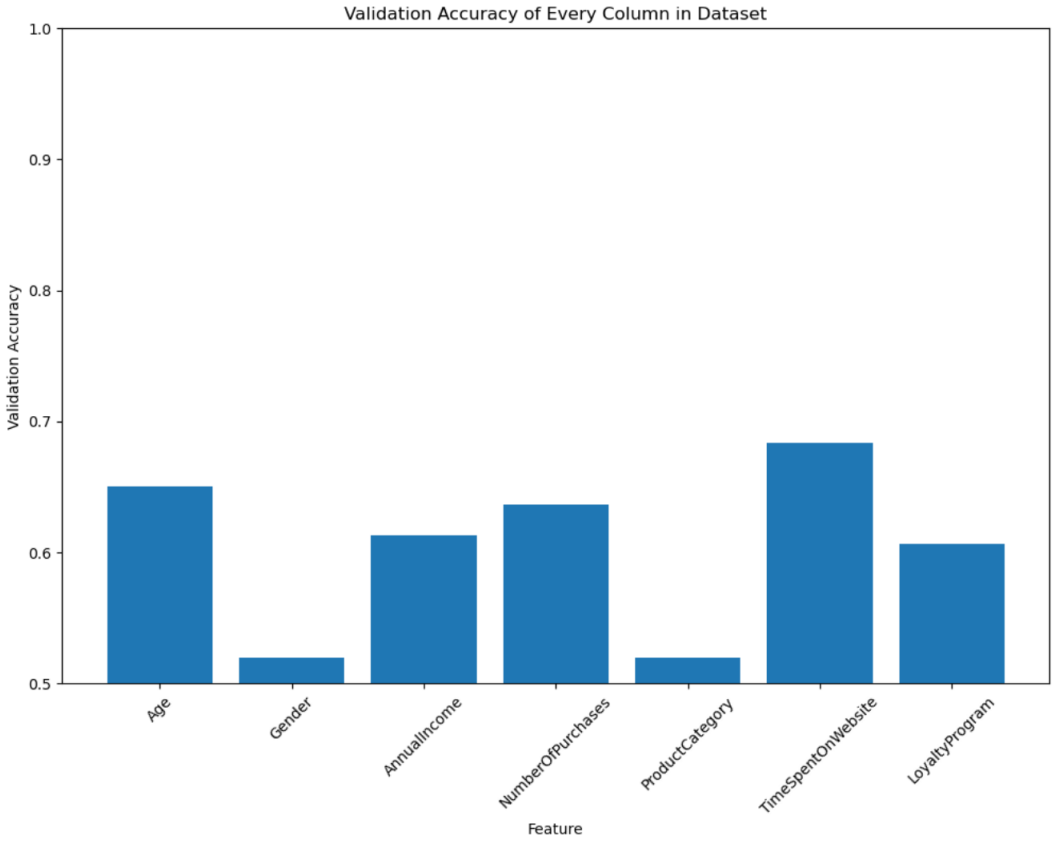


Figure 4: Validation accuracies for models trained on a single feature in the dataset

Next, I started removing the least important features (the features where the respective model had the lowest validation accuracy) from the full model and found the validation accuracy. I did this until the model only had the feature that gave the highest accuracy. In this case, I removed Gender, then ProductCategory, and so on until I was left with only TimeSpentOnWebsite. Then, I plotted the validation accuracies for each. The x-axis labels were too long, so I put them in a legend to make the graph easier to read. The model with the highest validation accuracy was the one where only gender and product category were removed.

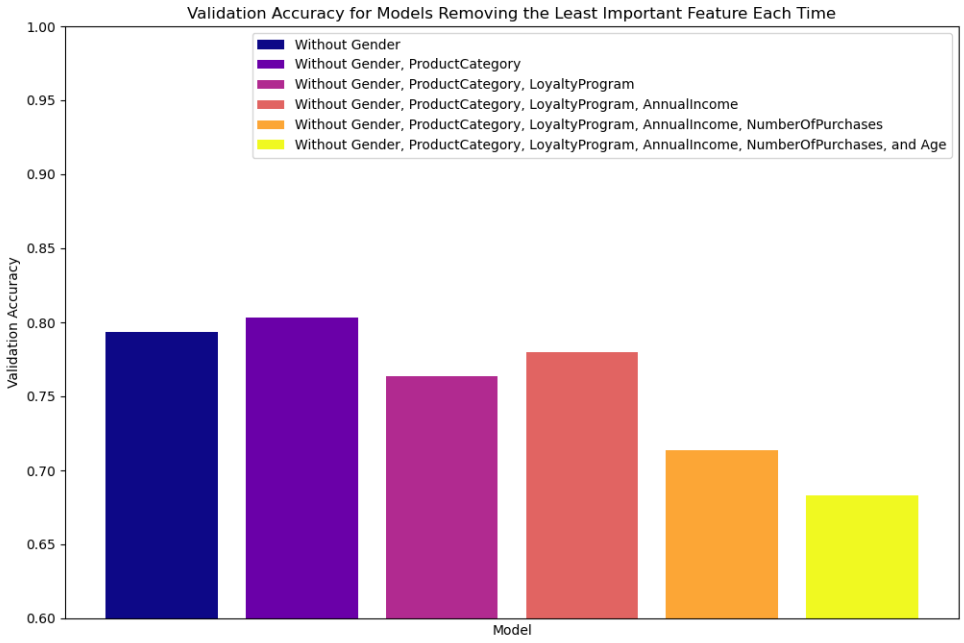


Figure 5: Validation accuracies for models trained on all features and removing the least important feature each time

After this, I compared the accuracies of the feature-reduced model with the best validation accuracy to the best model with full features. The feature-reduced model has an accuracy of 0.803 while the full-feature model has an accuracy of 0.817. Although the full-feature model had slightly better validation accuracy than the feature-reduced model, this improvement comes at the cost of increased model complexity. More features may increase the chance of capturing useful patterns, but they also introduce a higher risk of overfitting, greater computational cost, and reduced interpretability. In contrast, the reduced-feature model is simpler and easier to interpret, making it a better choice in contexts where model transparency or resource efficiency is important. This reflects a common trade-off in machine learning between maximizing accuracy and maintaining model simplicity.

The decrease in accuracy may be due to a few reasons. There may be some nonlinear interactions that are not captured by the methods used. A more informative method would be subset selection to determine which subset of features would produce the highest accuracy. Additionally, all features may contribute even a little to prediction, especially since there are only seven features to begin with. Lastly, the drop could be due to stochastic variation in training.

To investigate feature importance further, I found the Shapley value, which states how much each feature contributed to a prediction. I chose to do this instead of Local Interpretable Model-agnostic Explanations (LIME) because I didn't have too many features, I didn't need a prediction model at the end, and just wanted to investigate the strengths of each feature. I just did a single iteration because of a lack of compute power, but I would have averaged over the entire dataset. The Shapley value I calculated is around 0.79. The column that is most important is AnnualIncome and the least important column is Age. Below is the code used to find the Shapley value:

```
def shapley_value(best_model, X_train):
    instance = X_train.iloc[[1], :]
    train_pred = best_model.predict(X_train).flatten()
    base_pred = np.mean(train_pred)


```

```
    cols = range(len(X_train.columns))
    subsets = [[]]
    for col in cols:
        subsets += [sub + [col] for sub in subsets]

    predictions = {}
    for sub in subsets:
        mod_instance = instance.copy()
        for i in cols:
            if i not in sub:
                mod_instance.iloc[0, i] = 0

        pred = best_model.predict(mod_instance).flatten()
        predictions[tuple(sub)] = pred

    avg_marg_contr = {}
    for col in cols:
        total_contr = 0
        num = 0
        for sub in subsets:
            if col not in sub:
                pred_wo_feature = predictions[tuple(sub)]
                sub_w_feature = sub + [col]
                pred_w_feature = predictions[tuple(sorted(sub_w_feature))]
                marg_contr = pred_w_feature - pred_wo_feature
                total_contr += marg_contr
                num += 1

        avg_contr = total_contr / num
        avg_marg_contr[col] = avg_contr

    final_pred = base_pred
    for col in cols:
        final_pred += avg_marg_contr[col]

    return final_pred, avg_marg_contr
```

Listing 3: Function to calculate the Shapley value for the model and determine the order of importance based on marginal contribution of each feature.

Phase 5

In this phase, I drew the receiver operating characteristic (ROC) curve for the best model and found the area under the curve (AUC). The ROC curve is a popular way to evaluate classifiers. It plots the true positive rate (TPR) against the false positive rate (FPR) at various thresholds. These thresholds are defined by the unique values predicted by the classifier. The AUC measures how well the classifier separates classes. The possible values are from 0 to 1 with 1 being the perfect classifier.

I calculated the TPR and FPR from scratch and plotted them against each other. The following is ROC curve.

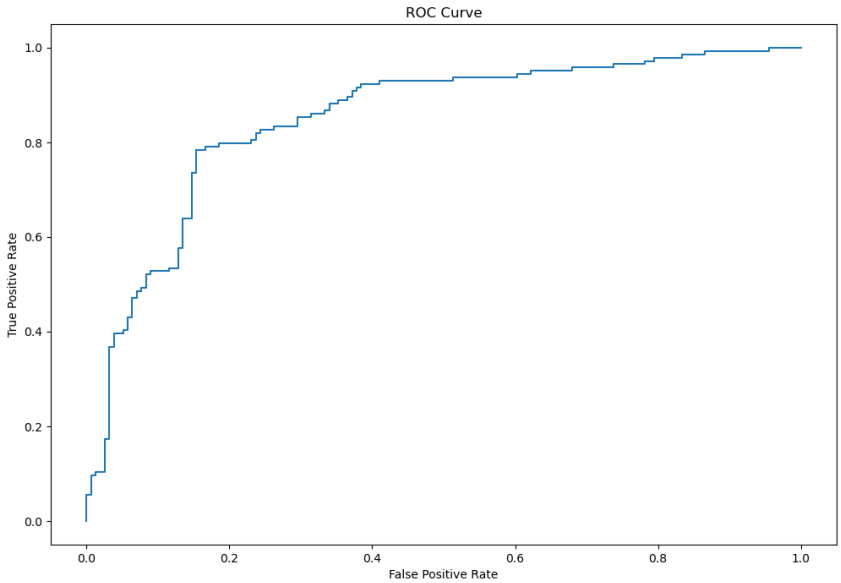


Figure 6: ROC curve for best model

I used the function np.trapz() to find the AUC value and it gave me around 0.85.

Conclusion

The goal for this project was to build a neural-network based binary classifier to predict whether a customer would make a purchase based on all or a portion of the features. Using a simulated dataset in place of real-world data, I explored various model architectures and evaluation metrics to determine the most effective predictive model.

The full-feature neural network with architecture 512-64-8-1 performed the best in terms of validation accuracy, achieving around 82%. This model balanced complexity and generalization, while avoiding over-fitting. Across all models, precision was consistently higher than recall, suggesting that they were more effective at correctly identifying customers likely to make a purchase. This behavior can be important in real-world applications where minimizing false positives is preferred.

This project also delved into feature importance analysis. I trained models on individual features, removed features one at a time, and calculated Shapley values to understand the contribution of each variable. Features like time spent on the website and loyalty membership were more significant predictors than others like age or gender. These insights suggest that behavioral engagement data may offer more predictive power than demographic information alone.

The simulated nature of the dataset is a limitation since real-world customer behavior can be more complex and noisy. However, this analysis helped establish a framework for building, interpreting, and evaluating predictive models. In the future, I plan to extend this approach to real customer data from the startup to generate more meaningful insights. Additionally, I aim to incorporate new features to enhance predictive performance and deepen the understanding of customer behavior.