Predicting whether Visitors Purchase after only one Click

Ayaan Gopalan · Tori Wang · Xi Wu · Ethan Young

Team Overfitters

Project Objective

Goal

Predict if a visitor to the Fingerhut site will make a purchase **using only the data from their first server call**

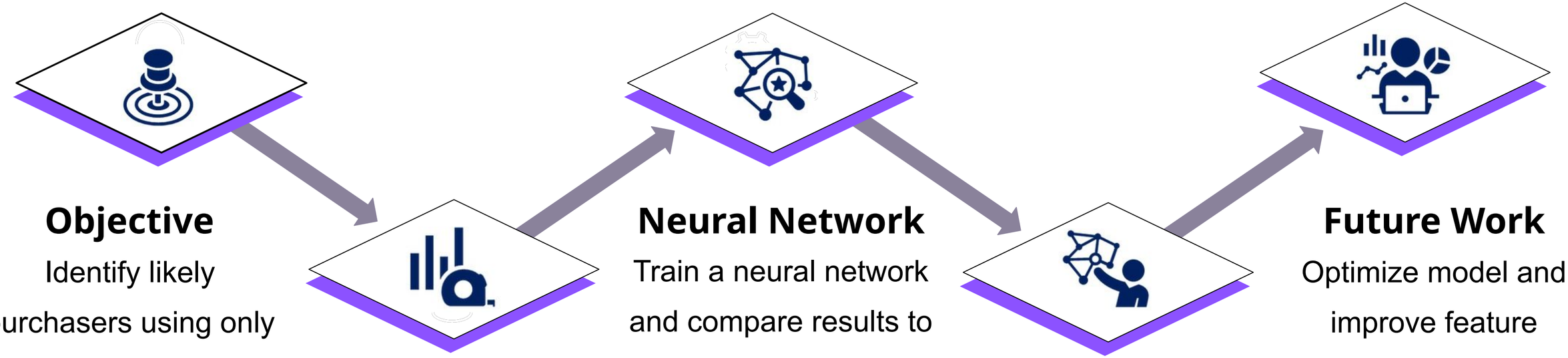
Challenge

Over 95% of visits **do not result in a purchase**, which presents difficulties in creating an accurate model

Application

Immediately generate predictions about **every visitor** to the Fingerhut site and tailor different marketing strategies for each one

Project Roadmap



Objective

Identify likely purchasers using only their first server call

Feature Selection

3-step procedure to select important features and narrow down large dataset

Neural Network

Train a neural network and compare results to baseline

Results

Different marketing strategies depending on the prediction

Future Work

Optimize model and improve feature selection to increase accuracy

Feature Selection (1/2)

Extracted the **first server calls** from all visits in the original dataset, then employed a **3-step feature selection** process to narrow down the variables:

1

Variables with missing values

Variables with at least 95% missing values were removed

2

Variance Threshold

Variables where at least 99% of the values were similar were removed

3

Random Forest Feature Importance

Trained a random forest classifier and extracted the top 20 variables in terms of feature importance

* More details about feature selection process are in the Technical Details slides

Feature Selection (2/2)

Final Data Set

The final subsetted data set consisted of **20 variables** from all of the first server calls, and a **binary variable** denoting if a visit resulted in a purchase

Random Forest Feature Importance: Top 5 Variables

	FEATURE	IMPORTANCE
IP address	ip	0.236056
Last Member ID value	post_evar23	0.195540
Pipeline Session ID	prop34	0.102365
List of events triggered	eventlist	0.085137
Last platform value	post_evar13	0.080618

Models

We implemented 2 models using the dataset with the 20 most important variables:

- **Logistic Regression (Baseline)**
  - Trained with class weight balances
  - Fast computation, but struggles with imbalanced data
- **Shallow Neural Network**
  - Trained using undersampled data
  - More flexible model that performs comparatively better

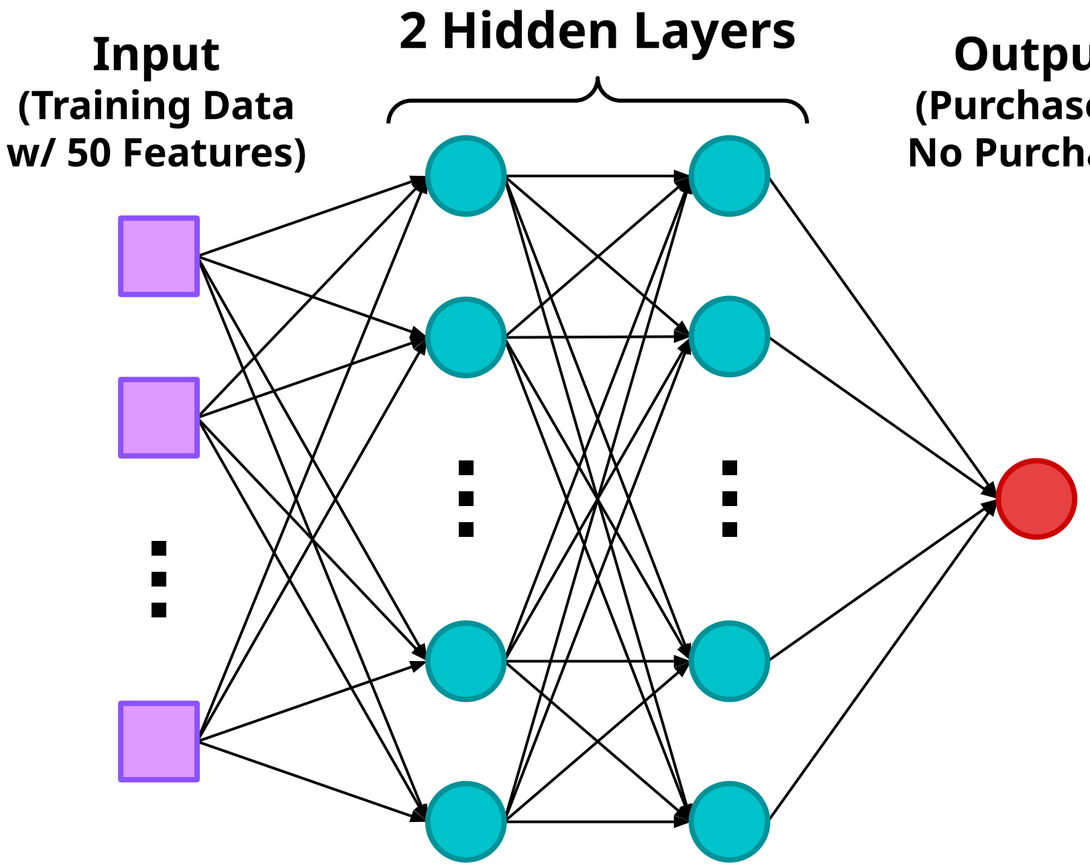


Diagram of the neural network
(A detailed version is in the Technical Details slides)

Applications of Results

Plug in data collected in real time to pre-trained model to make predictions

- ❏ Possible strategy based on if a user will:
  - **Purchase**: optimize search results and offer promotional codes based on subsequent page visits
  - **Not purchase**: show personalized homepage with targeted item recommendations

Retrain neural network seasonally using data from the previous year

- ❏ Opportunity to understand how customer behavior can vary throughout the year
- ❏ Using a larger dataset helps train a more robust model that predicts more consistently and with more confidence

* Details of model performance are in the Technical Details slides

Limitations & Future Work

Data

- ❏ Only 1 week of data limits applicability of results
- ❏ Features selected are predictive, but may not always be of practical interest

- ❖ Incorporate 2nd, 3rd, etc server calls to utilize more data as visitors spend more time on the site

Models

- ❏ Methods addressing class (purchase / no purchase) imbalance introduce bias
- ❏ Neural networks are flexible, but lack interpretability

- ❖ Optimize neural network to improve model accuracy and confidence of predictions

Summary

Project Objective Revisited

- ➔ Can we predict which users will make a purchase using only their first server call?
- ➔ Key challenge of most users not making a purchase

Feature Selection and Models

- ➔ 3-step feature selection and addressed the data imbalance
- ➔ Results suggest variables in the first server call contains strong insights about the likelihood of purchase

Sales and Advertising Strategy

- ➔ Real time prediction to more accurately target different types of visitors with different strategies
- ➔ Opportunity to retrain model with seasonal data

Thank you!

Technical Details

Feature Selection (1/4)

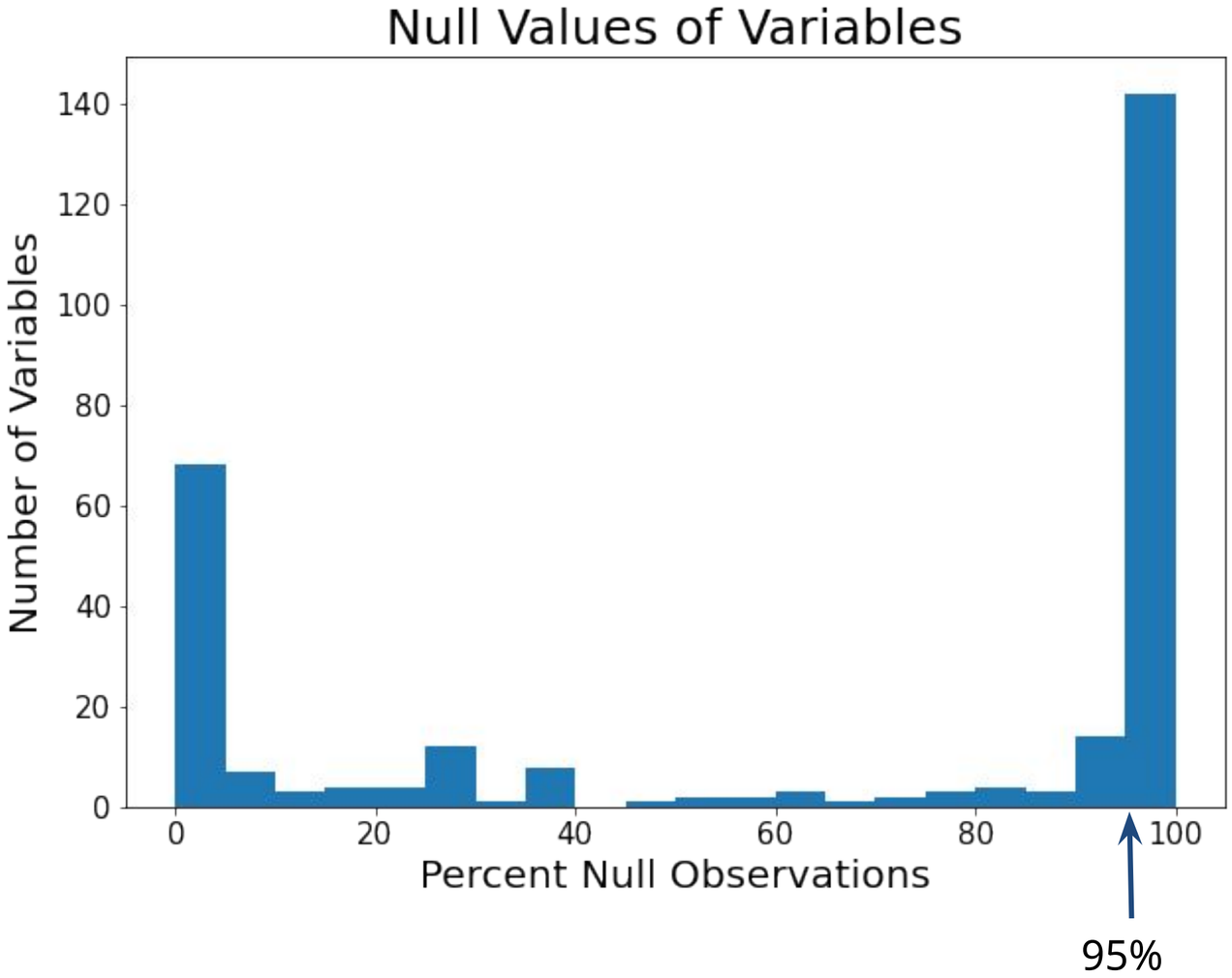
Variables with Missing Values

Because very few visits result in a purchase, we wanted to select variables that **returned non-null values in the purchase visit first server calls**

Step 1: Calculated the proportion of null values in the first server calls of purchase visits

Step 2: Graphed the distribution of null values in all the variables in a histogram

Step 3: Examining the histogram, we selected 95% or greater null values as the cutoff for values to remove



Feature Selection (2/4)

Variance Threshold

We wanted to **remove variables that return the same value almost every time**, as they are likely not as predictive of visit outcomes

Step 1: After removing variables from last step, converted null values in the remaining variables into their own category

- Fixes any missing values issues, and could unlock useful information in variables

Step 2: Calculated the variance of these variables

Step 3: Removed all variables with a variance of 0.01 or lower (99% or more of the values are similar)

Removed Variables

brandcode
homepage
excludehit
postcookies
postjavaenabled
postpersistentcookie
javascript
visitpagenum
prevpage
duplicatepurchase
newvisit
clicksourceid

Feature Selection (3/4)

Random Forest Classifier

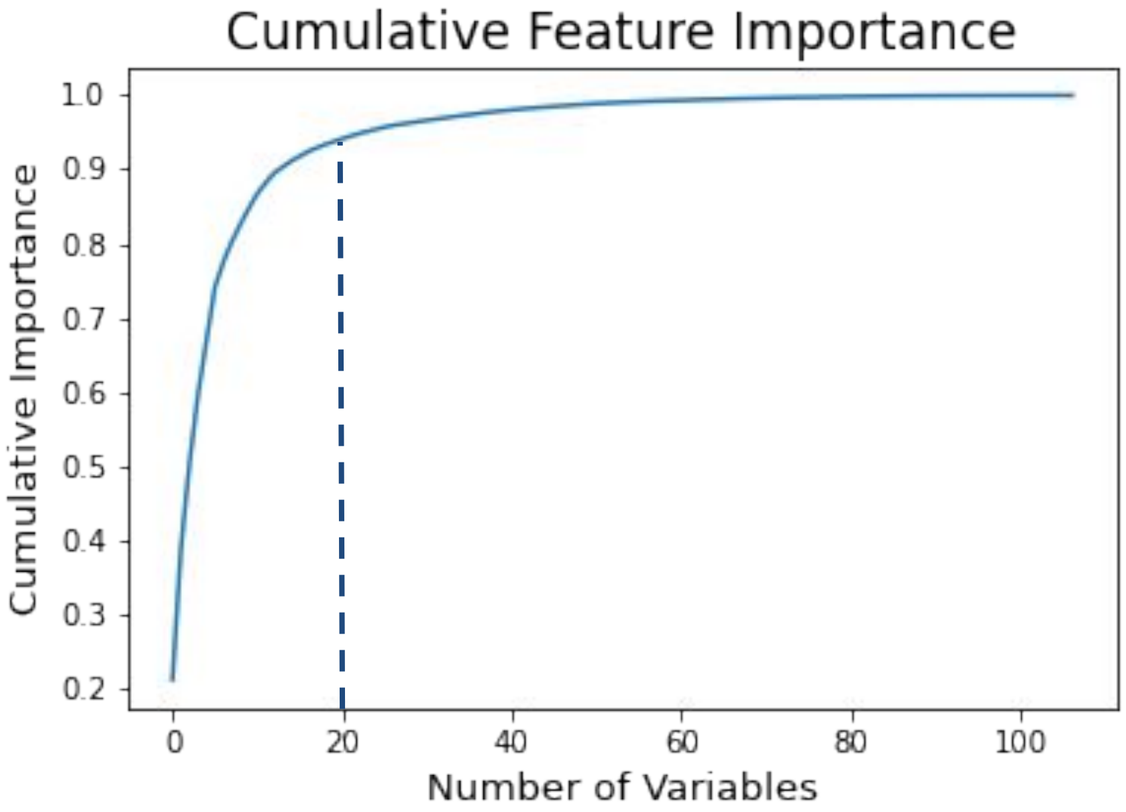
After subsetting based on missing values and variance, we wanted to further subset variables **based on their predictivity of visit outcomes**

Step 1: Trained a Random Forest Classifier on a undersampled data set of remaining variables

- Scaled the numerical variables between 0 and 1
- Encoded the categorical variables with target encoding

Step 2: Graphed the cumulative feature importance of the variables

Step 3: Examining the graph, selected the top 20 variables which captured about 94% of the feature importance



Feature Selection (4/4)

Final Data Set

FEATURE	IMPORTANCE
ip	0.177563
post_evar23	0.109907
prop34	0.109703
eventlist	0.095082
post_evar13	0.092875
crosssoldproduct	0.087115
evar23	0.061504
zip	0.036025
warrantycategory	0.034149
geocity	0.026755

FEATURE	IMPORTANCE
pageurl	0.022166
emailsubscriptionremove	0.018416
post_evar12	0.010415
useragent	0.008990
evar24	0.008817
post_evar30	0.008073
post_evar24	0.007258
pagename	0.007069
post_evar33	0.005333
institutionid	0.005331

Logistic Regression (1/3)

Basics

In (binary) classification, **logistic regression** is a regression technique that outputs a value between 0 and 1 (the input is labeled 0 if the output < 0.5 and 1 if > 0.5)

Mathematically, logistic regression is the logarithm of the odds of an event:

$$\log\left(\frac{p(y=1)}{1-p(y=1)}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_m x_m$$

3 key assumptions:

- Independent observations
- No multicollinearity
- No outliers

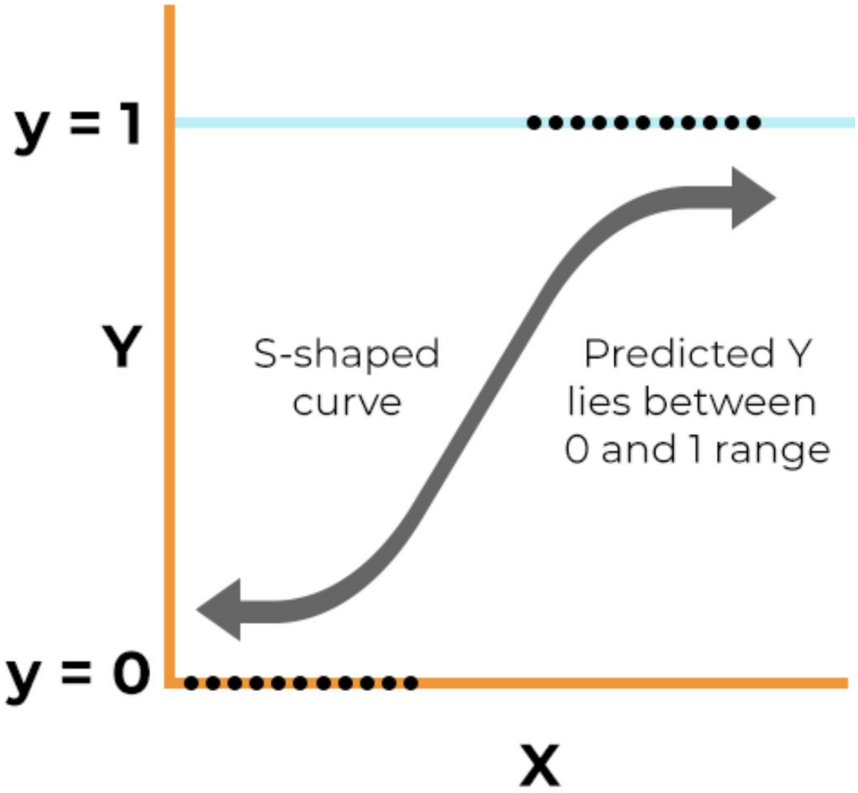


Figure from this site

Logistic Regression (2/3)

Training and Testing Sets

Variables used in this model were the **20 most important features** extracted during feature selection

Variables were then converted to numeric variables using **ordinal encoding** and then scaled, with outliers removed from the scaled dataset

Note: Our training and testing datasets were limited by issues working on computing clusters and the difficulty handling the large dataset on our personal computers

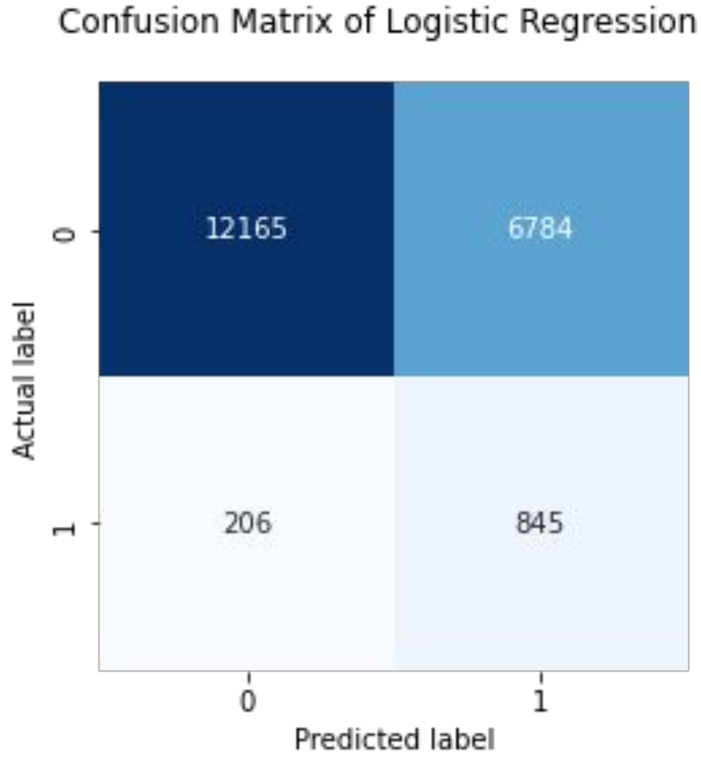
- Size of **training set** (sampled from day 1): **46657**
  - # users that purchase: **2439**
  - # users that do not purchase: **44218**
- Size of **testing set** (sampled from day 6): **20000**
  - # users that purchase: **1051**
  - # users that do not purchase: **18949**

Our model was trained using a dataset that had size: 4878. This is due to the fact that we used class weight balances in order to prevent our model from only predicting 0 or “no purchase’. This does result in some bias in our model.

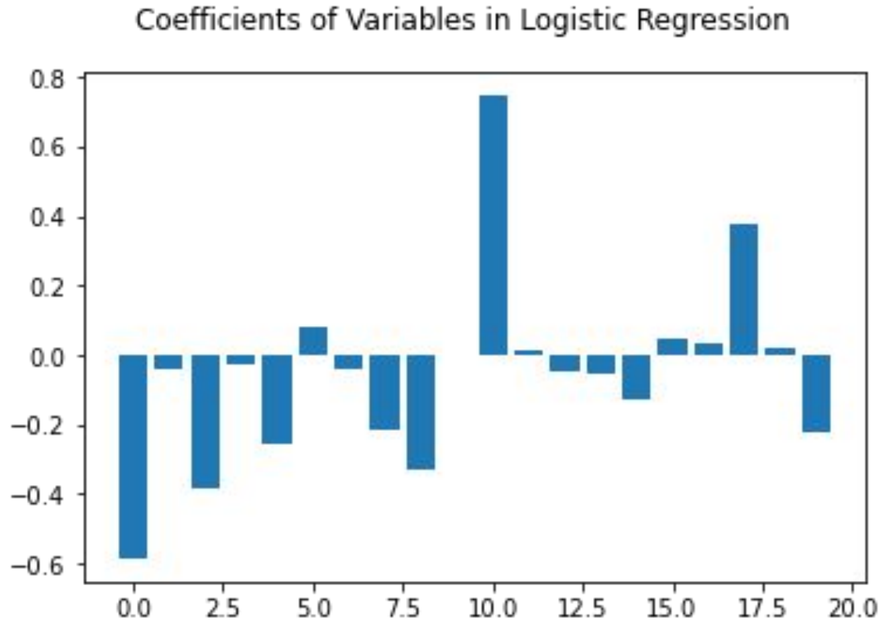
Logistic Regression (3/3)

Results

Test set accuracy rate: 0.6505



Variable Coefficients: (20 variables coded)



Neural Network (1/9)

Basics (1/3)

In (binary) classification, a (feedforward) **neural network** sends data through (at least one) layers of computation (i.e., matrix multiplications) to output a value between 0 and 1 (the input is labeled 0 if the output < 0.5 and 1 if > 0.5)

Conceptually, the **layers** of a neural network extract qualities of the input data $\mathbf{X}$ that allow the model to distinguish between classes

- Each layer is composed of a predetermined # of nodes with connections between them (called **weights**) that have randomly chosen values (initialized before training)
- The weights between layers compose a weight matrix $\mathbf{W}$

The **biases** $\mathbf{B}$ in each layer shift the activation function and influence the output values, not the input data

- The number of biases equals the number of weights in the hidden layers and the output layer (in our case, we have 64+128+1=**193** biases)

Neural Network (2/9)

Basics (2/3)

An **activation function** $f(z)$ (refer to this site) is applied in all layers except the input

- They apply nonlinearity by forcing input values to between 0 and 1

Mathematically, a 2-layer feedforward network can be written as:

$$\hat{y} = f(f(f(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{B}^{(1)})\mathbf{W}^{(2)} + \mathbf{B}^{(2)})\mathbf{W}^{(3)} + \mathbf{B}^{(3)})$$

$\hat{y}$ is the prediction, f the activation function, $\mathbf{X}$ the input data

The **output** $\hat{y}$ can be interpreted as the probability a user will purchase (if > 0.5) or not (if < 0.5)

Backpropagation (refer to this site) is the process of pushing the weights of a neural network toward the values that maximize model accuracy

- This is done with an optimization algorithm (e.g., gradient descent)
- In our case, we use backpropagation with the Adam algorithm (refer to this site)

Neural Network (3/9)

Basics (3/3)

Intuitively, the **hyperparameters** of a neural network are predetermined, adjustable settings that determine how the model performs before data are inputted

- Examples: # layers, # weights in each layer, choice of activation function

To gauge model performance, we use both **accuracy** and 'area under the ROC curve' (**AUC**) (see this site)

- The AUC considers unbalanced class distributions

Dropout is a regularization technique that prevents overfitting (refer to this site)

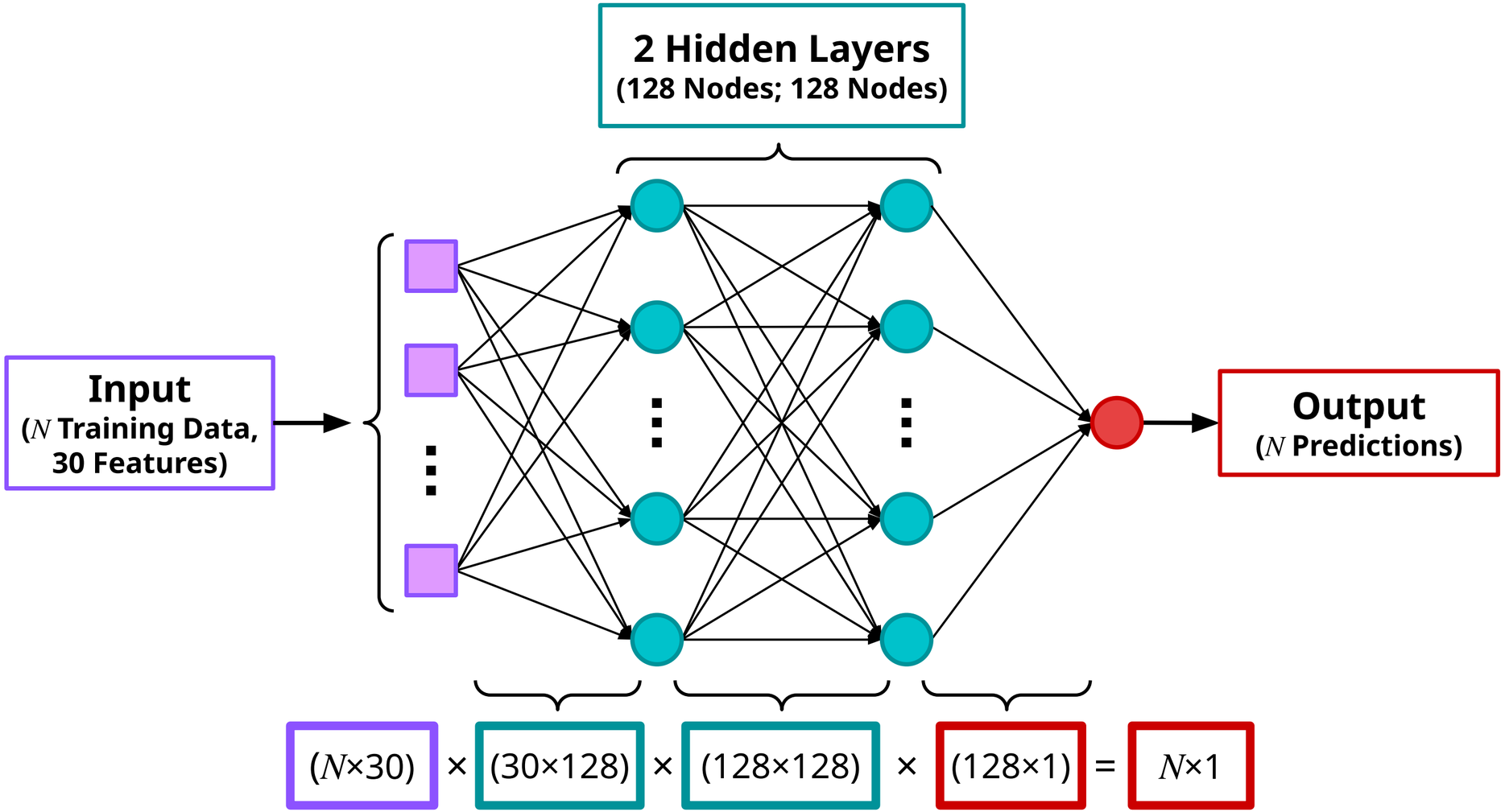
- It does so by ignoring a percentage of the weights during training

The **loss function** is **binary cross-entropy** (refer to this site)

- The model optimizes the weights that minimize this value

Neural Network (4/9)

Current Model Architecture



Neural Network (5/9)

Training and Testing Sets

Feature selection follows from the main presentation (slide 4) and is also explained in the Logistic Regression Technical Details

Instead of class weights, we oversampled users that do not purchase so the data we use slightly resembles the original data

- Size of **training set** (sampled from days 0–5): **34116**
    - # users that purchase: **14058**
    - # users that do not purchase: **20058**

- Size of **testing set** (sampled from day 6): **4908**
    - # users that purchase: **1954**
    - # users that do not purchase: **2954**

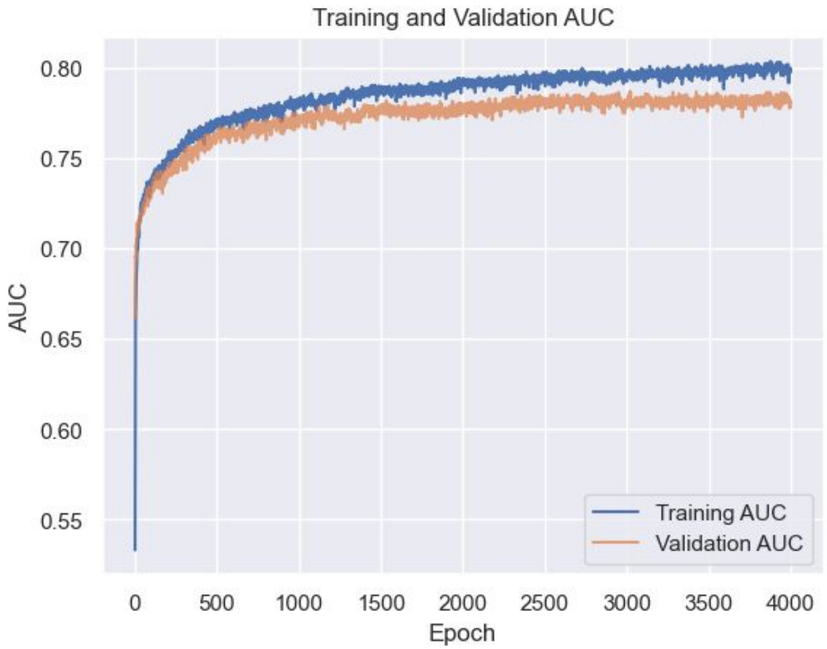
Neural Network (6/9)

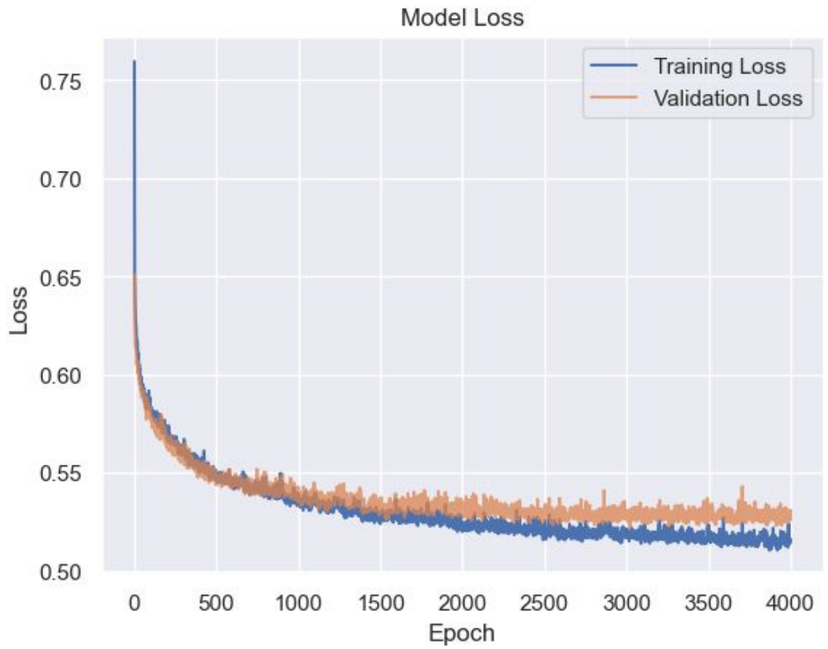
Current Hyperparameters

- # hidden layers: **2**
  - # nodes: **64**, **128**
    - Total # trainable parameters (incl. biases): **9793**
  - Dropout rates: **0.1**, **0.2**
- Batch size: **1024**
- # epochs: **4000**
- Optimizer: **Adam** (**AMSGrad** variant)
  - Step size: **0.001**
- Activation function: **sigmoid**
- Loss function: **binary cross-entropy**

Neural Network (7/9)

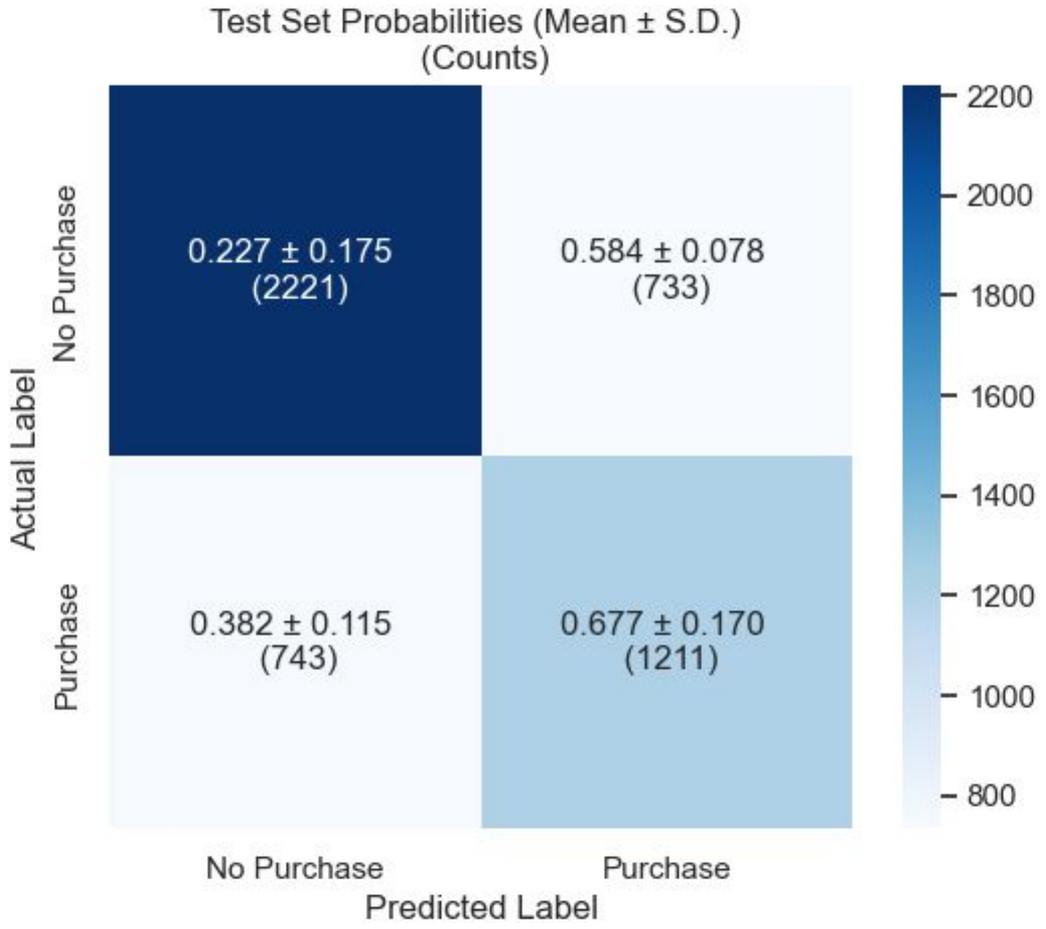
Results (1/2)





Neural Network (8/9)

Results (2/2)



Neural Network (9/9)

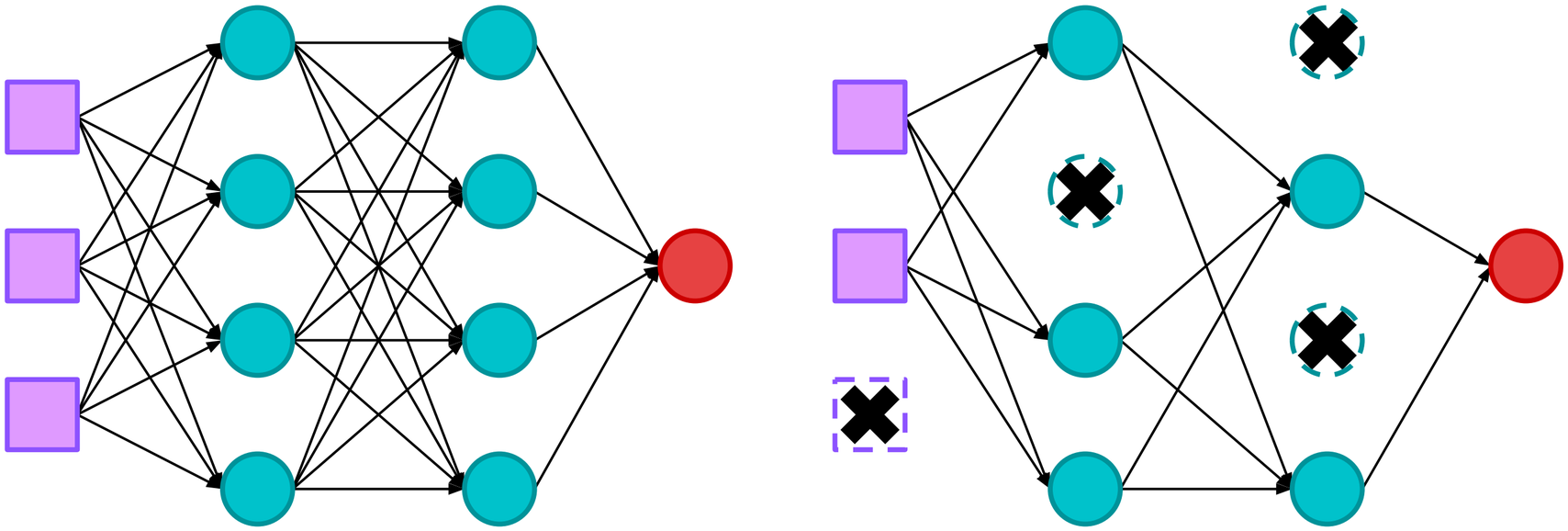
Hyperparameter Optimization

Our future work involves performing hyperparameter tuning using grid search with the *NNI* Python package (refer here for an overview of *NNI*)

Below is a table of the ranges of 8 hyperparameters we want to test

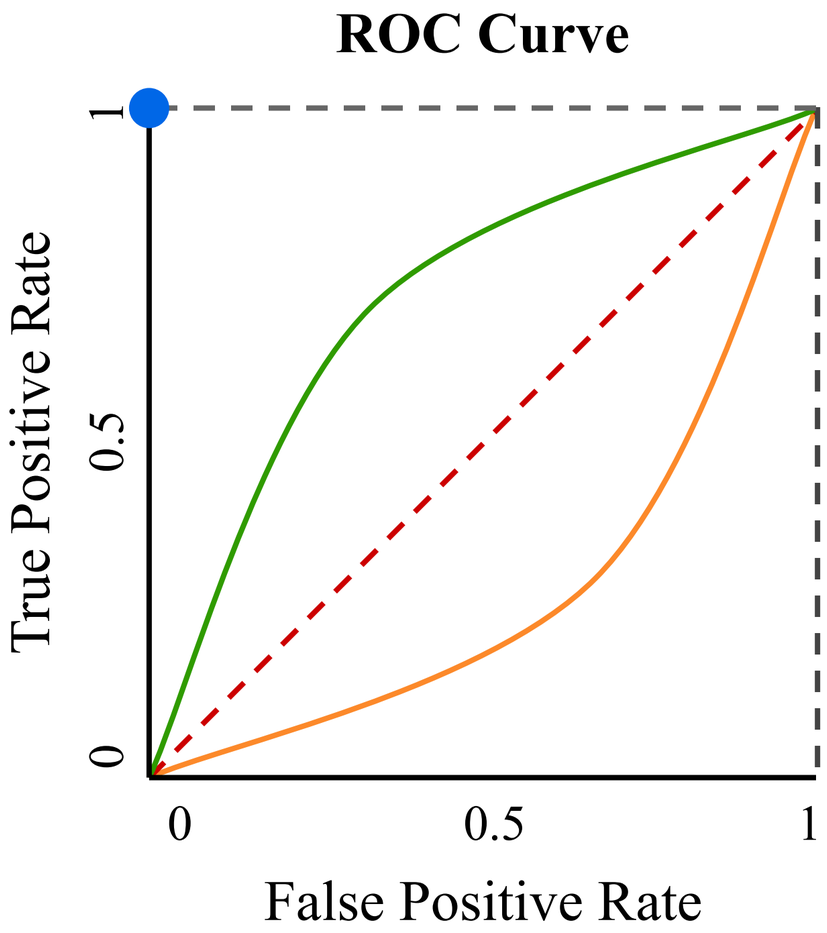
# Epochs	**500 1000 2000 4000**
Batch Size	**32 64 128 256 512 1024**
Step Size	**0.00075 0.001 0.0025**
# Nodes in Layer 1	**32 64 128 256**
Dropout Value in Layer 1	**0.1 0.2 0.3**
# Nodes in Layer 2	**32 64 128 256**
Dropout Value in Layer 2	**0.1 0.2 0.3**
Activation Function	**sigmoid tanh relu**

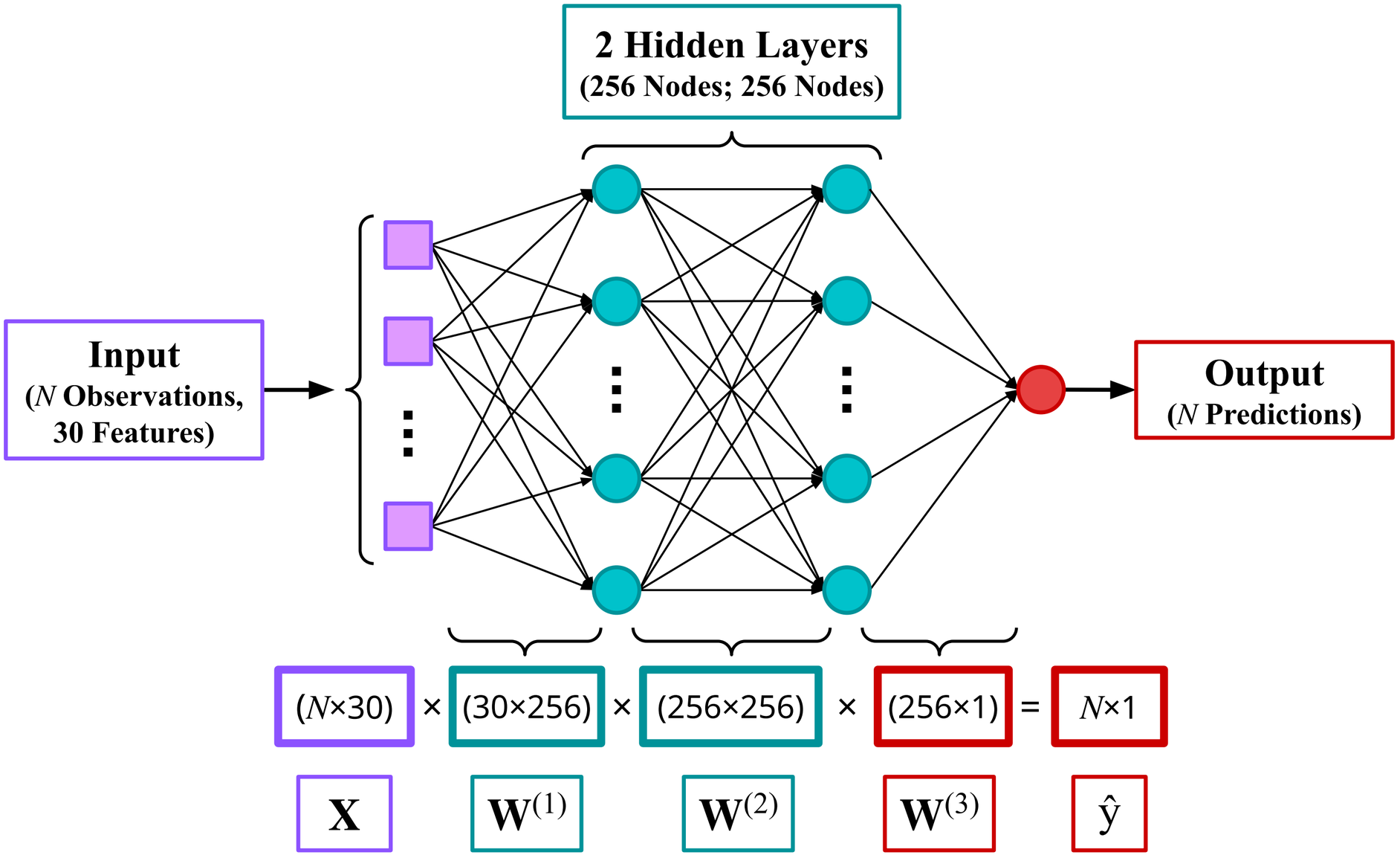
(indicates values used for results in the main presentation)



a) Standard neural network

b) Applying dropout


ROC Curve
True Positive Rate
1
0.5
0
0
0.5
1
False Positive Rate


2 Hidden Layers
(256 Nodes; 256 Nodes)
Input
(N Observations,
30 Features)
Output
(N Predictions)
(N×30) × (30×256) × (256×256) × (256×1) = N×1
X
W(1)
W(2)
W(3)
ŷ