(Individual Assignment II - MGSC-673-075 - Intro to AI & Deep Learning)

**Feed Forward Neural Network Architecture Experimentation**

**For Churn Dataset**

*Name: Arham Anwar*

*ID: 261137773*

**Section 1: Introduction**

This project presents an experimental study aimed at applying various deep learning techniques learned in class using PyTorch. The introduction section discusses the data source, cleansing steps, and the train split strategy, along with the rationale behind them. In Section 2, we delve into the experiments conducted, with emphasis on increasing performance. The philosophy followed in the experimental phase mirrors that of gradient descent in action, where changes are attempted in the direction of improving performance until practical optimality is reached. The objective of this project was also to understand common pitfalls. I hope this paper proves to be an engaging read!

* 1. Data: Predicting customer churn is paramount for businesses across various sectors, facilitating proactive retention strategies. Our analysis is based on a dataset sourced from Kaggle (source: Cell 2 Cell), comprising diverse features pertinent to customer behavior and engagement. Compared to other datasets, such as telco, this dataset is complex – complexity of features & complexity of problem. The Cell2Cell dataset, available through Teradata Duke University, contains a rich collection of 71,047 instances with 58 attributes. This dataset serves as a crucial resource for the telecom industry, specifically in predicting customer churn behavior. By leveraging deep learning algorithms, stakeholders can derive valuable insights to enhance customer retention strategies. Churn management is paramount in this sector, and accurate churn prediction facilitates proactive measures to retain valuable customers. The dataset's link on Kaggle provides access to a comprehensive set of features, empowering telecom companies to segment their customer base by profitability and propensity to churn. This segmentation enables targeted retention efforts, optimizing resources and maximizing customer satisfaction. [Click here to Access the dataset](https://www.kaggle.com/jpacse/datasets-for-churn-telecom)

1.2. EDA & Cleansing: Extensive EDA report was prepared with many important business findings based on which the models were prepared. For the purposes of this submission, only deep learning aspects are discussed in the main document. EDA report can be accessed at Appendix – EDA. In the initial phase of data preparation, we meticulously cleansed the dataset to ensure its integrity and reliability. The dataset was already in a nice place to start with, therefore major cleaning steps involved were as follows:

* Handling missing values: Any missing data points were addressed through appropriate techniques, such as imputation or removal.
* Removing duplicates: Duplicate entries were identified and eliminated to prevent redundancy and maintain dataset consistency.
* Ordinal Encoding: Since the values of validation and test set are supposed to be “unseen” and since neural networks require categorical variables to be encoded to numbers, ordinal encoding is fit made on train data is used to fit the test and validation set wherein new values are labelled as -1.

1.3. Train Validation Test Split & Scaling:

To facilitate model development and evaluation, we partitioned the dataset into three distinct subsets:

* Training set: This subset constituted the majority of the data and served as the foundation for model training.
* Validation set: Utilized for fine-tuning model hyperparameters and assessing performance, the validation set enabled iterative refinement of churn prediction models.
* Test set (Holdout set): Reserved for final model evaluation, the test set remained untouched during model development, providing an unbiased assessment of model performance on unseen data.

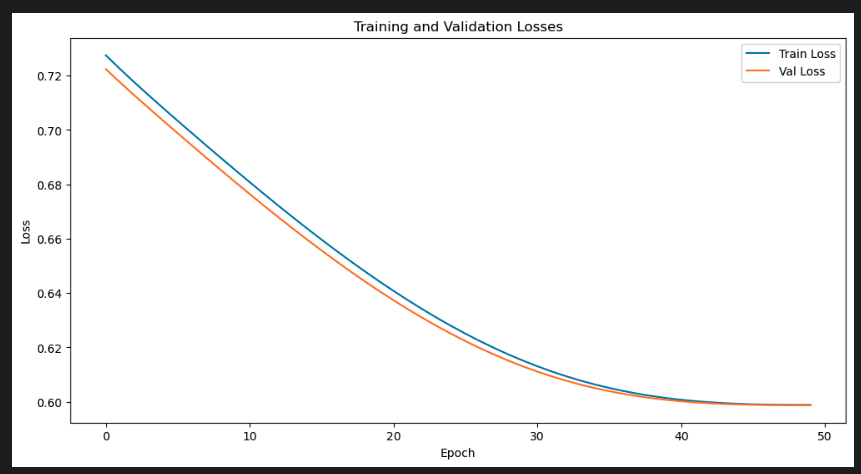
Scaling makes sure all features are equally important. We only scale the training set, then apply the same scaling to validation and holdout sets. This keeps things fair for our model.

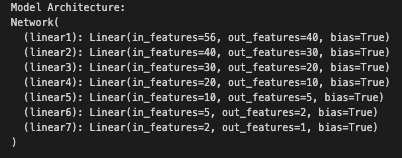
**Section 2: Experiments**

The Experiments section began by exploring some common pitfalls to understand why they are advised against. The first attempt involved using a simple feedforward neural network without normalization. This led to a very low F1 score as expected. However, just by adding normalization to a three-hidden-layer linear neural network, the accuracy increased slightly. A series of experiments were conducted, varying one hyperparameter/element of architecture at a time. First, we played with number of layers in a simple feedforward neural network, then activation functions, then initializations, then batch normalization, then optimization criterion, then mix of these and then finally other architectures such as RNN, LSTM.

2.1. Hidden Layers and Nodes Experiments [E1 to E11 in Code File, Best F1: 0.07]:

The first batch of experiments were done to understand the effect of number of layers. Initially I had compared 3 hidden layers versus 5, 7, 9, 12 and 15. Upon training, I found out the performance is very sensitive to the number of layers. Upon some research, sqrt(input\*output) was found to be a nice approximation of number of layers which was nicely validated when we had sqrt (56\*1) ~ 7 hidden layers which gave the best result of the batch. Still the results were not up to par with most predictions equal to “no churn”.

 A screenshot of a computer screen

Description automatically generated 

*Fig 1: Train-Validation curve with [simple 7 layers (Model Predicted Everything as* *0)]*

2.2. Activation Function Experiments [E12 to E17 in Code File, Best F1: 0.13]:

In this batch of experiments, various relevant activation functions were applied such as sigmoid at last layer, sigmoid at all layers relu, mix of relu and other activation functions, leaky relu and more. Initially, I had mistakenly used too many sigmoids, for multiple layers which led to poor results as shown in figure 2 (vanishing gradient). To fix this, sigmoid was applied only at last layer, for the rest a mix of dropout, and relu layers were used which led to model improvements. This led to slight improvement as shown in figure 2B.

A graph of a distribution of probabilities

Description automatically generated A graph with blue squares and numbers

Description automatically generated

*Fig 2A: Vanishing Gradient Problem(With all sigmoid) Fig 2B: Improvement In Confusion Matrix (sigmoid at end)*

2.3. Experiments on appropriate loss functions [E18-29, Best F1 = 0.43]:

Various loss functions were tried, such as binary cross entropy (BCE) loss, BCE with Logit, Hinge Loss. Some other loss functions were also tried for fun such as soft max which is better used for multi class classification. Respectable performance compared to previous experiments was achieved in these experiments as shown in figure 3, 4, 5

A graph showing a number of loss

Description automatically generated with medium confidence A screenshot of a graph

Description automatically generated

*Fig 3: Train-Validation curve with [BCE, and He Initialization] and its corresponding F1 and confusion matrix*

*Fig 3: Train-Validation curve with [BCE with Logit, and He Initialization] and its corresponding F1 and confusion matrix*

2.4. Experiments on Suitable Weight Initialization Techniques [E30: repeat of all previous experiments with initialization]:

Random Weight Initialization, He Initialization and Xavier initialization were implemented for the models so far. The best model was achieved with Xavier initialization. Best F1 of 0.43 was achieved as shown in figure 3 above.

2.5. Experiments on Optimization Algorithm:

ADAM, Ada, Batch, Mini Batch and momentum were tried. ADAM and Mini Batch had very competitive results.

A graph showing a line graph

Description automatically generated with medium confidence A blue squares with numbers

Description automatically generated

This experiment was the best experiment with the following performance summary:

A screenshot of a graph

Description automatically generated

2.7 Other Experiments (Layers, L1, L2, Early Stopping): Many other experiments were tried but to no avail as the performance did not increase too much.

2.8. Other Architectures (RNN, LSTM):

Some other complex architectures were also tried such as RNN and LSTM. Performance of RNN was lower than our feedforward best model.



RNN Architecture:

A computer screen shot of a program code

Description automatically generated

**Section 4: Learnings**

1. Activation Choice: ReLU is preferred over sigmoid for faster training due to its non-saturating nature and efficient gradient propagation, crucial for deep networks.
2. Output Activation: Sigmoid or softmax activations are chosen for classification tasks due to their probabilistic interpretation, facilitating decision-making and class prediction.
3. Loss Function: BCE and Hinge Loss are selected for binary classification due to their suitability for probabilistic outputs and margin maximization, enhancing model discriminative capability.
4. Hidden Layer Size: Optimizing node count strikes a balance between complexity and generalization, improving the network's ability to learn relevant features.
5. Batch Size Optimization: Experimenting with batch sizes balances computation and gradient accuracy, optimizing training efficiency and performance.
6. Sequential Data Handling: RNNs and LSTMs are preferred for sequential data processing due to their memory capabilities, facilitating context understanding and long-range dependencies modeling.
7. Regularization & Initialization: Dropout regularization and L2 penalty prevent overfitting by promoting feature independence and smoothing optimization, while proper weight initialization aids stable convergence.
8. Learning Rate Tuning: Proper adjustment of the learning rate during training ensures stable convergence without oscillation or divergence, optimizing the training process.
9. Data Preprocessing: Preprocessing techniques like normalization, imputation, and outlier detection enhance data quality and facilitate smoother model training and better generalization.
10. Model Evaluation: Employ cross-validation to robustly evaluate model performance, ensuring reliable estimates of performance metrics and reducing the risk of overfitting to specific subsets of data.

**Appendix – EDA Report**

**A screenshot of a black background

Description automatically generated**

**A screenshot of a computer

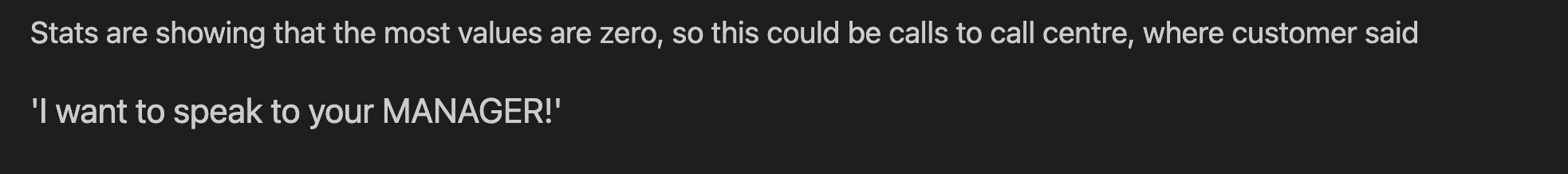
Description automatically generated A screenshot of a computer program

Description automatically generated A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

****

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated**

**A black screen with white text

Description automatically generated**

**A screenshot of a graph

Description automatically generated A screenshot of a graph

Description automatically generated**

**A screenshot of a graph

Description automatically generated A screenshot of a graph

Description automatically generated A screen shot of a graph

Description automatically generated**

**A screen shot of a graph

Description automatically generated A screenshot of a graph

Description automatically generated**

**A graph on a screen

Description automatically generated A graph on a screen

Description automatically generated**

**A graph with different colored boxes

Description automatically generated A graph with red and green lines

Description automatically generated**

**A graph of different colored bars

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated**

**A graph of a number of columns

Description automatically generated with medium confidence**

**A comparison of a graph

Description automatically generated**

**Thank You**