**Overview of the Analysis**

**Purpose of the Analysis**

The purpose of this analysis is to build a deep learning neural network model to predict the likelihood of applicants receiving funding from Alphabet Soup, a nonprofit organization. By analyzing historical application data, we aim to determine patterns that indicate whether an organization will receive funding or not.

This analysis will help Alphabet Soup allocate funding more efficiently by identifying promising applicants based on historical trends. Using a deep learning model allows us to uncover complex relationships between application features and the success of funding requests.

The key objective is to create a predictive model that meets a target accuracy of **75%** or higher. This report will discuss the data processing, model architecture, performance results, and recommendations for future improvements.

**Data Preprocessing**

✅ **What variable(s) are the target(s) for your model?**

* The **target variable** is **IS\_SUCCESSFUL**, which indicates if the application received funding (1) or not (0).

✅ **What variable(s) are the features for your model?**  
The following variables were used as input features to train the model:

* **APPLICATION\_TYPE**: Type of application submitted.
* **AFFILIATION**: The type of organization (Trust, Independent, etc.).
* **CLASSIFICATION**: Classification code representing the organization's purpose.
* **USE\_CASE**: The intended use of the funding.
* **ORGANIZATION**: The structure of the organization (Corporation, Trust, etc.).
* **STATUS**: Application status (active, inactive).
* **INCOME\_AMT**: Annual income of the organization.
* **ASK\_AMT**: Amount of funding requested.
* **SPECIAL\_CONSIDERATIONS**: Any special circumstances for the application.

✅ **What variable(s) should be removed from the input data because they are neither targets nor features?**  
The following columns were removed as they did not contribute to the model's predictive capabilities:

* **EIN (Employer Identification Number)**: A unique tax identifier with no predictive value.
* **NAME**: The name of the organization is irrelevant to predicting funding success.

**Data Cleaning Steps:**

1. Dropped **EIN** and **NAME** columns.
2. Consolidated low-frequency categories in **APPLICATION\_TYPE** and **CLASSIFICATION** under the label "Other."
3. Performed **one-hot encoding** on categorical variables.
4. Scaled numerical features using **StandardScaler** to ensure consistent input values.

**Compiling, Training, and Evaluating the Model**

✅ **How many neurons, layers, and activation functions did you select for your neural network model, and why?**

| **Layer** | **Neurons** | **Activation Function** | **Purpose** |
| --- | --- | --- | --- |
| Input | 43 (features) | None | Accept input features. |
| Hidden 1 | **80 neurons** | **ReLU** | Capture complex relationships in data. |
| Hidden 2 | **30 neurons** | **ReLU** | Further refine patterns and reduce dimensionality. |
| Output | 1 neuron | **Sigmoid** | Binary classification (Success or Not Successful). |

* **ReLU (Rectified Linear Unit)** activation function was used because it is efficient in reducing computational costs and minimizing the vanishing gradient problem.
* **Sigmoid activation function** was used in the output layer to predict probabilities for binary classification.

✅ **Why did you choose this architecture?**

* The architecture was chosen to balance computational efficiency and performance.
* Adding two hidden layers ensured the model could capture non-linear relationships without overfitting.
* The output layer used a **sigmoid activation** to produce a binary classification result (1 = success, 0 = failure).

**Model Compilation and Training**

The model was compiled using: nn.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

* **Loss Function:** Binary Cross-Entropy, as the model predicts binary outcomes.
* **Optimizer:** Adam Optimizer for efficient gradient descent.
* **Metric:** Accuracy to evaluate the model's performance.

The model was trained for **100 epochs** with a batch size of **32**. The model aimed to reach an accuracy of **75%** but achieved **72.91%**.

**Model Performance Results**

✅ **Was the target model performance achieved?**

* **No.** The target accuracy was **75%**, but the model achieved an accuracy of **72.91%**.
* Despite preprocessing and tuning efforts, the model fell short of the target by **2.09%**.

**Model Accuracy and Loss:**

| **Metric** | **Value** |
| --- | --- |
| **Loss** | 0.5412 |
| **Accuracy** | **72.91%** |

**Training and Validation Graphs:**

The graph showed that the model initially learned quickly but plateaued around **72% accuracy**, suggesting that the model hit a learning ceiling.

**1. Consolidating Rare Categories**

* Categories with fewer than **500 occurrences** in **APPLICATION\_TYPE** and **CLASSIFICATION** were grouped as **"Other"** to reduce noise.

**2. Increasing Hidden Layers and Neurons**

* Initially, the model had one hidden layer. Adding a second hidden layer (with **80 neurons** and **30 neurons**) improved performance by **1.2%**.

**3. Feature Scaling**

* Scaling numerical variables like **ASK\_AMT** and **INCOME\_AMT** ensured that large numbers did not dominate the model’s training process.

**4. Increasing Epochs**

* Increasing the epochs from **50 to 100** improved accuracy slightly. However, further training risked overfitting.

**5. Drop Irrelevant Columns**

* Removing **EIN** and **NAME** reduced unnecessary noise in the model.

**Recommendations for Future Improvements**

**What steps could further improve performance?**

1. **Use a Different Machine Learning Model.**

While a deep learning model is powerful, structured tabular data often performs better with:

* **Random Forest Classifier** – Handles categorical data with high accuracy.
* **XGBoost (Extreme Gradient Boosting)** – Provides high accuracy on structured data by minimizing bias and variance.
* **Logistic Regression** – Simple yet effective for binary classification problems.

1. **Perform Hyperparameter Tuning**

Using tools like **Keras Tuner** or **GridSearchCV**, we could fine-tune:

* **Number of neurons per layer.**
* **Learning rate.**
* **Batch size.**
* **Activation functions.**  
  This would optimize the model’s learning performance.

1. **Handle Class Imbalance**

If the dataset had a class imbalance (more successful applications than unsuccessful), the model might favor one outcome. Using:

* **SMOTE (Synthetic Minority Oversampling Technique)**
* **Class weight balancing**  
  could address this imbalance and improve accuracy.

1. **Engineering New Features**

Creating new features such as:

* **Success rate based on Classification Type.**
* **Success rate based on Application Type.**  
  could provide additional predictive power.

## **5. Conclusion**

The deep learning model for **Alphabet Soup's Charity Funding Prediction** provided **72.91% accuracy**, slightly below the target **75% accuracy**. While the model captured important patterns from the data, it struggled with accuracy due to:

* Imbalanced data.
* Limited training features.
* Lack of hyperparameter tuning.

To improve performance, it is recommended to:  
✅ Switch to a **Random Forest** or **XGBoost** model.  
✅ Perform hyperparameter tuning.  
✅ Engineer new predictive features.

By implementing these changes, **Alphabet Soup** can significantly enhance its ability to identify promising applicants and allocate funding more efficiently.