

## Analysis of Neural Network Results

### 1. Data Preprocessing

#### Initial Data Setup:

**Imported data:** The dataset (charity\_data.csv) was read into a pandas DataFrame.

**Dropped non-beneficial columns:** Columns like 'EIN' and 'NAME' were removed because they do not contribute to the model.

#### Categorical Data Processing:

**APPLICATION\_TYPE:** I replaced application types with low frequencies (those appearing less than 500 times) with a more general category "Other". This helps reduce the dimensionality caused by infrequent categories.

**CLASSIFICATION:** Similarly, the "CLASSIFICATION" column was cleaned by grouping classifications that appeared less than 100 times into the "Other" category.

#### Data Transformation:

**One-hot encoding:** Used `pd.get_dummies()` to convert categorical variables into numeric values, making them suitable for machine learning algorithms.

**Feature/Target Split:** The target variable (IS\_SUCCESSFUL) was separated from the feature variables, and the dataset was split into training and testing datasets using `train_test_split`.

**Scaling:** A `StandardScaler` was applied to scale the features, ensuring the neural network can learn effectively by preventing large numerical values from dominating.

### 2. Model Building

#### Model Definition:

A **Sequential neural network** was defined with:

**Input Layer:** The number of neurons in the input layer corresponds to the number of features in the dataset.

**Hidden Layers:** Two hidden layers were used:

First hidden layer with 80 units and ReLU activation.

Second hidden layer with 30 units and ReLU activation.

**Output Layer:** A single neuron using a **sigmoid** activation function to predict the probability of success (binary classification).

#### Compilation:

The model was compiled with:

**Loss function:** `binary_crossentropy`, which is suitable for binary classification.

**Optimizer:** `adam`, a popular optimizer that adapts the learning rate during training.

**Metric:** accuracy, to track how well the model is performing.

### 3. Model Training

The model was trained for **100 epochs** with a batch size of **32** on the preprocessed training data (X\_train\_scaled and y\_train). I also provided validation data (X\_test\_scaled, y\_test) to monitor overfitting.

#### Initial Results:

During training, the accuracy improved steadily (starting from 70% and reaching close to 74% after several epochs).

The **loss** decreased while the **validation accuracy** remained around the same range (72-74%).

### 4. Evaluation of Results

Looking at the training results from the logs:

**Accuracy** fluctuated slightly, ranging from **70% to 74%** across epochs.

**Validation accuracy** was consistently around **72%** and stabilized around this level.

**Loss** seemed to decrease slightly, suggesting some learning took place, but the improvement was slow.

### 5. Analysis of Model Performance

A few points to consider about model performance:

**Model underfitting:** The model might be underfitting. Despite training for 100 epochs, the accuracy is relatively low (around 74%), and the model doesn't seem to improve substantially. This could indicate that the model is not complex enough to capture the patterns in the data.

**Limited improvements:** The validation accuracy has plateaued, and the loss remains relatively constant, which means there may be a limit to the model's performance with the current setup.

### 6. Next Steps for Model Improvement

**Increase Model Complexity:** Adding more hidden layers or increasing the number of units per layer may help the model capture more complex patterns.

**Learning Rate Adjustment:** Adjusting the learning rate can help the optimizer converge faster or more accurately.

**More Epochs:** Increasing the number of epochs might give the model more time to learn, but be careful of overfitting.

**Hyperparameter Tuning:** Experiment with different activation functions (like LeakyReLU), optimizers, or loss functions.

**Early Stopping:** Implement early stopping during training to stop the model from overfitting once the validation accuracy starts to degrade.

**Cross-validation:** Use k-fold cross-validation to get more robust results.

**Feature Engineering:** Revisit feature engineering—perhaps certain features could be combined or new features could be derived from existing ones to improve model performance.

## 7. Potential Data Issues

**Imbalanced Classes:** If the dataset is imbalanced (e.g., more successful applications than unsuccessful ones), it could skew the results. We might need to consider techniques such as class weighting, oversampling, or undersampling.

**Data Leakage:** Ensure that no future information (such as outcome data) is leaking into the training set.

### Conclusion

The deep learning model has started to show promising results, but there is still room for optimization. Fine-tuning the model architecture and experimenting with more advanced techniques should help improve performance. Additionally, addressing potential data issues will ensure that the model generalizes well to new, unseen data.