Architecture of the ANN Model

# ANN Model Architecture Model

ANN model architecture consists of the layers structure, that is, how good the model learns from the data and makes predictions. A detailed analysis of every part of the architecture is below:

## Input Layer

The input layer is the initial layer of the ANN model and it receives the input data. This is determined by the structure of the input data, also known as the number of features in the dataset. Here, we use Input function to specify the input shape.

### Code

*# Define input shape based on the number of features input\_shape = (X\_train\_scaled.shape[1],)*

*input\_layer = Input(shape=input\_shape)*

## Hidden Layers

The hidden layers are responsible for doing most of the computation in the ANN model. These layers are made of neurons (or units), which perform transformations of the input data. A ReLU (Rectified Linear Unit) is usually used in hidden layers, adding non-linearity, and enabling the model to learn complex patterns.

### Code

*# First hidden layer with 64 neurons*

*hidden\_layer\_1 = Dense(units=64, activation='relu')(input\_layer)*

*# Second hidden layer with 32 neurons*

*hidden\_layer\_2 = Dense(units=32, activation='relu')(hidden\_layer\_1)*

*# Third hidden layer with 16 neurons*

*hidden\_layer\_3 = Dense(units=16, activation='relu')(hidden\_layer\_2)*

Unit is the number of significant features, when i tried putting around 16, 16, 8 it was giving 85%.

## Output Layer

The output layer is the last layer of the ANN model and only outputs the model’s predictions. Such as in binary-classification (e.g., predicting customer churn) cases, output layer consists of a single neuron that applies sigmoid function which returns a probability score ranging from 0 to 1.

### Code

*# Output layer for binary classification*

*output\_layer = Dense(units=1, activation='sigmoid')(hidden\_layer\_3)*

## Model Compilation

Once we have defined the layers, lets compile the model. The next step is to compile the model; you need to tell it what algorithm it should use to do the optimization, what loss function to use, and what metrics to track. The Adam optimizer is a popular adaptive learning rate optimization algorithm for training neural networks. The loss and evaluation metrics for the binary classification is binary cross entropy and accuracy, respectively.

### Code

*# Compile the ANN model*

*ann\_model = tf.keras.Model(inputs=input\_layer, outputs=output\_layer)*

*ann\_model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])*

# Trained ANN Model on Given Dataset

Training of the ANN model comprises the input and the output the data into the model to learn the pattern and then analyze the performance. Here’s a fuller breakdown, step by step:

## Data Preparation

First, we load the dataset and encode categorical variables. The data is then divided into training and testing sets, the features are normalized.

### Code

*# Load and preprocess the dataset*

*df = pd.read\_csv('Dataset (ATS).csv')*

*# Encode categorical variables*

*categorical\_cols = ['gender', 'Dependents', 'PhoneService', 'MultipleLines',*

*'InternetService', 'Contract', 'Churn']*

*le = LabelEncoder() for col in categorical\_cols:*

*df[col] = le.fit\_transform(df[col])*

*# Split the dataset into features (X) and target (y)*

*X = df.drop('Churn', axis=1) y = df['Churn']*

*# Split into training and testing sets*

*X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)*

*# Scale the features*

*scaler = StandardScaler()*

*X\_train\_scaled = scaler.fit\_transform(X\_train)*

*X\_test\_scaled = scaler.transform(X\_test)*

## Model Training

The ANN model is trained with the help of training data. Early stopping is used to avoid overfitting, halting the training if the validation loss does not improve after several epochs (patience).

### Code

*# Early stopping callback*

*early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)*

*# Train the model with early stopping*

*history = ann\_model.fit(X\_train\_scaled, y\_train, epochs=50, batch\_size=32, validation\_split=0.2, verbose=1, callbacks=[early\_stopping])*

## Model Evaluation

Then we carry out testing of the model performance using test data after training. And you can predict it by the model on test data and check the performance using metrics like accuracy and classification report.

### Code

*# Predict churn on the test data*

*y\_pred = (ann\_model.predict(X\_test\_scaled) > 0.5).astype(int)*

*# Evaluate the model's performance accuracy = accuracy\_score(y\_test, y\_pred)*

*classification\_report\_output = classification\_report(y\_test, y\_pred)*

*print(f"Accuracy: {accuracy}")*

*print(f"Classification Report: \n{classification\_report\_output}")*

## Saving the Trained Model

The ANN model which is trained is stored in a file which can be used for future purposes. This enables you to load, and utilize the trained model, without needing to train it again.

### Code

*# Save the trained model*

*ann\_model.save('trained\_ann\_model.h5')*