Exercise 1: Introduction to Neural Networks with Activation Functions

Objective

- Understand the concept of a basic Neural Network for classification.
- Learn how activation functions impact network output.
- Implement a simple feedforward neural network for binary classification.

Background

A Neural Network consists of layers of interconnected nodes (neurons) that process input features to predict an output.

- Activation Functions introduce non-linearity, enabling the network to model complex patterns.
- Common activation functions include:
  - Sigmoid: maps output between 0 and 1 (good for binary classification)
  - ReLU: introduces sparsity and avoids vanishing gradients
  - Tanh: maps output between -1 and 1 This exercise uses a simple feedforward neural network to predict loan approval (Yes/No) based on applicant data.

Step 1: Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
```

Step 2: Define Sample Dataset

In [2]:
```
# Sample dataset for Study Loan Approval (all binary/normalized values)
data = {
    'Income_High': [1,0,1,0,1,0,0,1],
    'Nationality_Local': [1,0,1,1,0,1,0,1],
    'Age_Above_30': [1,0,1,0,1,0,1,1],
    'CIBIL_Good': [1,1,0,0,1,0,1,1],
    'Collateral_Yes': [1,0,1,0,1,0,1,0],
    'Approval': [1,0,1,0,1,0,0,1]  # 1 = Approved, 0 = Not Approved
}
```

```
df = pd.DataFrame(data)
df
```

Out[2]:

	Income_High	Nationality_Local	Age_Above_30	CIBIL_Good	Collateral_Yes	Approval
0	1	1	1	1	1	1
1	0	0	0	1	0	0
2	1	1	1	0	1	1
3	0	1	0	0	0	0
4	1	0	1	1	1	1
5	0	1	0	0	0	0
6	0	0	1	1	1	0
7	1	1	1	1	0	1

Step 3: Split Dataset into Features and Labels

In [3]:

```
X = df.drop('Approval', axis=1).values
y = df['Approval'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_st
```

Step 4: Feature Scaling

In [4]:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Step 5: Build Neural Network Model

In [5]:

```
model = Sequential([
    Dense(4, input_dim=X_train.shape[1], activation='relu'),    # Hidden layer
    Dense(1, activation='sigmoid')                              # Output layer for b
])
model.compile(optimizer=Adam(learning_rate=0.01), loss='binary_crossentropy', metri
model.summary()
```

```
C:\Users\sarop\Envs\emoenv310\lib\site-packages\keras\src\layers\core\dense.py:93: U
serWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in the m
odel instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	
dense (Dense)	(None, 4)	
dense_1 (Dense)	(None, 1)	

Total params: 29 (116.00 B)

Trainable params: 29 (116.00 B)

Non-trainable params: 0 (0.00 B)

Step 6: Train the Model

In [6]:
```
history = model.fit(X_train, y_train, epochs=50, batch_size=2, validation_split=0.2
```

```
Epoch 1/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 463ms/step - accuracy: 0.6667 - loss: 0.7598 - val_accur
acy: 0.5000 - val_loss: 0.9068
Epoch 2/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 117ms/step - accuracy: 0.6667 - loss: 0.7016 - val_accur
acy: 0.5000 - val_loss: 0.9208
Epoch 3/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 128ms/step - accuracy: 0.8333 - loss: 0.5853 - val_accur
acy: 0.5000 - val_loss: 0.9364
Epoch 4/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 127ms/step - accuracy: 0.6667 - loss: 0.6161 - val_accur
acy: 0.5000 - val_loss: 0.9534
Epoch 5/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 126ms/step - accuracy: 0.8333 - loss: 0.4983 - val_accur
acy: 0.5000 - val_loss: 0.9738
Epoch 6/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step - accuracy: 0.6667 - loss: 0.5172 - val_accur
acy: 0.5000 - val_loss: 0.9930
Epoch 7/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 147ms/step - accuracy: 1.0000 - loss: 0.4352 - val_accur
acy: 0.5000 - val_loss: 1.0140
Epoch 8/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 129ms/step - accuracy: 1.0000 - loss: 0.4406 - val_accur
acy: 0.5000 - val_loss: 1.0349
Epoch 9/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 182ms/step - accuracy: 1.0000 - loss: 0.3542 - val_accur
acy: 0.5000 - val_loss: 1.0582
Epoch 10/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step - accuracy: 1.0000 - loss: 0.3168 - val_accura
cy: 0.5000 - val_loss: 1.0818
Epoch 11/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 105ms/step - accuracy: 1.0000 - loss: 0.3275 - val_accur
acy: 0.5000 - val_loss: 1.1040
Epoch 12/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 134ms/step - accuracy: 1.0000 - loss: 0.2602 - val_accur
acy: 0.5000 - val_loss: 1.1281
Epoch 13/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step - accuracy: 1.0000 - loss: 0.2396 - val_accura
cy: 0.5000 - val_loss: 1.1518
Epoch 14/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 122ms/step - accuracy: 1.0000 - loss: 0.2472 - val_accur
acy: 0.0000e+00 - val_loss: 1.1738
Epoch 15/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step - accuracy: 1.0000 - loss: 0.1945 - val_accura
cy: 0.0000e+00 - val_loss: 1.1972
Epoch 16/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 170ms/step - accuracy: 1.0000 - loss: 0.2014 - val_accur
acy: 0.0000e+00 - val_loss: 1.2189
Epoch 17/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 120ms/step - accuracy: 1.0000 - loss: 0.1816 - val_accur
acy: 0.0000e+00 - val_loss: 1.2403
Epoch 18/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 116ms/step - accuracy: 1.0000 - loss: 0.1394 - val_accur
acy: 0.0000e+00 - val_loss: 1.2627
Epoch 19/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step - accuracy: 1.0000 - loss: 0.1276 - val_accura
```

```
cy: 0.0000e+00 - val_loss: 1.2841
Epoch 20/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step - accuracy: 1.0000 - loss: 0.1132 - val_accur
acy: 0.0000e+00 - val_loss: 1.3049
Epoch 21/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step - accuracy: 1.0000 - loss: 0.1102 - val_accura
cy: 0.0000e+00 - val_loss: 1.3242
Epoch 22/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step - accuracy: 1.0000 - loss: 0.0942 - val_accura
cy: 0.0000e+00 - val_loss: 1.3434
Epoch 23/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 111ms/step - accuracy: 1.0000 - loss: 0.0858 - val_accur
acy: 0.0000e+00 - val_loss: 1.3622
Epoch 24/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step - accuracy: 1.0000 - loss: 0.0865 - val_accur
acy: 0.0000e+00 - val_loss: 1.3804
Epoch 25/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step - accuracy: 1.0000 - loss: 0.0862 - val_accur
acy: 0.0000e+00 - val_loss: 1.3978
Epoch 26/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 1.0000 - loss: 0.0718 - val_accura
cy: 0.0000e+00 - val_loss: 1.4153
Epoch 27/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 89ms/step - accuracy: 1.0000 - loss: 0.0670 - val_accura
cy: 0.0000e+00 - val_loss: 1.4319
Epoch 28/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - accuracy: 1.0000 - loss: 0.0678 - val_accura
cy: 0.0000e+00 - val_loss: 1.4480
Epoch 29/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 74ms/step - accuracy: 1.0000 - loss: 0.0571 - val_accura
cy: 0.0000e+00 - val_loss: 1.4635
Epoch 30/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 84ms/step - accuracy: 1.0000 - loss: 0.0588 - val_accura
cy: 0.0000e+00 - val_loss: 1.4784
Epoch 31/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 77ms/step - accuracy: 1.0000 - loss: 0.0502 - val_accura
cy: 0.0000e+00 - val_loss: 1.4928
Epoch 32/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 80ms/step - accuracy: 1.0000 - loss: 0.0459 - val_accura
cy: 0.0000e+00 - val_loss: 1.5065
Epoch 33/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step - accuracy: 1.0000 - loss: 0.0442 - val_accur
acy: 0.0000e+00 - val_loss: 1.5197
Epoch 34/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 83ms/step - accuracy: 1.0000 - loss: 0.0455 - val_accura
cy: 0.0000e+00 - val_loss: 1.5323
Epoch 35/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 78ms/step - accuracy: 1.0000 - loss: 0.0376 - val_accura
cy: 0.0000e+00 - val_loss: 1.5443
Epoch 36/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 151ms/step - accuracy: 1.0000 - loss: 0.0371 - val_accur
acy: 0.0000e+00 - val_loss: 1.5559
Epoch 37/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step - accuracy: 1.0000 - loss: 0.0301 - val_accur
acy: 0.0000e+00 - val_loss: 1.5670
Epoch 38/50
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 182ms/step - accuracy: 1.0000 - loss: 0.0365 - val_accur
acy: 0.0000e+00 - val_loss: 1.5774
Epoch 39/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 153ms/step - accuracy: 1.0000 - loss: 0.0317 - val_accur
acy: 0.0000e+00 - val_loss: 1.5877
Epoch 40/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 166ms/step - accuracy: 1.0000 - loss: 0.0328 - val_accur
acy: 0.0000e+00 - val_loss: 1.5975
Epoch 41/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 140ms/step - accuracy: 1.0000 - loss: 0.0241 - val_accur
acy: 0.0000e+00 - val_loss: 1.6072
Epoch 42/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 165ms/step - accuracy: 1.0000 - loss: 0.0230 - val_accur
acy: 0.0000e+00 - val_loss: 1.6162
Epoch 43/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 160ms/step - accuracy: 1.0000 - loss: 0.0262 - val_accur
acy: 0.0000e+00 - val_loss: 1.6250
Epoch 44/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 181ms/step - accuracy: 1.0000 - loss: 0.0272 - val_accur
acy: 0.0000e+00 - val_loss: 1.6334
Epoch 45/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - accuracy: 1.0000 - loss: 0.0221 - val_accur
acy: 0.0000e+00 - val_loss: 1.6415
Epoch 46/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 99ms/step - accuracy: 1.0000 - loss: 0.0230 - val_accura
cy: 0.0000e+00 - val_loss: 1.6494
Epoch 47/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 132ms/step - accuracy: 1.0000 - loss: 0.0183 - val_accur
acy: 0.0000e+00 - val_loss: 1.6570
Epoch 48/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 115ms/step - accuracy: 1.0000 - loss: 0.0230 - val_accur
acy: 0.0000e+00 - val_loss: 1.6642
Epoch 49/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step - accuracy: 1.0000 - loss: 0.0221 - val_accura
cy: 0.0000e+00 - val_loss: 1.6714
Epoch 50/50
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step - accuracy: 1.0000 - loss: 0.0212 - val_accura
cy: 0.0000e+00 - val_loss: 1.6782
```

Step 7: Evaluate Model

In [7]:
```python
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy*100:.2f}%")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 440ms/step - accuracy: 0.5000 - loss: 1.3683
Test Accuracy: 50.00%
```

Step 8: Sample Predictions

In [8]:
```python
predictions = model.predict(X_test)
pred_labels = (predictions > 0.5).astype(int)
print("Predicted Approval Status:", pred_labels.flatten())
print("Actual Approval Status:", y_test)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 182ms/step
Predicted Approval Status: [0 1]
Actual Approval Status: [0 0]
```

Results and Observation

- Training and validation accuracy curves show how well the model learned.
- Predicted loan approval matches most of the actual outcomes.
- Using ReLU for hidden layers and Sigmoid for output is effective for binary classification.

Insights

- Activation functions determine neuron output and model performance.
- Sigmoid outputs are suitable for binary decisions.
- Small datasets can be overfitted; scaling input features improves learning.
- Adding more hidden layers can improve representation but may require more data.

Conclusion

- Students implemented a simple neural network for loan approval prediction.
- Learned the effect of activation functions and basic feedforward network training.
- Gained hands-on experience with TensorFlow/Keras for binary classification.

In [ ]: