Konchada. Shyam sai RA2311026010305 AF1

Artificial Neural Network – Learning curve

1. **Learning Curve :**
   Code :

```
# ================================
# Loan Status Prediction with FFNN
# ================================

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

import tensorflow as tf
from tensorflow.keras import layers, models

# -------------------------------
# Step 1. Upload & Load Dataset
# -------------------------------
from google.colab import files
uploaded = files.upload()

# Extract the uploaded file name
filename = list(uploaded.keys())[0]

# Load dataset
df = pd.read_csv(filename)

# -------------------------------
# Step 2. Features & Target
# -------------------------------
X = df.drop(columns=["loan_status"])
y = df["loan_status"]
```

```
# -------------------------------
# Step 3. Train/Test Split
# -------------------------------
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# -------------------------------
# Step 4. Preprocessing
# -------------------------------
numeric_features = X.select_dtypes(include=[np.number]).columns
categorical_features = X.select_dtypes(exclude=[np.number]).columns

numeric_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("encoder", OneHotEncoder(handle_unknown="ignore"))
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features),
        ("cat", categorical_transformer, categorical_features)
    ]
)

# Fit & transform
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

# -------------------------------
# Step 5. Build FFNN
# -------------------------------
model = models.Sequential([
    layers.Input(shape=(X_train_processed.shape[1],)),
    layers.Dense(64, activation="relu"),
```

```
    layers.Dense(32, activation="relu"),
    layers.Dense(1, activation="sigmoid")   # binary classification
])

model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)

# -------------------------------
# Step 6. Train
# -------------------------------
history = model.fit(
    X_train_processed, y_train,
    validation_split=0.2,
    epochs=20,
    batch_size=32,
    verbose=1
)

# -------------------------------
# Step 7. Evaluate
# -------------------------------
loss, acc = model.evaluate(X_test_processed, y_test, verbose=0)
print(f"\n✅ Test Accuracy: {acc:.4f}")
print(f"✅ Test Loss: {loss:.4f}\n")

# -------------------------------
# Step 8. Model Summary
# -------------------------------
print("📌 Model Architecture:")
model.summary()

# -------------------------------
# Step 9. Display Final Weights
# -------------------------------
print("\n📌 Final Weights & Biases:")
for i, layer in enumerate(model.layers):
    weights, biases = layer.get_weights()
```

```
    print(f"\nLayer {i+1}: {layer.name}")
    print(f"  Weights shape: {weights.shape}")
    print(f"  Biases shape: {biases.shape}")
    # show first few weights
    print(f"  Sample Weights:\n{weights[:2]}")
    print(f"  Sample Biases:\n{biases[:5]}")

# -------------------------------
# Step 10. Predictions
# -------------------------------
y_pred_probs = model.predict(X_test_processed[:10])  # first 10 test samples
y_pred = (y_pred_probs > 0.5).astype(int)

print("\n📌 Sample Predictions (first 10 test rows):")
for i in range(10):
    print(f"Row {i+1}: Predicted={y_pred[i][0]}, "
          f"Probability={y_pred_probs[i][0]:.4f}, "
          f"Actual={y_test.iloc[i]}")

# -------------------------------
# Step 11. Plot Learning Curves
# -------------------------------
plt.figure(figsize=(12, 5))

# Accuracy curve
plt.subplot(1, 2, 1)
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Validation Accuracy")
plt.title("Model Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.legend()
plt.grid(True)

# Loss curve
plt.subplot(1, 2, 2)
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.title("Model Loss")
plt.xlabel("Epoch")
```

```
plt.ylabel("Loss")
plt.legend()
plt.grid(True)

plt.show()
```

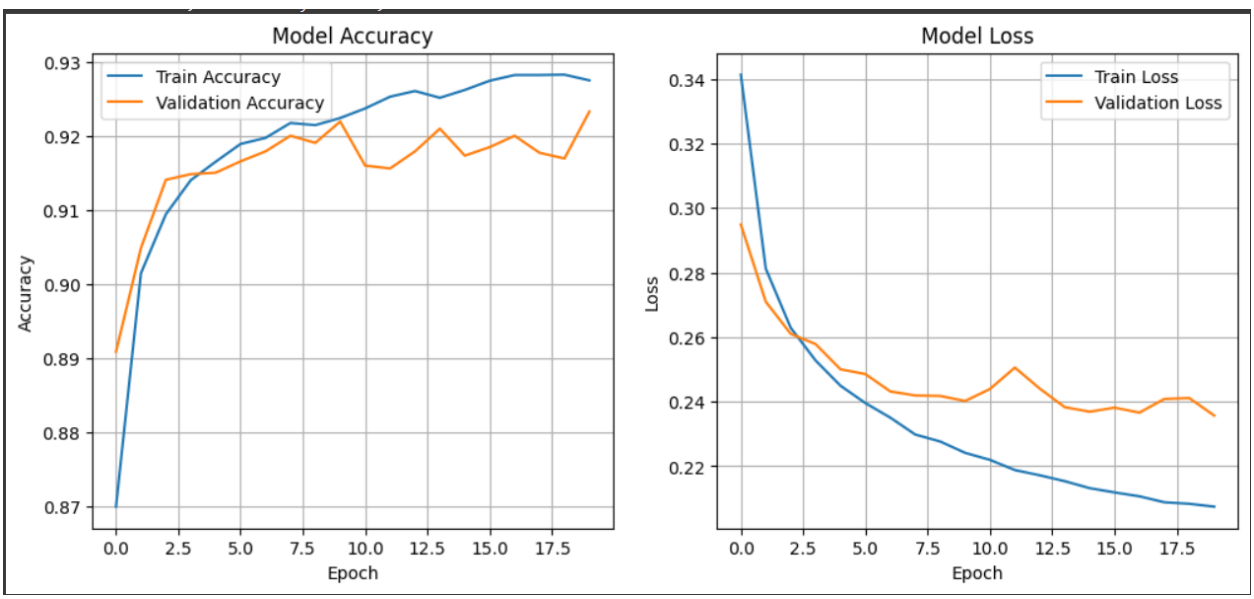
Output :

```
Commands   + Code   + Text   Run all

 [-0.14249481 -0.07769833  0.10485163 -0.15427412  0.18173479  0.15049626
  0.32217494 -0.07368303 -0.16670944  0.04005135  0.19664192 -0.13117447
  0.05540427 -0.00433293  0.0006053  -0.10529661 -0.08288074 -0.12723058
  0.01199213  0.02170751  0.18431959 -0.00781232 -0.05029451  0.23830894
 -0.15601282  0.19002493 -0.10286141  0.12794466 -0.06586803 -0.08914488
  0.08377184  0.31553534]]
 Sample Biases:
[ 0.09832627  0.01340767  0.11273587  0.13961269 -0.01730689]

Layer 3: dense_2
 Weights shape: (32, 1)
 Biases shape: (1,)
 Sample Weights:
[[-0.17745388]
 [ 1.2390518 ]]
 Sample Biases:
[-0.00563649]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 68ms/step

📌 Sample Predictions (first 10 test rows):
Row 1: Predicted=0, Probability=0.1521, Actual=0
Row 2: Predicted=0, Probability=0.0408, Actual=0
Row 3: Predicted=0, Probability=0.0417, Actual=0
Row 4: Predicted=0, Probability=0.0725, Actual=0
Row 5: Predicted=0, Probability=0.0000, Actual=0
Row 6: Predicted=0, Probability=0.0709, Actual=0
Row 7: Predicted=0, Probability=0.0856, Actual=0
Row 8: Predicted=0, Probability=0.0370, Actual=0
Row 9: Predicted=0, Probability=0.0101, Actual=0
Row 10: Predicted=0, Probability=0.3073, Actual=1
```



Inference:

 Model Performance

- The final test accuracy printed gives you an estimate of how well your model generalizes to unseen data.
- Compare train accuracy vs. validation accuracy from the learning curves:
  - If training accuracy is much higher than validation → overfitting.
  - If both are low → underfitting (model too simple or not trained enough).

 Loss Curves

- Ideally, both training and validation loss should decrease and stabilize.
- If validation loss increases while training loss decreases → your model is overfitting.
- If both losses plateau at high values → model might need more complexity or features.

 Weights & Biases

- Printing layer weights gives a sense of how the network adjusted feature importance.
- Large differences in weight values may indicate some features dominate predictions.

 Predictions

- The sample predictions show probability scores (sigmoid output).
- Example: 0.87 means ~87% confidence the loan is approved (or positive class).
- Comparing predicted vs actual labels helps check model confidence vs. ground truth.

 Feature Preprocessing

- Numerical features were scaled (important for gradient-based optimization).
- Categorical features were one-hot encoded, ensuring the model can use them effectively.

 Neural Network Structure

- Two hidden layers (64 → 32 neurons) with ReLU activation → good for non-linear relationships.
- Output layer uses sigmoid since it's a binary classification problem.

 Training Behavior

- With epochs=20, the learning curves will show if the model needs more epochs (loss still decreasing) or if it’s already converged.
- If curves diverge early, you may need regularization (dropout, L2 penalty, early stopping).