perceptron_hardcorded

June 5, 2025

[1]:
```
import numpy as np
```

[2]:
```
class Perceptron:
    def __init__(self, learning_rate=0.1, epochs=100):
        self.lr = learning_rate
        self.epochs = epochs
        self.weights = None
        self.bias = None

    def activation(self, z):
        return 1 if z >= 0 else 0

    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.random.rand(n_features)
        self.bias = np.random.rand()

        for _ in range(self.epochs):
            for i in range(n_samples):
                z = np.dot(X[i], self.weights) + self.bias
                y_pred = self.activation(z)
                error = y[i] - y_pred
                self.weights += self.lr * error * X[i]
                self.bias += self.lr * error

    def predict(self, X):
        z = np.dot(X, self.weights) + self.bias
        return np.array([self.activation(zi) for zi in z])
```

[3]:
```
# Data for AND gate
X_and = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_and = np.array([0, 0, 0, 1])
```

[4]:
```
# Data for OR gate
X_or = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_or = np.array([0, 1, 1, 1])
```

[5]:
```python
# Train and test AND gate
print("Training Perceptron for AND gate:")
and_perceptron = Perceptron(learning_rate=0.1, epochs=100)
and_perceptron.fit(X_and, y_and)
predictions_and = and_perceptron.predict(X_and)
print("AND Gate Predictions:", predictions_and)
print("AND Gate Weights:", and_perceptron.weights, "Bias:", and_perceptron.bias)
```

```
Training Perceptron for AND gate:
AND Gate Predictions: [0 0 0 1]
AND Gate Weights: [0.26050667 0.12968379] Bias: -0.386287838887039586
```

[6]:
```python
# Train and test OR gate
print("\nTraining Perceptron for OR gate:")
or_perceptron = Perceptron(learning_rate=0.1, epochs=100)
or_perceptron.fit(X_or, y_or)
predictions_or = or_perceptron.predict(X_or)
print("OR Gate Predictions:", predictions_or)
print("OR Gate Weights:", or_perceptron.weights, "Bias:", or_perceptron.bias)
```

```

Training Perceptron for OR gate:
OR Gate Predictions: [0 1 1 1]
OR Gate Weights: [0.47223509 0.90652439] Bias: -0.0726942328043195
```

[ ]:
```python
#make necessary adjustments to the code
# Save the model if needed
```

perceptron_tensorflow

June 5, 2025

[1]:
```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
```

[2]:
```python
# Load Iris dataset and select two classes (Setosa and Versicolor)
iris = load_iris()
X = iris.data[iris.target != 2, :4]  # Use all four features, exclude Virginica
y = iris.target[iris.target != 2]    # Binary classification (Setosa=0,␣
 ↪Versicolor=1)
```

[3]:
```python
# Split the dataset
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4,␣
 ↪random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5,␣
 ↪random_state=42)
```

[4]:
```python
# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

[5]:
```python
# Create the Perceptron model
def create_perceptron(learning_rate=0.01):
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(1, activation='sigmoid', input_shape=(4,))
    ])
    model.compile(optimizer=tf.keras.optimizers.
 ↪SGD(learning_rate=learning_rate),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model
```

[6]:
```python
# Train the Perceptron
def train_perceptron(model, X_train, y_train, X_val, y_val, epochs=100):
    history = model.fit(X_train, y_train, epochs=epochs,␣
 ↪validation_data=(X_val, y_val), verbose=1)
    return history
```

[7]:
```python
# Evaluate the Perceptron
def evaluate_perceptron(model, X_test, y_test):
    loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
    print(f"\nTest Loss: {loss:.4f}")
    print(f"Test Accuracy: {accuracy:.4f}")
    predictions = (model.predict(X_test) >= 0.5).astype(int)
    print("\nSample Predictions (First 10 instances):")
    for i in range(10):
        print(f"True: {y_test[i]}, Predicted: {predictions[i][0]}")
    cm = confusion_matrix(y_test, predictions)
    print("\nConfusion Matrix:")
    print(cm)
```

[8]:
```python
# Plot training history
def plot_training_history(history):
    plt.figure(figsize=(12, 4))
    # Plot accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.legend()
    # Plot loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('Training and Validation Loss')
    plt.legend()
    plt.show()
```

[9]:
```python
# Execute the pipeline
model = create_perceptron(learning_rate=0.01)
history = train_perceptron(model, X_train, y_train, X_val, y_val, epochs=100)
evaluate_perceptron(model, X_test, y_test)
plot_training_history(history)
```

```
Epoch 1/100
```

```
c:\Users\sajee\anaconda3\Lib\site-packages\keras\src\layers\core\dense.py:87:
UserWarning: 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 model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
2/2            1s 280ms/step -
accuracy: 0.6194 - loss: 0.6525 - val_accuracy: 0.7500 - val_loss: 0.6067
Epoch 2/100
2/2            0s 70ms/step -
accuracy: 0.6507 - loss: 0.6318 - val_accuracy: 0.7500 - val_loss: 0.5947
Epoch 3/100
2/2            0s 78ms/step -
accuracy: 0.6840 - loss: 0.6250 - val_accuracy: 0.7500 - val_loss: 0.5831
Epoch 4/100
2/2            0s 86ms/step -
accuracy: 0.7063 - loss: 0.6071 - val_accuracy: 0.8000 - val_loss: 0.5718
Epoch 5/100
2/2            0s 75ms/step -
accuracy: 0.7597 - loss: 0.5870 - val_accuracy: 0.8500 - val_loss: 0.5608
Epoch 6/100
2/2            0s 78ms/step -
accuracy: 0.7507 - loss: 0.5971 - val_accuracy: 0.8500 - val_loss: 0.5503
Epoch 7/100
2/2            0s 83ms/step -
accuracy: 0.8035 - loss: 0.5705 - val_accuracy: 0.8500 - val_loss: 0.5399
Epoch 8/100
2/2            0s 71ms/step -
accuracy: 0.8146 - loss: 0.5589 - val_accuracy: 0.9000 - val_loss: 0.5299
Epoch 9/100
2/2            0s 78ms/step -
accuracy: 0.8049 - loss: 0.5597 - val_accuracy: 0.9000 - val_loss: 0.5201
Epoch 10/100
2/2            0s 69ms/step -
accuracy: 0.7944 - loss: 0.5570 - val_accuracy: 1.0000 - val_loss: 0.5107
Epoch 11/100
2/2            0s 75ms/step -
accuracy: 0.8049 - loss: 0.5422 - val_accuracy: 1.0000 - val_loss: 0.5016
Epoch 12/100
2/2            0s 70ms/step -
accuracy: 0.8153 - loss: 0.5267 - val_accuracy: 1.0000 - val_loss: 0.4926
Epoch 13/100
2/2            0s 64ms/step -
accuracy: 0.8917 - loss: 0.5177 - val_accuracy: 1.0000 - val_loss: 0.4839
Epoch 14/100
2/2            0s 84ms/step -
accuracy: 0.9125 - loss: 0.5025 - val_accuracy: 1.0000 - val_loss: 0.4754
Epoch 15/100
```

```
2/2              0s 88ms/step -
accuracy: 0.9451 - loss: 0.5052 - val_accuracy: 1.0000 - val_loss: 0.4673
Epoch 16/100
2/2              0s 75ms/step -
accuracy: 0.9347 - loss: 0.4875 - val_accuracy: 1.0000 - val_loss: 0.4593
Epoch 17/100
2/2              0s 82ms/step -
accuracy: 0.9563 - loss: 0.4805 - val_accuracy: 1.0000 - val_loss: 0.4515
Epoch 18/100
2/2              0s 78ms/step -
accuracy: 0.9569 - loss: 0.4869 - val_accuracy: 1.0000 - val_loss: 0.4440
Epoch 19/100
2/2              0s 83ms/step -
accuracy: 0.9674 - loss: 0.4734 - val_accuracy: 1.0000 - val_loss: 0.4366
Epoch 20/100
2/2              0s 106ms/step -
accuracy: 0.9569 - loss: 0.4663 - val_accuracy: 1.0000 - val_loss: 0.4295
Epoch 21/100
2/2              0s 97ms/step -
accuracy: 0.9674 - loss: 0.4491 - val_accuracy: 1.0000 - val_loss: 0.4225
Epoch 22/100
2/2              0s 83ms/step -
accuracy: 0.9674 - loss: 0.4464 - val_accuracy: 1.0000 - val_loss: 0.4158
Epoch 23/100
2/2              0s 76ms/step -
accuracy: 0.9569 - loss: 0.4472 - val_accuracy: 1.0000 - val_loss: 0.4092
Epoch 24/100
2/2              0s 90ms/step -
accuracy: 0.9785 - loss: 0.4281 - val_accuracy: 1.0000 - val_loss: 0.4027
Epoch 25/100
2/2              0s 78ms/step -
accuracy: 0.9785 - loss: 0.4309 - val_accuracy: 1.0000 - val_loss: 0.3965
Epoch 26/100
2/2              0s 86ms/step -
accuracy: 0.9889 - loss: 0.4204 - val_accuracy: 1.0000 - val_loss: 0.3904
Epoch 27/100
2/2              0s 75ms/step -
accuracy: 0.9785 - loss: 0.4144 - val_accuracy: 1.0000 - val_loss: 0.3845
Epoch 28/100
2/2              0s 65ms/step -
accuracy: 0.9889 - loss: 0.4024 - val_accuracy: 1.0000 - val_loss: 0.3787
Epoch 29/100
2/2              0s 84ms/step -
accuracy: 0.9785 - loss: 0.4020 - val_accuracy: 1.0000 - val_loss: 0.3730
Epoch 30/100
2/2              0s 84ms/step -
accuracy: 0.9785 - loss: 0.3945 - val_accuracy: 1.0000 - val_loss: 0.3676
Epoch 31/100
```

```
2/2            0s 78ms/step -
accuracy: 0.9785 - loss: 0.3884 - val_accuracy: 1.0000 - val_loss: 0.3622
Epoch 32/100
2/2            0s 84ms/step -
accuracy: 0.9785 - loss: 0.3846 - val_accuracy: 1.0000 - val_loss: 0.3570
Epoch 33/100
2/2            0s 83ms/step -
accuracy: 0.9785 - loss: 0.3802 - val_accuracy: 1.0000 - val_loss: 0.3519
Epoch 34/100
2/2            0s 131ms/step -
accuracy: 0.9889 - loss: 0.3683 - val_accuracy: 1.0000 - val_loss: 0.3470
Epoch 35/100
2/2            0s 90ms/step -
accuracy: 0.9785 - loss: 0.3713 - val_accuracy: 1.0000 - val_loss: 0.3421
Epoch 36/100
2/2            0s 81ms/step -
accuracy: 0.9889 - loss: 0.3552 - val_accuracy: 1.0000 - val_loss: 0.3374
Epoch 37/100
2/2            0s 74ms/step -
accuracy: 0.9785 - loss: 0.3569 - val_accuracy: 1.0000 - val_loss: 0.3328
Epoch 38/100
2/2            0s 67ms/step -
accuracy: 0.9785 - loss: 0.3477 - val_accuracy: 1.0000 - val_loss: 0.3283
Epoch 39/100
2/2            0s 75ms/step -
accuracy: 0.9889 - loss: 0.3482 - val_accuracy: 1.0000 - val_loss: 0.3239
Epoch 40/100
2/2            0s 90ms/step -
accuracy: 0.9785 - loss: 0.3425 - val_accuracy: 1.0000 - val_loss: 0.3196
Epoch 41/100
2/2            0s 65ms/step -
accuracy: 0.9889 - loss: 0.3349 - val_accuracy: 1.0000 - val_loss: 0.3154
Epoch 42/100
2/2            0s 90ms/step -
accuracy: 0.9889 - loss: 0.3289 - val_accuracy: 1.0000 - val_loss: 0.3113
Epoch 43/100
2/2            0s 72ms/step -
accuracy: 0.9889 - loss: 0.3218 - val_accuracy: 1.0000 - val_loss: 0.3073
Epoch 44/100
2/2            0s 80ms/step -
accuracy: 0.9785 - loss: 0.3279 - val_accuracy: 1.0000 - val_loss: 0.3034
Epoch 45/100
2/2            0s 78ms/step -
accuracy: 0.9785 - loss: 0.3187 - val_accuracy: 1.0000 - val_loss: 0.2995
Epoch 46/100
2/2            0s 82ms/step -
accuracy: 0.9785 - loss: 0.3197 - val_accuracy: 1.0000 - val_loss: 0.2958
Epoch 47/100
```

```
2/2            0s 98ms/step -
accuracy: 0.9785 - loss: 0.3136 - val_accuracy: 1.0000 - val_loss: 0.2921
Epoch 48/100
2/2            0s 87ms/step -
accuracy: 0.9889 - loss: 0.3023 - val_accuracy: 1.0000 - val_loss: 0.2885
Epoch 49/100
2/2            0s 86ms/step -
accuracy: 0.9889 - loss: 0.3000 - val_accuracy: 1.0000 - val_loss: 0.2850
Epoch 50/100
2/2            0s 95ms/step -
accuracy: 0.9785 - loss: 0.3014 - val_accuracy: 1.0000 - val_loss: 0.2816
Epoch 51/100
2/2            0s 77ms/step -
accuracy: 0.9785 - loss: 0.3019 - val_accuracy: 1.0000 - val_loss: 0.2782
Epoch 52/100
2/2            0s 71ms/step -
accuracy: 0.9889 - loss: 0.2935 - val_accuracy: 1.0000 - val_loss: 0.2750
Epoch 53/100
2/2            0s 80ms/step -
accuracy: 0.9785 - loss: 0.2913 - val_accuracy: 1.0000 - val_loss: 0.2718
Epoch 54/100
2/2            0s 73ms/step -
accuracy: 0.9785 - loss: 0.2911 - val_accuracy: 1.0000 - val_loss: 0.2686
Epoch 55/100
2/2            0s 83ms/step -
accuracy: 0.9785 - loss: 0.2879 - val_accuracy: 1.0000 - val_loss: 0.2656
Epoch 56/100
2/2            0s 88ms/step -
accuracy: 0.9785 - loss: 0.2824 - val_accuracy: 1.0000 - val_loss: 0.2626
Epoch 57/100
2/2            0s 75ms/step -
accuracy: 0.9889 - loss: 0.2756 - val_accuracy: 1.0000 - val_loss: 0.2596
Epoch 58/100
2/2            0s 72ms/step -
accuracy: 0.9785 - loss: 0.2769 - val_accuracy: 1.0000 - val_loss: 0.2567
Epoch 59/100
2/2            0s 74ms/step -
accuracy: 0.9785 - loss: 0.2723 - val_accuracy: 1.0000 - val_loss: 0.2539
Epoch 60/100
2/2            0s 66ms/step -
accuracy: 1.0000 - loss: 0.2630 - val_accuracy: 1.0000 - val_loss: 0.2511
Epoch 61/100
2/2            0s 79ms/step -
accuracy: 1.0000 - loss: 0.2671 - val_accuracy: 1.0000 - val_loss: 0.2484
Epoch 62/100
2/2            0s 82ms/step -
accuracy: 1.0000 - loss: 0.2631 - val_accuracy: 1.0000 - val_loss: 0.2458
Epoch 63/100
```

```
2/2            0s 91ms/step -
accuracy: 1.0000 - loss: 0.2562 - val_accuracy: 1.0000 - val_loss: 0.2432
Epoch 64/100
2/2            0s 75ms/step -
accuracy: 1.0000 - loss: 0.2583 - val_accuracy: 1.0000 - val_loss: 0.2406
Epoch 65/100
2/2            0s 74ms/step -
accuracy: 1.0000 - loss: 0.2581 - val_accuracy: 1.0000 - val_loss: 0.2381
Epoch 66/100
2/2            0s 82ms/step -
accuracy: 1.0000 - loss: 0.2530 - val_accuracy: 1.0000 - val_loss: 0.2356
Epoch 67/100
2/2            0s 85ms/step -
accuracy: 1.0000 - loss: 0.2504 - val_accuracy: 1.0000 - val_loss: 0.2332
Epoch 68/100
2/2            0s 57ms/step -
accuracy: 1.0000 - loss: 0.2474 - val_accuracy: 1.0000 - val_loss: 0.2308
Epoch 69/100
2/2            0s 75ms/step -
accuracy: 1.0000 - loss: 0.2443 - val_accuracy: 1.0000 - val_loss: 0.2285
Epoch 70/100
2/2            0s 77ms/step -
accuracy: 1.0000 - loss: 0.2404 - val_accuracy: 1.0000 - val_loss: 0.2262
Epoch 71/100
2/2            0s 81ms/step -
accuracy: 1.0000 - loss: 0.2353 - val_accuracy: 1.0000 - val_loss: 0.2240
Epoch 72/100
2/2            0s 79ms/step -
accuracy: 1.0000 - loss: 0.2364 - val_accuracy: 1.0000 - val_loss: 0.2218
Epoch 73/100
2/2            0s 82ms/step -
accuracy: 1.0000 - loss: 0.2372 - val_accuracy: 1.0000 - val_loss: 0.2196
Epoch 74/100
2/2            0s 73ms/step -
accuracy: 1.0000 - loss: 0.2371 - val_accuracy: 1.0000 - val_loss: 0.2175
Epoch 75/100
2/2            0s 138ms/step -
accuracy: 1.0000 - loss: 0.2308 - val_accuracy: 1.0000 - val_loss: 0.2154
Epoch 76/100
2/2            0s 87ms/step -
accuracy: 1.0000 - loss: 0.2316 - val_accuracy: 1.0000 - val_loss: 0.2134
Epoch 77/100
2/2            0s 87ms/step -
accuracy: 1.0000 - loss: 0.2243 - val_accuracy: 1.0000 - val_loss: 0.2114
Epoch 78/100
2/2            0s 66ms/step -
accuracy: 1.0000 - loss: 0.2216 - val_accuracy: 1.0000 - val_loss: 0.2094
Epoch 79/100
```

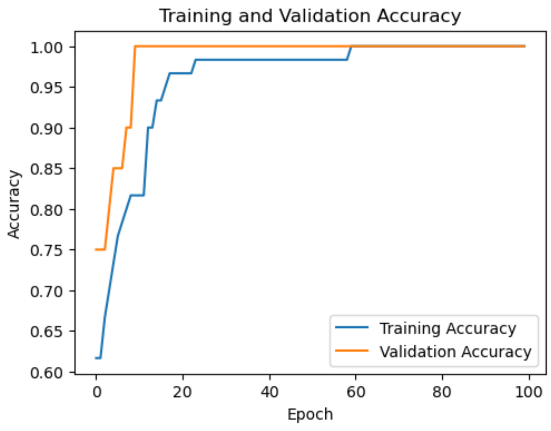
```
2/2             0s 68ms/step -
accuracy: 1.0000 - loss: 0.2257 - val_accuracy: 1.0000 - val_loss: 0.2074
Epoch 80/100
2/2             0s 88ms/step -
accuracy: 1.0000 - loss: 0.2152 - val_accuracy: 1.0000 - val_loss: 0.2055
Epoch 81/100
2/2             0s 88ms/step -
accuracy: 1.0000 - loss: 0.2180 - val_accuracy: 1.0000 - val_loss: 0.2037
Epoch 82/100
2/2             0s 72ms/step -
accuracy: 1.0000 - loss: 0.2120 - val_accuracy: 1.0000 - val_loss: 0.2018
Epoch 83/100
2/2             0s 84ms/step -
accuracy: 1.0000 - loss: 0.2126 - val_accuracy: 1.0000 - val_loss: 0.2000
Epoch 84/100
2/2             0s 83ms/step -
accuracy: 1.0000 - loss: 0.2107 - val_accuracy: 1.0000 - val_loss: 0.1982
Epoch 85/100
2/2             0s 72ms/step -
accuracy: 1.0000 - loss: 0.2152 - val_accuracy: 1.0000 - val_loss: 0.1965
Epoch 86/100
2/2             0s 81ms/step -
accuracy: 1.0000 - loss: 0.2073 - val_accuracy: 1.0000 - val_loss: 0.1948
Epoch 87/100
2/2             0s 83ms/step -
accuracy: 1.0000 - loss: 0.2060 - val_accuracy: 1.0000 - val_loss: 0.1931
Epoch 88/100
2/2             0s 87ms/step -
accuracy: 1.0000 - loss: 0.2057 - val_accuracy: 1.0000 - val_loss: 0.1914
Epoch 89/100
2/2             0s 70ms/step -
accuracy: 1.0000 - loss: 0.2071 - val_accuracy: 1.0000 - val_loss: 0.1898
Epoch 90/100
2/2             0s 76ms/step -
accuracy: 1.0000 - loss: 0.2031 - val_accuracy: 1.0000 - val_loss: 0.1882
Epoch 91/100
2/2             0s 84ms/step -
accuracy: 1.0000 - loss: 0.2006 - val_accuracy: 1.0000 - val_loss: 0.1866
Epoch 92/100
2/2             0s 73ms/step -
accuracy: 1.0000 - loss: 0.1966 - val_accuracy: 1.0000 - val_loss: 0.1850
Epoch 93/100
2/2             0s 85ms/step -
accuracy: 1.0000 - loss: 0.1927 - val_accuracy: 1.0000 - val_loss: 0.1835
Epoch 94/100
2/2             0s 74ms/step -
accuracy: 1.0000 - loss: 0.1956 - val_accuracy: 1.0000 - val_loss: 0.1820
Epoch 95/100
```

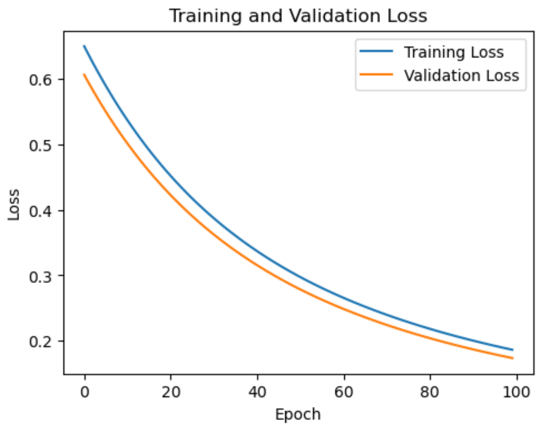
```
2/2             0s 86ms/step -
accuracy: 1.0000 - loss: 0.1907 - val_accuracy: 1.0000 - val_loss: 0.1805
Epoch 96/100
2/2             0s 70ms/step -
accuracy: 1.0000 - loss: 0.1915 - val_accuracy: 1.0000 - val_loss: 0.1790
Epoch 97/100
2/2             0s 83ms/step -
accuracy: 1.0000 - loss: 0.1860 - val_accuracy: 1.0000 - val_loss: 0.1776
Epoch 98/100
2/2             0s 65ms/step -
accuracy: 1.0000 - loss: 0.1905 - val_accuracy: 1.0000 - val_loss: 0.1762
Epoch 99/100
2/2             0s 76ms/step -
accuracy: 1.0000 - loss: 0.1862 - val_accuracy: 1.0000 - val_loss: 0.1748
Epoch 100/100
2/2             0s 83ms/step -
accuracy: 1.0000 - loss: 0.1850 - val_accuracy: 1.0000 - val_loss: 0.1734

Test Loss: 0.1410
Test Accuracy: 1.0000
1/1             0s 60ms/step

Sample Predictions (First 10 instances):
True: 0, Predicted: 0
True: 1, Predicted: 1
True: 0, Predicted: 0
True: 0, Predicted: 0
True: 0, Predicted: 0
True: 1, Predicted: 1
True: 1, Predicted: 1
True: 1, Predicted: 1
True: 0, Predicted: 0
True: 0, Predicted: 0

Confusion Matrix:
[[12  0]
 [ 0  8]]
```


Training and Validation Accuracy
Accuracy
Epoch
Training Accuracy
Validation Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
20
40
60
80
100


Training and Validation Loss
Loss
Epoch
Training Loss
Validation Loss
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100

```
[ ]:
```

perceptron_sklearn

June 5, 2025

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

[3]:
```python
# Load Iris dataset and select two classes (Setosa and Versicolor)
iris = load_iris()
X = iris.data[iris.target != 2, :2]  # Use first two features, exclude Virginica
y = iris.target[iris.target != 2]    # Binary classification (Setosa=0,␣
 ↪Versicolor=1)
```

[5]:
```python
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[7]:
```python
# Scale the features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[9]:
```python
# Create and train the Perceptron
model = Perceptron(max_iter=1000, tol=1e-3, random_state=42)
model.fit(X_train, y_train)
```

[9]:
```
Perceptron(random_state=42)
```

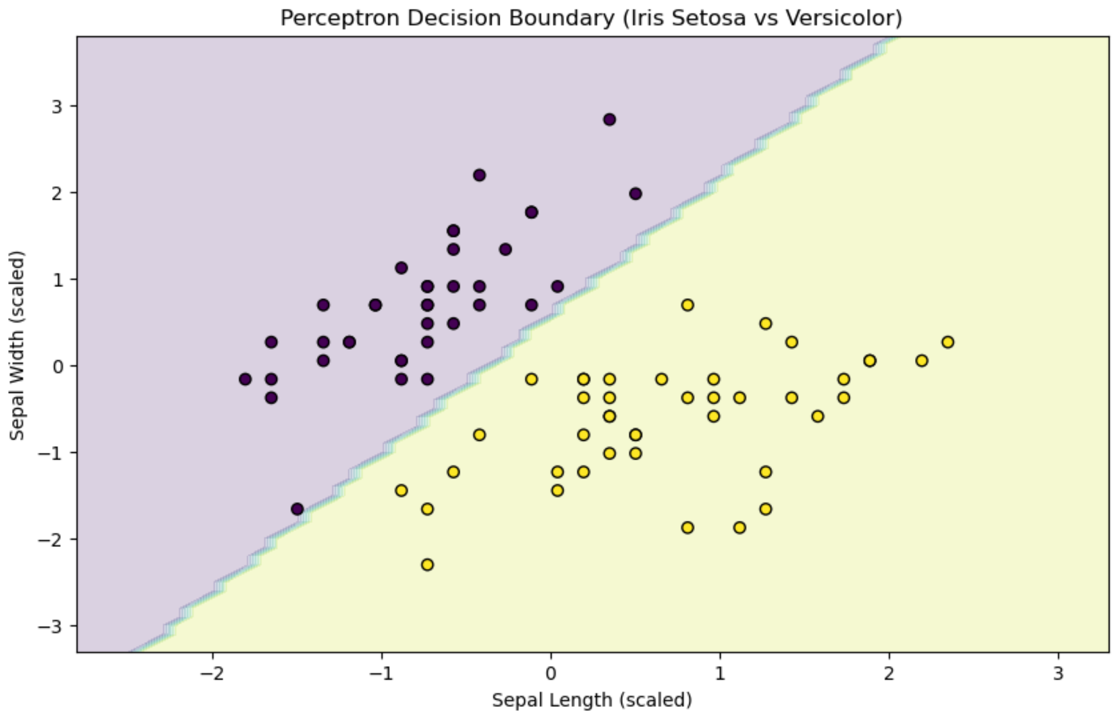
[11]:
```python
# Make predictions and evaluate
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

[13]:
```python
# Visualize decision boundary
plt.figure(figsize=(10, 6))
```

```
x_min, x_max = X_train[:, 0].min() - 1, X_train[:, 0].max() + 1
y_min, y_max = X_train[:, 1].min() - 1, X_train[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), np.arange(y_min, y_max, 0.1))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.2, cmap='viridis')
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='viridis',␣
 ↪marker='o', edgecolor='k')
plt.xlabel('Sepal Length (scaled)')
plt.ylabel('Sepal Width (scaled)')
plt.title('Perceptron Decision Boundary (Iris Setosa vs Versicolor)')
plt.show()
```



```
[ ]: #make necessary adjustments to the code
     # Save the model if needed
```