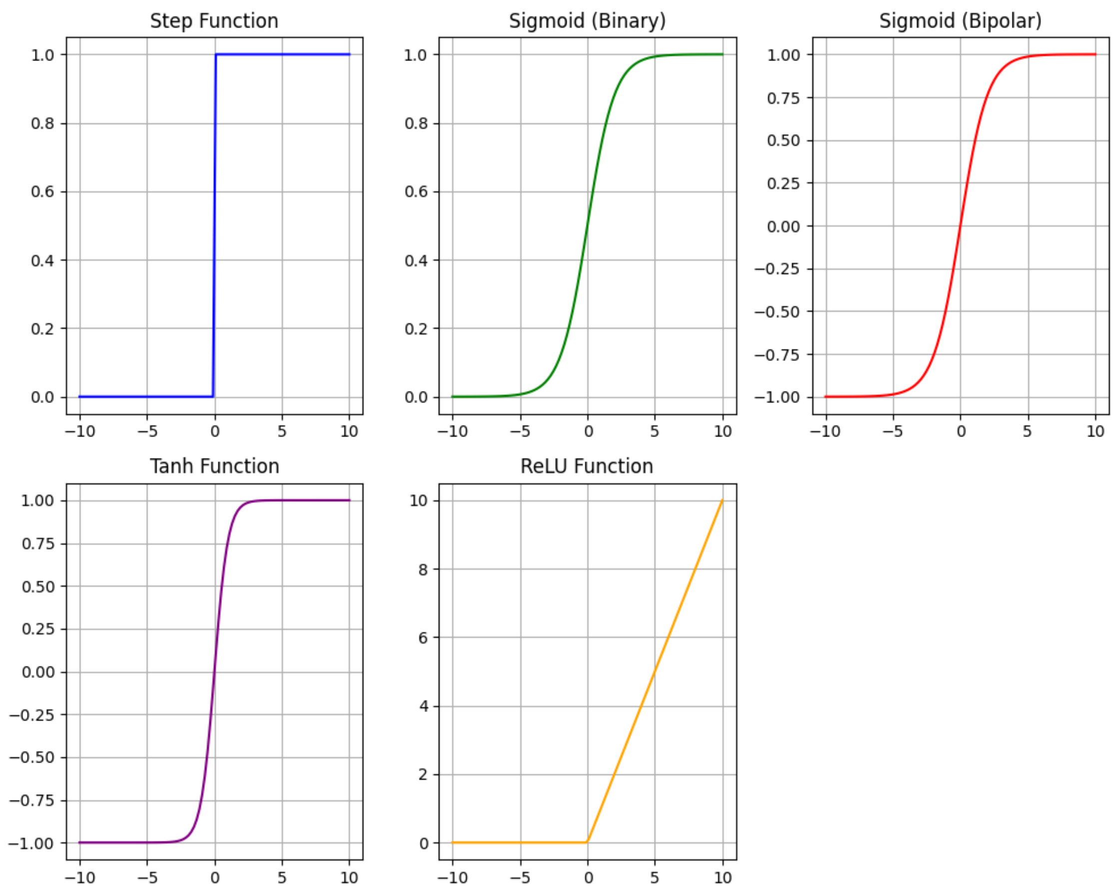
riya-151-lab2

September 24, 2024

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt

# 1. Define Activation Functions

# Step Function: Outputs 1 if input >= 0, otherwise 0. Simple, but not commonly␣
↪used in modern neural networks.
def step_function(x):
    return np.where(x >= 0, 1, 0)

# Sigmoid (Binary): Maps input to the range (0, 1), useful for binary␣
↪classification tasks.
def sigmoid_binary(x):
    return 1 / (1 + np.exp(-x))

# Sigmoid (Bipolar): Maps input to the range (-1, 1), useful when outputs need␣
↪negative values.
def sigmoid_bipolar(x):
    return 2 / (1 + np.exp(-x)) - 1

# Tanh: Outputs values between -1 and 1, often used in hidden layers as it␣
↪centers data around zero.
def tanh_function(x):
    return np.tanh(x)

# ReLU: Outputs the input if positive, else 0. Widely used in deep networks for␣
↪its simplicity and efficiency.
def relu_function(x):
    return np.maximum(0, x)

# 2. Visualization

# Create 100 values between -10 and 10 for plotting.
x = np.linspace(-10, 10, 100)

# Set up a grid of plots with a figure size of 10x8.
plt.figure(figsize=(10, 8))
```

```python
# Step Function plot
plt.subplot(2, 3, 1)  # 2 rows, 3 columns, position 1
plt.plot(x, step_function(x), label="Step Function", color='blue')  # Blue line
plt.title("Step Function")  # Add title
plt.grid(True)  # Add grid

# Sigmoid (Binary) plot
plt.subplot(2, 3, 2)  # Position 2
plt.plot(x, sigmoid_binary(x), label="Sigmoid (Binary)", color='green')  # Green line
plt.title("Sigmoid (Binary)")
plt.grid(True)

# Sigmoid (Bipolar) plot
plt.subplot(2, 3, 3)  # Position 3
plt.plot(x, sigmoid_bipolar(x), label="Sigmoid (Bipolar)", color='red')  # Red line
plt.title("Sigmoid (Bipolar)")
plt.grid(True)

# Tanh plot
plt.subplot(2, 3, 4)  # Position 4
plt.plot(x, tanh_function(x), label="Tanh", color='purple')  # Purple line
plt.title("Tanh Function")
plt.grid(True)

# ReLU plot
plt.subplot(2, 3, 5)  # Position 5
plt.plot(x, relu_function(x), label="ReLU", color='orange')  # Orange line
plt.title("ReLU Function")
plt.grid(True)

# Adjust layout to avoid overlapping elements.
plt.tight_layout()

# Show the plots
plt.show()
```



[3]:
```
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.metrics import accuracy_score
import numpy as np

# Set seed values for consistent results across runs
np.random.seed(42)  # NumPy seed
tf.random.set_seed(42)  # TensorFlow seed

# XOR dataset (input and corresponding output pairs)
# X: Inputs for XOR problem (binary combinations)
# y: Expected outputs (1 for different, 0 for same input pairs)
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # XOR inputs
y = np.array([0, 1, 1, 0])  # XOR outputs

# Function to build a neural network model
# The hidden layer has 4 neurons with a customizable activation function
# The output layer has 1 neuron using 'sigmoid' for binary classification
def create_model(activation_function):
```

```python
    model = models.Sequential()  # Initialize a Sequential model
    model.add(layers.Dense(4, input_dim=2, activation=activation_function))  # Hidden layer: 4 neurons, custom activation
    model.add(layers.Dense(1, activation='sigmoid'))  # Output layer: 1 neuron, using sigmoid for binary output

    # Compile the model: Adam optimizer, binary crossentropy for loss, and track accuracy
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Loop through different activation functions for comparison
activations = ['sigmoid', 'tanh', 'relu']  # List of activation functions to try

for activation in activations:
    print(f"\nTraining with {activation} activation:")  # Indicate the current activation function being used

    # Build and compile model with the selected activation function
    model = create_model(activation)

    # Train the model with XOR data (100 epochs), suppress verbose output
    model.fit(X, y, epochs=100, verbose=0)

    # Generate predictions for the XOR inputs
    # Predictions are probabilities, convert them to binary (0 or 1) using a threshold of 0.5
    predictions = (model.predict(X) > 0.5).astype("int32")

    # Evaluate the model's accuracy by comparing predictions with actual outputs (y)
    accuracy = accuracy_score(y, predictions)

    # Print the accuracy for the current model
    print(f"Accuracy with {activation}: {accuracy * 100:.2f}%")
```

```
Training with sigmoid activation:
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/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)
```

```
1/1            0s 13ms/step
Accuracy with sigmoid: 50.00%
```

```
Training with tanh activation:

/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/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)

WARNING:tensorflow:5 out of the last 5 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x31b911bc0> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1            0s 12ms/step
Accuracy with tanh: 50.00%

Training with relu activation:

/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/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)

WARNING:tensorflow:6 out of the last 6 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x31a0c2200> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1            0s 14ms/step
Accuracy with relu: 100.00%
```

Interpretation: Tanh: Scales inputs between -1 and 1, centered at zero, making it useful for handling negative values. ReLU: Efficient for deep networks, outputs zero for negative inputs and the input itself for positive values, preventing saturation. Sigmoid: Squeezes inputs between 0 and 1, prone to vanishing gradients, but suitable for binary classification tasks.