Oishi_136_Lab2

September 24, 2024

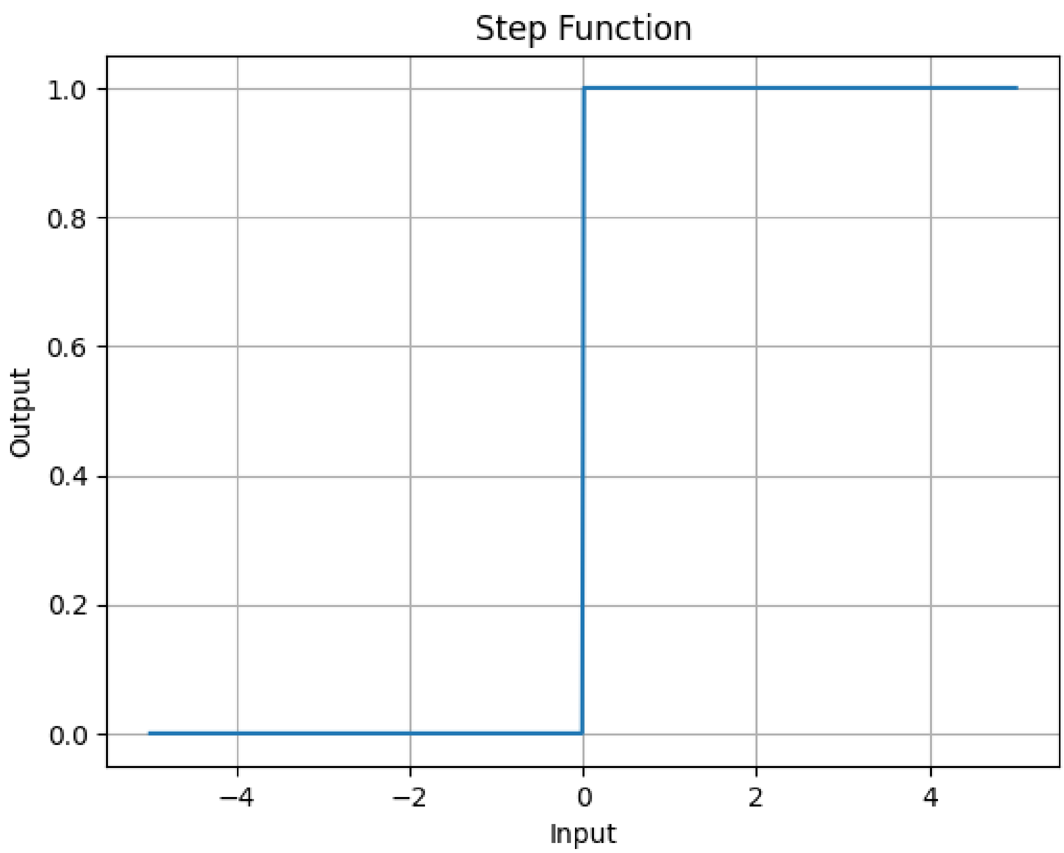
Regular lab Question – 2 1. Exploring Activation Functions in Neural Networks

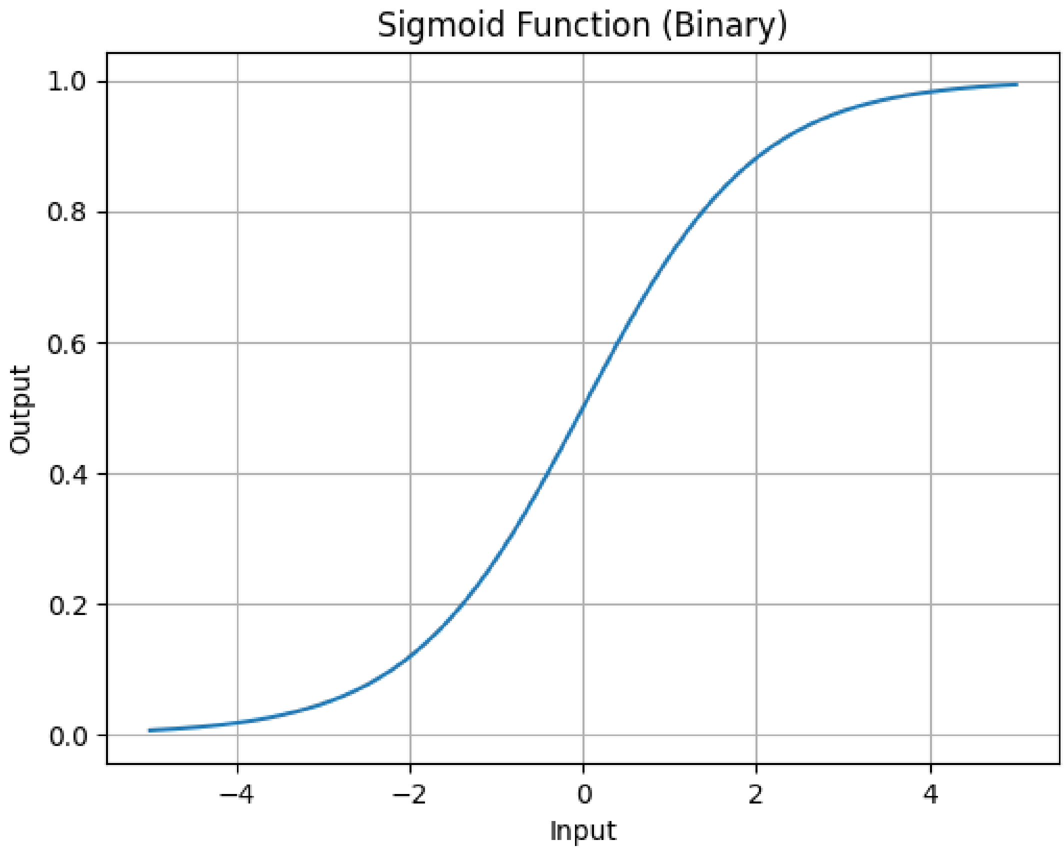
1. Implement and Visualize Activation Functions: o Implement the following activation functions in Python: Step Function Sigmoid Function (Binary and Bipolar) Tanh Function ReLU Function

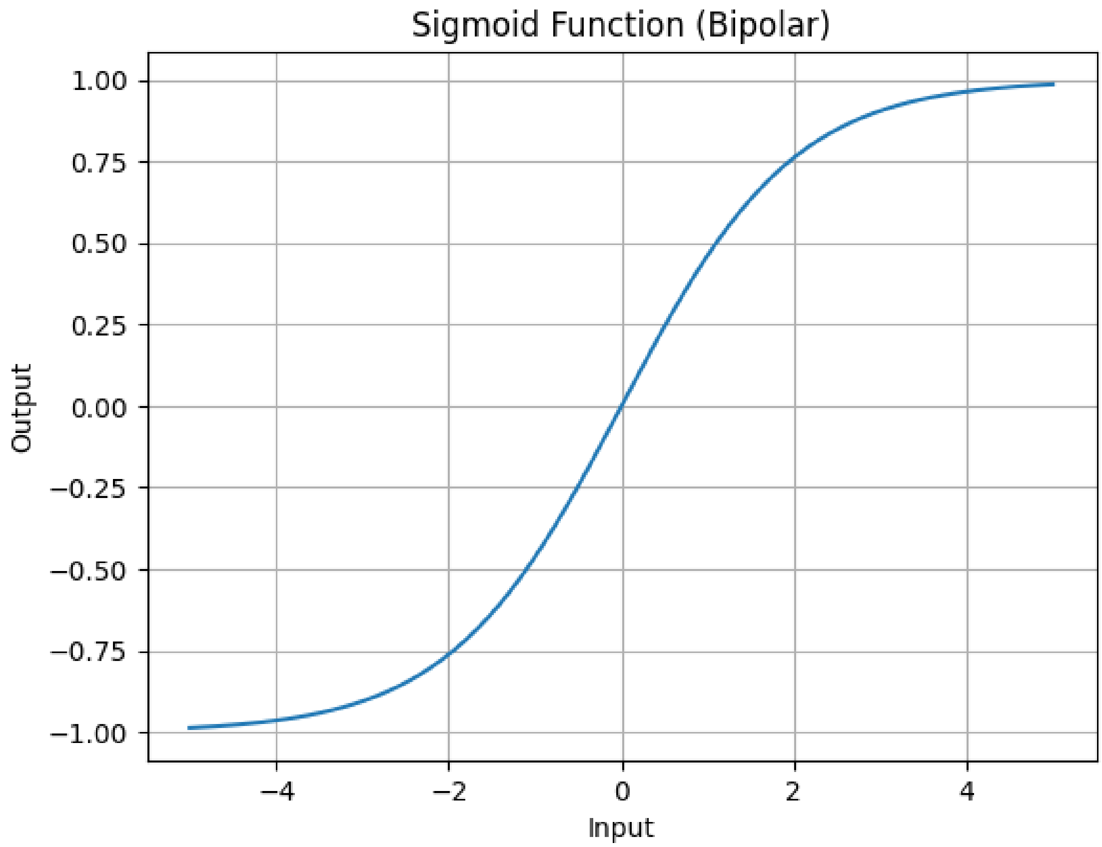
[7]:
```
import numpy as np
import matplotlib.pyplot as plt
```

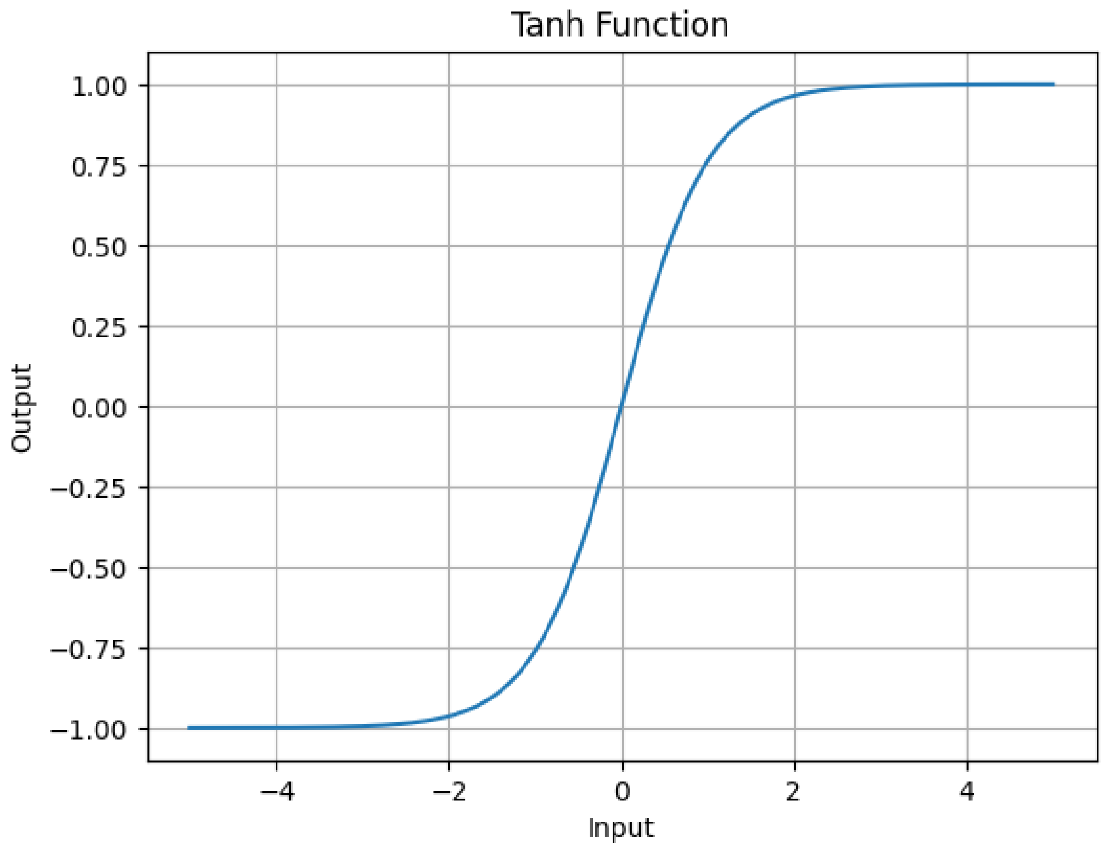
[8]:
```
# Step Function
def step_function(x):
    return np.where(x >= 0, 1, 0)

# Sigmoid Function (Binary)
def sigmoid_binary(x):
    return 1 / (1 + np.exp(-x))

# Sigmoid Function (Bipolar)
def sigmoid_bipolar(x):
    return 2 / (1 + np.exp(-x)) - 1

# Tanh Function
def tanh_function(x):
    return np.tanh(x)

# ReLU Function
def relu_function(x):
    return np.maximum(0, x)

# Function to plot activation functions
def plot_activation_function(x, y, title):
    plt.plot(x, y)
    plt.title(title)
    plt.xlabel('Input')
    plt.ylabel('Output')
    plt.grid(True)
    plt.show()
```

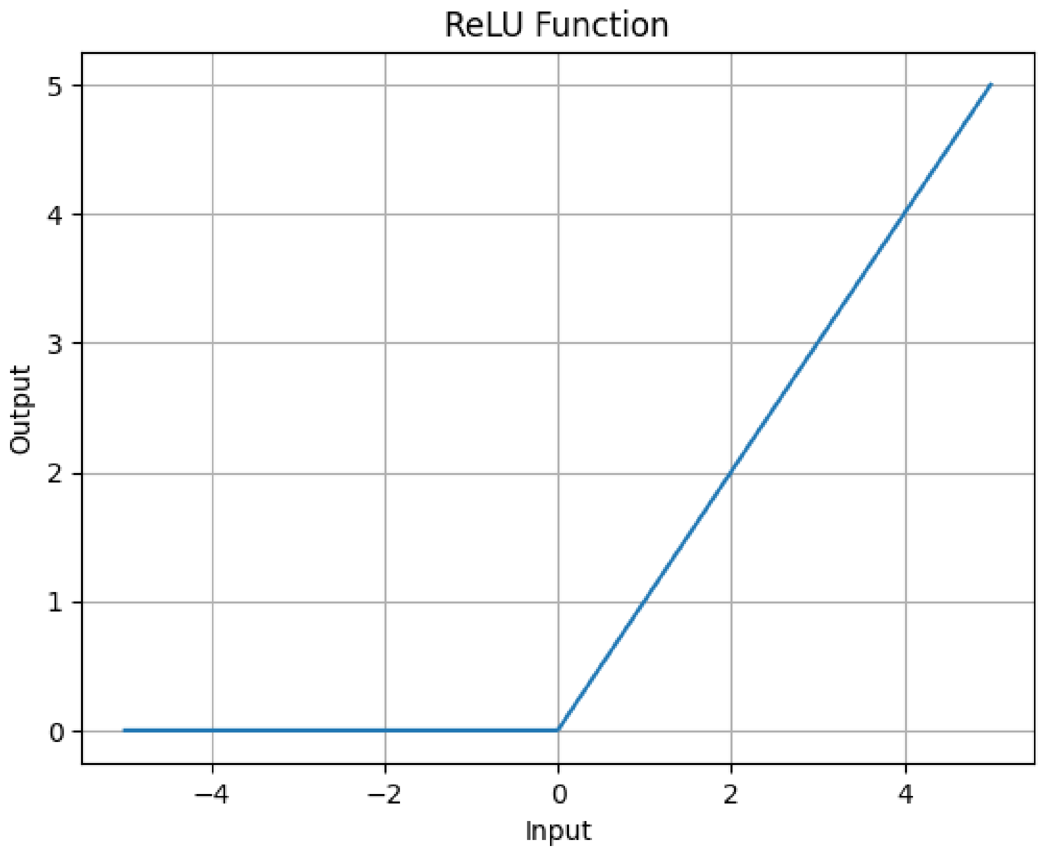
[9]:
```
# Input values for plotting
x = np.linspace(-5, 5, 500)


# Plot each function
plot_activation_function(x, step_function(x), "Step Function")
plot_activation_function(x, sigmoid_binary(x), "Sigmoid Function (Binary)")
plot_activation_function(x, sigmoid_bipolar(x), "Sigmoid Function (Bipolar)")
plot_activation_function(x, tanh_function(x), "Tanh Function")
plot_activation_function(x, relu_function(x), "ReLU Function")
```




Sigmoid Function (Binary)
Output
1.0
0.8
0.6
0.4
0.2
0.0
−4
−2
0
2
4
Input


Sigmoid Function (Bipolar)
Output
Input
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−4
−2
0
2
4


Tanh Function
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Output
−4
−2
0
2
4
Input



2. Implement a Simple Neural Network: • Create a simple neural network with one hidden layer using each activation function (sigmoid, tanh, and ReLU). • Train the network on a binary classification task (e.g., XOR problem) using a small dataset. • Compare the performance of the neural network with different activation functions.

[10]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.metrics import accuracy_score
```

[11]:
```
# XOR dataset
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 0])

# Function to create and train neural network with different activation␣
↪functions
def create_and_train_nn(activation_function, name):
    model = Sequential()
    model.add(Dense(2, input_dim=2, activation=activation_function))  # hidden␣
↪layer
```

```
    model.add(Dense(1, activation='sigmoid'))  # output layer (binary␣
↪classification)

    model.compile(loss='binary_crossentropy', optimizer='adam',␣
↪metrics=['accuracy'])

    # Train the model
    model.fit(X, y, epochs=100, verbose=0)

    # Predictions and performance
    predictions = model.predict(X)
    predictions = np.where(predictions > 0.5, 1, 0)
    accuracy = accuracy_score(y, predictions)
    print(f"{name} Activation - Accuracy: {accuracy * 100:.2f}%")

# Train and evaluate the neural network with different activation functions
create_and_train_nn('sigmoid', 'Sigmoid')
create_and_train_nn('tanh', 'Tanh')
create_and_train_nn('relu', 'ReLU')
```

```
C:\Users\USER\AppData\Roaming\Python\Python312\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 96ms/step
Sigmoid Activation - Accuracy: 50.00%
```

```
C:\Users\USER\AppData\Roaming\Python\Python312\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
0x0000021F7CB116C0> 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 93ms/step
Tanh Activation - Accuracy: 50.00%
```

```
C:\Users\USER\AppData\Roaming\Python\Python312\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
0x0000021F7DCF40E0> 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 73ms/step
ReLU Activation - Accuracy: 75.00%
1/1            0s 73ms/step
ReLU Activation - Accuracy: 75.00%
```

According to the accuracy of the activation functions we can decide the performance of the neural network. ReLU has 75% accuracy making it the best activation function for this neural network compared to the Sigmoid and TanH functions which is only 50%.