Assignment 1

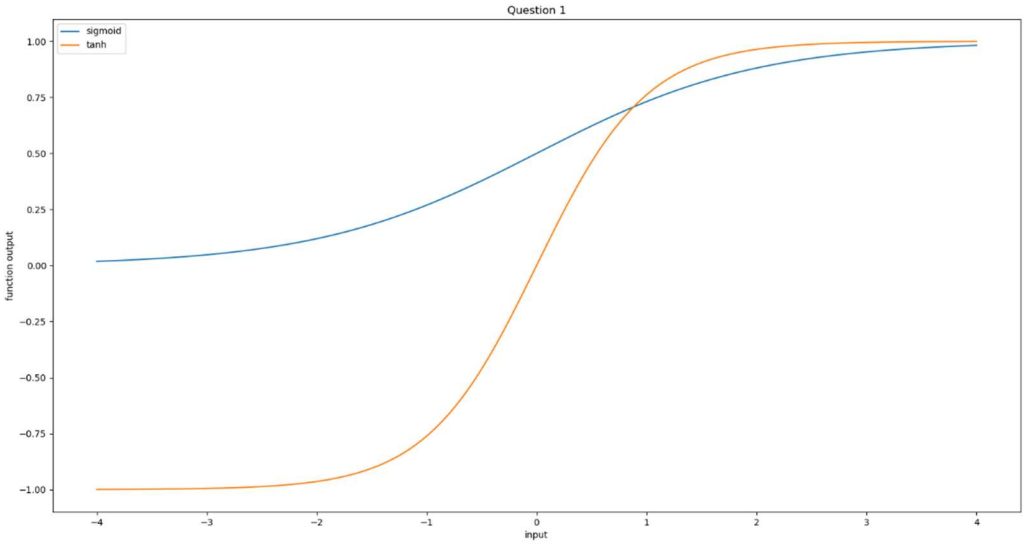
Jacob Taylor Cassady

Deep Learning

January 23, 2020

1 SHOW THAT THE TANH FUNCTION IS A RE-SCALED SIGMOID FUNCTION WITH BOTH HORIZONTAL AND VERTICAL STRETCHING, AS WELL AS VERTICAL TRANSLATION: $\tanh(v) = 2sigmoid(2v) - 1$

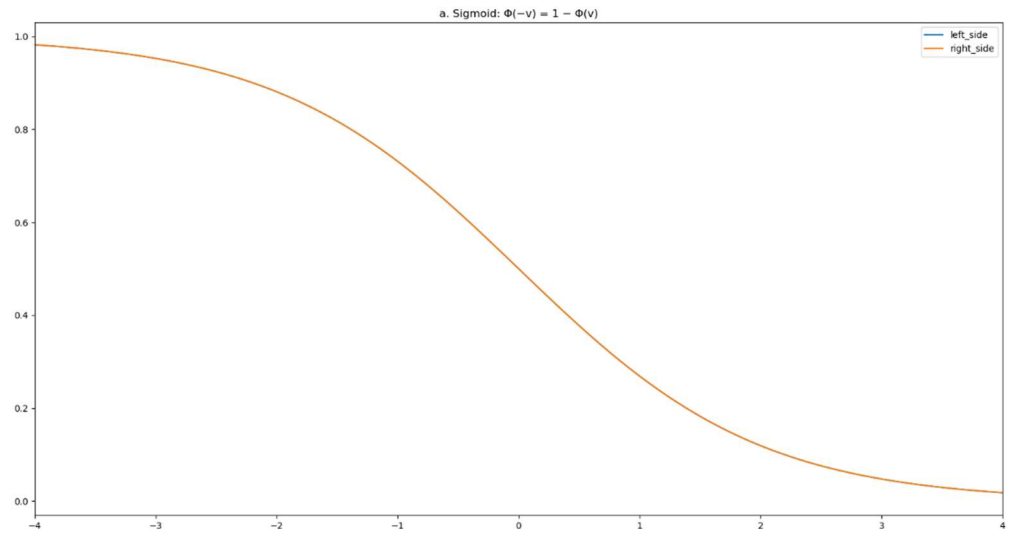
```
def question_1():
    """
        Show that the tanh function is a re-scaled sigmoid function with both
        horizontal and vertical streching, as well as vertical translation:
            tanh(v) = 2sigmoid(2v) - 1
    """
    print("how that the tanh function is a re-scaled sigmoid function with both horizontal and vertical streching, as well as vertical translation:")
    print("tanh(v) = 2*sigmoid(2v) - 1")

    input_array = np.linspace(-4, 4, 8000)

    sigmoid_output = sigmoid(input_array)
    tanh_output = tanh(input_array)

    plt.plot(input_array, sigmoid_output, label="sigmoid")
    plt.plot(input_array, tanh_output, label="tanh")
    plt.title("Question 1")
    plt.xlabel("input")
    plt.ylabel("function output")
    plt.legend()
    plt.show()
```



2 SHOW THE FOLLOWING PROPERTIES OF THE SIGMOID AND TANH ACTIVATION FUNCTIONS:

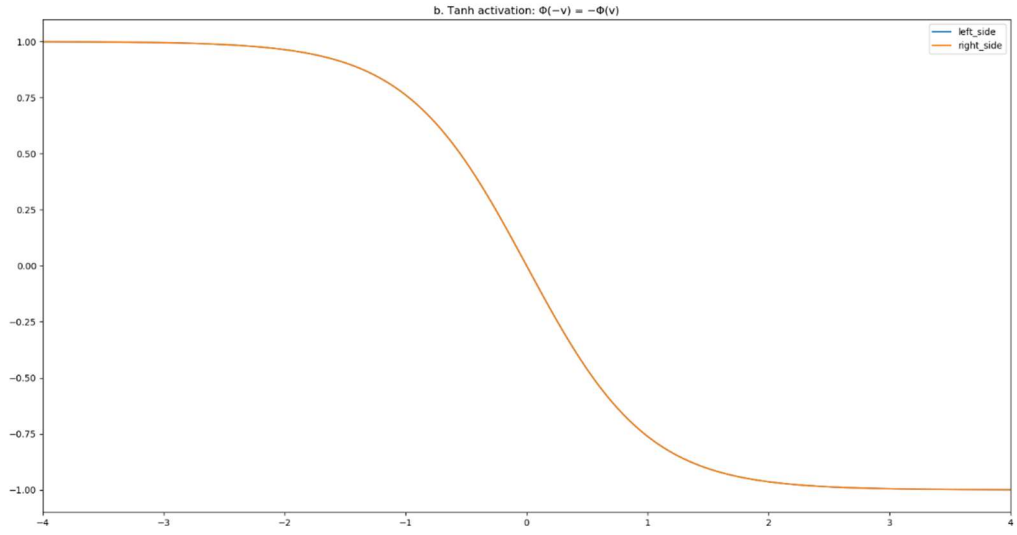
```
def question_2():
    """
        2. Show the following properties of the sigmoid and tanh activation functions:
            a. Sigmoid: Φ(-v) = 1 - Φ(v)
            b. Tanh activation: Φ(-v) = -Φ(v)
            c. Hard tanh activation: Φ(-v) = -Φ(v)
    """
    print("2. Show the following properties of the sigmoid and tanh activation functions:")
    input_array = np.linspace(-4, 4, 8000)

    print("\ta. Sigmoid: Φ(-v) = 1 - Φ(v)")
    left_side = sigmoid(-input_array)
    right_side = 1 - sigmoid(input_array)
    plt.title("a. Sigmoid: Φ(-v) = 1 - Φ(v)")
    plt.plot(input_array, left_side, label="left_side")
    plt.plot(input_array, right_side, label="right_side")
    plt.xlim(-4, 4)
    plt.legend()
    plt.show()

    print("\tb. Tanh activation: Φ(-v) = -Φ(v)")
    left_side = tanh(-input_array)
    right_side = -1 * tanh(input_array)
    plt.title("b. Tanh activation: Φ(-v) = -Φ(v)")
    plt.plot(input_array, left_side, label="left_side")
    plt.plot(input_array, right_side, label="right_side")
    plt.xlim(-4, 4)
    plt.legend()
    plt.show()

    print("\tc. Hard tanh activation: Φ(-v) = -Φ(v)")
    left_side = hardtanh(-input_array)
    right_side = -1 * hardtanh(input_array)
    plt.title("c. Hard tanh activation: Φ(-v) = -Φ(v)")
    plt.plot(input_array, left_side, label="left_side")
    plt.plot(input_array, right_side, label="right_side")
    plt.xlim(-4, 4)
    plt.legend()
    plt.show()
```

2.1 SIGMOID: $\Phi(-v) = 1 - \Phi(v)$



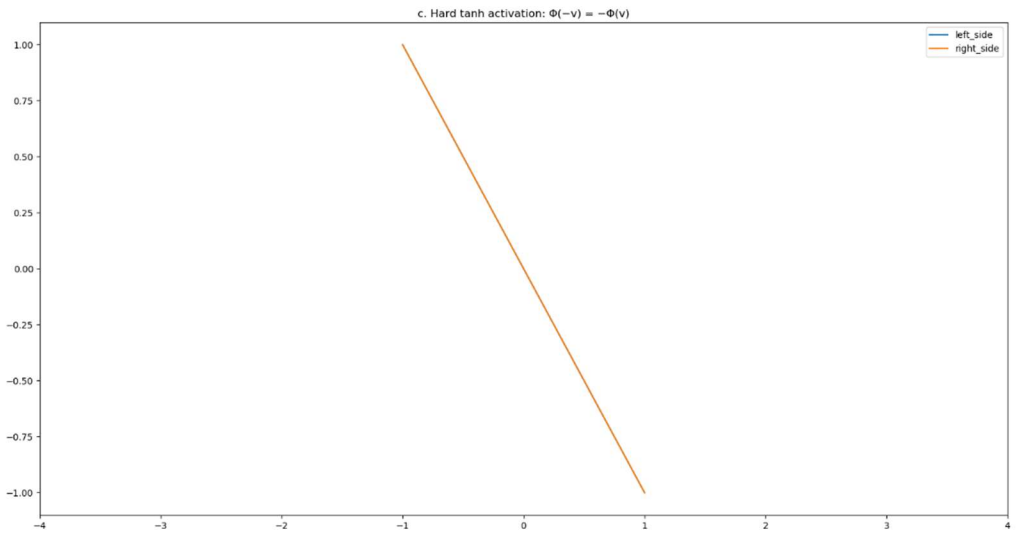
Note: Both function outputs are directly on top of each other.

2.2 TANH ACTIVATION: $\Phi(-v) = -\Phi(v)$



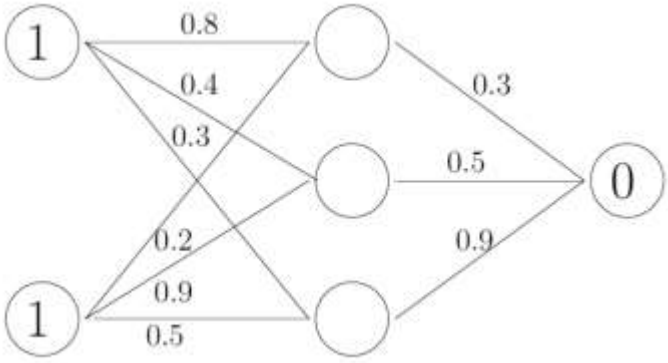
Note: Both function outputs are directly on top of each other.

2.3 HARD TANH ACTIVATION: Φ(−v) = −Φ(v)



Note: Both function outputs are directly on top of each other.

3 CONSIDER THE NEURAL NETWORK:



4 ALL THE ACTIVATION FUNCTIONS FROM THE NEURONS IN THE HIDDEN LAYER ARE SIGMOIDS AND THE ERROR IS CALCULATED BY USING SQUARED ERROR FUNCTION:

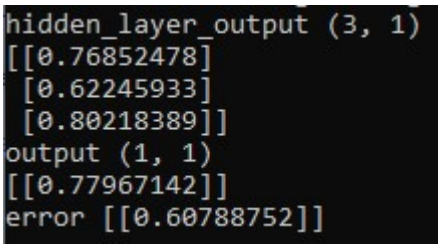
4.1 DESCRIBE ALL THE ESSENTIAL PARTS FROM THE NEURAL NETWORK.

There are three neurons in the hidden layer and one output neuron. The initial weights for the neurons are given in the lines connected to the left of the neuron and the bias is to the right of the neuron. The input features are {1, 1}. The target is {0}.

4.2 GIVEN THE INPUT {1,1} COMPUTE THE PREDICTED OUTPUT OF THE NETWORK STEP BY STEP AND CALCULATE THE ERROR IF THE TARGET OUTPUT IS 0.

```
class Perceptron(object):
    def __init__(self, number_of_neurons, input_size, Weights=None, bias=None, transfer_function=sigmoid):
        if Weights is None:
            self.Weights = np.random.rand(number_of_neurons, input_size)
        else:
            self.Weights = Weights
        if bias is None:
            self.bias = np.random.rand(number_of_neurons, 1)
        else:
            self.bias = bias
        self.activation_function = sigmoid

    def classify(self, prototype):
        net_input = self.Weights.dot(prototype) + self.bias
        return self.activation_function(net_input)
```

```
def question_3():
    input_array = np.array([1, 1]).reshape((2, 1))
    weights = np.array([[0.8, 0.4, 0.3],
                        [0.2, 0.9, 0.5]]).reshape((3, 2))
    bias = np.array([0, 0, 0]).reshape((3, 1))
    hidden_layer = Perceptron(number_of_neurons=3, input_size=2, Weights=weights, bias=bias)
    hidden_layer_output = hidden_layer.classify(input_array)

    print("hidden_layer_output", hidden_layer_output.shape)
    print(hidden_layer_output)

    weights = np.array([0.3, 0.5, 0.9]).reshape((1, 3))
    bias = np.array([0]).reshape((1, 1))
    output_layer = Perceptron(number_of_neurons=1, input_size=3, Weights=weights, bias=bias)
    output = output_layer.classify(hidden_layer_output)
    print("output", output.shape)
    print(output)

    def error(expected, actual):
        return (expected - actual)**2

    print("error", error(output, 0))
```



4.3 COMPUTE STEP BY STEP 2 TRAINING EPOCHS USING BACK-PROPAGATION ALGORITHM.