Activation functions

Autumn 2020

```
###############################
##                           ##
##  Deep Learning in Python  ##
##                           ##
###############################
```

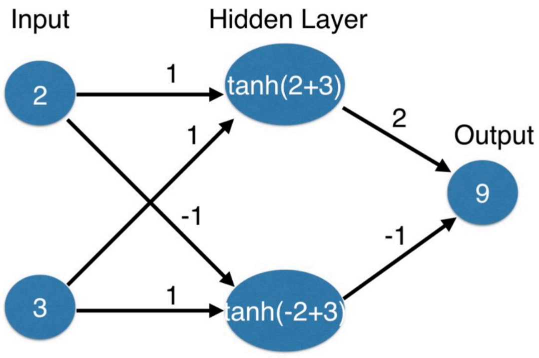
§1 Introduction to Deep Learning in Python

§1.1 Basics of deep learning and neural networks

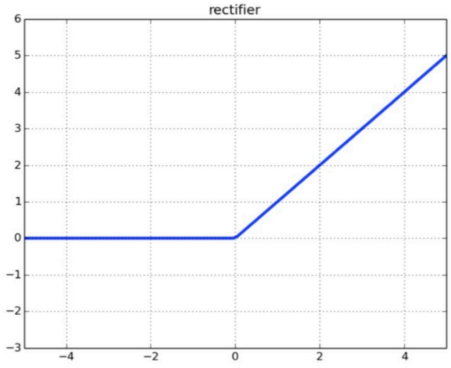
§1.1.3 Activation functions

1. How do activation functions work?

It is applied to node inputs to produce node output.



2. What is the rectified linear activation (ReLU)?



$$RELU(x) = \begin{cases} 0 & if \quad x < 0 \\ x & if \quad x \geq 0 \end{cases}$$

3. Code of activation functions:

[1]:

```python
import numpy as np

input_data = np.array([-1, 2])
weights = {
    'node_0': np.array([3, 3]),
    'node_1': np.array([1, 5]),
    'output': np.array([2, -1])
}
node_0_input = (input_data * weights['node_0']).sum()
node_0_output = np.tanh(node_0_input)
node_1_input = (input_data * weights['node_1']).sum()
node_1_output = np.tanh(node_1_input)
hidden_layer_outputs = np.array([node_0_output, node_1_output])
output = (hidden_layer_outputs * weights['output']).sum()

print(output)
```

```
0.9901095378334199
```

4. Practice exercises for activation functions:

▶ **Data pre-loading:**

[2]:

```python
import numpy as np

input_data = np.array([3, 5])

weights = {
    'node_0': np.array([2, 4]),
    'node_1': np.array([4, -5]),
    'output': np.array([2, 7])
}
```

▶ **The rectified linear activation function practice:**

[3]:

```python
def relu(input):
    '''Define your relu activation function here'''
    # Calculate the value for the output of the relu function: output
    output = max(0, input)

    # Return the value just calculated
    return (output)
```

```python

# Calculate node 0 value: node_0_output
node_0_input = (input_data * weights['node_0']).sum()
node_0_output = relu(node_0_input)

# Calculate node 1 value: node_1_output
node_1_input = (input_data * weights['node_1']).sum()
node_1_output = relu(node_1_input)

# Put node values into array: hidden_layer_outputs
hidden_layer_outputs = np.array([node_0_output, node_1_output])

# Calculate model output (do not apply relu)
model_output = (hidden_layer_outputs * weights['output']).sum()

# Print model output
print(model_output)
```

```
52
```

▶ **Data re-pre-loading:**

[4]:

```python
input_data = [
    np.array([3, 5]),
    np.array([1, -1]),
    np.array([0, 0]),
    np.array([8, 4])
]
```

▶ **Network to many observations/rows of data applying practice:**

[5]:

```python
# Define predict_with_network()
def predict_with_network(input_data_row, weights):

    # Calculate node 0 value
    node_0_input = (input_data_row * weights['node_0']).sum()
    node_0_output = relu(node_0_input)

    # Calculate node 1 value
    node_1_input = (input_data_row * weights['node_1']).sum()
    node_1_output = relu(node_1_input)

    # Put node values into array: hidden_layer_outputs
    hidden_layer_outputs = np.array([node_0_output, node_1_output])

    # Calculate model output
    input_to_final_layer = (hidden_layer_outputs * weights['output']).sum()
    model_output = relu(input_to_final_layer)
    # Return model output
    return (model_output)


# Create empty list to store prediction results
results = []
for input_data_row in input_data:
    # Append prediction to results
    results.append(predict_with_network(input_data_row, weights))

# Print results
print(results)
```

```
[52, 63, 0, 148]
```