

Basics of deep learning and neural networks

Puteaux, Fall/Winter 2020-2021

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

§1 Introduction to Deep Learning in Python

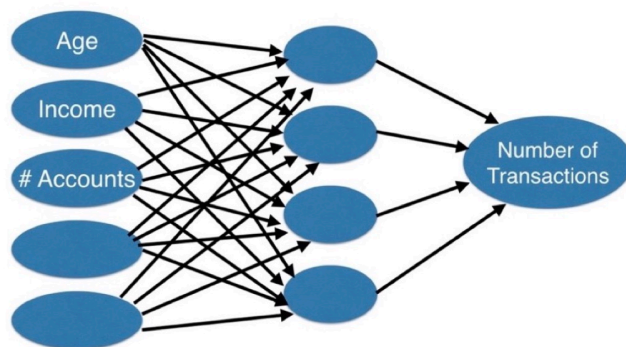
§1.1 Basics of deep learning and neural networks

1 Introduction to deep learning

1. What is the value of interactions for neural networks?

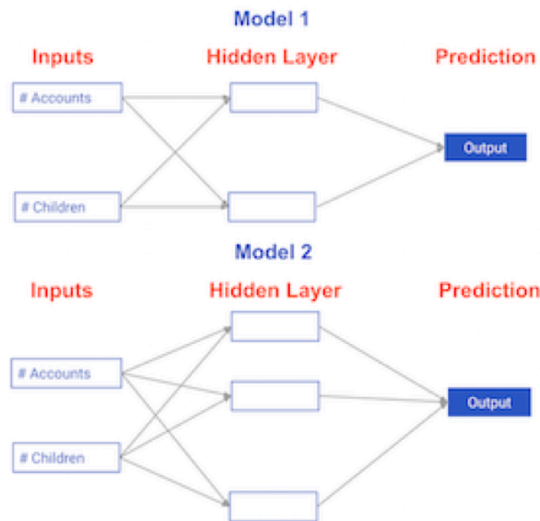
Deep learning uses especially powerful neural networks with almost anything such as *text*, *images*, *videos*, *audio*, or *source code* because neural networks really well account for interactions.

2. How do interactions work in neural networks?



3. Practice question for comparing neural network models to classical regression models:

- Which of the models in the diagrams has greater ability to account for interactions?

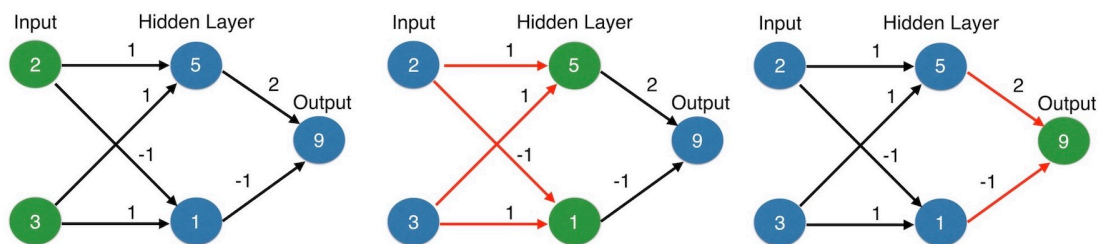


- ☐ Model 1.
- ☒ Model 2.
- ☐ They are both the same.

2 Forward propagation

1. How does the forward propagation function?

- Multiply - add process.
- Dot product.
- Forward propagation is for one data point at a time.
- The output is the prediction for that data point.



2. Code of the forward propagation:

```
[1]: import numpy as np

input_data = np.array([2, 3])
weights = {
```

```
'node_0': np.array([1, 1]),
'node_1': np.array([-1, 1]),
'output': np.array([2, -1])
}
node_0_value = (input_data * weights['node_0']).sum()
node_1_value = (input_data * weights['node_1']).sum()

hidden_layer_values = np.array([node_0_value, node_1_value])
print(hidden_layer_values)
```

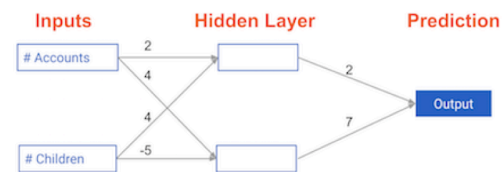
[5 1]

```
[2]: output = (hidden_layer_values * weights['output']).sum()
print(output)
```

9

3. Practice exercises for the forward propagation:

► Diagram of the forward propagation:



► Package pre-loading:

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

► Data pre-loading:

```
[4]: 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 forward propagation algorithm practice:

```
[5]: # Calculate node 0 value: node_0_value
node_0_value = (input_data * weights['node_0']).sum()

# Calculate node 1 value: node_1_value
node_1_value = (input_data * weights['node_1']).sum()
```

```
# Put node values into array: hidden_layer_outputs
hidden_layer_outputs = np.array([node_0_value, node_1_value])

# Calculate output: output
output = (hidden_layer_outputs * weights['output']).sum()

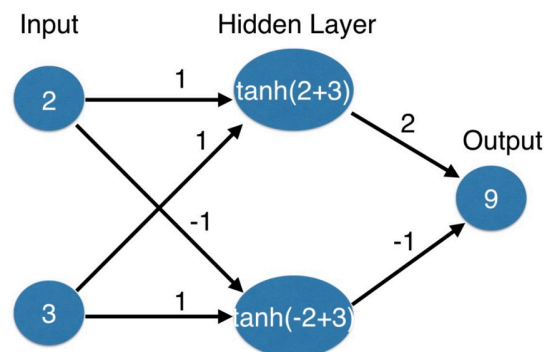
# Print output
print(output)
```

-39

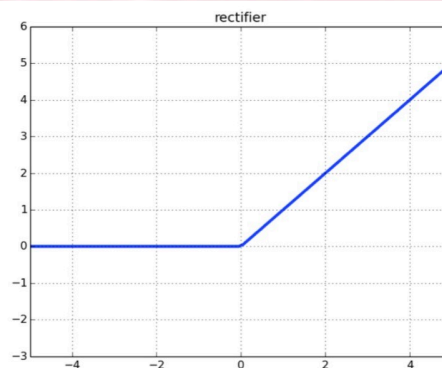
3 Activation functions

1. How do activation functions work?

It is applied to node inputs to produce node output.



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



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

3. Code of activation functions:

```
[6]: 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:

► Package pre-loading:

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

► Data pre-loading:

```
[8]: 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:

```
[9]: 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)

# 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:

```
[10]: 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:

```
[11]: # 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)
```

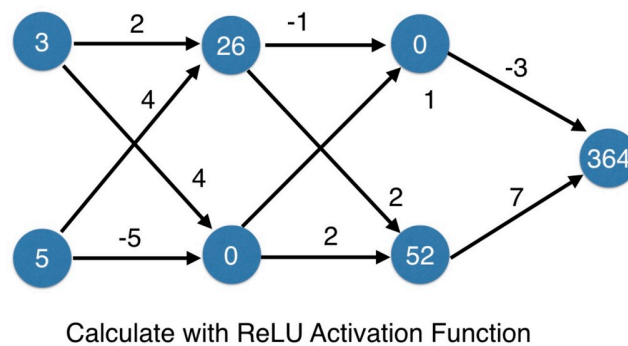
```
# Create empty list to store prediction results
results = []
for input_data_row in input_data:
    # Append prediction to results
    results.append(predict_with_network(input_data_row, weights))

# Print results
print(results)
```

[52, 63, 0, 148]

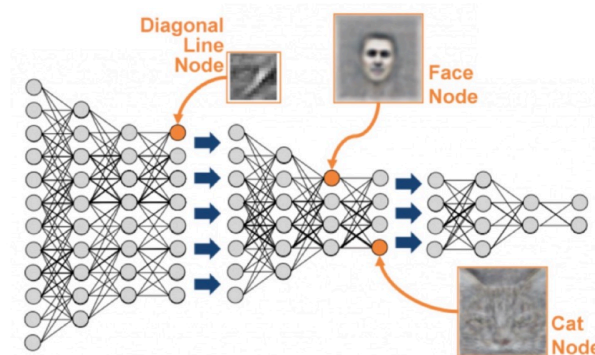
4 Deeper networks

1. How do multiple hidden layers function?



2. Why is deep learning also sometimes called representation learning?

- Deep networks internally build representations of patterns in the data; in this way, partially replace the need for feature engineering.
- Subsequent layers build increasingly sophisticated representations of raw data.

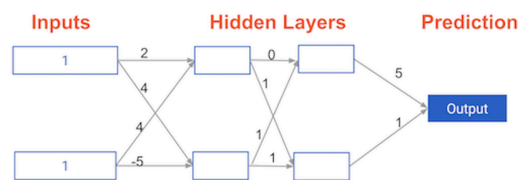


3. How does the deep learning process?

- The modeler doesn't need to specify the interactions.
- When training the model, the neural network gets weights that find the relevant patterns to make better predictions.

4. Practice question for the forward propagation in a deeper network:

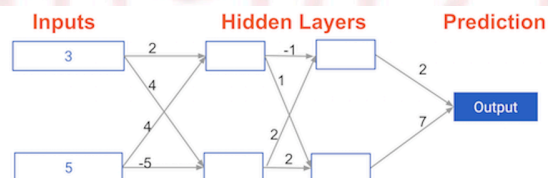
- There is a model with two hidden layers. The values for an input data point are shown inside the input nodes. The weights are shown on the edges/lines. What prediction would this model make on this data point?
- Assume the activation function at each node is the *identity function*. That is, each node's output will be the same as its input. So the value of the bottom node in the first hidden layer is -1 , and not 0 , as it would be if the ReLU activation function was used.



- ☒ 0.
- ☐ 7.
- ☐ 9.

5. Practice exercises for deeper networks:

► Diagram of the forward propagation:



► Package pre-loading:

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

► Data pre-loading:

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

weights = {
    'node_0_0': np.array([2, 4]),
    'node_0_1': np.array([4, -5]),
    'node_1_0': np.array([-1, 2]),
    'node_1_1': np.array([1, 2]),
    'output': np.array([2, 7])
}
```



```
}
```

► Code pre-loading:

```
[14]: def relu(input):  
        output = max(0, input)  
        return (output)
```

► Multi-layer neural networks practice:

```
[15]: def predict_with_network(input_data):  
        # Calculate node 0 in the first hidden layer  
        node_0_0_input = (input_data * weights['node_0_0']).sum()  
        node_0_0_output = relu(node_0_0_input)  
  
        # Calculate node 1 in the first hidden layer  
        node_0_1_input = (input_data * weights['node_0_1']).sum()  
        node_0_1_output = relu(node_0_1_input)  
  
        # Put node values into array: hidden_0_outputs  
        hidden_0_outputs = np.array([node_0_0_output, node_0_1_output])  
  
        # Calculate node 0 in the second hidden layer  
        node_1_0_input = (hidden_0_outputs * weights['node_1_0']).sum()  
        node_1_0_output = relu(node_1_0_input)  
  
        # Calculate node 1 in the second hidden layer  
        node_1_1_input = (hidden_0_outputs * weights['node_1_1']).sum()  
        node_1_1_output = relu(node_1_1_input)  
  
        # Put node values into array: hidden_1_outputs  
        hidden_1_outputs = np.array([node_1_0_output, node_1_1_output])  
  
        # Calculate model output: model_output  
        model_output = (hidden_1_outputs * weights['output']).sum()  
  
        # Return model_output  
        return (model_output)  
  
output = predict_with_network(input_data)  
print(output)
```

182

6. Practice question for learned representations:

- How are the weights that determine the features/interactions in Neural Networks created?

- ☐ A user chooses them when creating the model.
- ☒ The model training process sets them to optimize predictive accuracy.
- ☐ The weights are random numbers.

7. Practice question for levels of representation:

- Which layers of a model capture more complex or “higher level” interactions?
 - ☐ The first layers capture the most complex interactions.
 - ☒ The last layers capture the most complex interactions.
 - ☐ All layers capture interactions of similar complexity.

