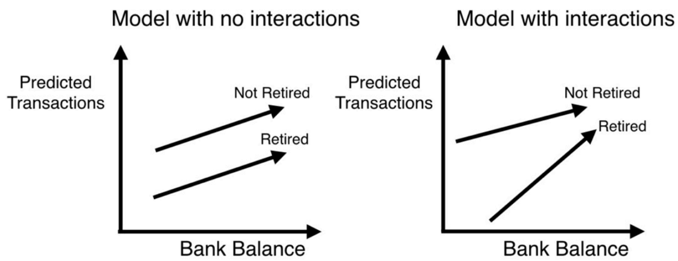
Basics of Deep Learning and Neural Networks

LaTeX, Puteaux, 2020, 2021

Table of Contents

1 Introduction to deep learning

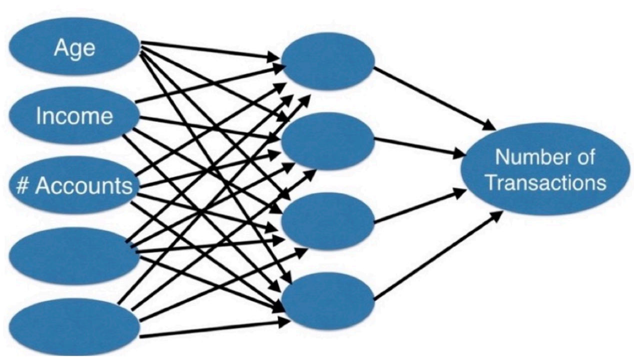
1.1 [note-1] Example as seen by linear regression



1.2 [note-2] Interactions

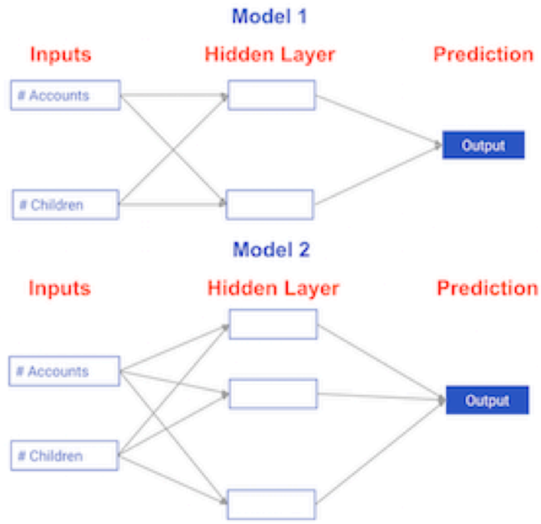
- Neural networks account for interactions really well.
- Deep learning uses especially powerful neural networks:
  - text
  - images
  - videos
  - audio
  - source code

1.3 [note-3] Interactions in neural network



1.4 [quiz-1] Comparing neural network models to classical regression models

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



☐ Model 1.

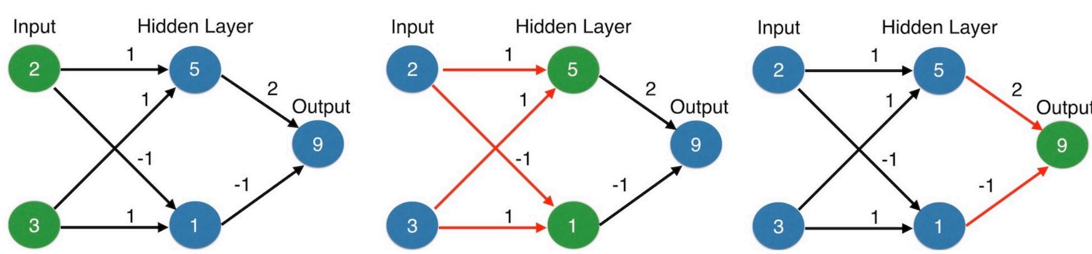
☒ Model 2.

☐ They are both the same.

2 Forward propagation

2.1 [note-1] Forward propagation

- Multiply - add process.
- Dot product.
- Propagate forward for one data point at a time.
- Output is the prediction for that data point.



2.2 [code-1] 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)
```

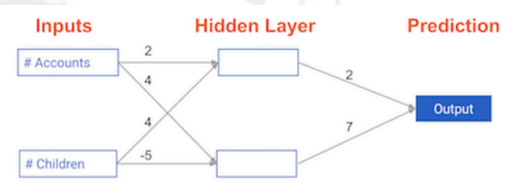
```
[5 1]
```

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

```
9
```

2.3 [task-1] Coding the forward propagation algorithm

▶ Task diagram



▶ 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])
}
```

▶ Task practice

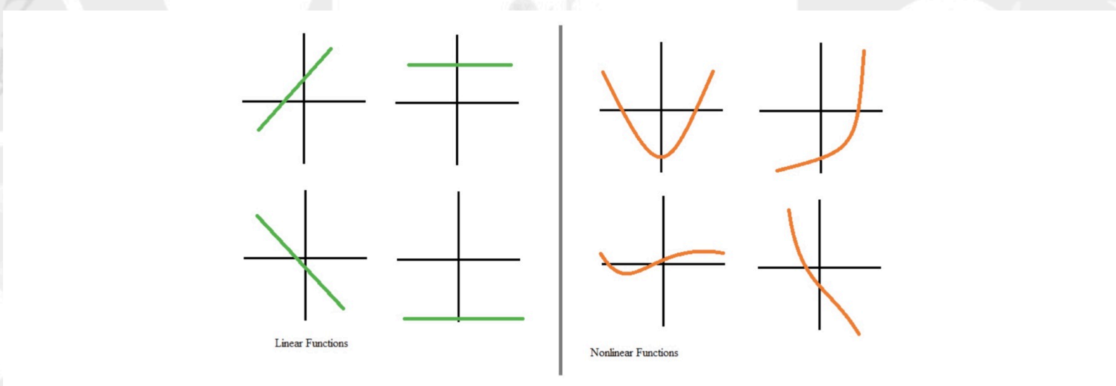
[5]:
```
# Calculate node 0 value: node_0_value
node_0_value = (input_data * weights['node_0']).sum()

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

```python
# 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
```

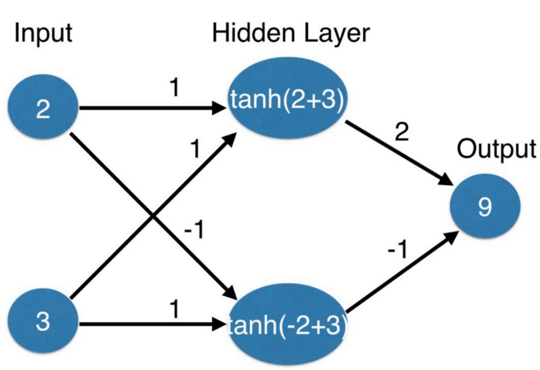
3 Activation functions

3.1 [note-1] Linear vs. nonlinear functions

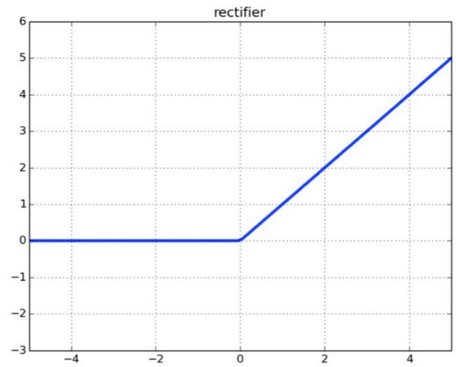


3.2 [note-2] Activation functions

- Activation functions are applied to node inputs to produce node output.



3.3 [note-3] ReLU (Rectified Linear Activation)



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

3.4 [code-1] 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
```

3.5 [task-1] The rectified linear activation function

▶ **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])
}
```

▶ **Task 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
```

3.6 [task-2] Applying the network to many observations/rows of data

▶ **Data pre-loading**

[10]:
```
input_data = [
    np.array([3, 5]),
    np.array([1, -1]),
    np.array([0, 0]),
    np.array([8, 4])
]
```

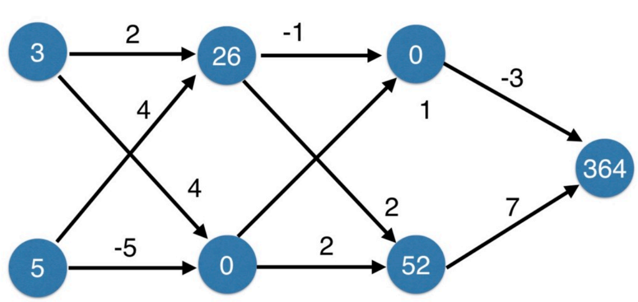
▶ **Task practice**

[11]:
```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]
```

4 Deeper networks

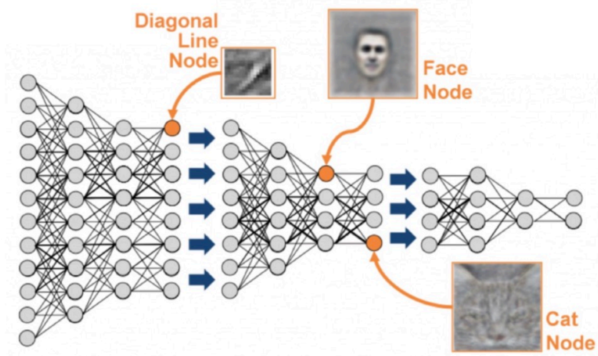
4.1 [note-1] Multiple hidden layers



Calculate with ReLU Activation Function

4.2 [note-2] Representation learning

- Deep networks internally build representations of patterns in the data.
- Partially replace the need for feature engineering.
- Subsequent layers build increasingly sophisticated representations of raw data.

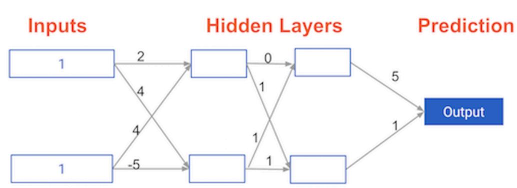


4.3 [note-3] Deep learning

- 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.

4.4 [quiz-1] Forward propagation in a deeper network

- Ther 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.



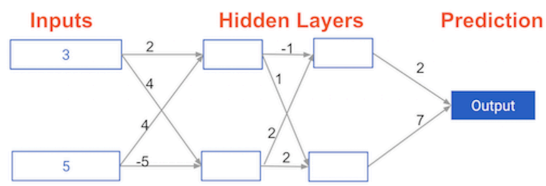
☒ 0.

☐ 7.

☐ 9.

4.5 [task-1] Multi-layer neural networks

▶ **Task diagram**



▶ **Package pre-loading**

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

▶ **Data pre-loading**

[13]:
```python
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]:
```python
def relu(input):
    output = max(0, input)
    return (output)
```

▶ **Task practice**

[15]:
```python
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
```

4.6 [quiz-2] Representations are learned

- 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.

4.7 [quiz-3] 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.

5 Execution environment

[16]:
```
from platform import python_version

python_version = ('python=={}'.format(python_version()))
numpy_version = ('numpy=={}'.format(np.__version__))

writepath = '../../requirements.txt'
requirements = []
packages = [numpy_version]

with open(writepath, 'w+') as file:
    for line in file:
```

```
        requirements.append(line.strip('\n'))
    for package in packages:
        if package not in requirements:
            file.write(package + '\n')

max_characters = len(python_version)
for package in packages:
    if max(max_characters, len(package)) > max_characters:
        max_characters = max(max_characters, len(package))

print('#' * (max_characters + 8))
print('#' * 2 + ' ' * (max_characters + 4) + '#' * 2)
print('#' * 2 + ' ' * 2 + python_version + ' ' *
      (max_characters - len(python_version) + 2) + '#' * 2)
print('#' * 2 + ' ' * 2 + numpy_version + ' ' *
      (max_characters - len(numpy_version) + 2) + '#' * 2)
print('#' * 2 + ' ' * (max_characters + 4) + '#' * 2)
print('#' * (max_characters + 8))
```

```
#####################
##                 ##
##  python==3.7.9  ##
##  numpy==1.19.5  ##
##                 ##
#####################
```