Neural Network Architectures and Backpropagation

Fundamentals of Artificial Intelligence

Fabien Cromieres
Kyoto University

http://lotus.kuee.kyoto-u.ac.jp/~fabien/lectures/IA/

Neural Network Architectures and Backpropagation

- What we are going to discuss today:
  - An overview of Neural Network Architectures
  - The backpropagation algorithm that allows us to compute the gradient in neural network and apply Gradient Descent to learn parameters

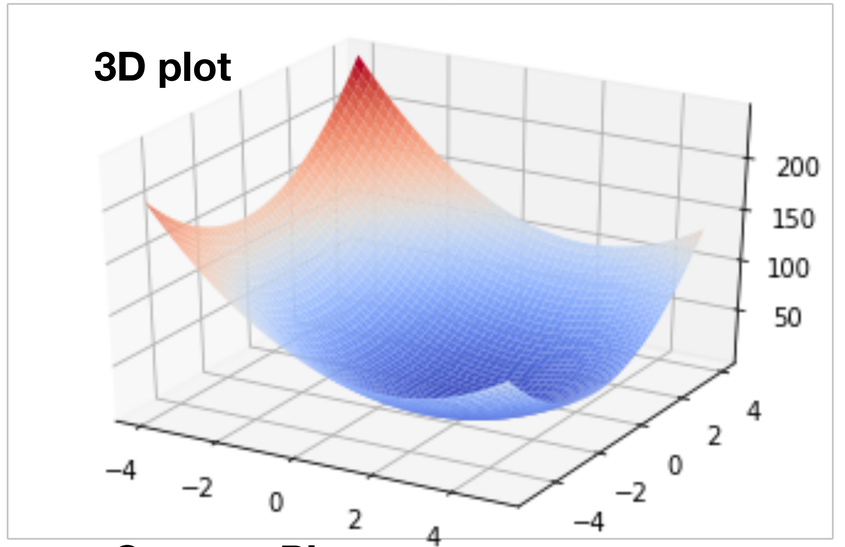
Previously, in this class

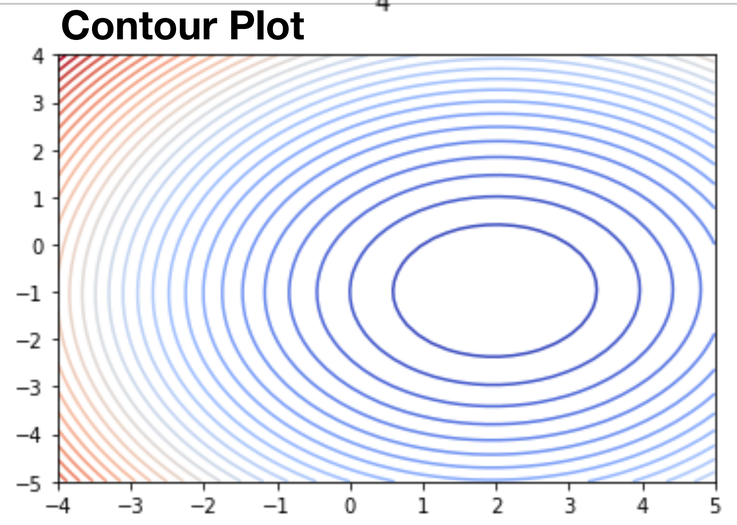
- Let us recap what we have seen so far

Minimizing a function of several variables

$$f(x, y) = 4(x-2)^2 + 4(y+1)^2 - 0.1xy$$

- We have seen that, given a function of several variables, we could find its minimum by gradient descent



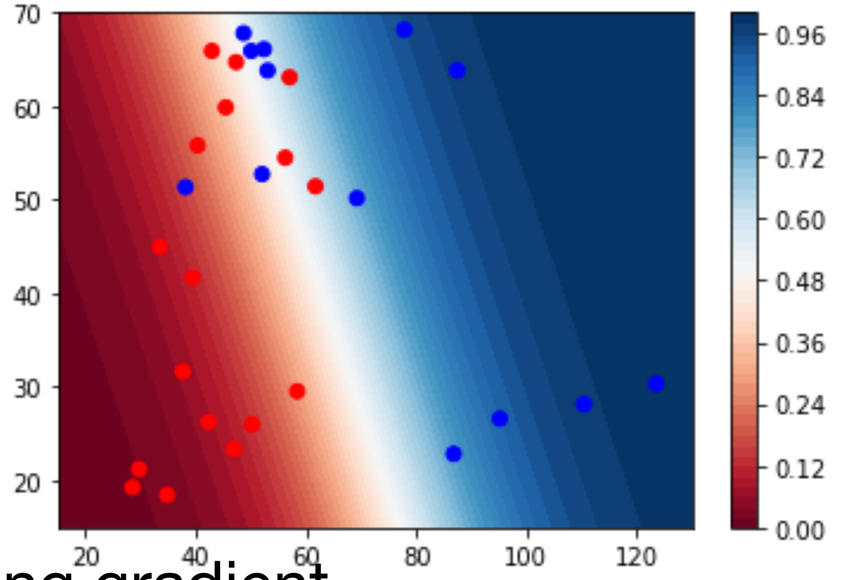


We have seen

- We have seen that we can learn to predict classes with a simple parameterized function called a **logistic classifier**

$$score(income, age) = \theta_0 + \theta_1 \times income + \theta_2 \times age$$

$$V_{model} = \sigma(score)$$



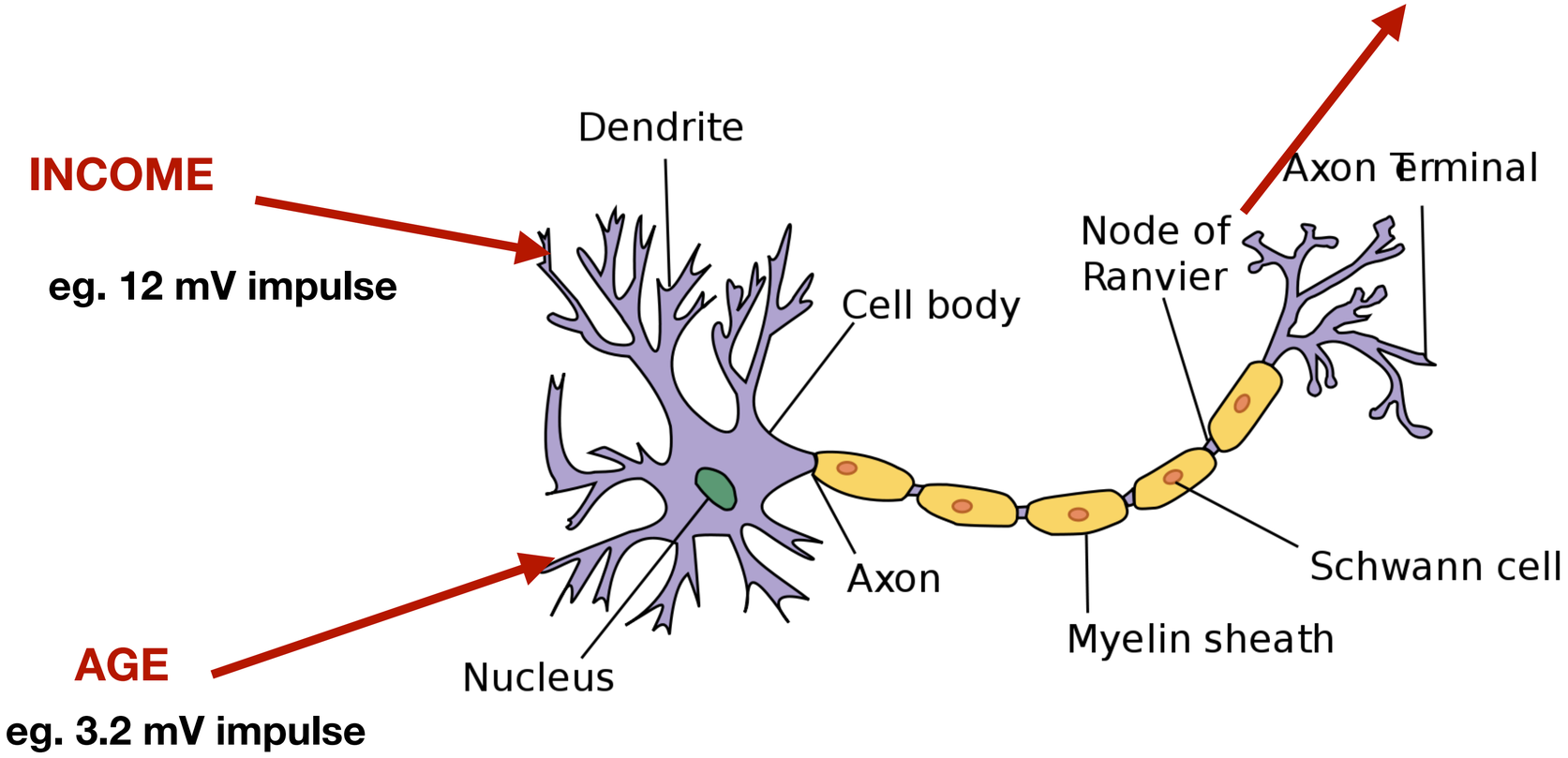
- And that we can learn the proper parameters by applying gradient descent on the loss given some examples

Previously

- We have seen that human neurons actually behave like logistic classifiers

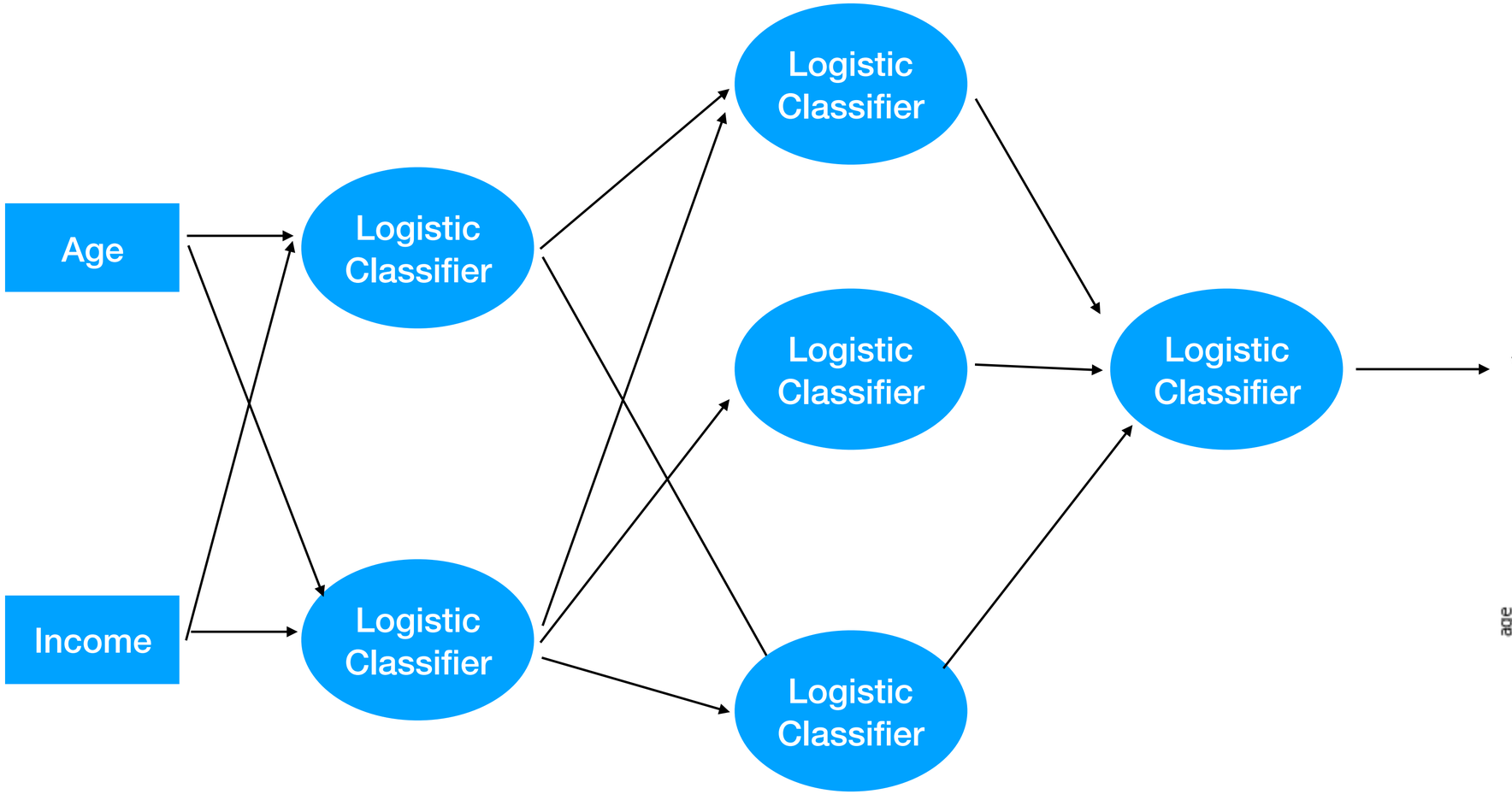
-70mV or 30mV impulse
-70mV: Left-Wing
30mV: Right Wing

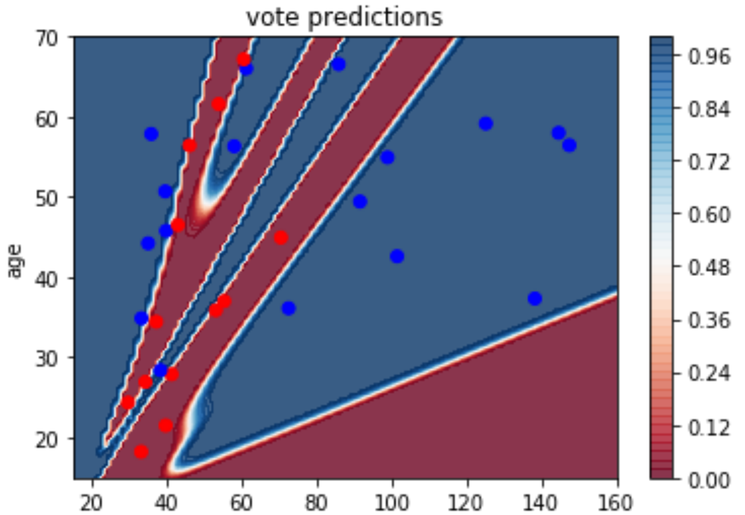
VOTE



Previously

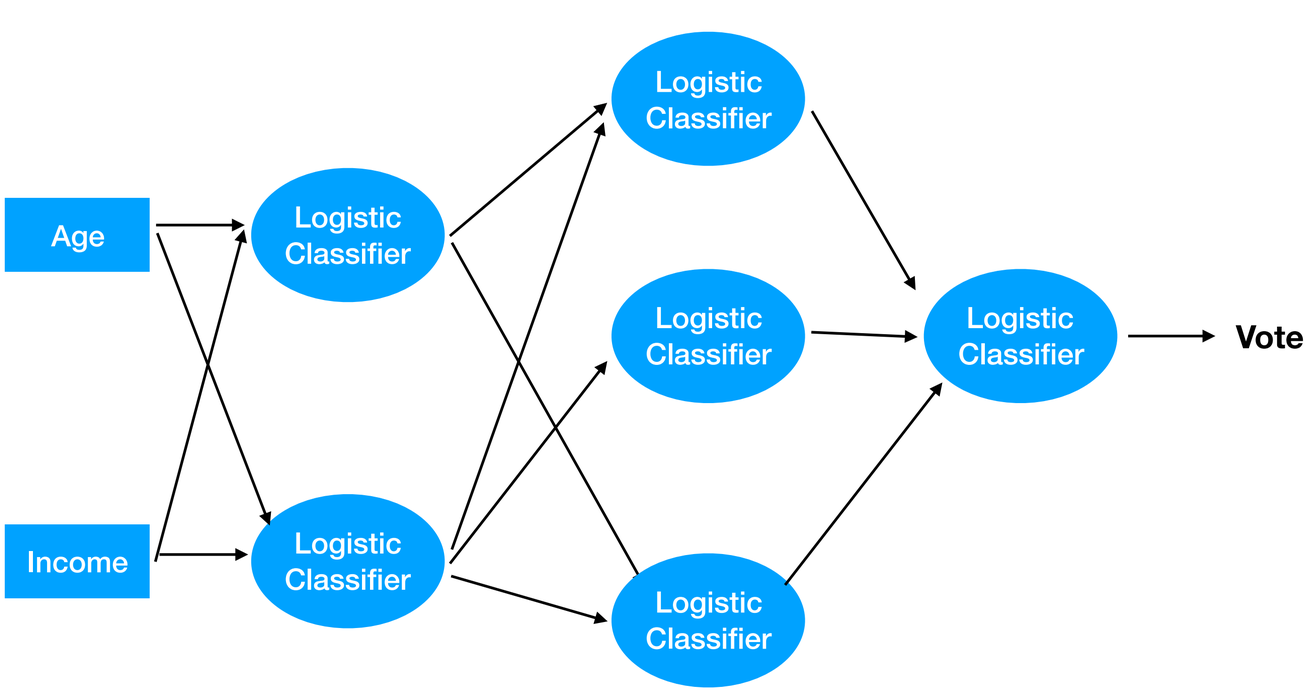
- We have seen that we can obtain more powerful classifiers by combining the neuron-like logistic classifiers



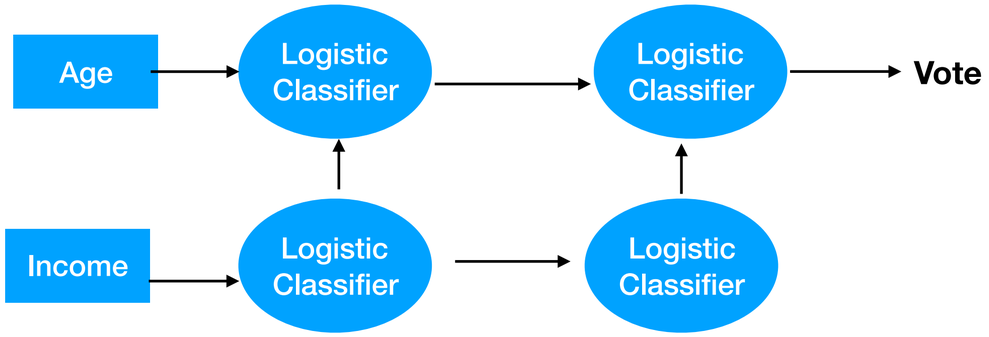


Neural Network Architectures

- We have seen that we could connect neurons to get more powerful classifiers
- How do we design the connections in practice ?
- —> Neural Networks Architecture

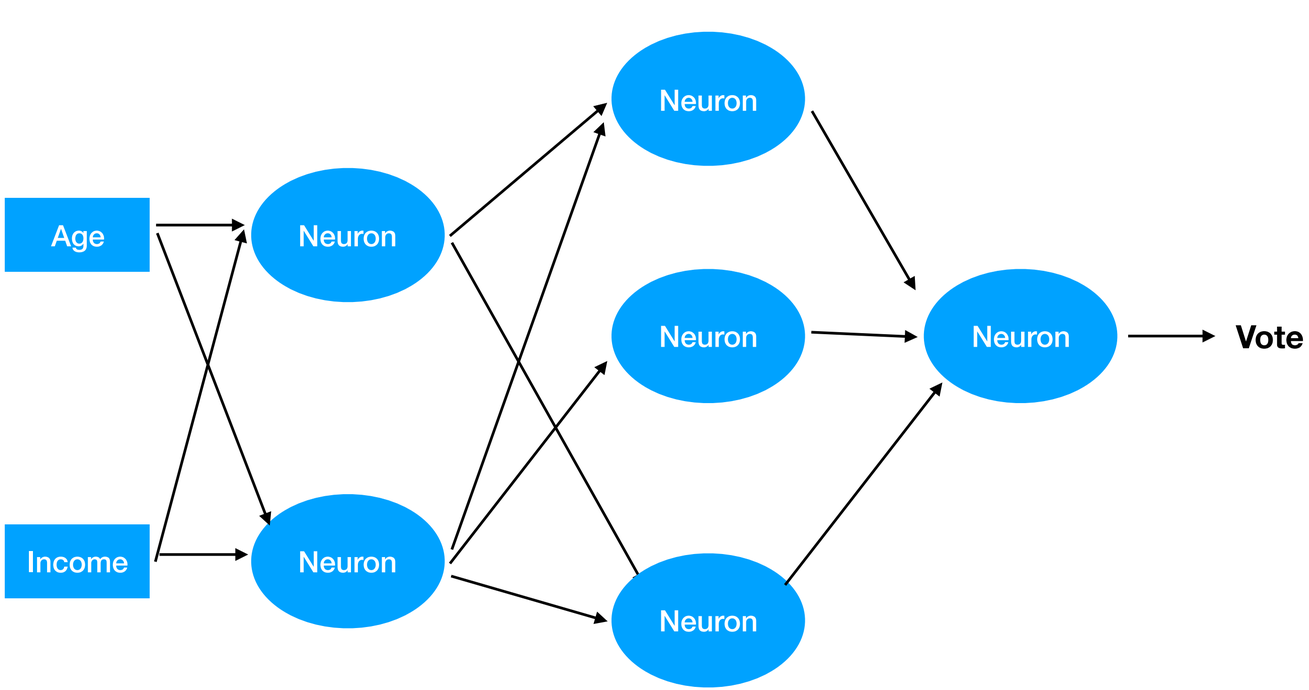


OR ?

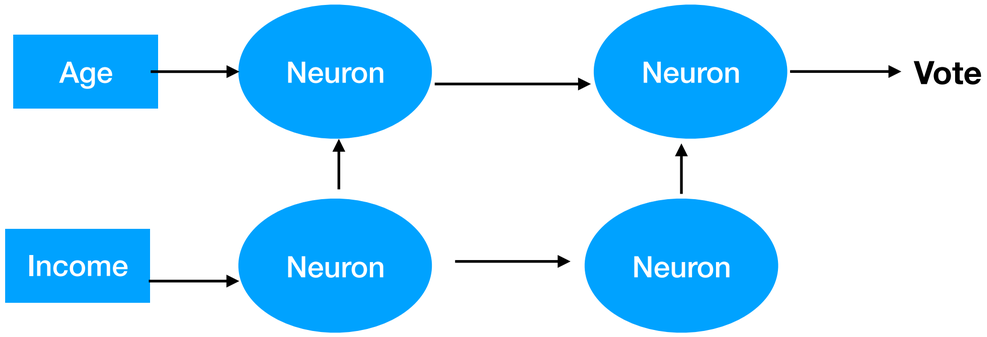


Neural Network Architectures

- We have seen that we could connect neurons to get more powerful classifiers
- How do we design the connections in practice ?
- —> Neural Networks Architecture

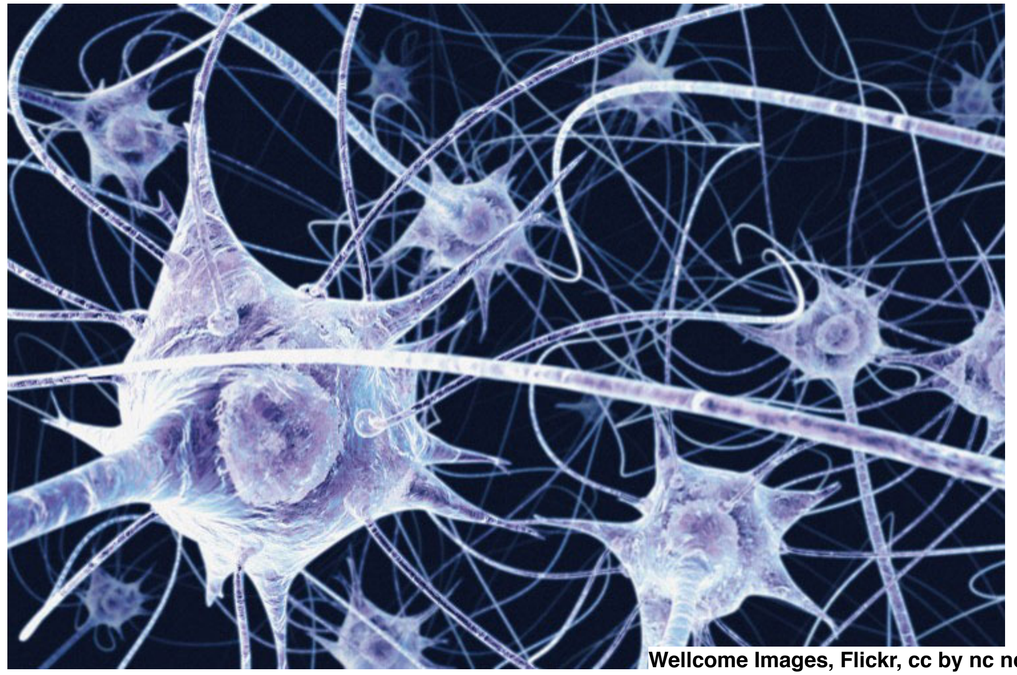


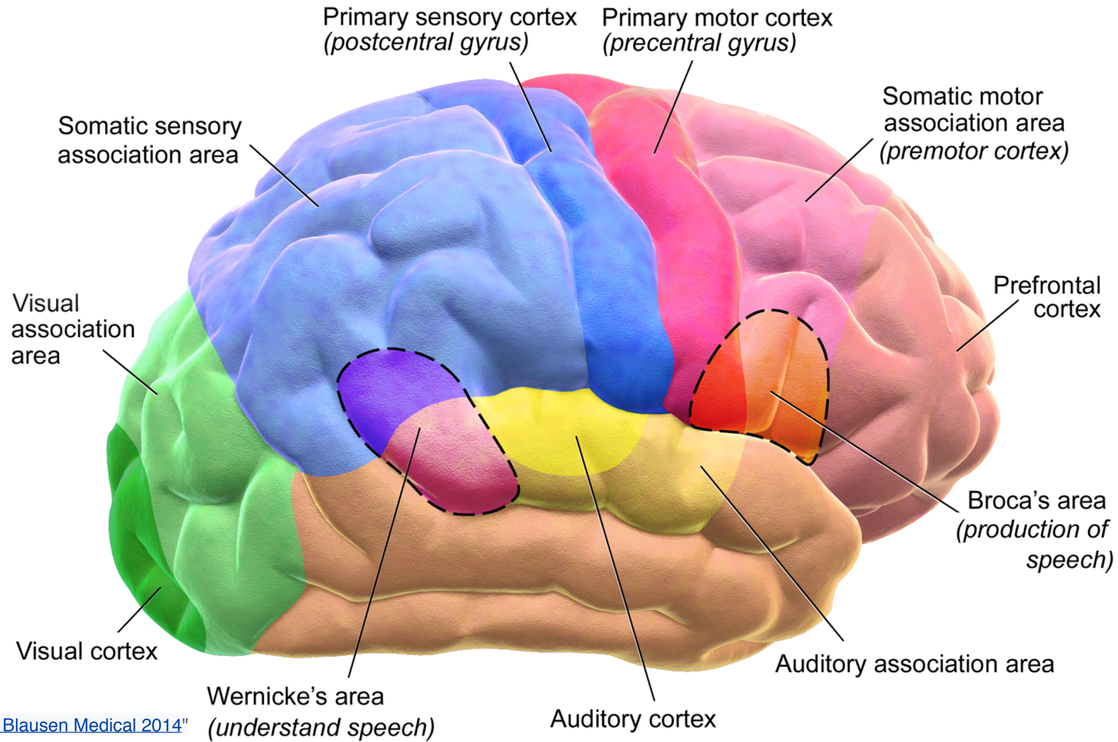
OR ?



Quick biological Analogy

- In our brain too, Neurons are organized in complex elaborated ways
- Remember that an average neuron can connect to 10 000 other neurons
- It seems the organization of the neuron in a zone of the brain will depending on what this zone of the brain is processing
- Neural Network Architecture matters!





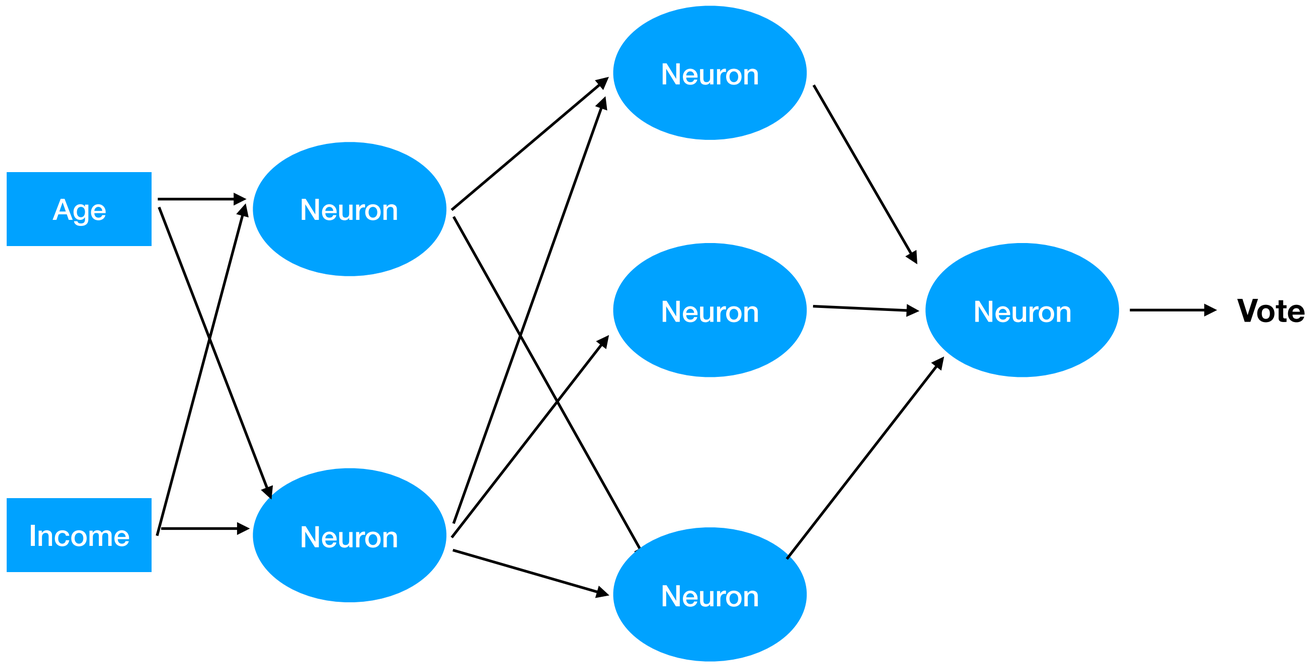
Blausen.com staff (2014). "Medical gallery of Blausen Medical 2014"

Overview of Neural Network Architectures

- First, we will distinguish **two** broad categories of architecture:
  - Feed-Forward Architectures
  - Recurrent Architectures

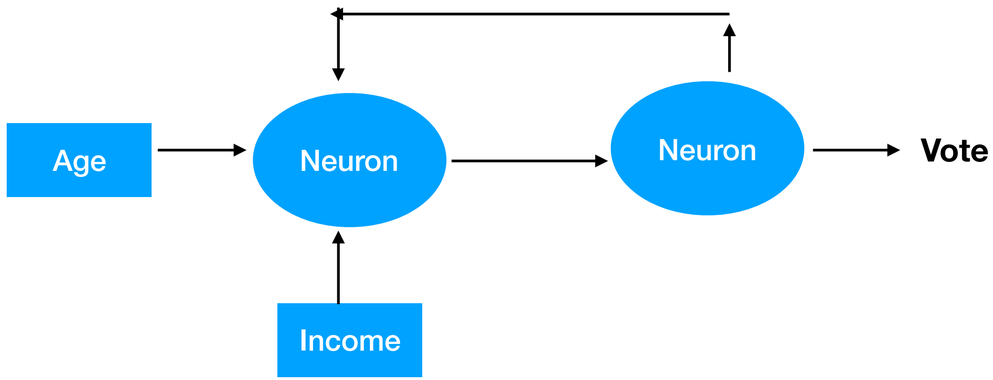
Feed Forward Architectures

- In a Feed Forward Architecture, the “flow” of computation always goes forward



Recurrent Architectures

- In a Recurrent Architecture, the output of a neuron can flow back to a previous neuron



Overview of Neural Network Architectures

- First, we will distinguish **two** broad categories of architecture:
  - Feed-Forward Architectures
    - Used for image processing or general classification
  - Recurrent Architectures
    - Used for processing sequences (especially text)

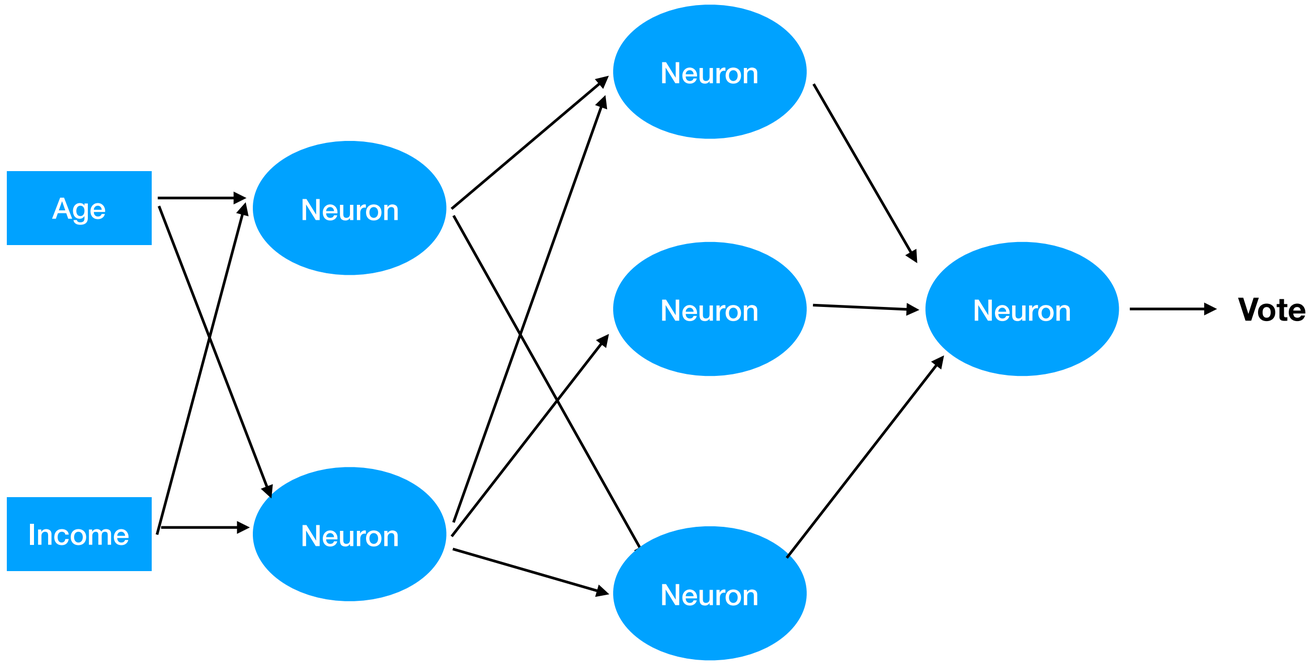
Overview of Neural Network Architectures

- First, we will distinguish **two** broad categories of architecture:
  - Feed-Forward Architectures
    - Used for image processing or general classification
  - Recurrent Architectures
    - Used for processing sequences (especially text)

Today and next 2 sessions

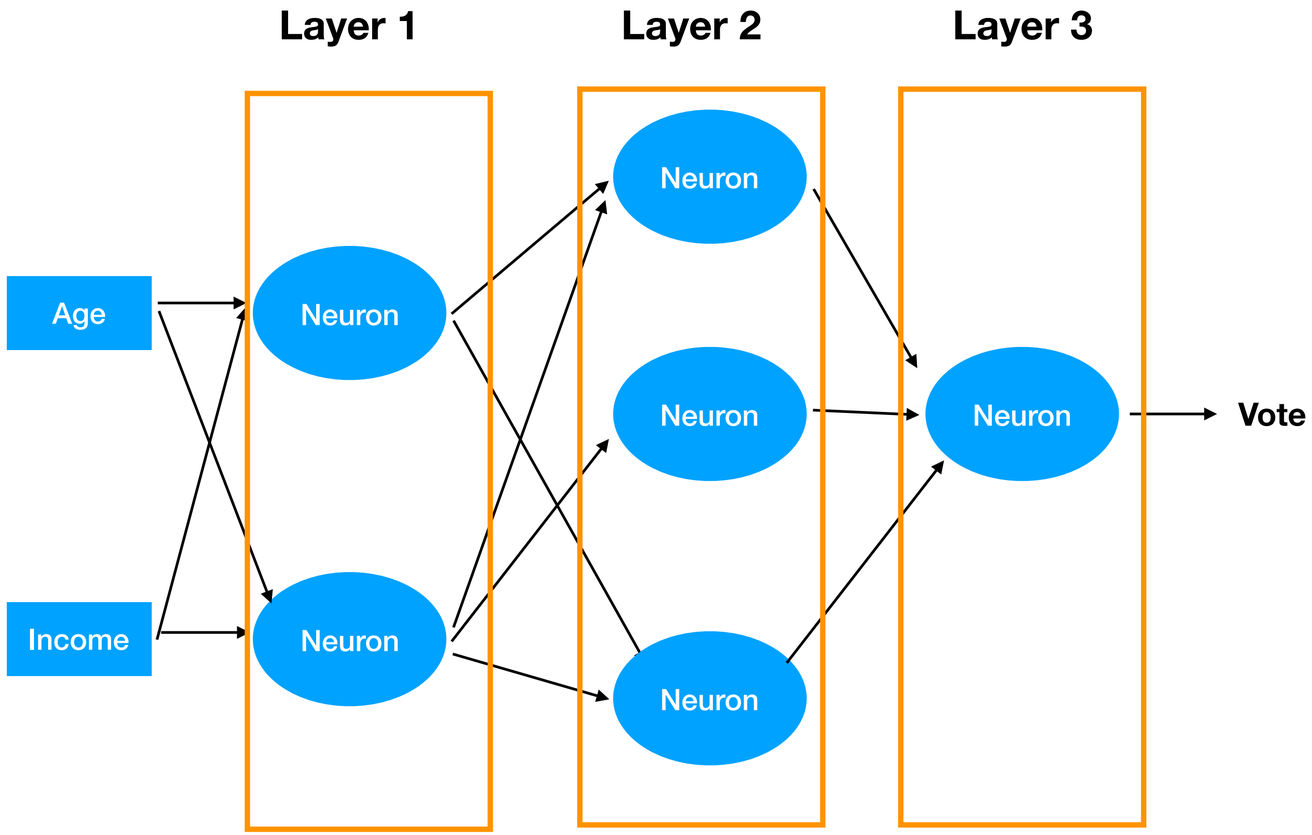
Feed Forward Architectures

- In the case of a feed-forward architecture, we often organize neurons in ***layers***



Feed Forward Architectures

- In the case of a feed-forward architecture, we often organize neurons in ***layers***



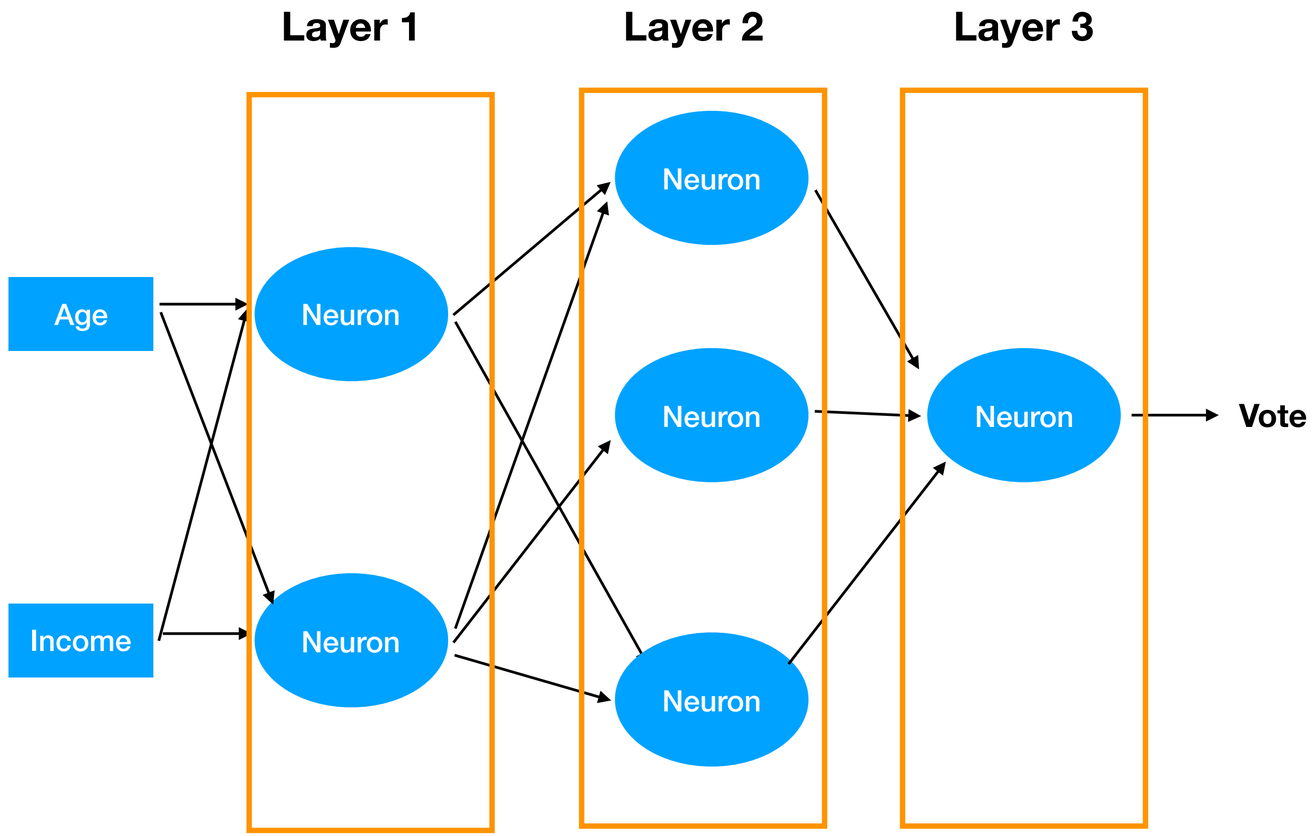
Feed Forward Architectures

- In the case of a feed-forward architecture, we often organize neurons in ***layers***

- Rules:

1. A neuron is never connected to a neuron in the same layer

2. A neuron output only goes in the input of a neuron in the next layer



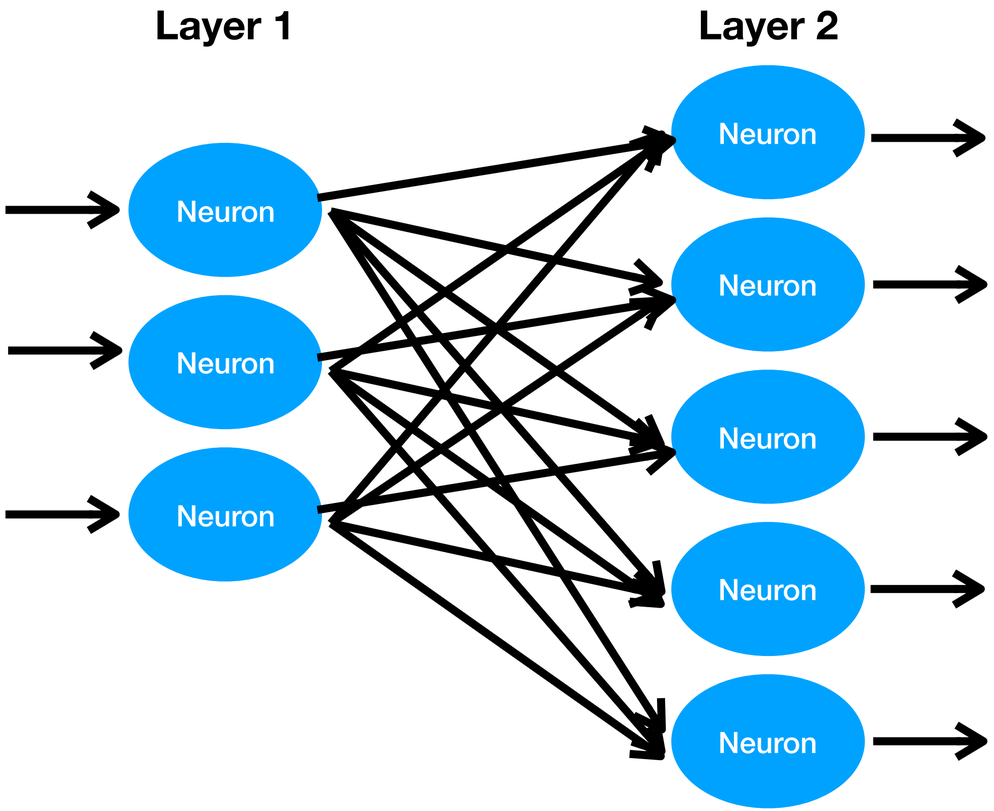
- We will call this a Feed Forward Multi-Layer Architecture

Type of Feed-Forward layers

- We will consider **two** types of Feed-Forward Layers:
  - Fully Connected Layers
  - Convolutional Layers

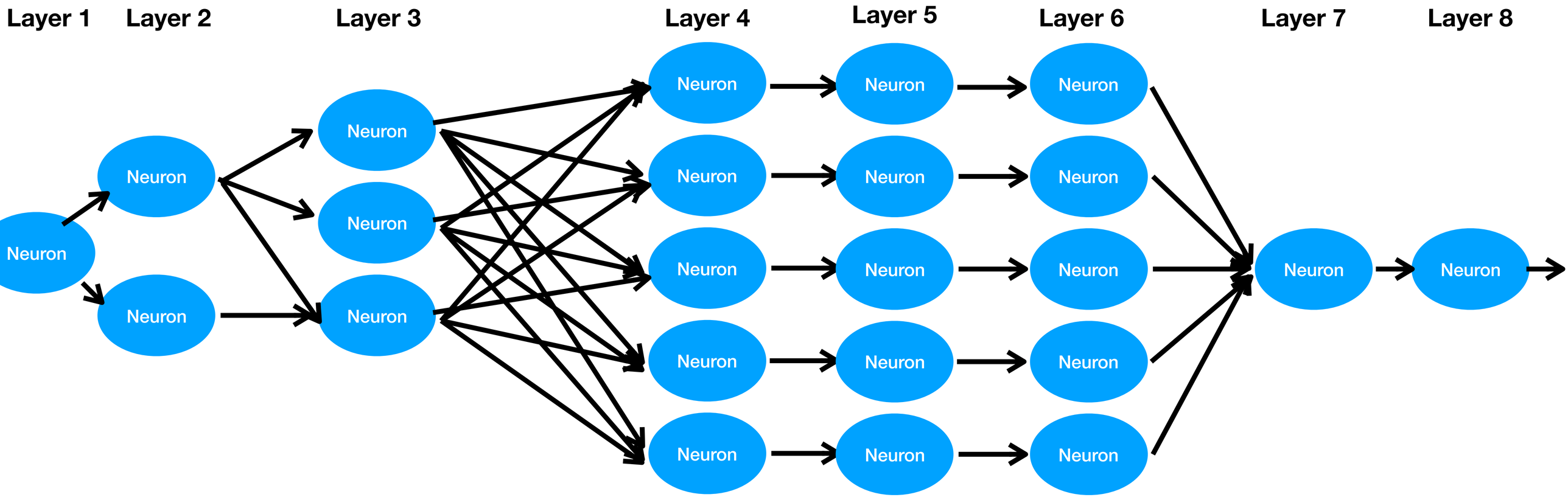
Fully Connected Layers

- We call a layer "*Fully Connected*" if **EACH** neuron in the layer is connected to **ALL** neurons in the **previous** layer



Quiz

- Which Layers are fully connected?



Convolutional Layers

- Among the layers that are **NOT** fully connected, there is a special type of layer called ***Convolutional layer***
  - Very used for processing images
- Neurons are organized in 2-dimensional layers
- Neurons in 2 layers are only connected if they roughly belong to the same area of their respective layer
  - Eg. The neuron in the top-left corner of layer 2 is only connected to the 9 neurons in the top-left corner of layer 1

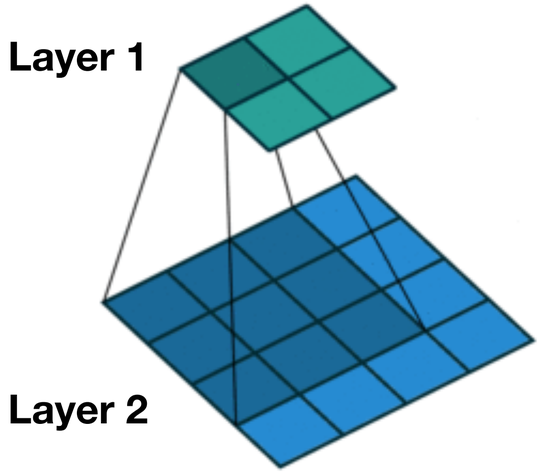
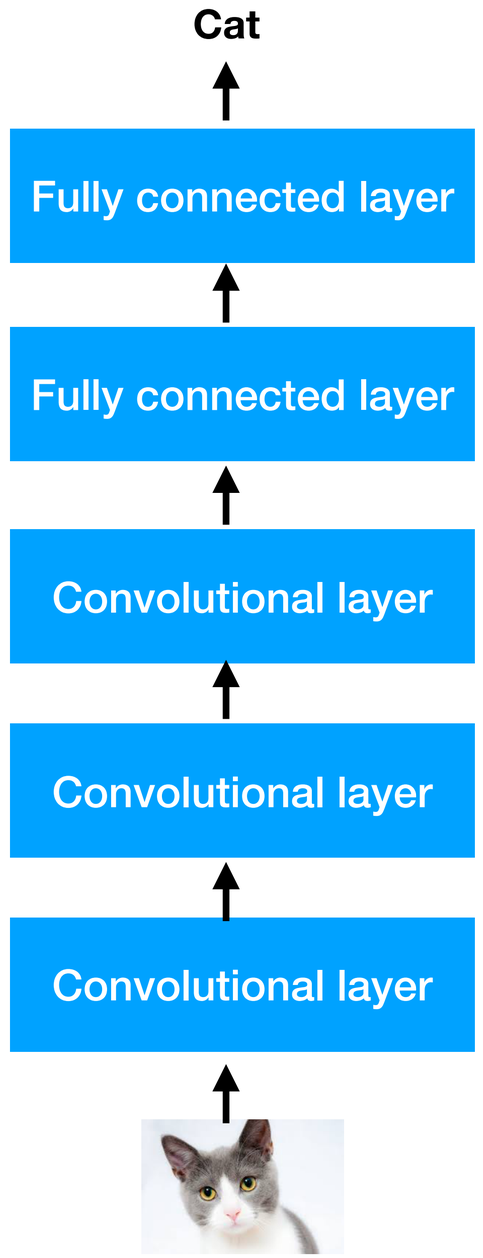


Image: Vincent Dumoulin, Francesco Visin - A guide to convolution arithmetic for deep learning

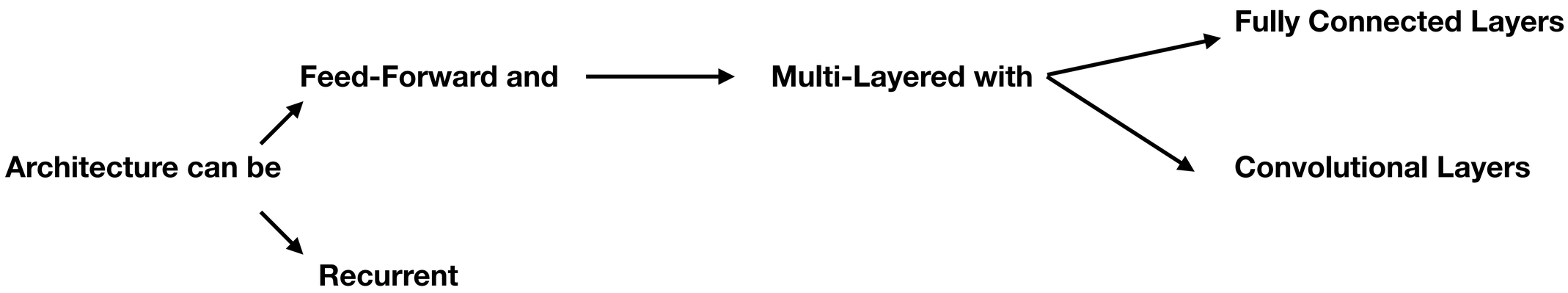
Mix of layers

- A typical Neural Network for Image classification will include many ***convolutional layers*** followed by a few ***fully connected layers***



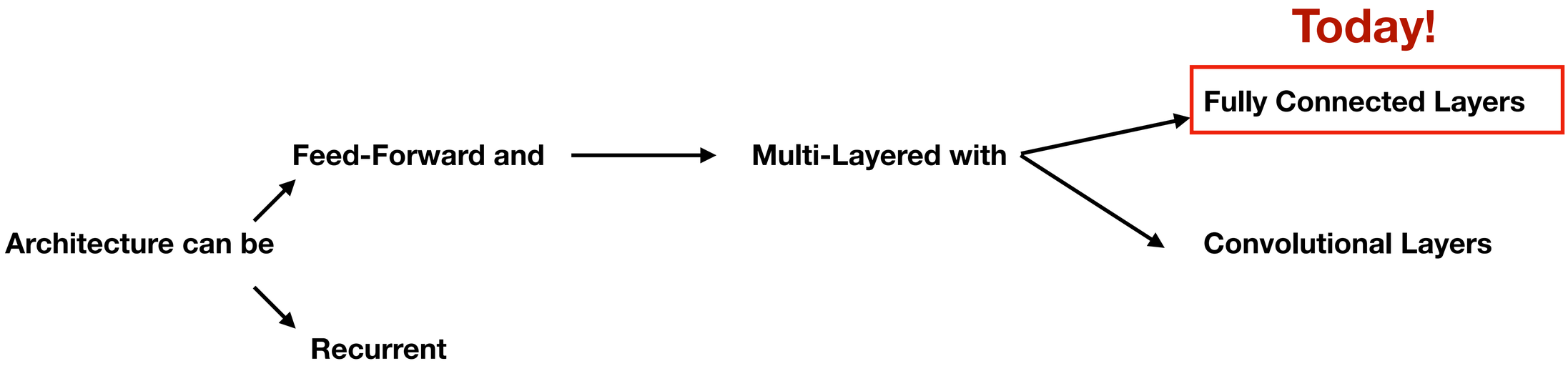
Neural Network Architectures

- In short:



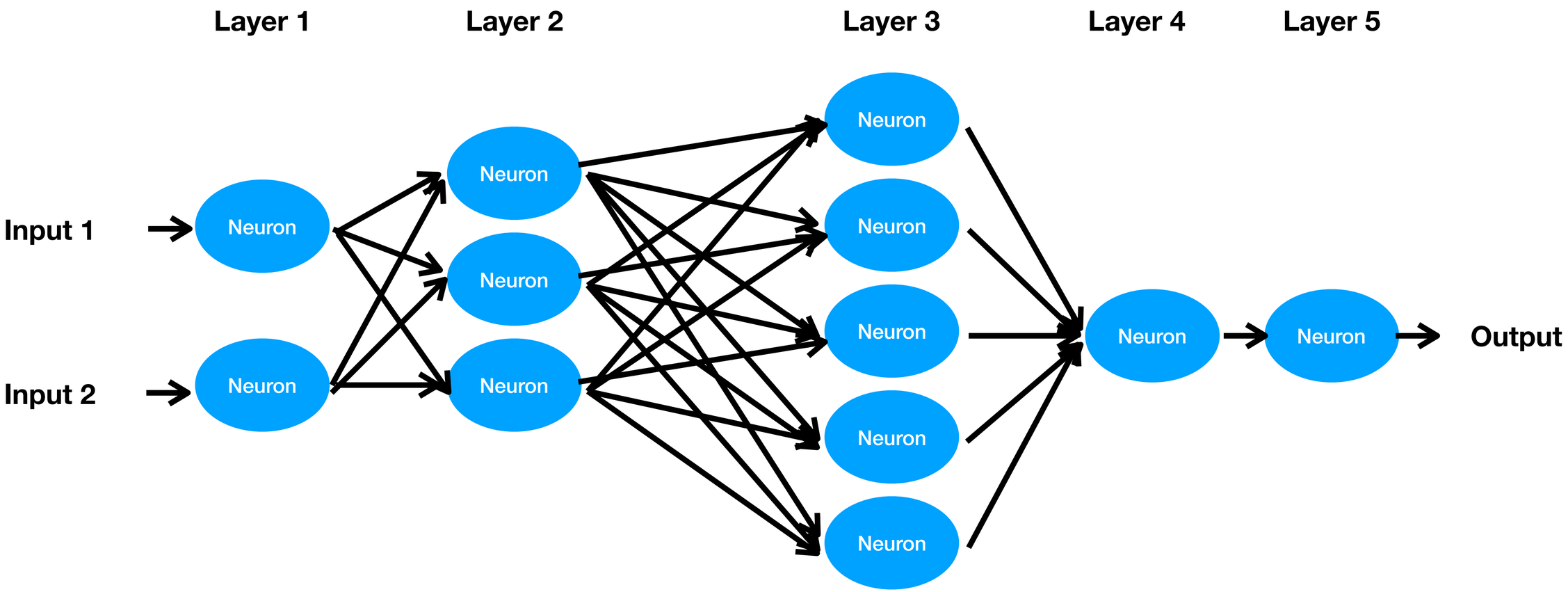
Neural Network Architectures

- In short:



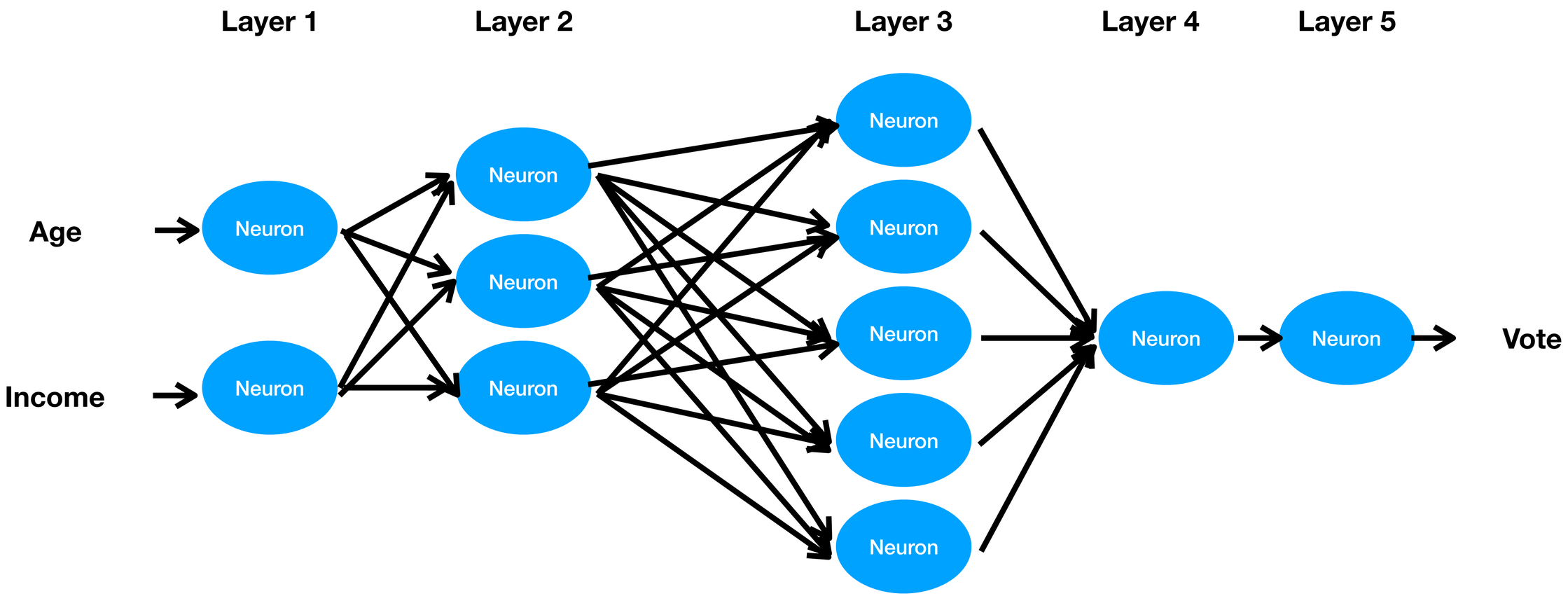
Feed-Forward networks with fully connected layers

- Therefore, we are going to consider this type of Neural Network:



Feed-Forward networks with fully connected layers

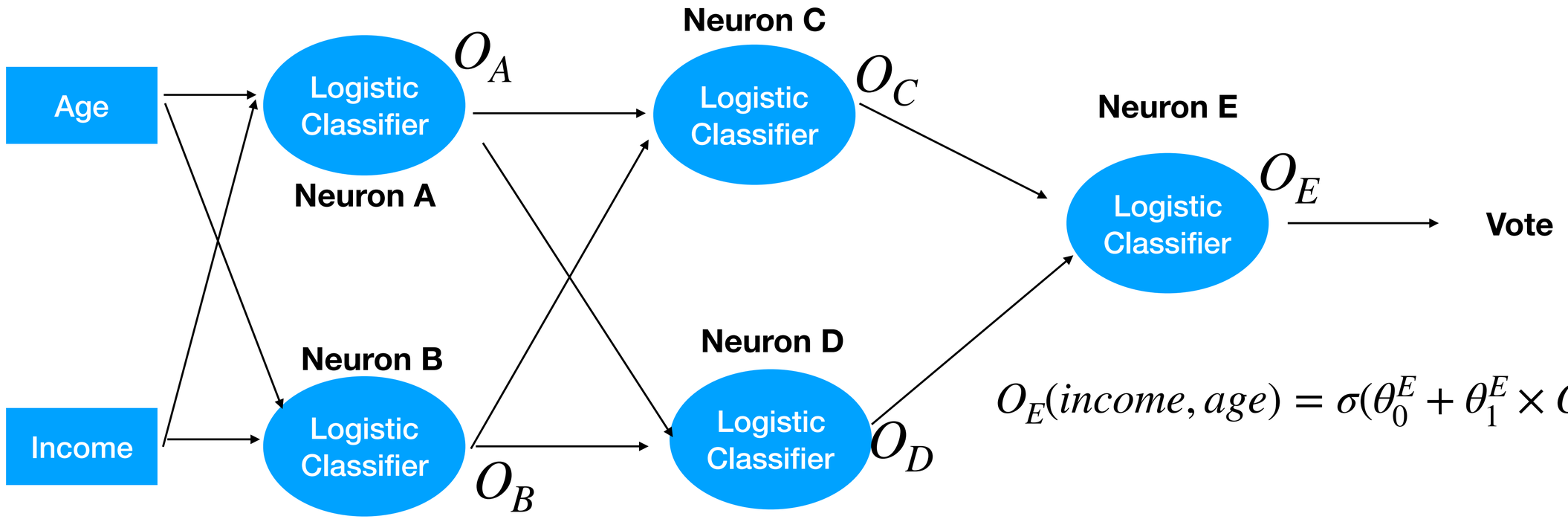
- Therefore, we are going to consider this type of Neural Network:



Keeping in mind what this type of graph mean

$$O_A(income, age) = \sigma(\theta_0^A + \theta_1^A \times income + \theta_2^A \times age)$$

$$O_C(income, age) = \sigma(\theta_0^C + \theta_1^C \times O_A + \theta_2^C \times O_B)$$



$$O_E(income, age) = \sigma(\theta_0^E + \theta_1^E \times O_C + \theta_2^E \times O_D)$$

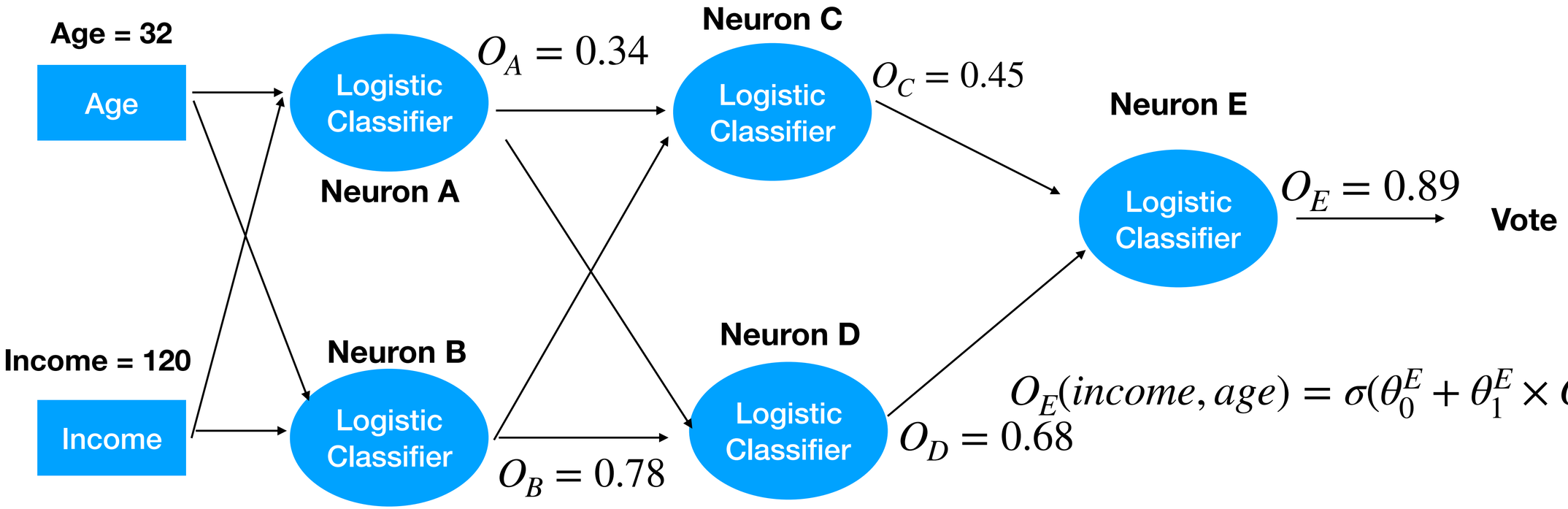
$$O_B(income, age) = \sigma(\theta_0^B + \theta_1^B \times income + \theta_2^B \times age)$$

$$O_D(income, age) = \sigma(\theta_0^D + \theta_1^D \times O_A + \theta_2^D \times O_B)$$

Keeping in mind what this type of graph mean

$$O_A(income, age) = \sigma(\theta_0^A + \theta_1^A \times income + \theta_2^A \times age)$$

$$O_C(income, age) = \sigma(\theta_0^C + \theta_1^C \times O_A + \theta_2^C \times O_B)$$



$$O_E(income, age) = \sigma(\theta_0^E + \theta_1^E \times O_C + \theta_2^E \times O_D)$$

$$O_B(income, age) = \sigma(\theta_0^B + \theta_1^B \times income + \theta_2^B \times age)$$

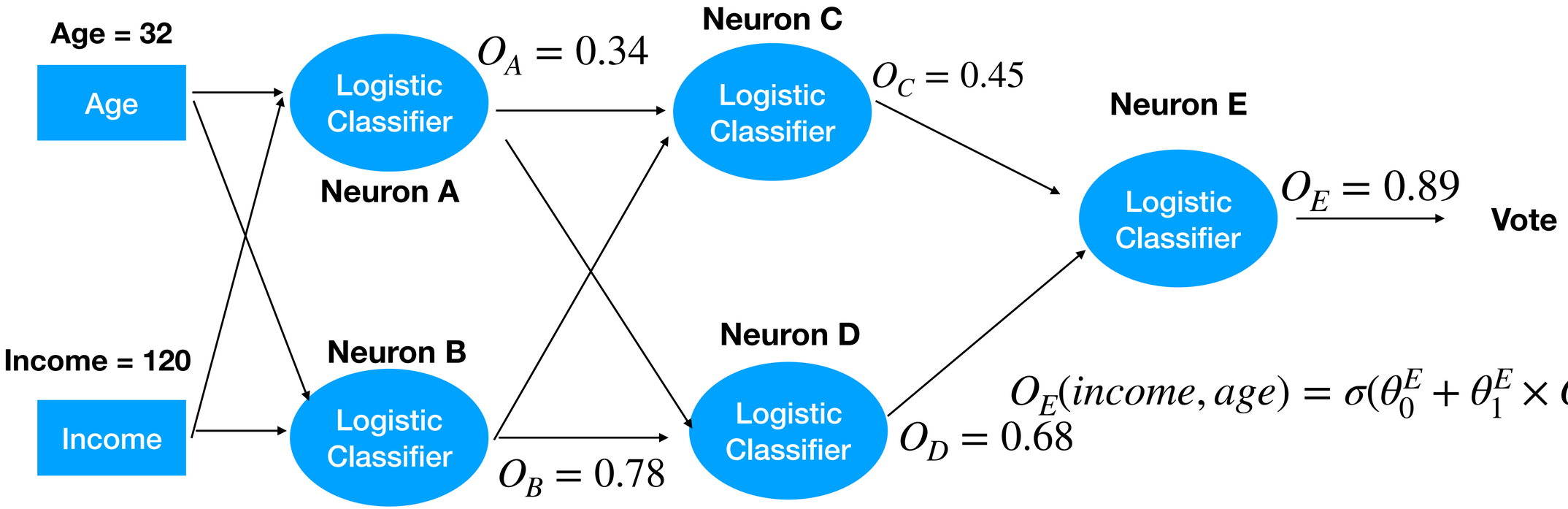
$$O_D(income, age) = \sigma(\theta_0^D + \theta_1^D \times O_A + \theta_2^D \times O_B)$$

Keeping in mind what this type of graph mean

—> Each Neural Network architecture defines a function of the input with parameters θ

$$O_A(income, age) = \sigma(\theta_0^A + \theta_1^A \times income + \theta_2^A \times age)$$

$$O_C(income, age) = \sigma(\theta_0^C + \theta_1^C \times O_A + \theta_2^C \times O_B)$$



$$O_E(income, age) = \sigma(\theta_0^E + \theta_1^E \times O_C + \theta_2^E \times O_D)$$

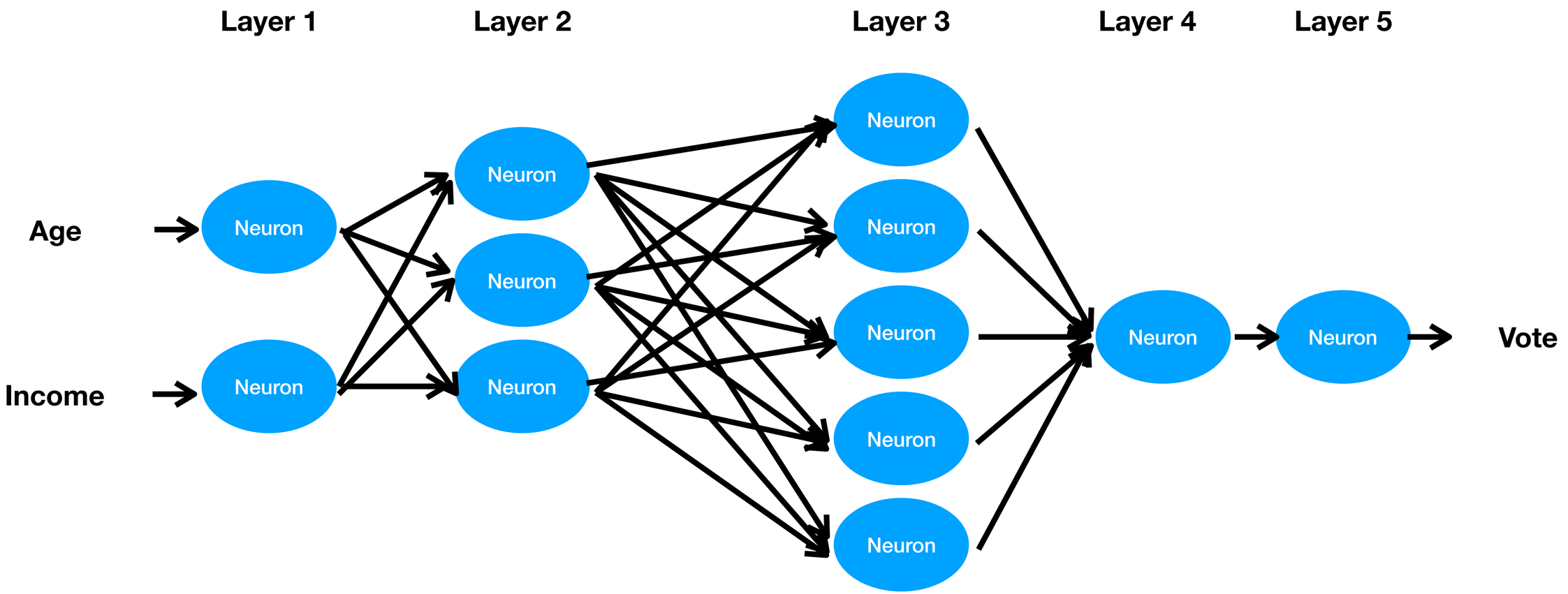
$$O_B(income, age) = \sigma(\theta_0^B + \theta_1^B \times income + \theta_2^B \times age)$$

$$O_D(income, age) = \sigma(\theta_0^D + \theta_1^D \times O_A + \theta_2^D \times O_B)$$

Feed-Forward networks with fully connected layers

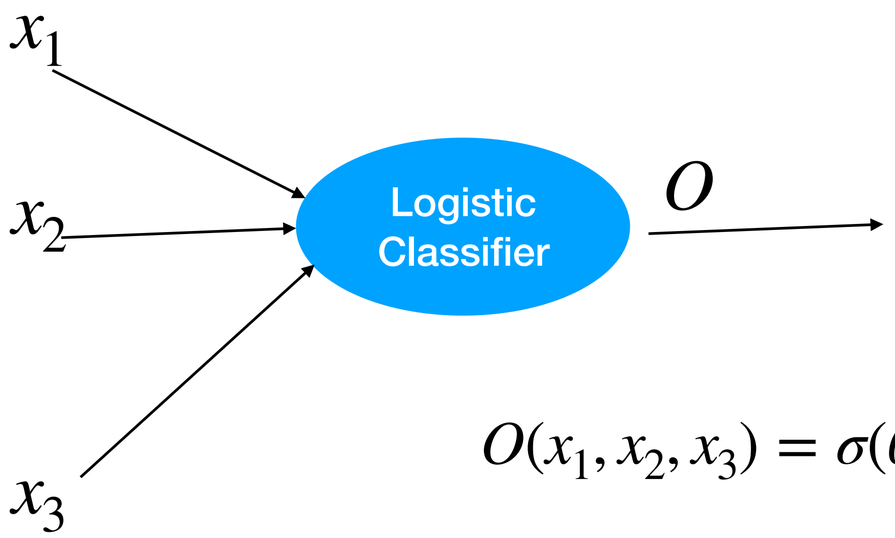
—> Each Neural Network architecture defines a function of the input with parameters θ

- Therefore, this is just a visual way of defining a complicated parameterized function of ***Vote*** given ***Age*** and ***Income***:



Parameters

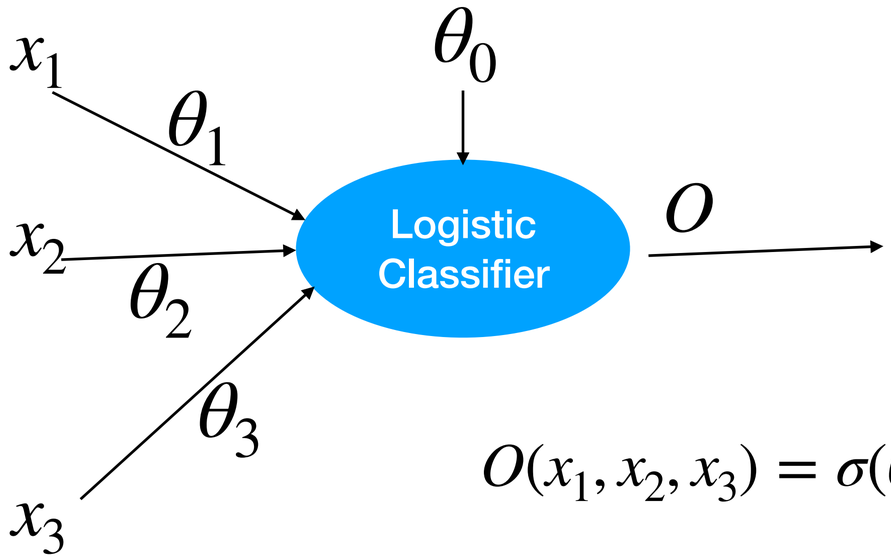
- If a neuron has N inputs, it has N+1 parameters



$$O(x_1, x_2, x_3) = \sigma(\theta_0 + \theta_1 \times x_1 + \theta_2 \times x_2 + \theta_3 \times x_3)$$

Parameters

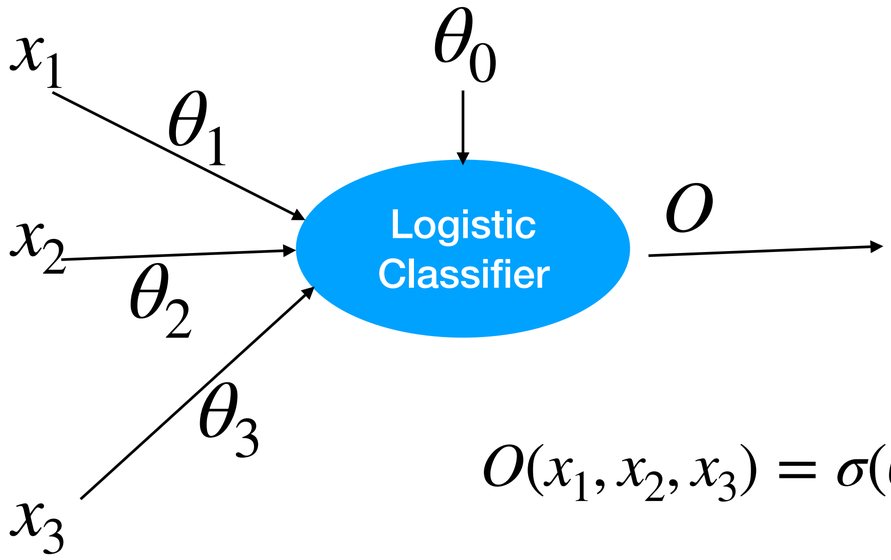
- If a neuron has N inputs, it has N+1 parameters
  - Visually, we can associate a parameter to each input, and show θ_0 separately



$$O(x_1, x_2, x_3) = \sigma(\theta_0 + \theta_1 \times x_1 + \theta_2 \times x_2 + \theta_3 \times x_3)$$

Parameters: Terminology

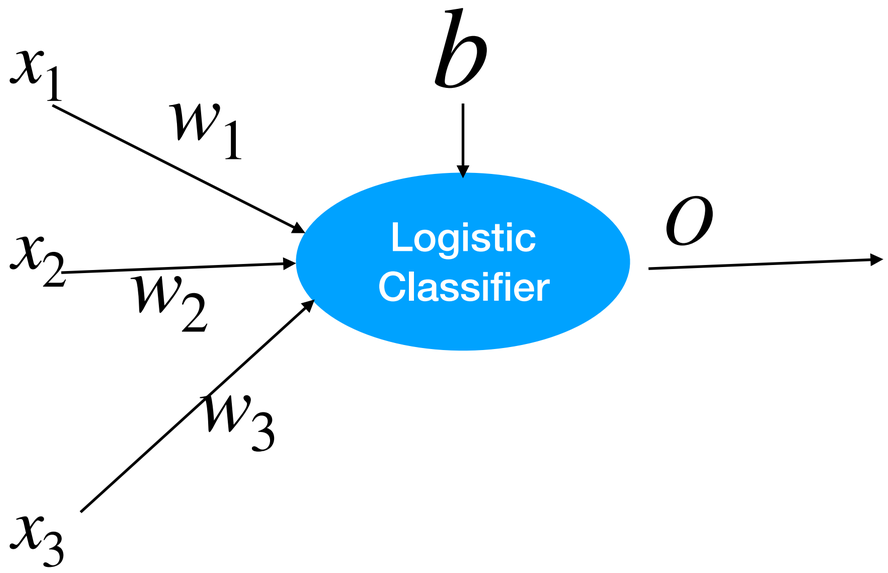
- θ_1, θ_2, θ_3 are often called the ***weights*** of the neuron
- θ_0 is often called the ***bias*** of the neuron



$$O(x_1, x_2, x_3) = \sigma(\theta_0 + \theta_1 \times x_1 + \theta_2 \times x_2 + \theta_3 \times x_3)$$

Parameters: Terminology

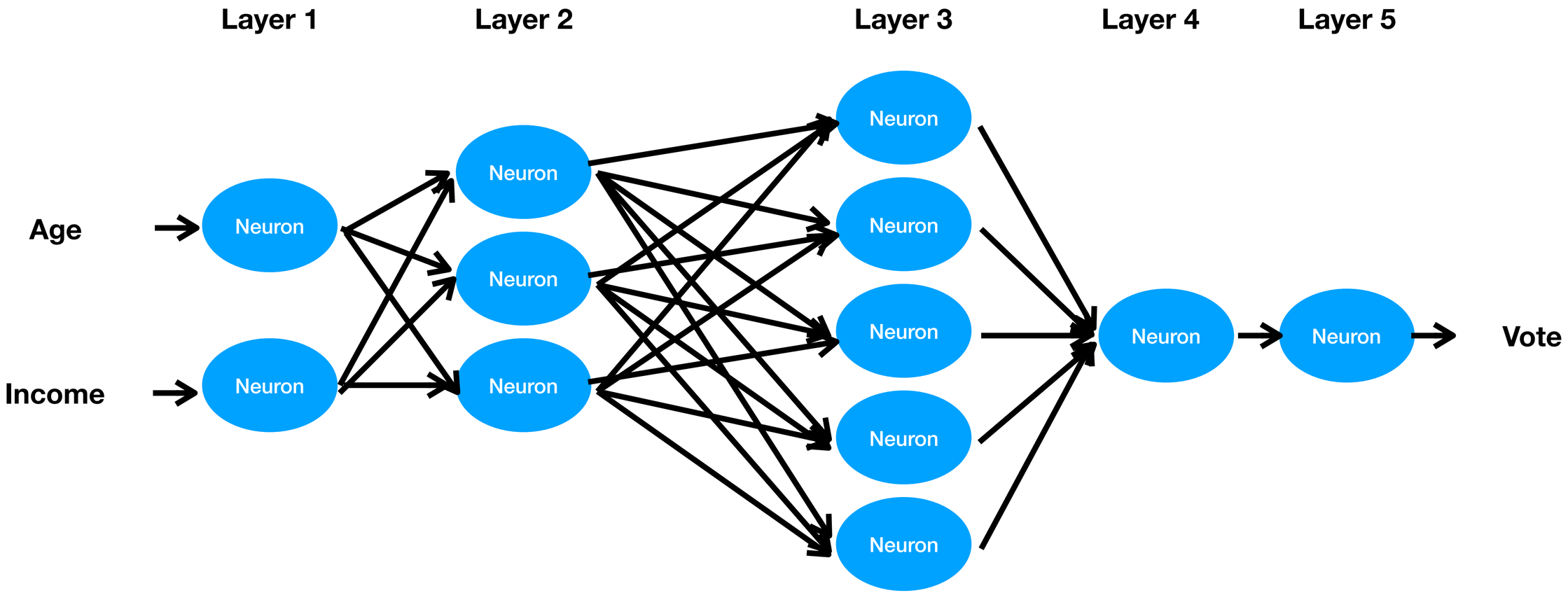
- θ_1, θ_2, θ_3 are often called the ***weights*** of the neuron
  - They are therefore often also noted w_1, w_2, w_3
- θ_0 is often called the ***bias*** of the neuron
  - It is often noted ***b***



$$O(x_1, x_2, x_3) = \sigma(b + w_1 \times x_1 + w_2 \times x_2 + w_3 \times x_3)$$

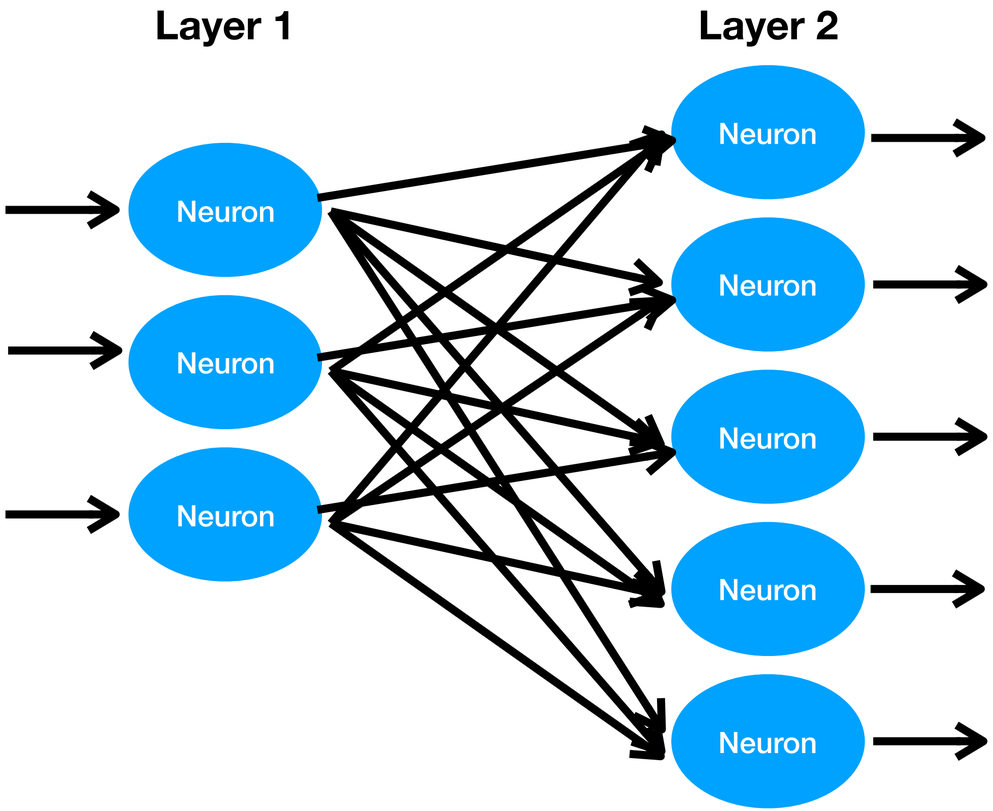
Quiz

- How many parameters for this Neural Network?



Parameters of Fully Connected Layers

- For a fully connected layer of N neurons, and with M neurons in the previous layer, the number of parameters is : N x M + N

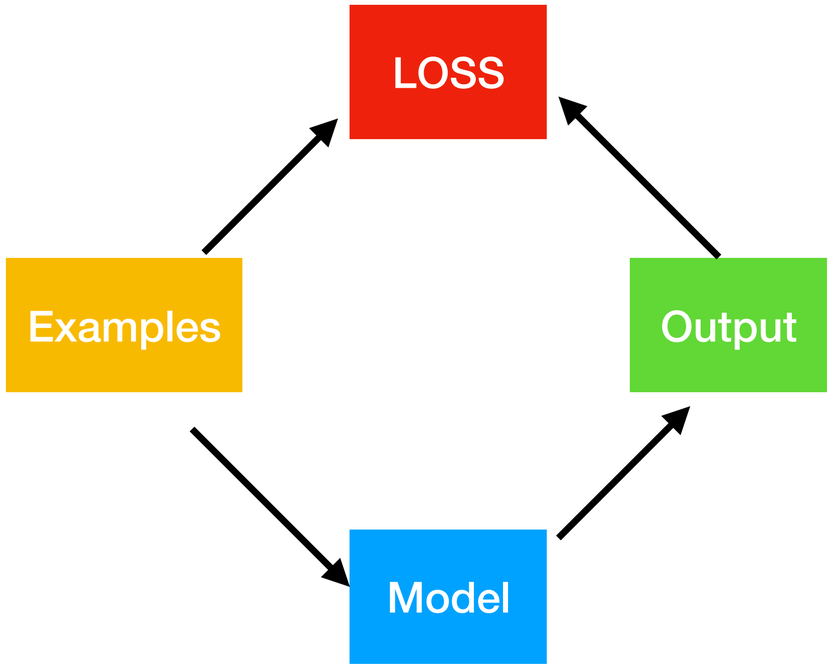


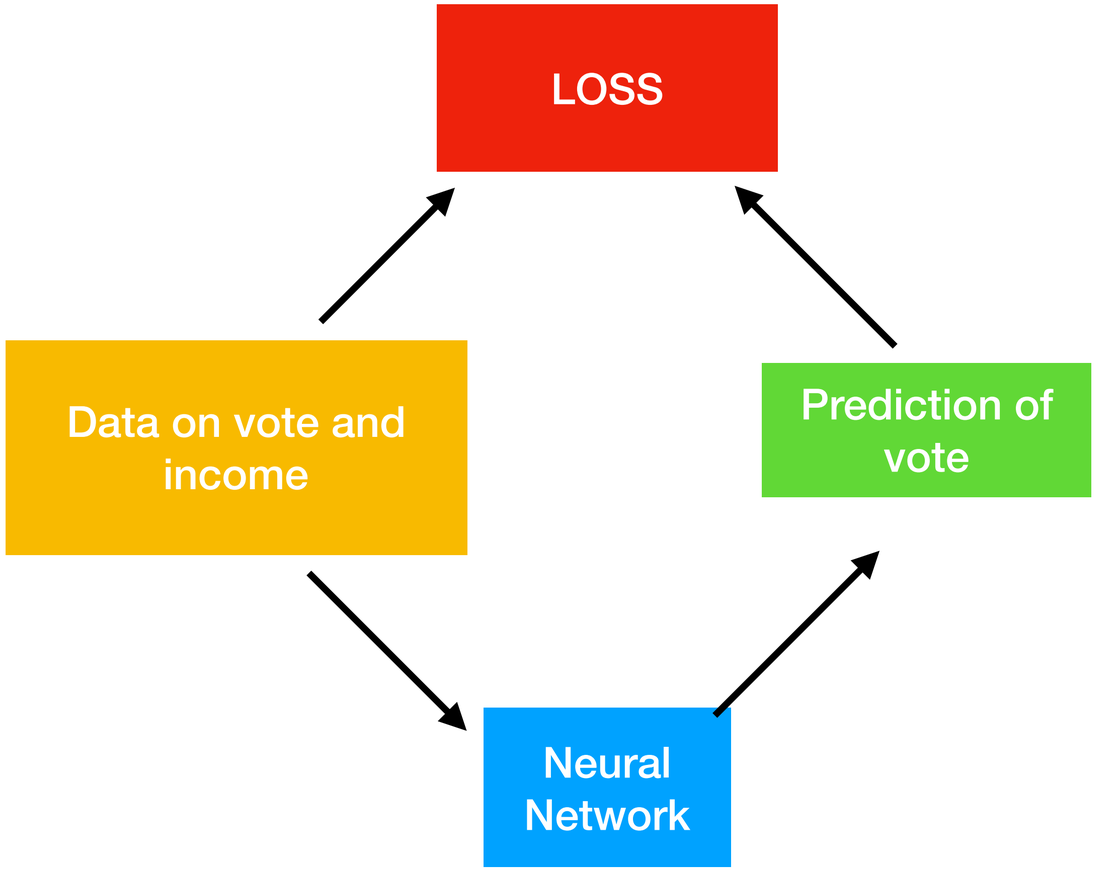
Layer 2 has 5x3+5 = 20 parameters

How to find good parameters

- The result of our Neural Network will depend on the value of the parameters
- How do we find good parameters?

How to find the parameters θ?





We have some examples

We have a model with some parameters

We have a loss function that compute the difference between the example and the prediction of our model

We minimize the loss to obtain the best parameters for our model by GRADIENT DESCENT

How to find the parameters θ?

Yeah ok, we have seen that before…
But if we have a complicated neural network, won't it be <u>very complicated</u> to compute the gradient for all these parameters?

Exam

Model

LOSS

Data on vote and income

Prediction of vote

Neural Network

have some <u>examples</u>

<u>model</u> with some <u>parameters</u>

We have a <u>loss function</u> that compute the difference between the example and the prediction of our model

We <u>minimize the loss</u> to obtain the best parameters for our model by GRADIENT DESCENT

The Backpropagation algorithm

- Actually there is a method for **automatically** computing the *gradient* of a loss for a given *Feed Forward Neural Network*

  - And as you should know now, if we can compute the gradient of the loss, we can find the parameters that minimize the loss by *gradient descent*

The Backpropagation algorithm

- We will not see the details of the algorithm
  - It is actually quite simple, but involves some notions not everybody here is familiar with:
    - Partial derivatives
    - Dynamic programming
  - Anyway, in practice, you will use software that will do the backpropagation for you
    - -> You can actually train a Neural Network without understanding the Backpropagation algorithm (but you should know it exists)
- But let us see the **general idea**

The backpropagation algorithm

- The role of the backpropagation is to compute the gradient
  - Remember that the gradient is a vector of partial derivatives
- Now, remember the ***composition rule*** (a.k.a ***chain rule***) for derivatives (1 variable case here, but there is a similar rule for the case with several variables):

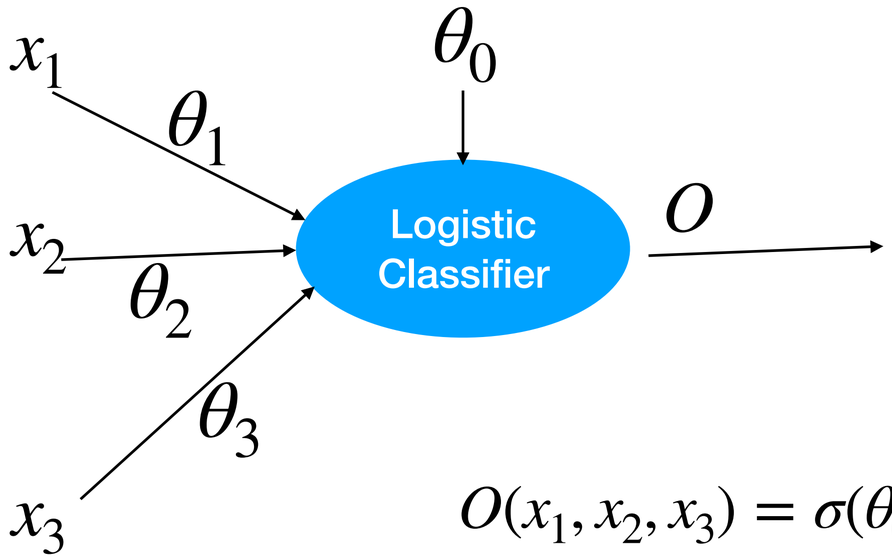
Chain rule

$$f(x) = g(h(x)) \quad \Rightarrow \quad f'(x) = h'(x) \times g'((h(x))$$

This rule says that if I know how to compute the derivative of functions g(x) and h(x), you know how to compute the derivative of g(h(x))

The backpropagation algorithm

- We know how to compute the gradient for a single neuron
  - (see the lecture on logistic classifier for a formula)

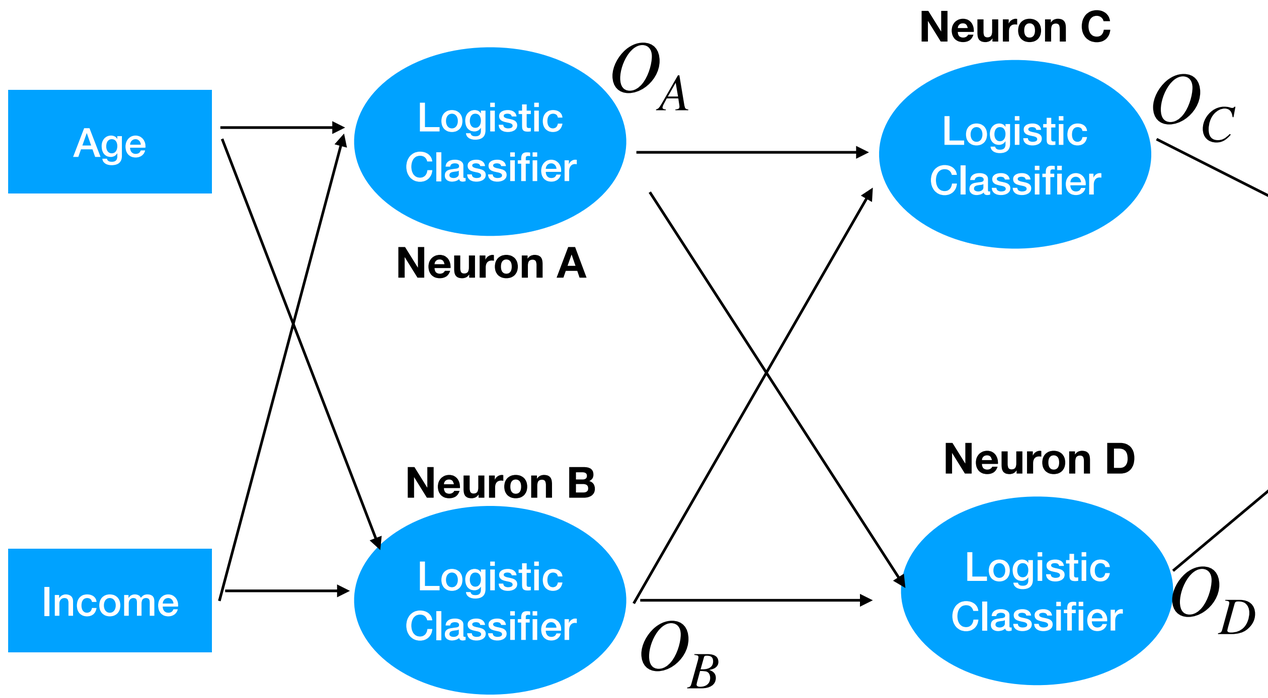


$$O(x_1, x_2, x_3) = \sigma(\theta_0 + \theta_1 \times x_1 + \theta_2 \times x_2 + \theta_3 \times x_3)$$

The backpropagation algorithm

- Actually, a neural network is just a composition of functions
  - And we know how to compute the gradient for one of these functions

$$O_A(income, age) = \sigma(\theta_0^A + \theta_1^A \times income + \theta_2^A \times age)$$



$$O_C(income, age) = \sigma(\theta_0^C + \theta_1^C \times O_A + \theta_2^C \times O_B)$$

$$O_E(income, age) = \sigma(\theta_0^E + \theta_1^E \times O_C + \theta_2^E \times O_D)$$

$$O_B(income, age) = \sigma(\theta_0^B + \theta_1^B \times income + \theta_2^B \times age)$$

$$O_D(income, age) = \sigma(\theta_0^D + \theta_1^D \times O_A + \theta_2^D \times O_B)$$

The backpropagation algorithm

- Actually, a neural network is just a composition of functions
  - And we know how to compute the gradient for one of these functions
- And we have seen there is a chain rule that says that if we know how to compute the derivative of simple functions, we can compute the derivative of their composition

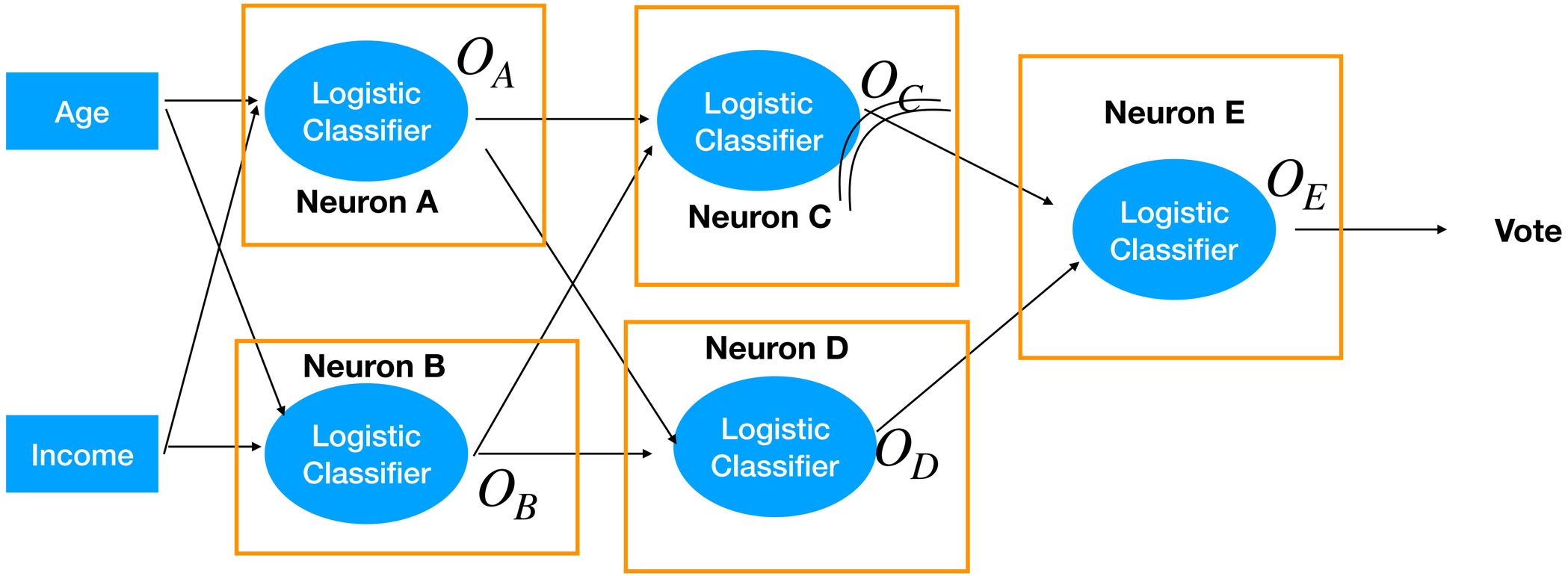
Chain rule

$$f(x) = g(h(x)) \quad \Rightarrow \quad f'(x) = h'(x) \times g'((h(x))$$

- This is the fundamental principle of the back propagation algorithm

The backpropagation algorithm

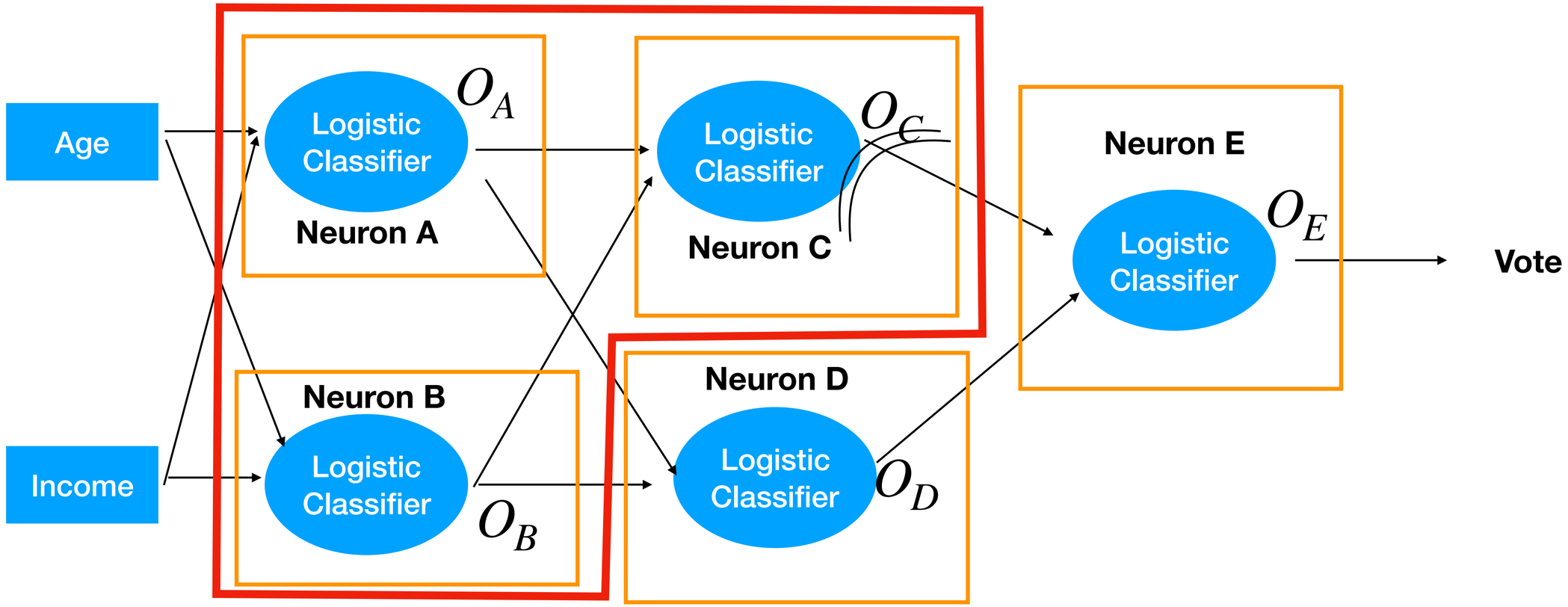
We know how to compute the gradient for the individual neurons:



The backpropagation algorithm

We know how to compute the gradient for the individual neurons

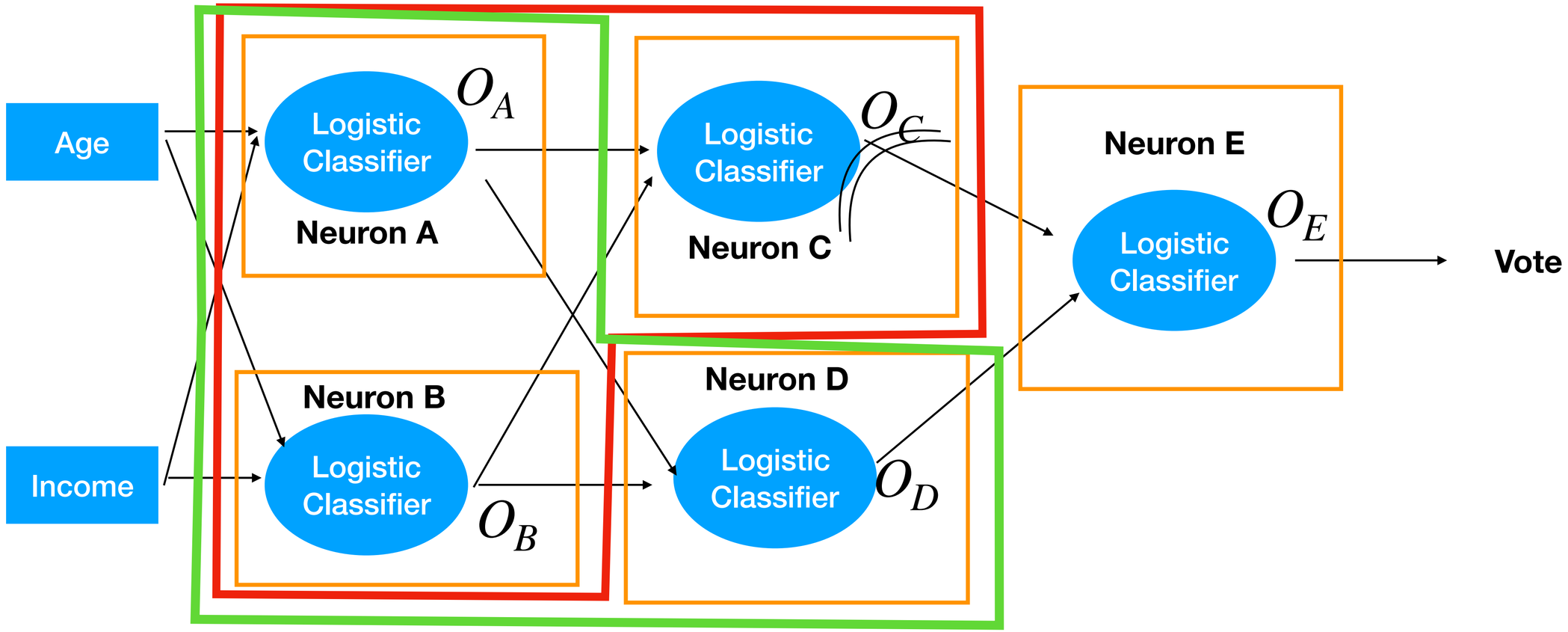
Thanks to the chain rule, we therefore can compute the gradient for this part of the network:



The backpropagation algorithm

We know how to compute the gradient for the individual neurons

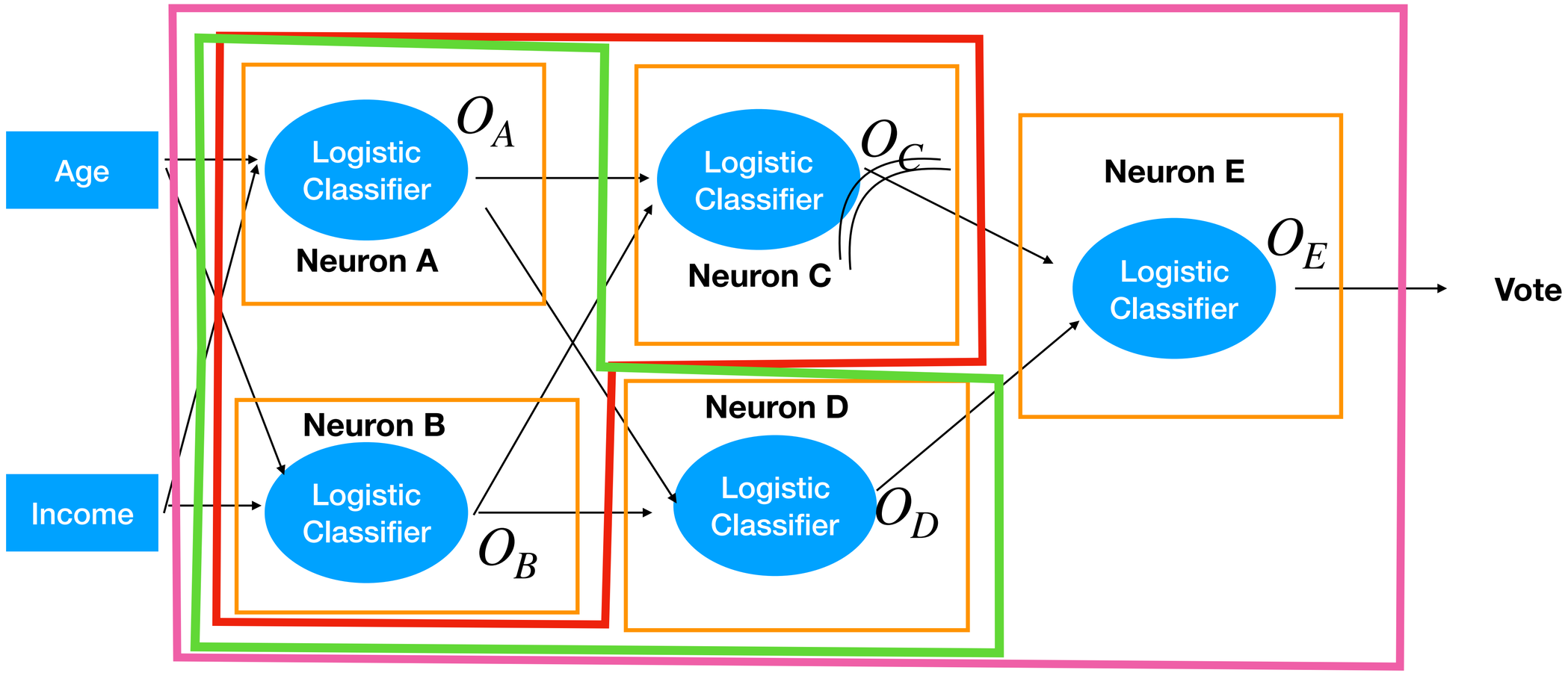
Thanks to the chain rule, we therefore can compute the gradient for this other part of the network:



The backpropagation algorithm

We know how to compute the gradient for the individual neurons

Finally, thanks to the chain rule, we can compute the gradient for the whole network:



The backpropagation algorithm

- That is the general idea: if you have the formula for computing the gradient for **each** part of the neural network, you can compute the gradient for the **whole** network

Backpropagation in practice

- Note that backpropagation do not work only with Neural Networks
- It can be used to compute the derivative of any composition of function
- In order to make you “feel” the process of the back propagation algorithm (rather than describe it), let us apply it on a simple composition of functions

Backpropagation in practice

- Note that backpropagation do not work only with Neural Networks
- It can be used to compute the derivative of any composition of function
- In order to make you “feel” the process of the back propagation algorithm (rather than describe it), let us apply it on a simple composition of functions

Backpropagation in practice:

- To make you feel how it actually work, let us consider the composition of simple functions:

$$h(x) = x + 1$$
$$g(x) = 2(x - 1)^2$$
$$f(x) = x^2 - x + 1$$

- We define K(x) = f(g(h(x)))

- We want to compute the value and the derivative of K for x=1 (for example)

Backpropagation in practice:

- To make you feel how it actually work, let us consider the composition of simple functions:

$$h(x) = x + 1 \qquad g(x) = 2(x-1)^2 \qquad f(x) = x^2 - x + 1$$

- We define K(x) = f(g(h(x)))
- We want to compute the value and the derivative of K for x=1 (for example)
- One way is to compute K explicitly: $K(x) = 4x^4 - 2x^2 + 1$
  - Then K(1) = 4-2+1 = 3
- We can also compute K'(x) explicitly: $K'(x) = 16x^3 - 4x$
  - Then K'(1) = 16 - 4 = 12

Backpropagation in practice:

- To make you feel how it actually work, let us consider the composition of simple functions:

$$h(x) = x + 1 \qquad g(x) = 2(x-1)^2 \qquad f(x) = x^2 - x + 1$$

- We define K(x) = f(g(h(x)))
- We want to compute the value and the derivative of K for x=1 (for example)
- Now, let us do it using backpropagation!

Backpropagation in practice:

- First, let us make sure we know the derivative of each individual function:

$$h(x) = x + 1 \qquad h'(x) = 1$$

$$g(x) = 2(x-1)^2 \qquad g'(x) = 4(x-1)$$

$$f(x) = x^2 - x + 1 \qquad f'(x) = 2x - 1$$

- Let us try to see K(x) = f(g(h(x))) as if it was a neural network:

Note: this graph is called the “computation graph” of K

Backpropagation in practice:

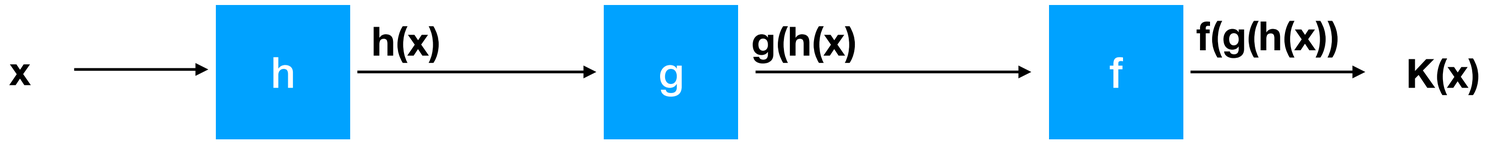
- First, let us make sure we know the derivative of each individual function:

$$h(x) = x + 1 \qquad h'(x) = 1$$

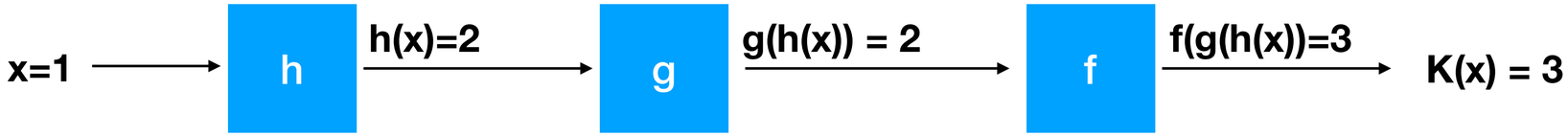
$$g(x) = 2(x-1)^2 \qquad g'(x) = 4(x-1)$$

$$f(x) = x^2 - x + 1 \qquad f'(x) = 2x - 1$$

- Let us try to see K(x) = f(g(h(x))) as if it was a neural network:



- Then let us compute K(1):



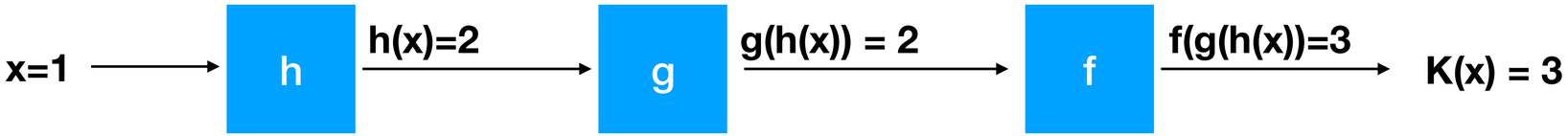
Backpropagation in practice:

- First, let us make sure we know the derivative of each individual function:

$$h(x) = x + 1$$
$$g(x) = 2(x-1)^2$$
$$f(x) = x^2 - x + 1$$

$$K(x) = f(g(h(x)))$$

$$h'(x) = 1$$
$$g'(x) = 4(x-1)$$
$$f'(x) = 2x - 1$$



- By applying the chain rule twice, we have: $K'(x) = f'(g(h(x))) \times g'(h(x)) \times h'(x)$
- Note that we have already computed h(x) and g(h(x))!
- Let us compute K'(1):

Backpropagation in practice:

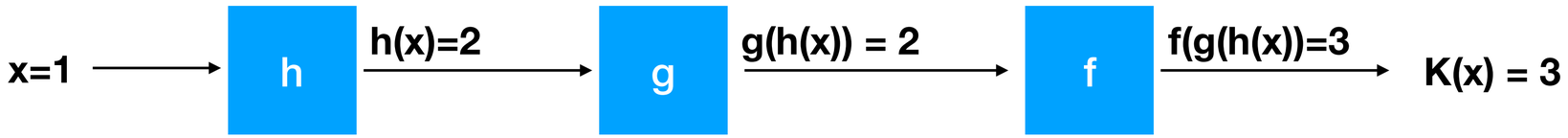
- First, let us make sure we know the derivative of each individual function:

$$h(x) = x + 1$$
$$g(x) = 2(x-1)^2$$
$$f(x) = x^2 - x + 1$$

$$K(x) = f(g(h(x)))$$

$$h'(x) = 1$$
$$g'(x) = 4(x-1)$$
$$f'(x) = 2x - 1$$

- By applying the chain rule twice, we have: $K'(x) = f'(g(h(x))) \times g'(h(x)) \times h'(x)$
- Note that we have already computed h(x) and g(h(x))!
- Let us compute K'(1):



K'(1) = 3 x 4 x 1 = 12

h'(x) = 1

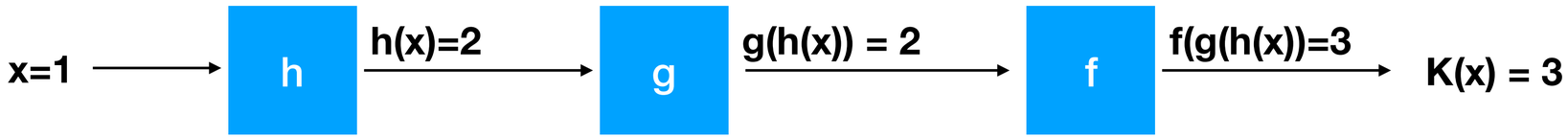
g'(h(x)) = 4

f'(g(h(x))) = 3

Backpropagation in practice:

- Note that we did the computation in two passes:
  - First pass compute K(1)
  - Second pass reuse the computations of the first pass to compute K’(1)
- Backpropagation works always like that:
  - The first pass is called the ***forward pass***
  - The second pass is called the ***backward pass***

$$K'(x) = f'(g(h(x))) \times g'(h(x)) \times h'(x)$$



K’(1) = 3 x 4 x 1 = 12

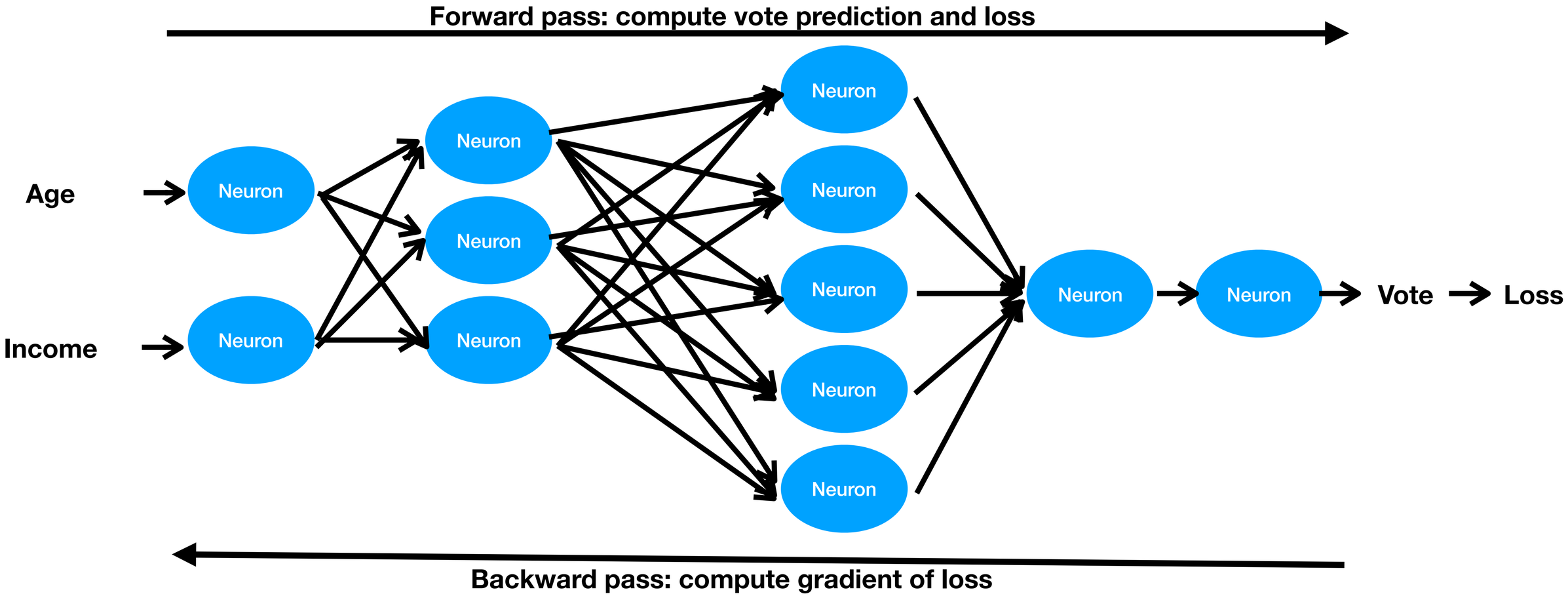
h’(x) = 1

g’(h(x)) = 4

f’(g(h(x))) = 3

The backpropagation algorithm

- Backpropagation works exactly the same way on neural networks:



A word about the different approaches for computing derivatives

- Note that ***backpropagation*** is a specific case of ***Automatic Differentiation***

- There exists 3 methods for computing a derivative with a computer:

  - Automatic differentiation

  - Symbolic differentiation

  - Numerical differentiation

A word about the different approaches for computing derivatives

- Note that ***backpropagation*** is a specific case of ***Automatic Differentiation***

- There exists 3 methods for computing a derivative with a computer:

  - Automatic differentiation

  - Symbolic differentiation

  - Numerical differentiation

A word about the different approaches for computing derivatives

- Going back to our example, let us illustrate how each method compute the derivative of K(x) = f(g(h(x))):
  - Automatic differentiation use the computation graph (like we saw)
  - Symbolic differentiation is the first thing we tried: compute K and K' explicitly:

$$K(x) = 4x^4 - 2x^2 + 1 \qquad K'(x) = 16x^3 - 4x \qquad \longrightarrow \qquad K'(1) = 12$$

  - Numerical differentiation K(1.0001)-K(1)

$$K'(1) \approx \frac{K(1.0001) - K(1)}{0.0001} = \frac{3.0012002 - 3}{0.0001} \approx 12$$

A word about the different approaches for computing derivatives

- Note that ***backpropagation*** is a specific case of ***Automatic Differentiation***
- There exists 3 methods for computing a derivative with a computer:
  - Automatic differentiation
    - Commonly used for neural network
  - Symbolic differentiation
    - Sometimes used in combination with backpropagation for neural network (but usually less efficient)
  - Numerical differentiation
    - Way too slow! (and approximative)

How about you try?

- We define K(x) = f(g(h(x))) with these functions. Compute K(1) and K’(1) with both the symbolic method and the backpropagation method

$$h(x) = x^2 \qquad g(x) = 3x \qquad f(x) = 1 + x^2$$

$$K(x) = f(g(h(x)))$$

$$K'(x) = f'(g(h(x))) \times g'(h(x)) \times h'(x)$$

Neural Network Libraries

- As mentioned, we normally use libraries that will do all this backpropagation work for us

Flow of training with a library:

- Define Model M

We will see how next time

- Repeat:
  - Take some example (input, desired_output) (eg. ([age,income], vote))
  - prediction: = Model(x)
  - loss := loss_function(prediction, desired_output)
  - loss.backward()
  - optimizer.update(model)

Forward pass

Ask library to compute backward pass

Ask library to perform a gradient descent update

Which libraries

- There are many existing libraries for using Neural Network: Tensorflow, Torch, PyTorch, Chainer, Keras,….
- Let us describe a few of them

Which libraries

- ***Tensorflow*** is the library developed by Google
  - It is used internally by Google developpers (eg. Google Translate run on tensorflow)
  - Open Source
  - Use Python or C++ programming language

Which libraries

- ***Chainer*** is a library developed the Japanese company Preferred Networks
  - Open Source
  - Uses Python
  - Easy to use

Which libraries

- ***Torch*** and **PyTorch**
  - Currently sponsored and used by Facebook
  - Open Source
  - Torch uses the LUA programming language
  - PyTorch uses the Python programming language
  - (PyTorch was initially a fork of ***chainer*** that got adapted to uses the torch libraries)

Which libraries

- ***Theano***
  - One of the oldest library
  - Developed “Universite de Montreal”
  - Open Source
  - Can use symbolic differentiation
  - A bit difficult to use

Which libraries

- ***Keras***
  - Library that run on top of Tensorflow or Theano
  - Make them easier to use

For next time

- From next week, we will train real neural networks using ***chainer*** or ***PyTorch***
- Please try to install both by running:
  - conda install pytorch-cpu -c pytorch
  - pip install chainer