Neural Networks with Fully Connected Layers in Practice

Fundamentals of Artificial Intelligence

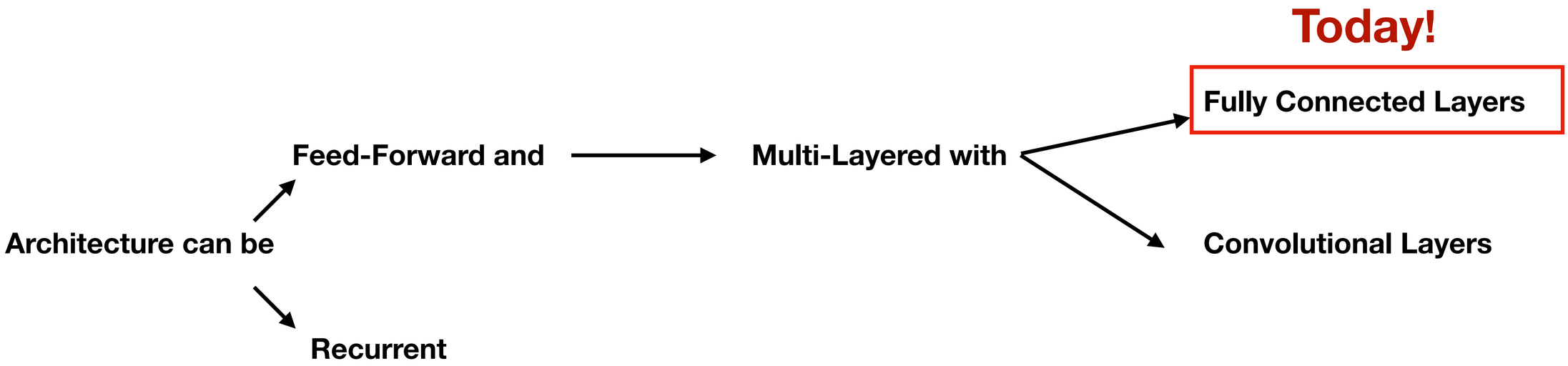
Fabien Cromieres
Kyoto University

Program for today:

- 1- Discuss the mathematical representation of Fully-connected layers in Neural Networks
- 2- Define and train a real Neural Network in a Jupyter Notebook

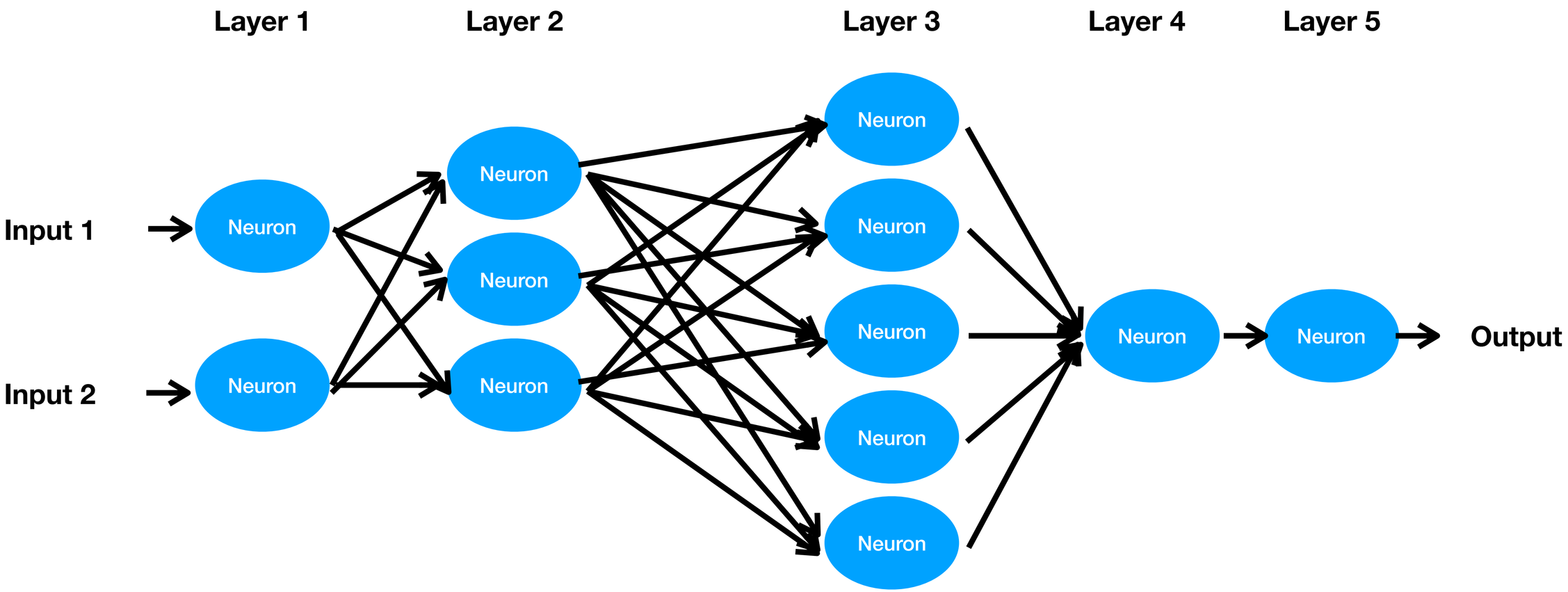
Neural Network Architectures

- We are still here:



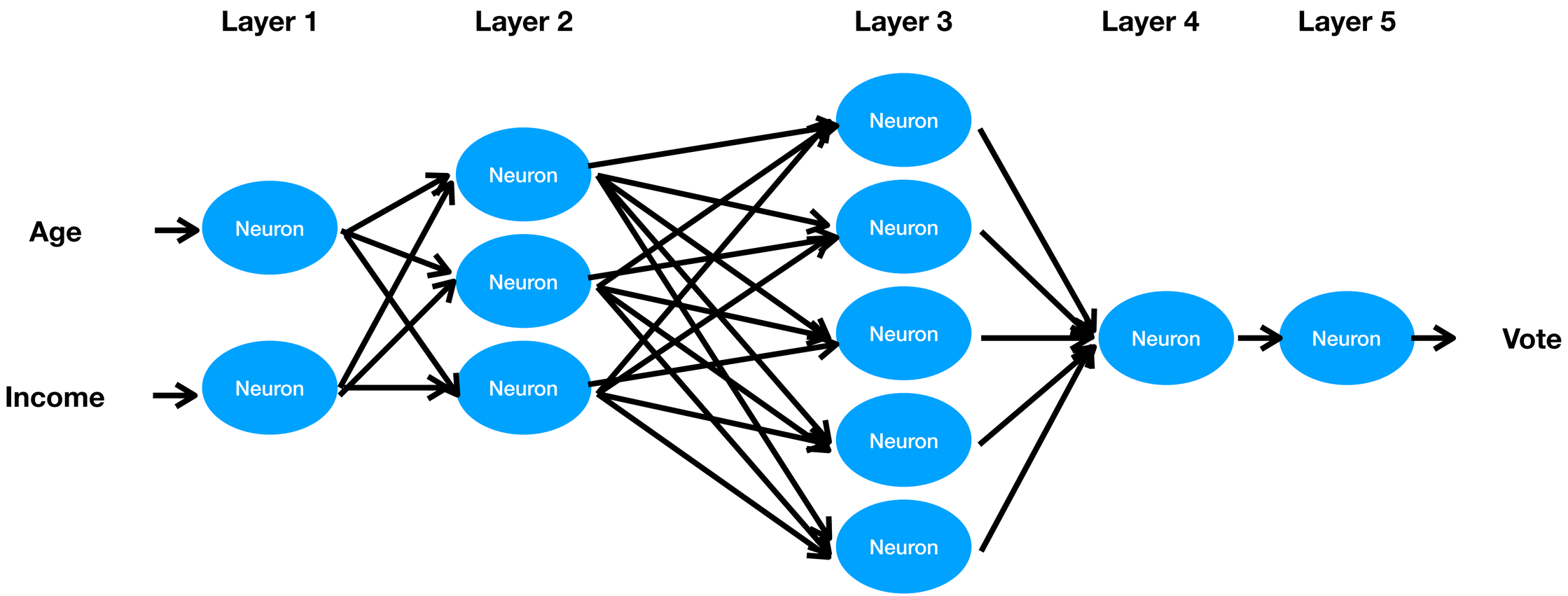
Feed-Forward networks with fully connected layers

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



Feed-Forward networks with fully connected layers

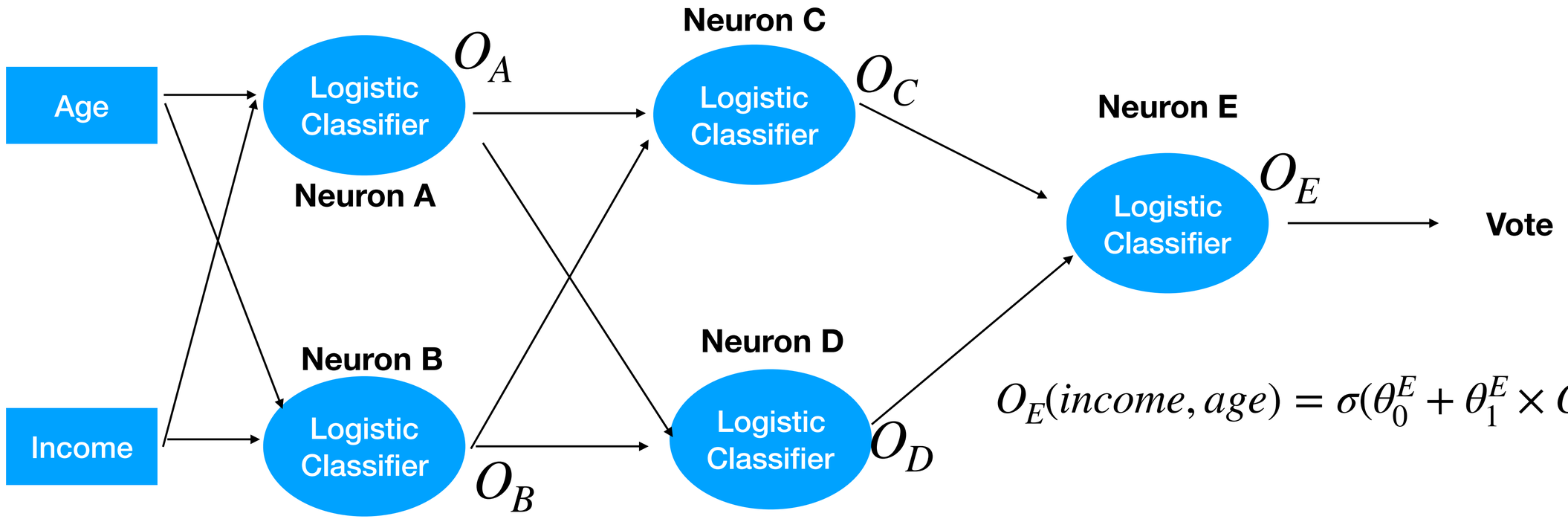
- Therefore, we are going to consider this type of Neural Network:



Keeping in mind what this type of graph mean

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

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



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

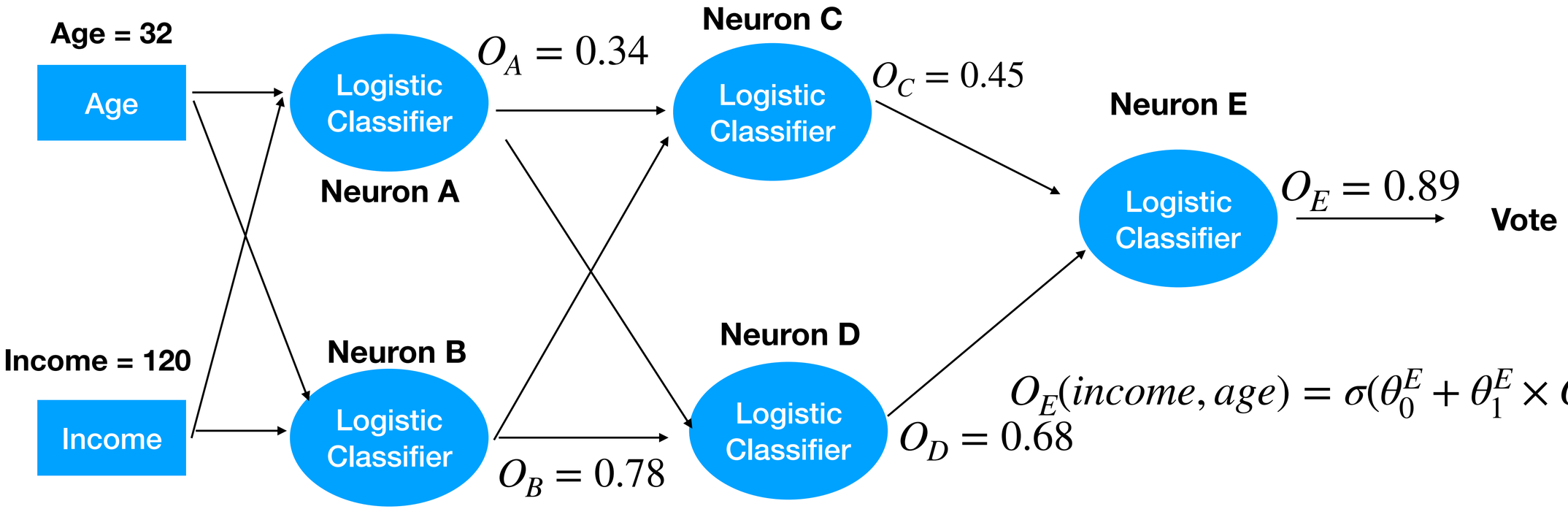
$$O_B(income, age) = \sigma(\theta_0^B + \theta_1^B \times income + \theta_2^B \times age)$$

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

Keeping in mind what this type of graph mean

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

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



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

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

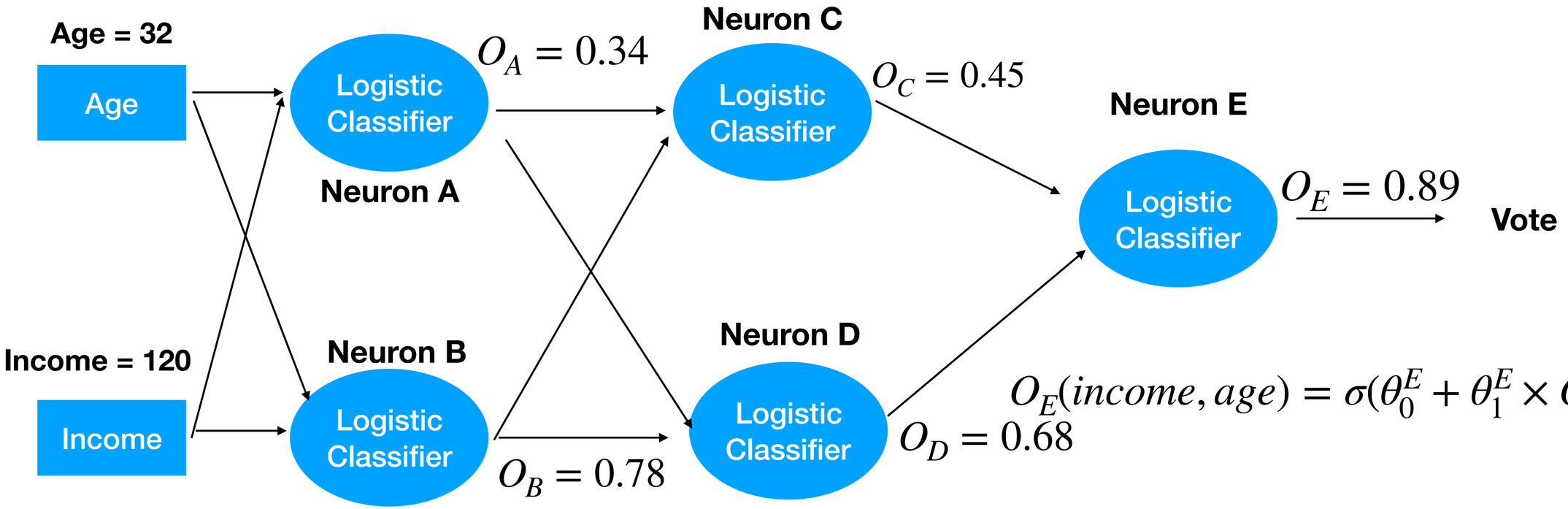
$$O_D(income, age) = \sigma(\theta_0^D + \theta_1^D \times O_A + \theta_2^D \times O_B)$$

Keeping in mind what this type of graph mean

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

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

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



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

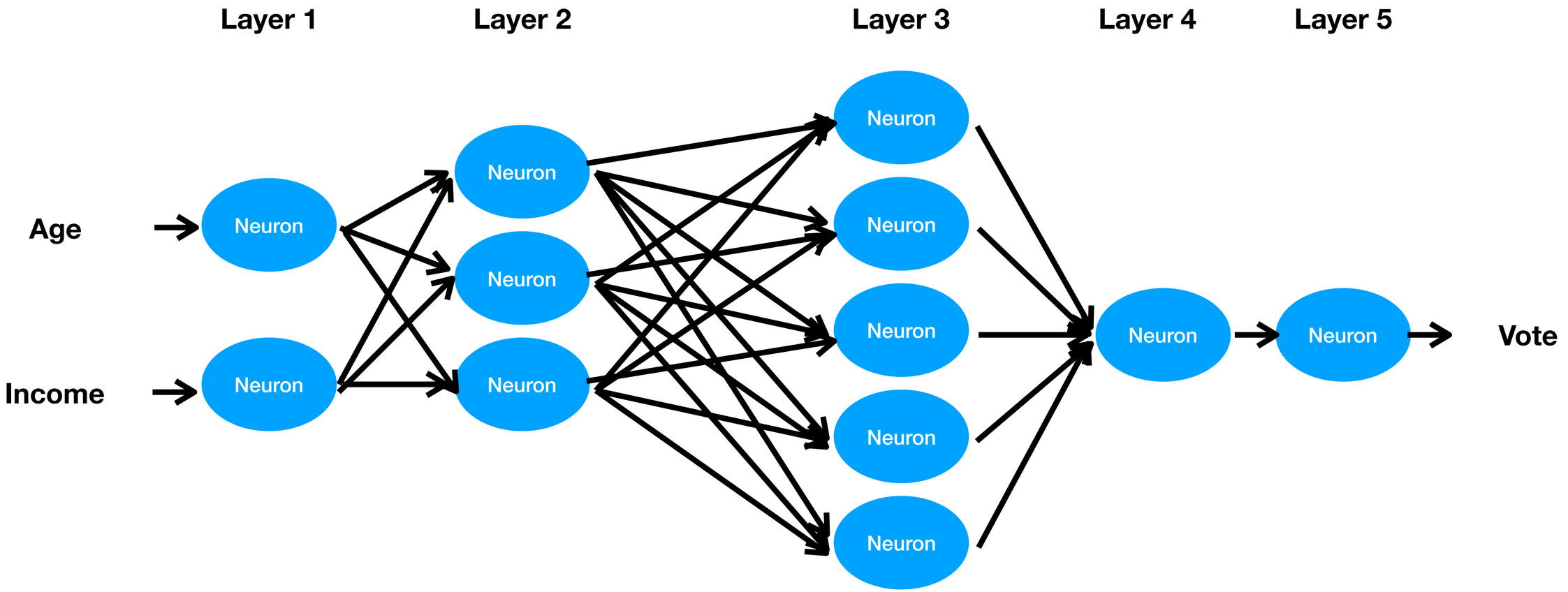
$$O_B(income, age) = \sigma(\theta_0^B + \theta_1^B \times income + \theta_2^B \times age)$$

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

Feed-Forward networks with fully connected layers

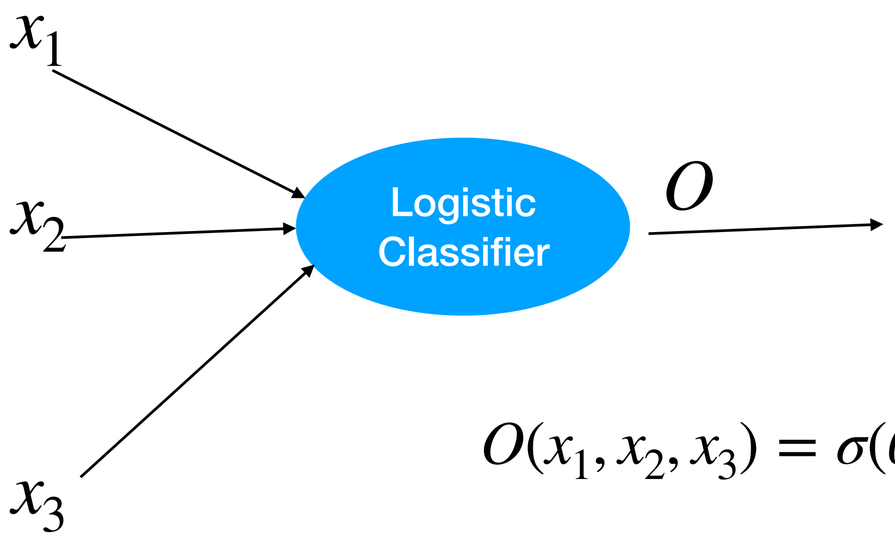
—> Each Neural Network architecture defines a function of the input with parameters θ

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



Parameters

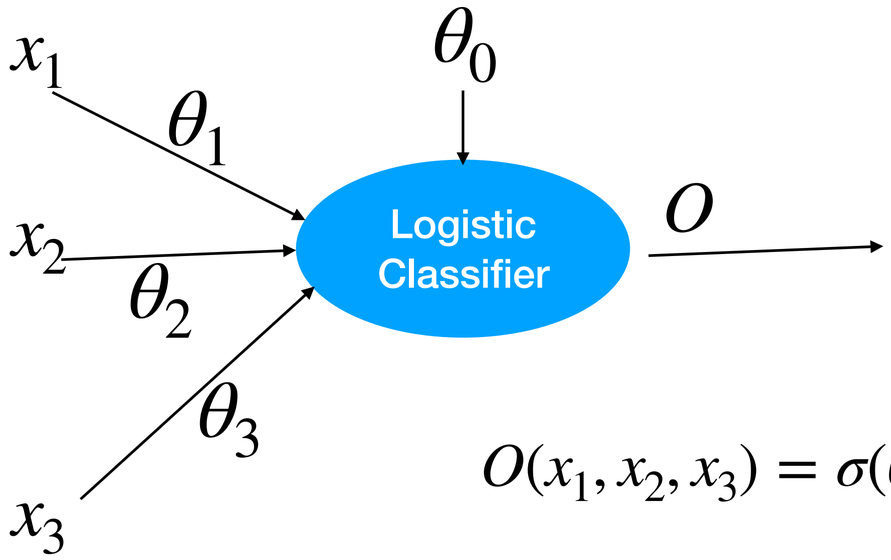
- If a neuron has N inputs, it has N+1 parameters



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

Parameters

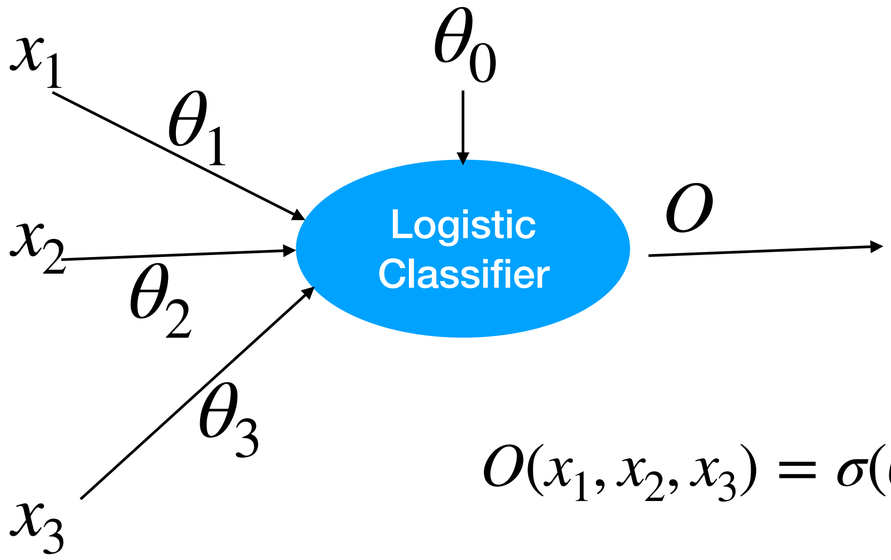
- If a neuron has N inputs, it has N+1 parameters
  - Visually, we can associate a parameter to each input, and show θ_0 separately



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

Parameters: Terminology

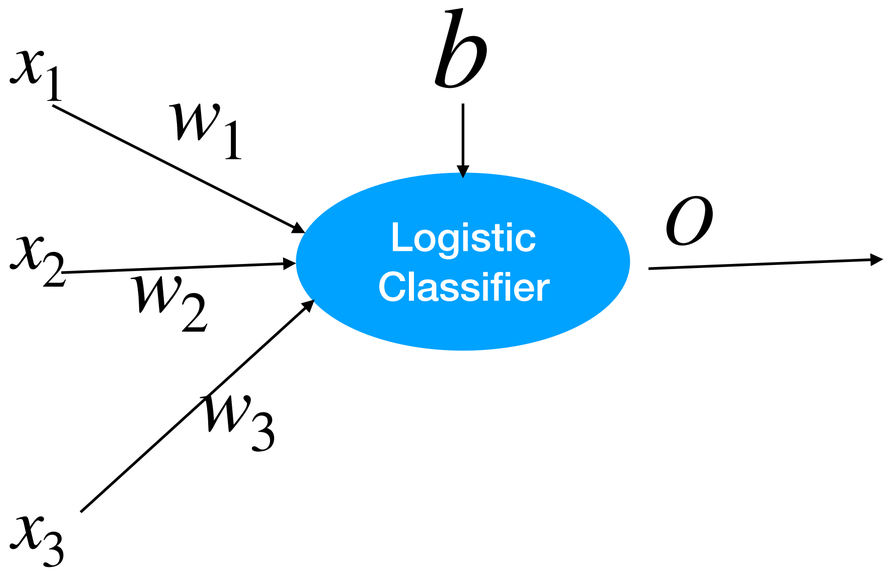
- θ_1, θ_2, θ_3 are often called the ***weights*** of the neuron
- θ_0 is often called the ***bias*** of the neuron



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

Parameters: Terminology

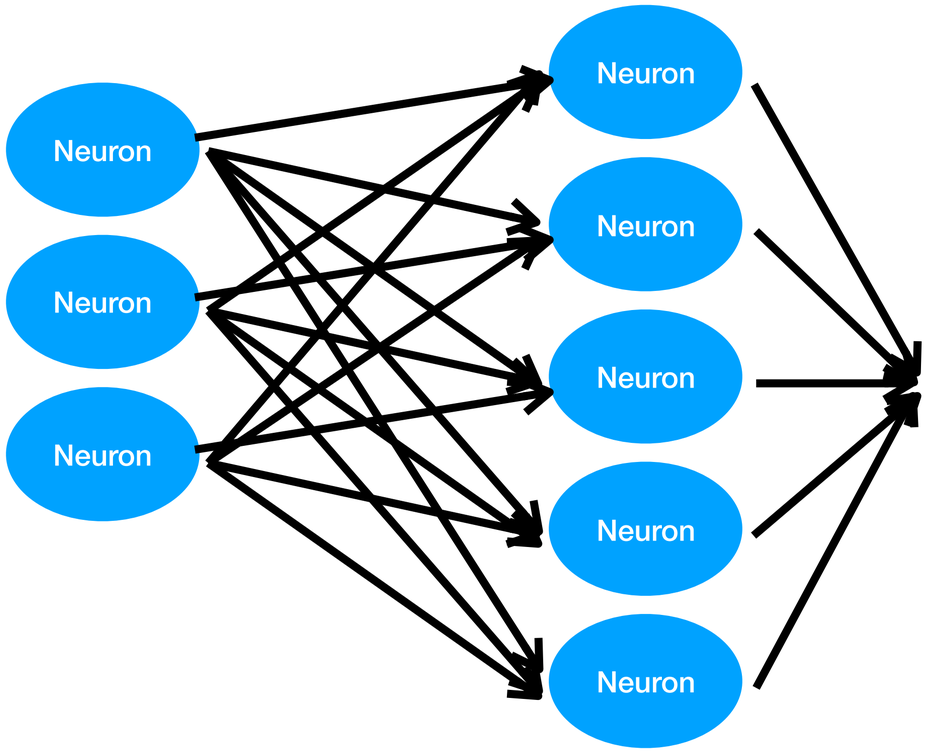
- θ_1, θ_2, θ_3 are often called the ***weights*** of the neuron
  - They are therefore often also noted w_1, w_2, w_3
- θ_0 is often called the ***bias*** of the neuron
  - It is often noted ***b***



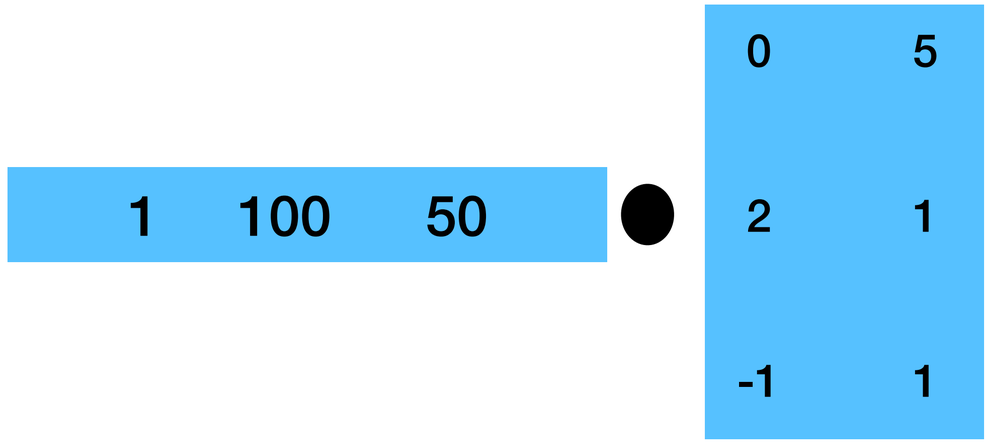
$$O(x_1, x_2, x_3) = \sigma(b + w_1 \times x_1 + w_2 \times x_2 + w_3 \times x_3)$$

“Connectionist” view vs “Mathematical” view

- “Connections” viewpoint
  - *Fully Connected Layers*

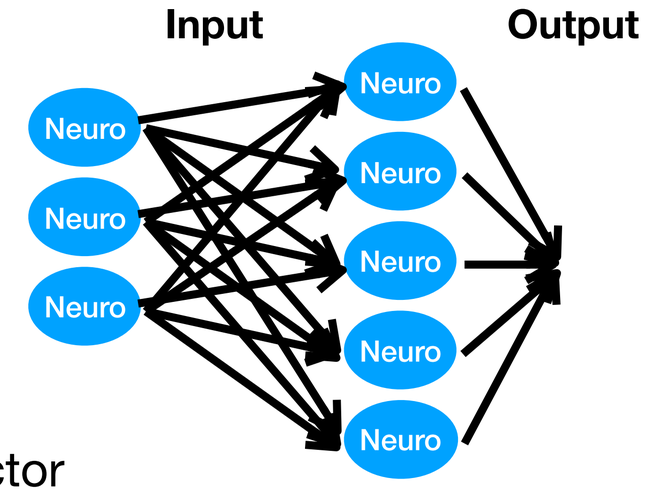


- Math operations viewpoint
  - **Matrix multiplications**



“Connectionist” view vs “Mathematical” view

- In practice, computing the output of a fully connected layer is equivalent to computing:
  - a Matrix multiplication
  - followed by an activation
- Equivalences:
  - Input of the layer (= output of the previous layer) <-> row vector
  - Output of the layer (= set of output of each neuron in a layer) <-> row vector
  - Weights of a neuron <-> column vector
  - Set weights of all neurons in a fully connected layer <-> matrix
  - Batched input of a layer (= several inputs at once) <-> matrix



Linear Algebra

- In order to discuss the way Neural Networks are actually implemented, we need to discuss some mathematical concepts
- We discuss:
  - Vector scalar products (a.k.a inner product)
  - Matrices
  - Matrix-vector multiplication
  - Matrix-Matrix multiplication
- Warning: it is a bit ambitious to explain that in 30 minutes. Hopefully you understand most of it.

Row Vectors and Column Vectors

- We have seen that ***vectors*** are just a “list” of numbers
- We are actually going to distinguish 2 “types” of vectors: ***row vectors*** and ***column vectors***

(5-dimensional)
Column vector:

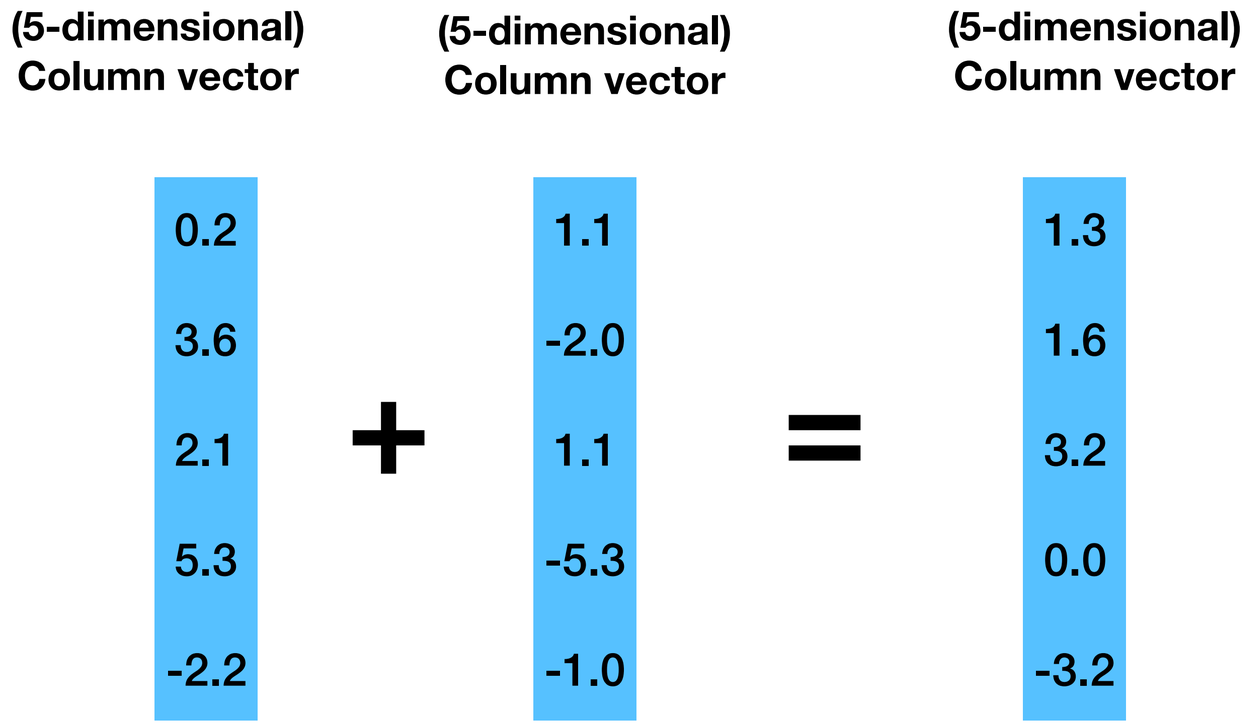
0.2
3.6
2.1
5.3
-2.2

(5-dimensional)
Row vector:

0.2	3.6	2.1	5.3	-2.2

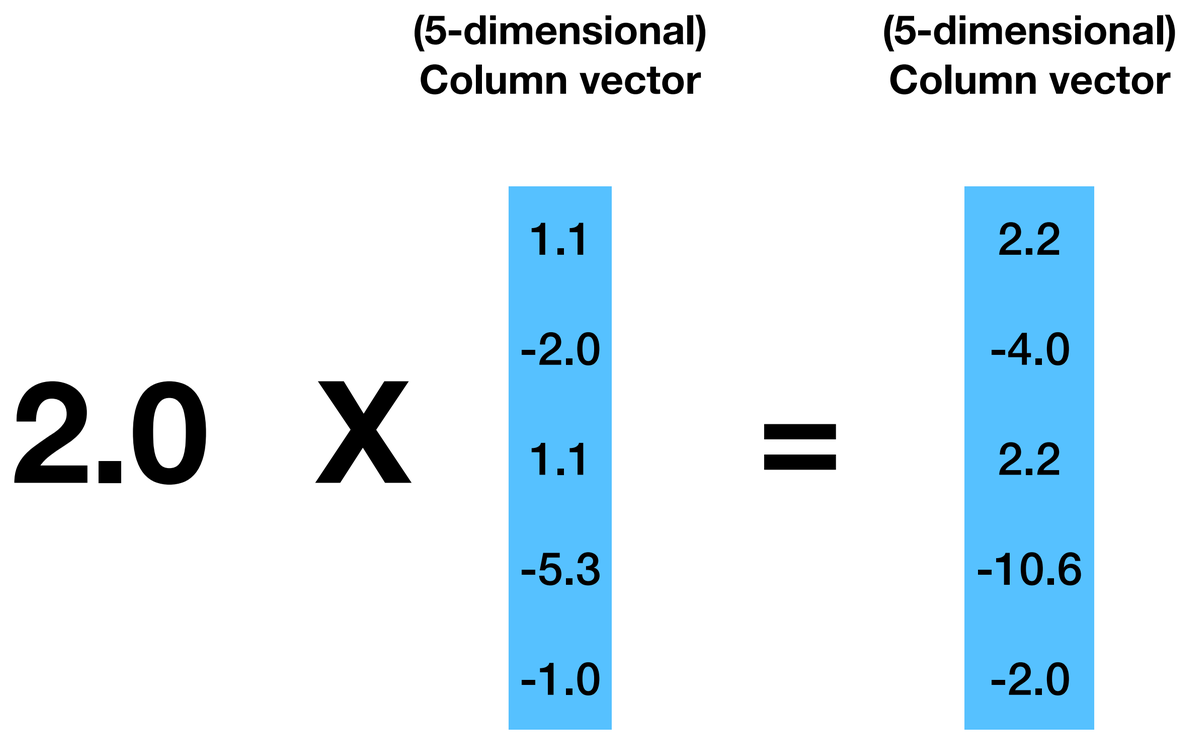
Column vectors

- Column vectors of the same dimension can be added



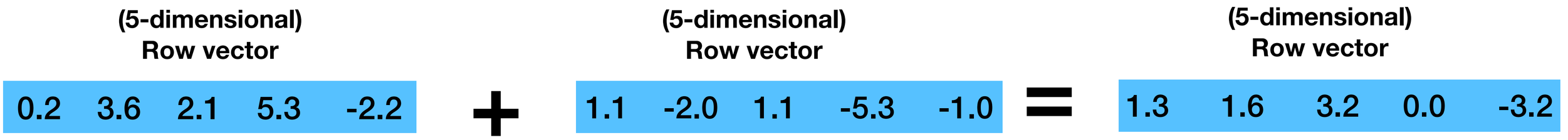
Column vectors

- Column vectors can be multiplied by a number



Row vectors

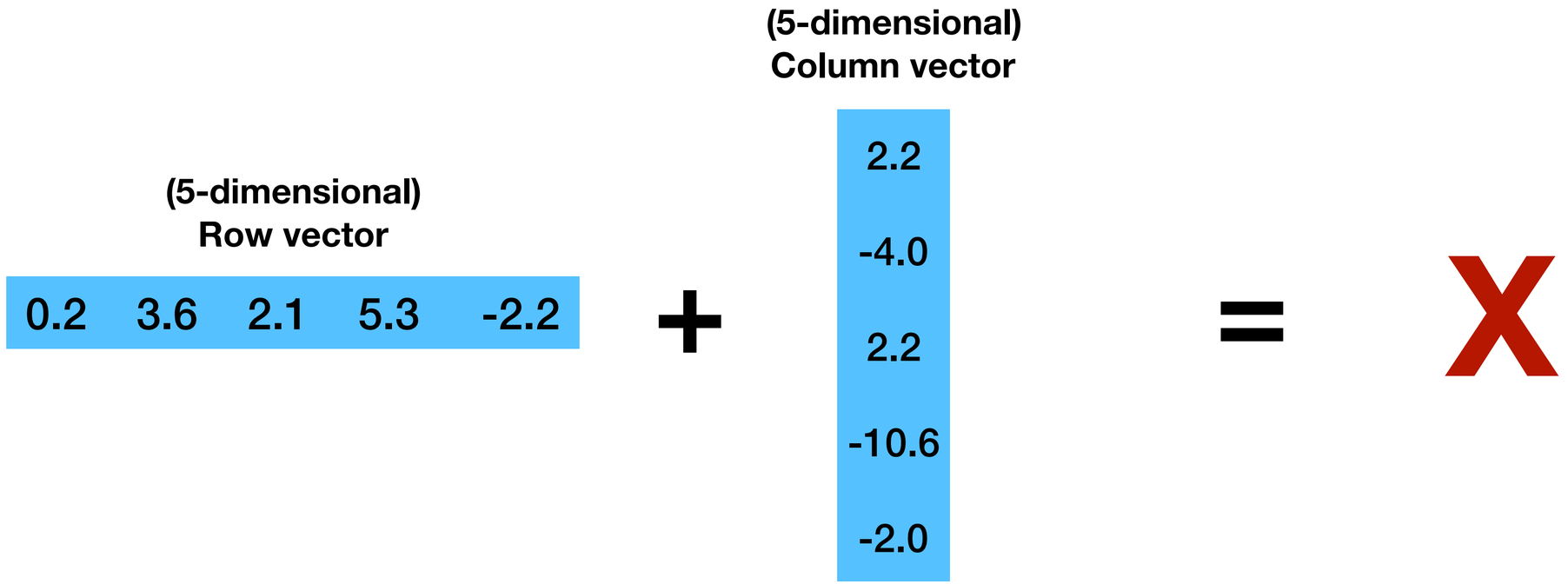
- Row vectors have the same operations as Column vectors
- They can be added:



- They can be multiplied by a number:

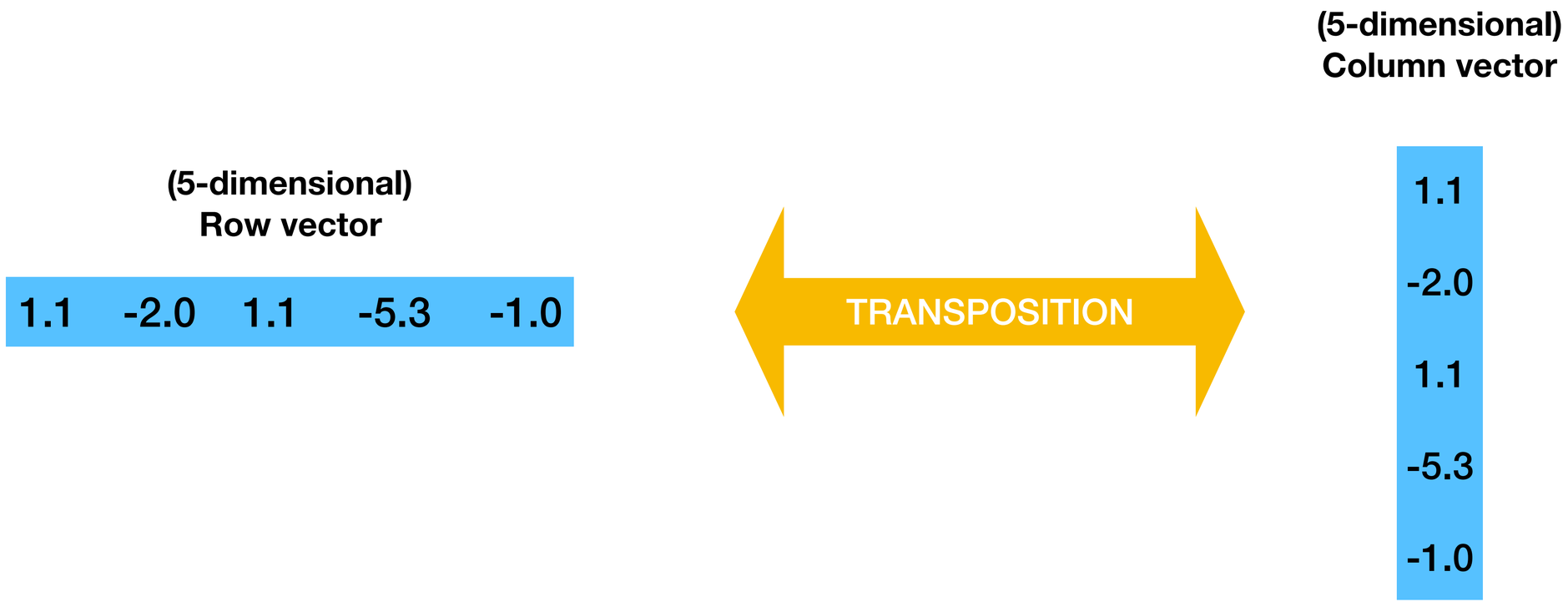
Row vectors and Column Vectors

- Row vectors and Column vectors cannot be added together



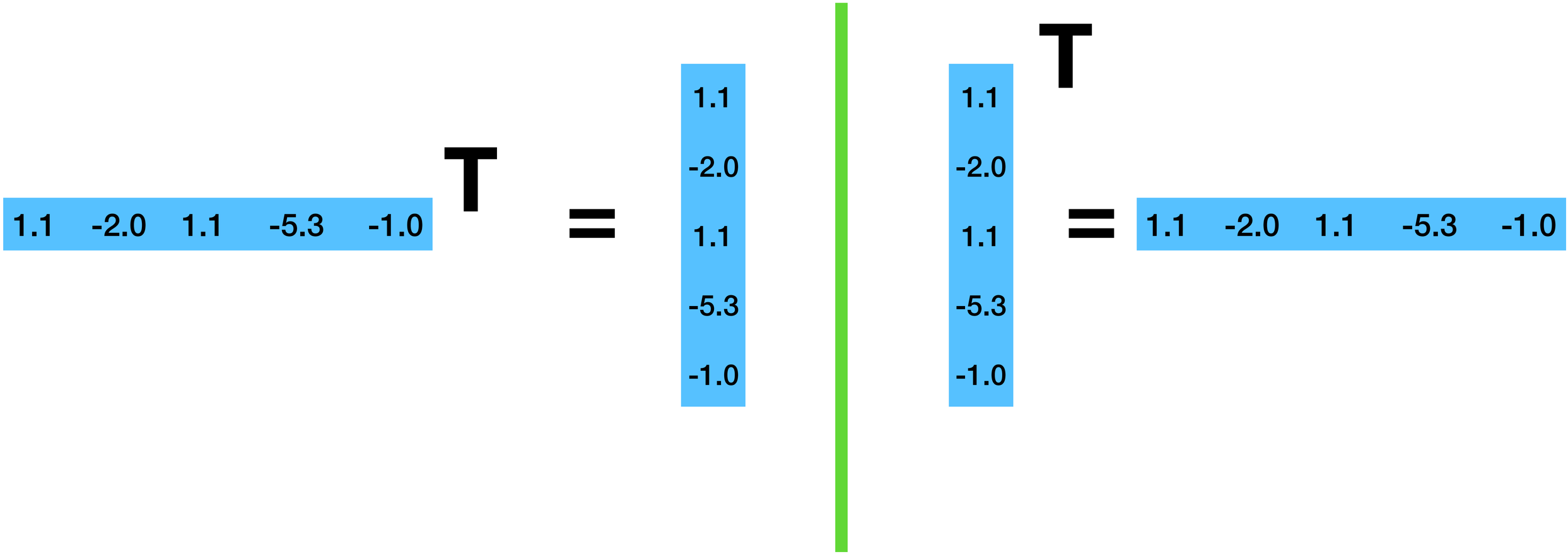
Transposition

- However, *row vectors* can be transformed in *column vectors* by an operation called ***transposition*** (and vice-versa)



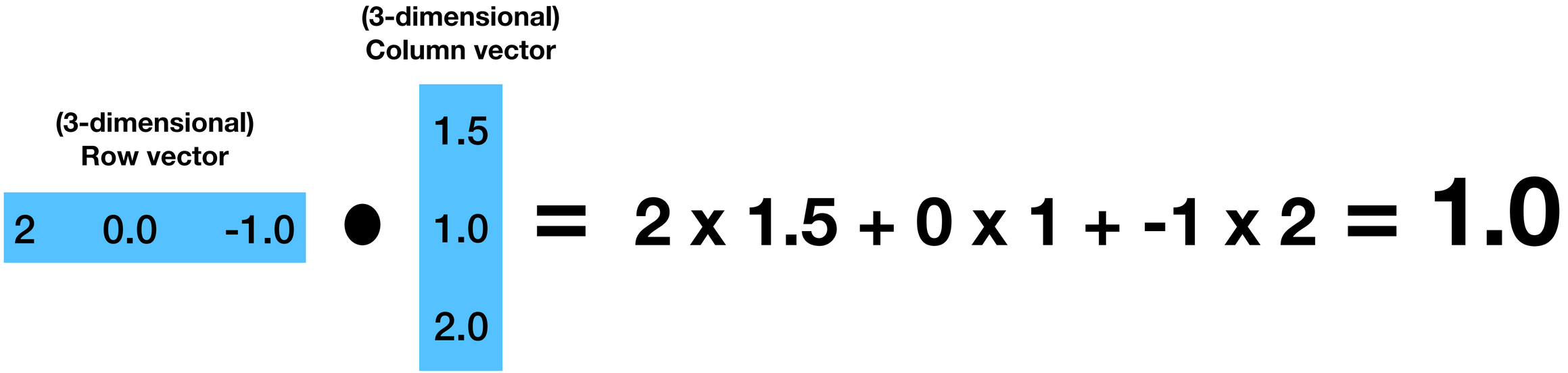
Transposition

- Transposition is usually noted with a ***T*** in exponent:



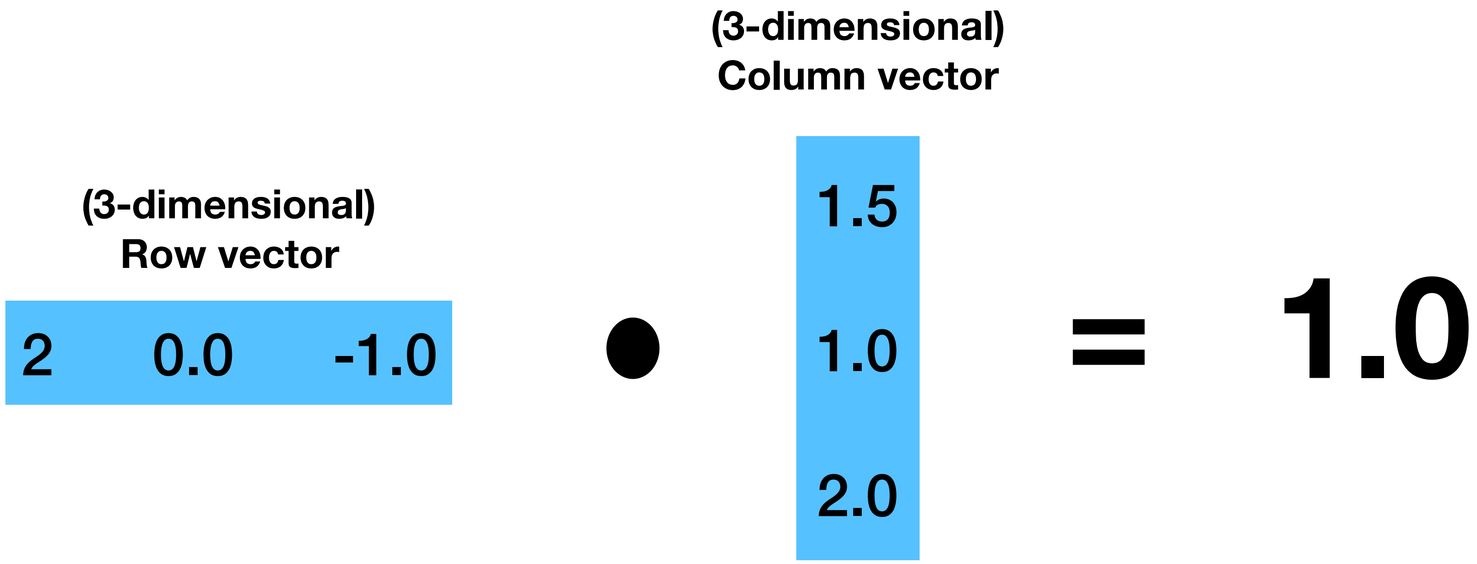
Inner product (a.k.a Scalar product)

- We can compute the ***inner product*** of a *row vector* and a *column vector* to obtain a single number
- It consists in taking the product of the number dimension by dimension and then taking the sum

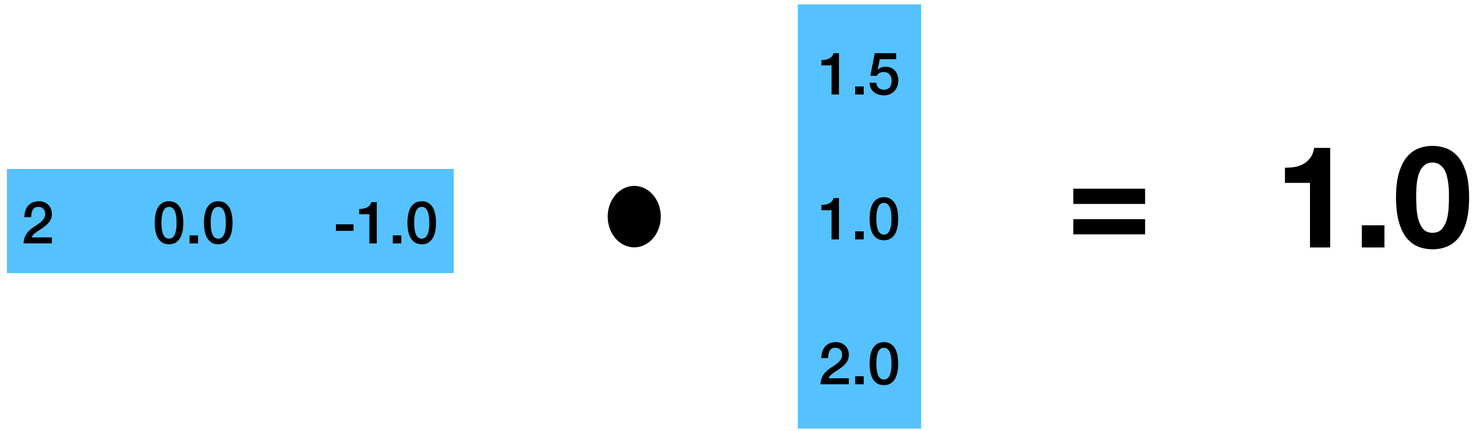


Inner product (a.k.a Scalar product)

- The vectors should have the **same dimension**
- You should have the *row vector* on the left, and the *column vector* on the right
  - (if it is the opposite, the operation is called *outer product* and gives a different result)

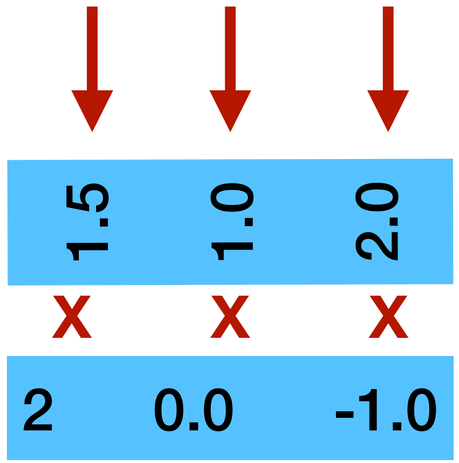


Inner product (a.k.a Scalar product)



You can try to visualize this as the column vector lying down on the row vector to produce the number

Inner product (a.k.a Scalar product)



$\bullet$ = **1.0**

3.0 + 0.0 + -2.0 → 1.0

You can try to visualize this as the column vector lying down on the row vector to produce the number

Why inner product is important?

- It allows us to **express linear functions efficiently**
- And remember that linear functions are one of the **fundamental components** of Machine Learning

Linear regression

$$f(x, y) = \theta_0 + \theta_1 \times x + \theta_2 \times y$$

Logistic Classifier

$$score(x, y) = \theta_0 + \theta_1 \times x + \theta_2 \times y$$

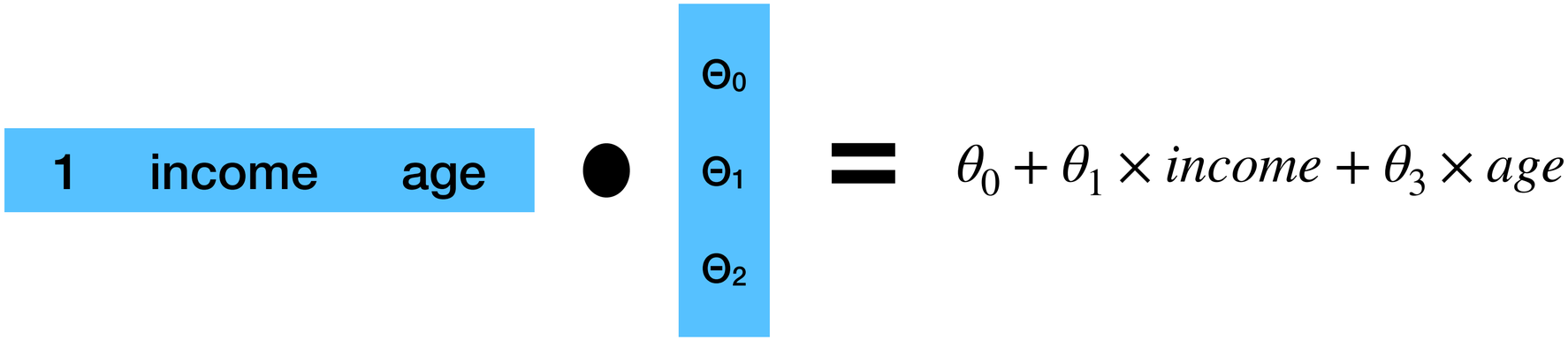
$$prediction = \sigma(score)$$

Neuron (same formula as Logistic Classifier)

$$output(x, y) = \sigma(\theta_0 + \theta_1 \times x + \theta_2 \times y)$$

Why inner product is important?

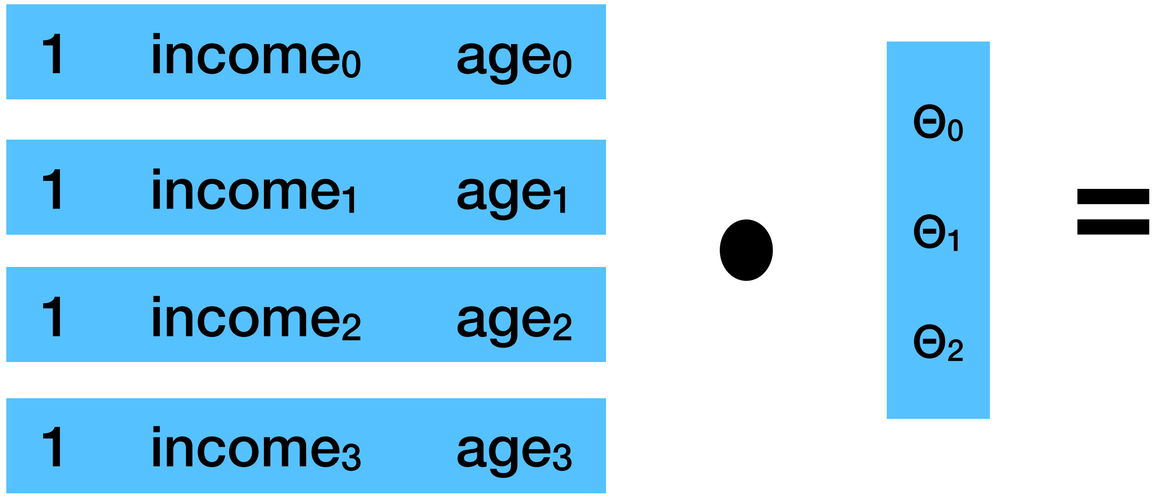
$$vote(age, income) = \sigma(\theta_0 + \theta_1 \times age + \theta_2 \times income)$$



$$vote(age, income) = \sigma\left(\begin{bmatrix} 1 & income & age \end{bmatrix} \bullet \begin{bmatrix} \Theta_0 \\ \Theta_1 \\ \Theta_2 \end{bmatrix} \right)$$

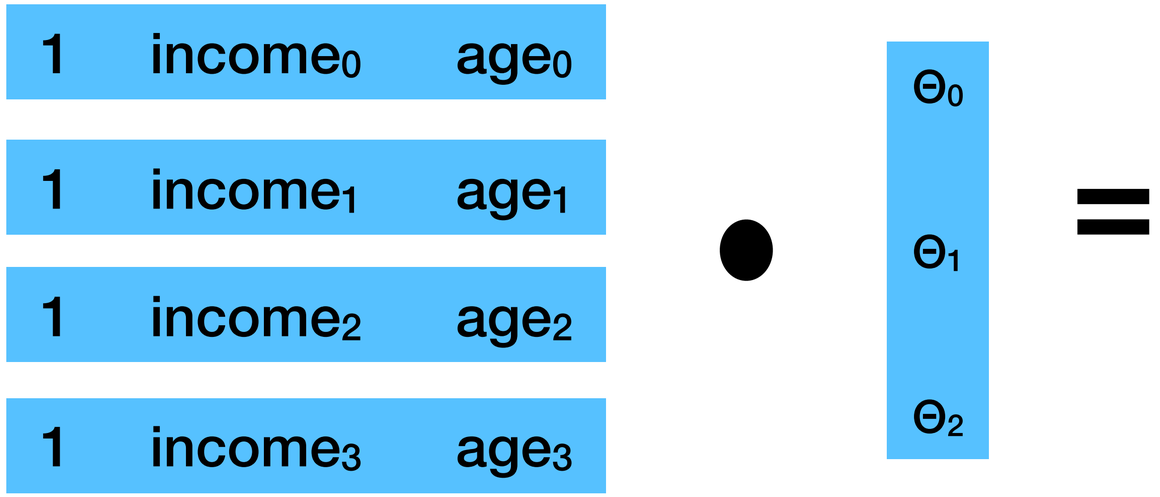
Representing many inner product at once

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



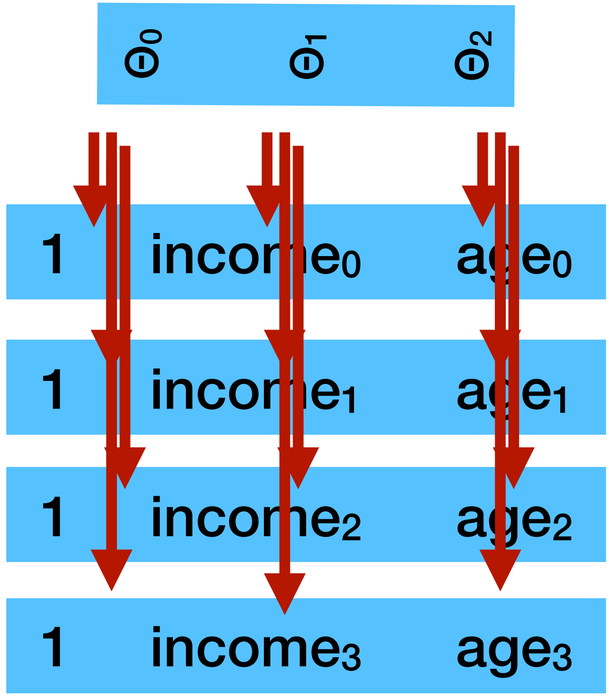
Representing many inner product at once

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



Representing many inner product at once

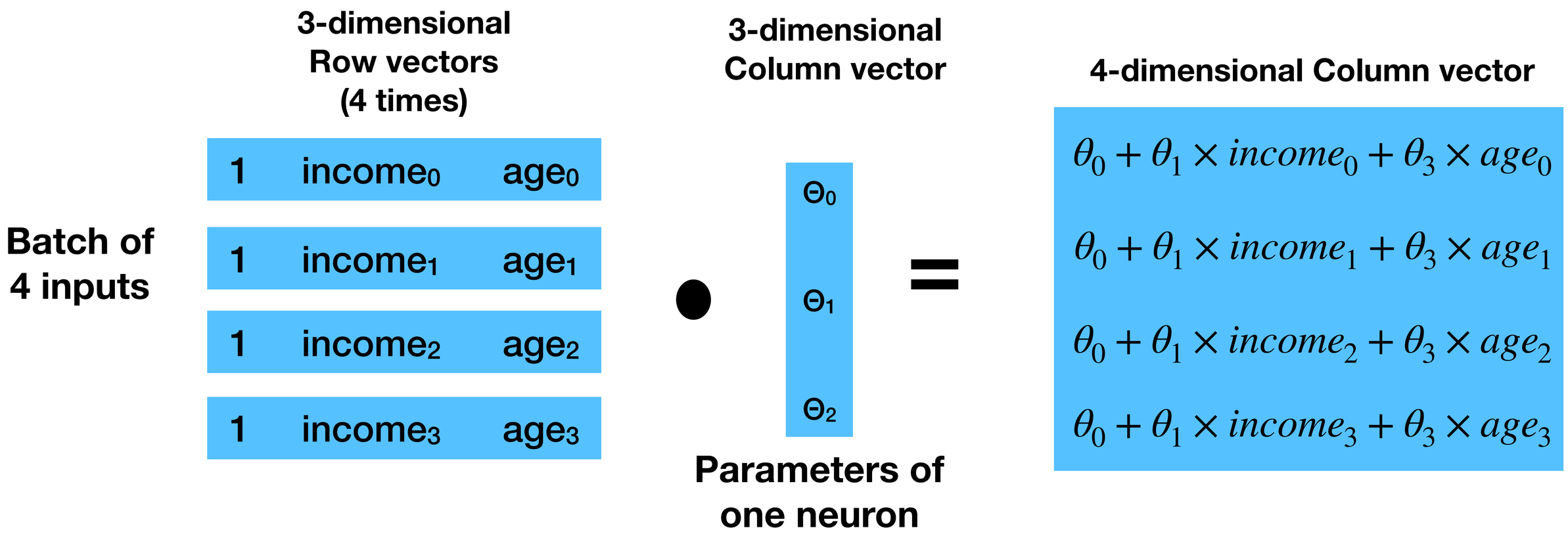
$$vote(age, income) = \sigma(\theta_0 + \theta_1 \times age + \theta_2 \times income)$$



$$\bullet \quad = \quad \begin{matrix} \theta_0 + \theta_1 \times income_0 + \theta_3 \times age_0 \\ \theta_0 + \theta_1 \times income_1 + \theta_3 \times age_1 \\ \theta_0 + \theta_1 \times income_2 + \theta_3 \times age_2 \\ \theta_0 + \theta_1 \times income_3 + \theta_3 \times age_3 \end{matrix}$$

Representing many inner product at once

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



This represents the computation of the score output of a single neuron for 4 inputs at the same time!

Representing many inner product at once

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

3-dimensional Row vectors (4 times)

3-dimensional Column vector

4-dimensional Column vector

Batch of 4 inputs

1	$income_0$	age_0
1	$income_1$	age_1
1	$income_2$	age_2
1	$income_3$	age_3

$\bullet$

Θ_0
Θ_1
Θ_2

$=$

$$\theta_0 + \theta_1 \times income_0 + \theta_3 \times age_0$$

$$\theta_0 + \theta_1 \times income_1 + \theta_3 \times age_1$$

$$\theta_0 + \theta_1 \times income_2 + \theta_3 \times age_2$$

$$\theta_0 + \theta_1 \times income_3 + \theta_3 \times age_3$$

This is called a 4x3 matrix

Representing many inner product at once

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

3-dimensional Row vectors (4 times)

1	$income_0$	age_0
1	$income_1$	age_1
1	$income_2$	age_2
1	$income_3$	age_3

This is called a 4x3 matrix

$\bullet$

3-dimensional Column vector

Θ_0

Θ_1

Θ_2

$=$

4-dimensional Column vector

$$\theta_0 + \theta_1 \times income_0 + \theta_3 \times age_0$$

$$\theta_0 + \theta_1 \times income_1 + \theta_3 \times age_1$$

$$\theta_0 + \theta_1 \times income_2 + \theta_3 \times age_2$$

$$\theta_0 + \theta_1 \times income_3 + \theta_3 \times age_3$$

Representing many inner product at once

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

3-dimensional Row vectors (2 times)

3-dimensional Column vector

2-dimensional Column vector

Batch of 2 inputs

1	10	20
1	30	40

$\bullet$

1
2
-1

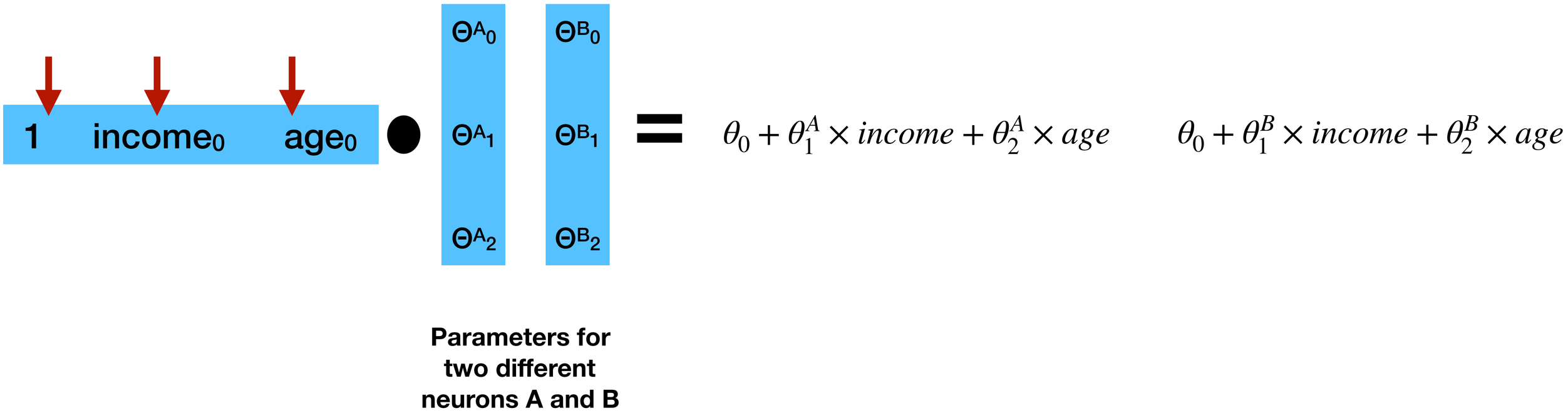
$=$

?

Parameters of one Neuron

Representing many inner product at once

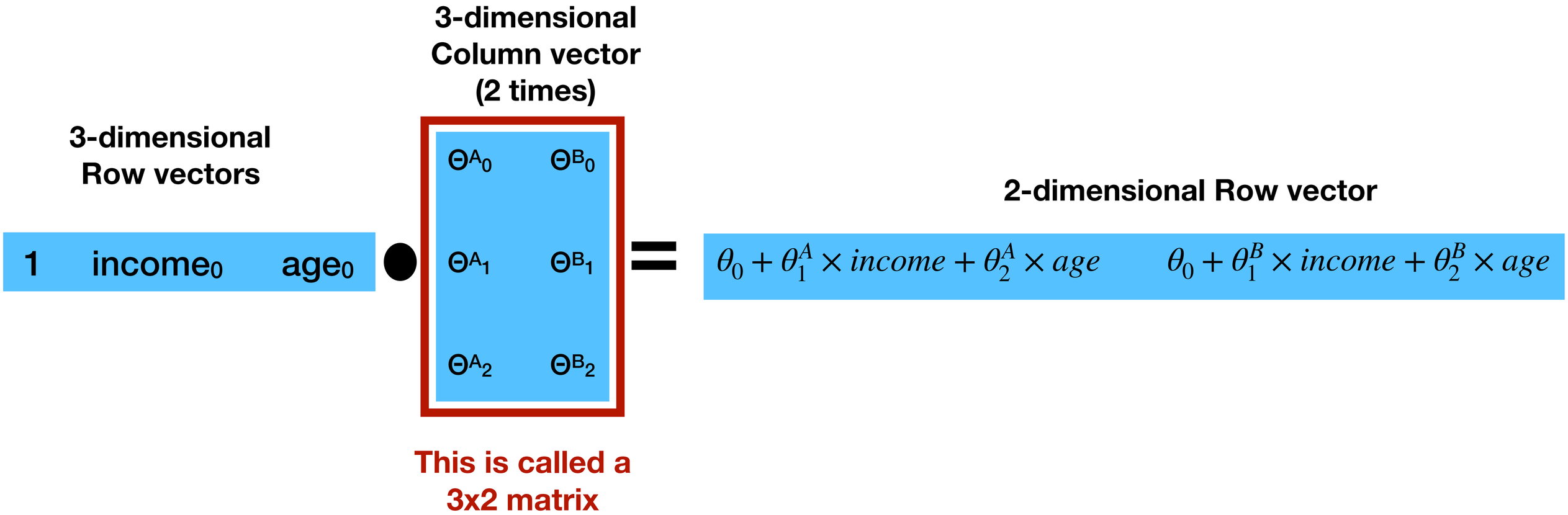
$$vote(age, income) = \sigma(\theta_0 + \theta_1 \times income + \theta_2 \times age)$$



Parameters for two different neurons A and B

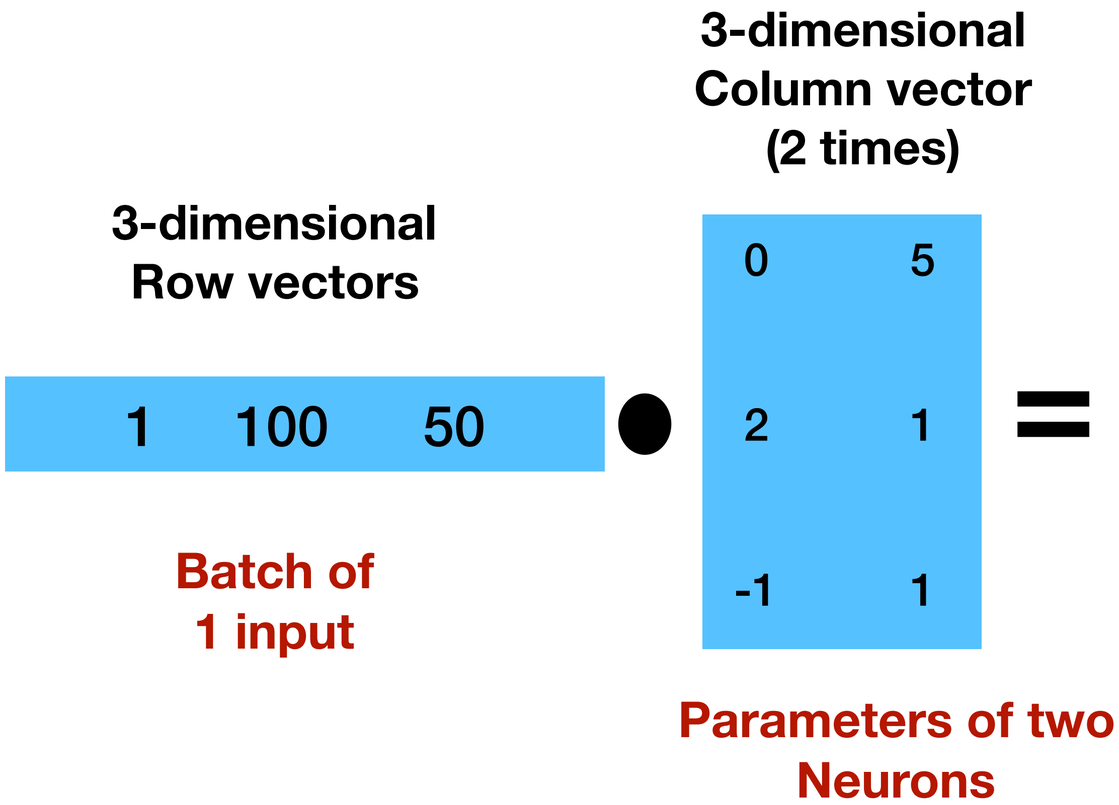
Representing many inner product at once

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



Representing many inner product at once

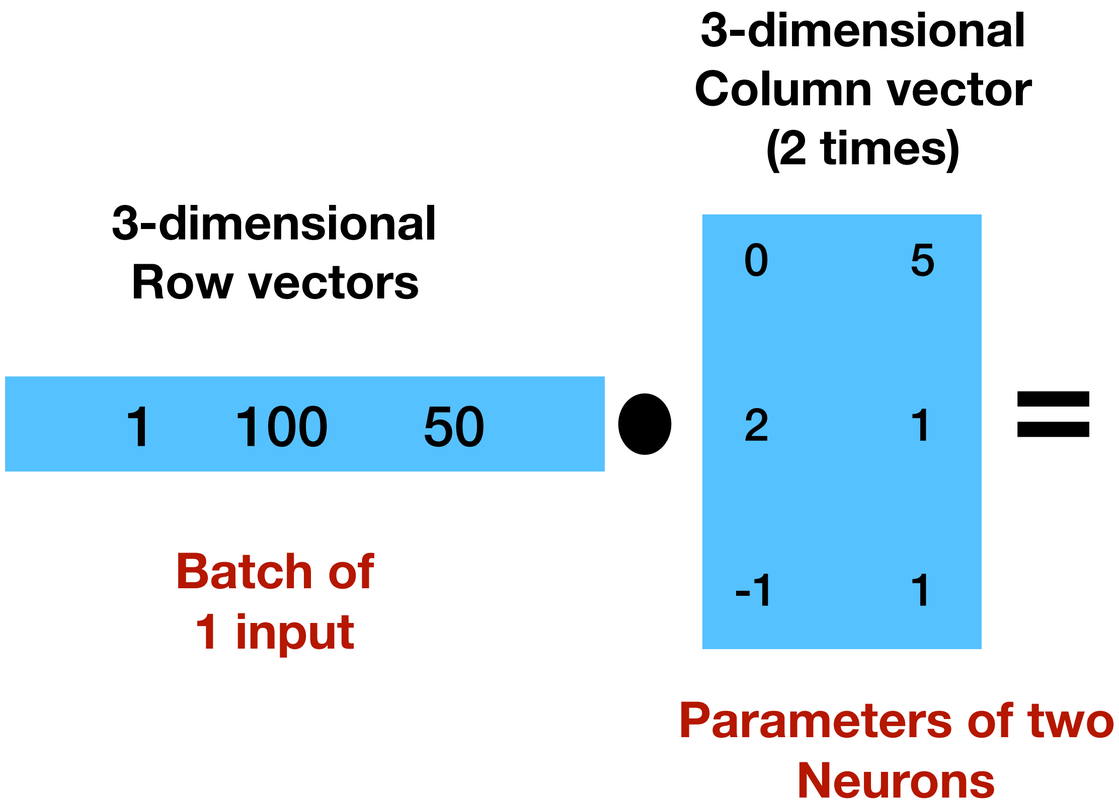
$$vote(age, income) = \sigma(\theta_0 + \theta_1 \times age + \theta_2 \times income)$$



2-dimensional Row vector

Representing many inner product at once

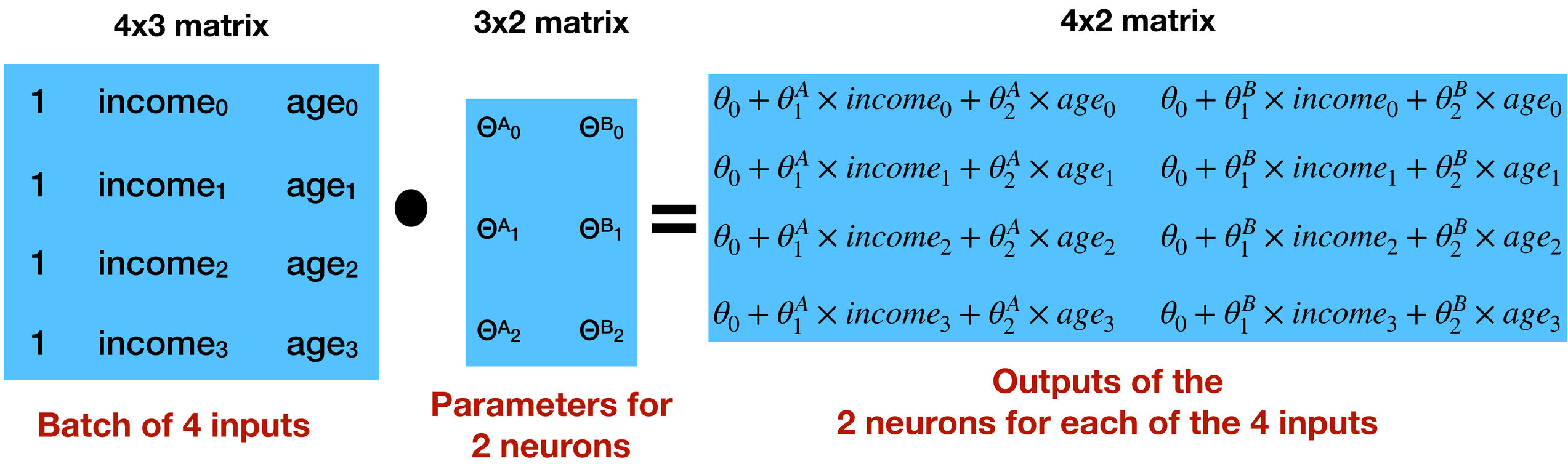
$$vote(age, income) = \sigma(\theta_0 + \theta_1 \times age + \theta_2 \times income)$$



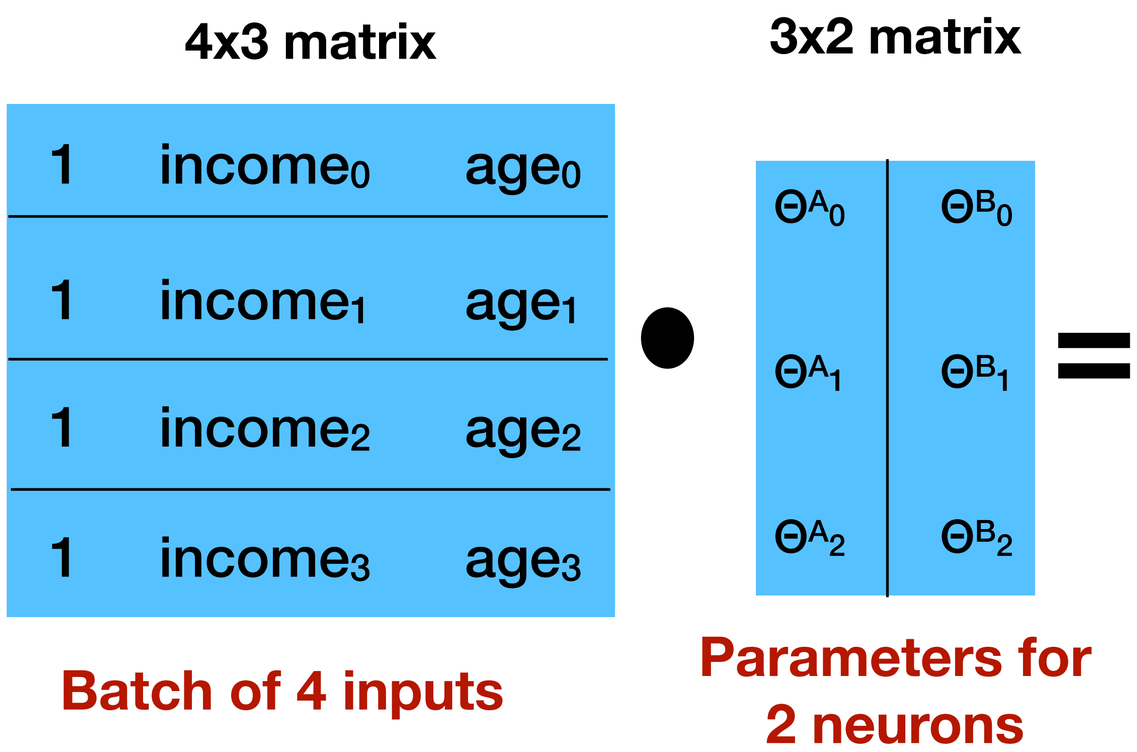
2-dimensional Row vector

Representing many inner product at once

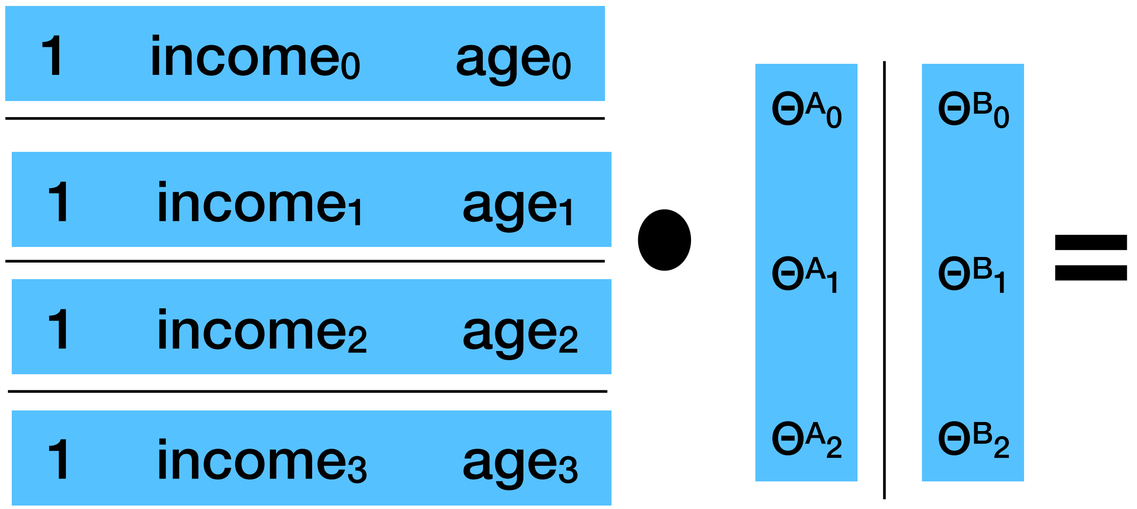
- If we combine the 2 aspects of having several inputs and several neurons, we have what is called a ***matrix multiplication***



Matrix Multiplication



Matrix Multiplication

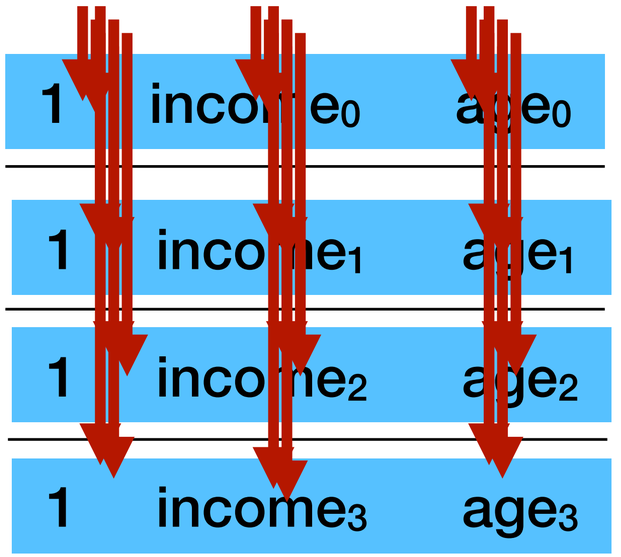


4x2 matrix

Batch of 4 inputs

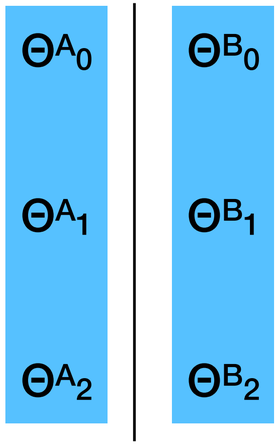
Parameters for 2 neurons

Matrix Multiplication



Batch of 4 inputs

$\bullet$



Parameters for 2 neurons

$=$

4x2 matrix

$$\begin{bmatrix} \theta_0^A + \theta_1^A \times income_0 + \theta_2^A \times age_0 & \theta_0^B + \theta_1^B \times income_0 + \\ \theta_0^A + \theta_1^A \times income_1 + \theta_2^A \times age_1 & \theta_0^B + \theta_1^B \times income_1 + \\ \theta_0^A + \theta_1^A \times income_2 + \theta_2^A \times age_2 & \theta_0^B + \theta_1^B \times income_2 + \\ \theta_0^A + \theta_1^A \times income_3 + \theta_3^A \times age_3 & \theta_0^B + \theta_1^B \times income_3 + \end{bmatrix}$$

Matrix Multiplication

- We represent many inputs as a matrix
- We represent many neurons as a matrix
- Matrix multiplication compute the output score of all the neurons for all the inputs

4x3 matrix

1	$income_0$	age_0
1	$income_1$	age_1
1	$income_2$	age_2
1	$income_3$	age_3

Batch of 4 inputs

$\bullet$

3x2 matrix

Θ^A_0	Θ^B_0
Θ^A_1	Θ^B_1
Θ^A_2	Θ^B_2

Parameters for 2 neurons

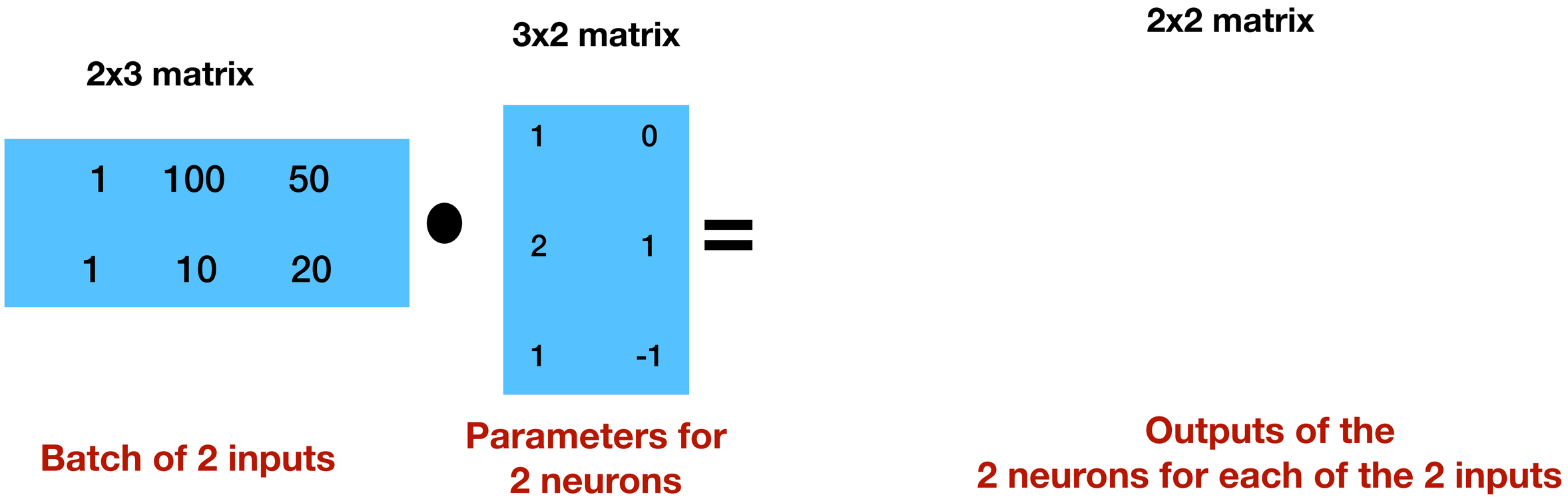
$=$

4x3 matrix

$\theta_0 + \theta_1^A \times income_0 + \theta_2^A \times age_0$	$\theta_0 + \theta_1^B \times income_0 + \theta_2^B \times age_0$
$\theta_0 + \theta_1^A \times income_1 + \theta_2^A \times age_1$	$\theta_0 + \theta_1^B \times income_1 + \theta_2^B \times age_1$
$\theta_0 + \theta_1^A \times income_2 + \theta_2^A \times age_2$	$\theta_0 + \theta_1^B \times income_2 + \theta_2^B \times age_2$
$\theta_0 + \theta_1^A \times income_3 + \theta_2^A \times age_3$	$\theta_0 + \theta_1^B \times income_3 + \theta_2^B \times age_3$

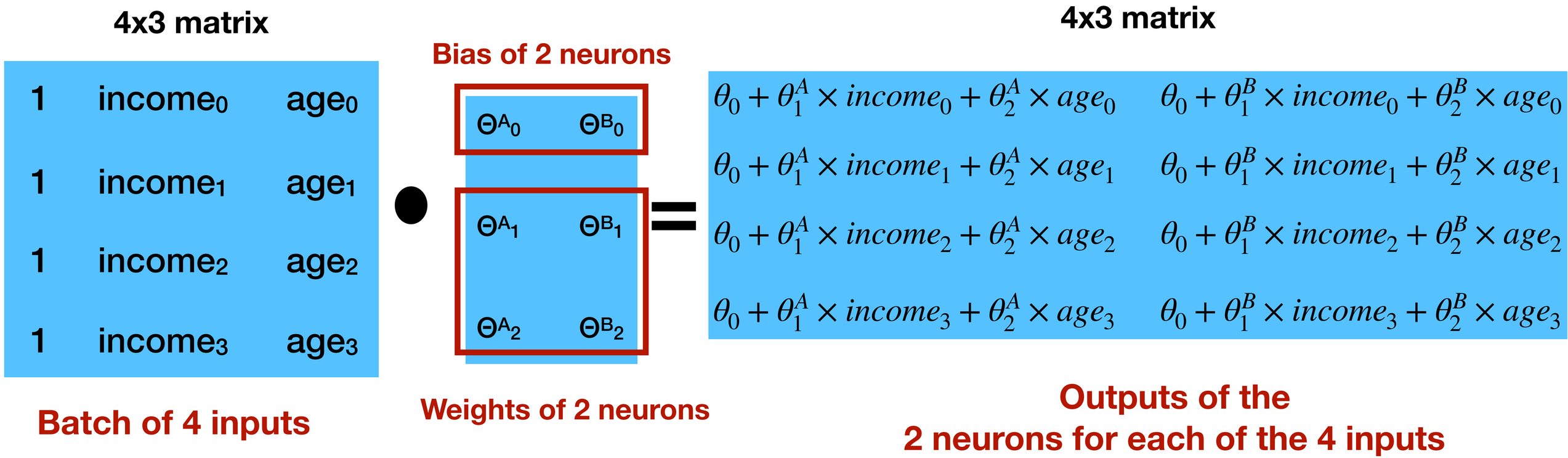
Outputs of the 2 neurons for each of the 4 inputs

Matrix Multiplication

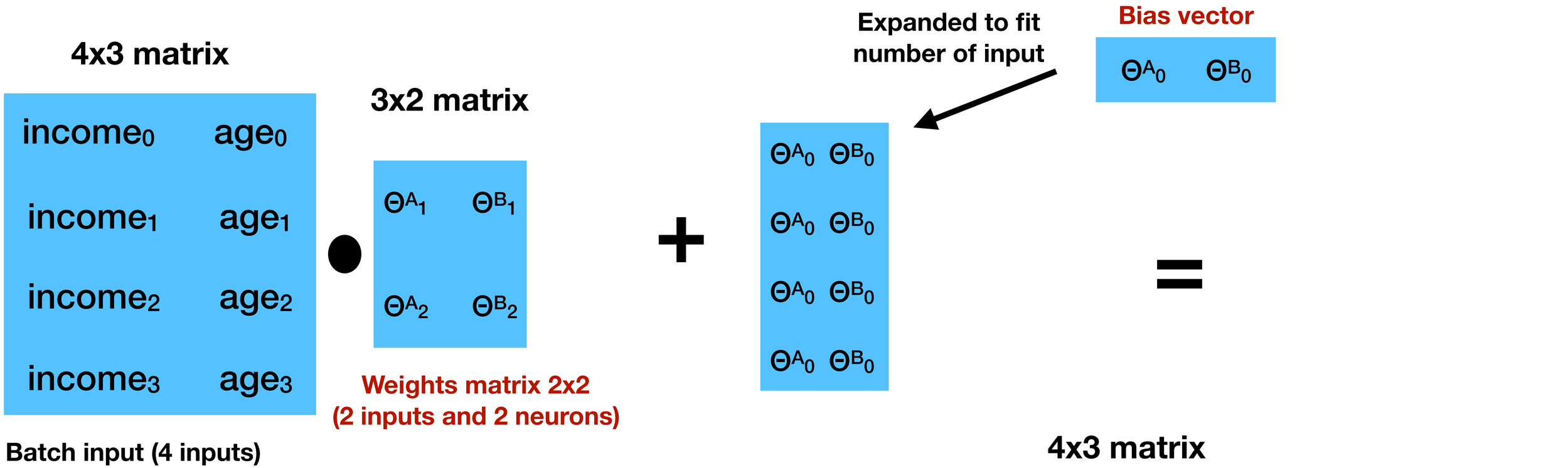


Computing the output of many neurons for many inputs

- In practice, we separate the ***weights*** from the ***bias***



Computing the output of many neurons for many inputs



4x3 matrix

$$\begin{bmatrix} \theta_0 + \theta_1^A \times income_0 + \theta_2^A \times age_0 & \theta_0 + \theta_1^B \times income_0 + \theta_2^B \times age_0 \\ \theta_0 + \theta_1^A \times income_1 + \theta_2^A \times age_1 & \theta_0 + \theta_1^B \times income_1 + \theta_2^B \times age_1 \\ \theta_0 + \theta_1^A \times income_2 + \theta_2^A \times age_2 & \theta_0 + \theta_1^B \times income_2 + \theta_2^B \times age_2 \\ \theta_0 + \theta_1^A \times income_3 + \theta_2^A \times age_3 & \theta_0 + \theta_1^B \times income_3 + \theta_2^B \times age_3 \end{bmatrix}$$

Matrix Multiplication

Bias for 2 neurons

1	0

2x2 matrix

100	50
10	20

Batch of 2 inputs

•

2x2 matrix

2	1
1	-1

Weights for 2 neurons

=

2x2 matrix

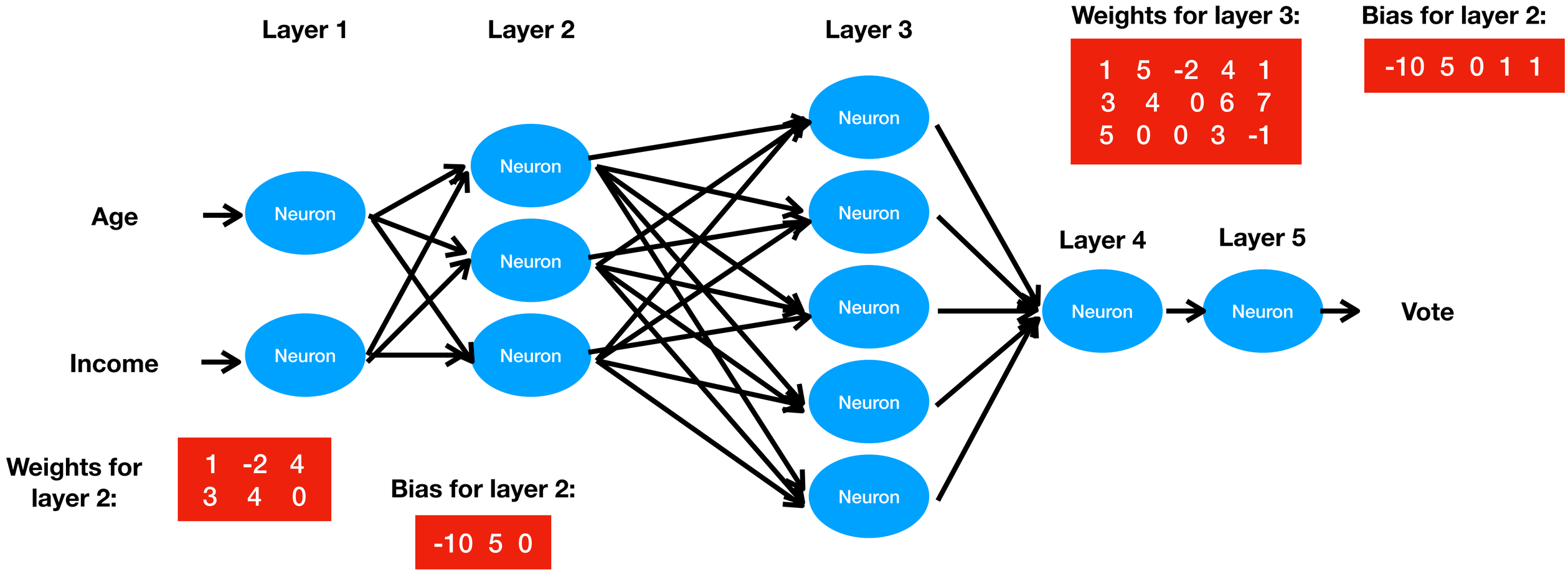
Score outputs of the 2 neurons for each of the 2 inputs

- Important: using these matrix representations only work if all the neurons have the same input and are fully connected

- Then, in practice, a **Fully Connected layer of N neurons with K input** can be represented by a ***matrix of weights W*** of shape KxN, and a ***bias vector b*** of dimension N.

Feed-Forward networks with fully connected layers

- Then, in practice, a **Fully Connected layer of N neurons with K input** can be represented by a ***matrix of weights W*** of shape KxN, and a ***bias vector b*** of dimension N.



Yet another visualization

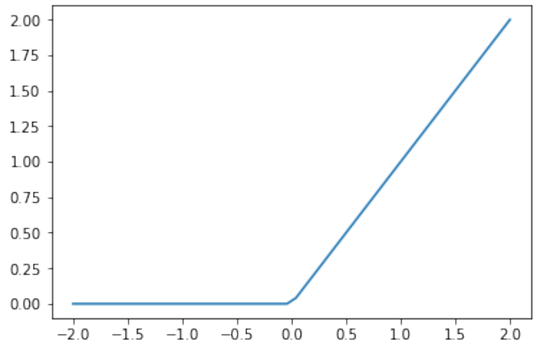
- An interactive Neural Network with Fully Connected Layers
  - From some nice people at Google
  - https://playground.tensorflow.org

Activations

- Some possible activation functions:

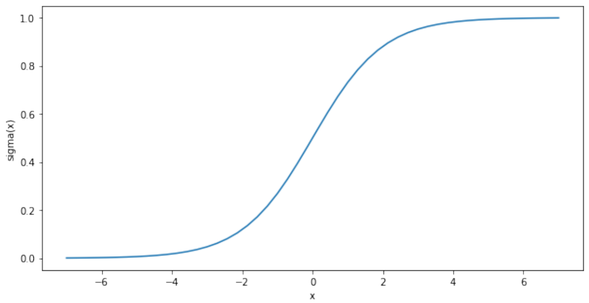
Rectified Linear Unit

$$ReLU(x) = max(x,0)$$



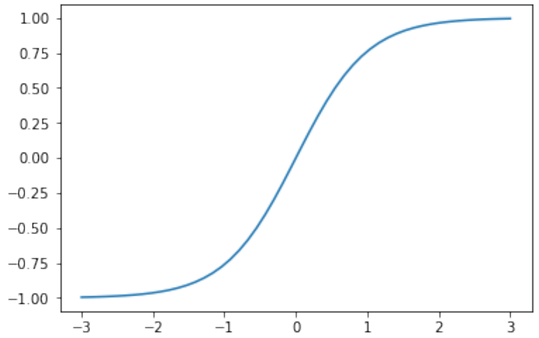
Sigmoid
(a.k.a logistic function)

$$\sigma(x) = \frac{1}{1 + exp(-x)}$$



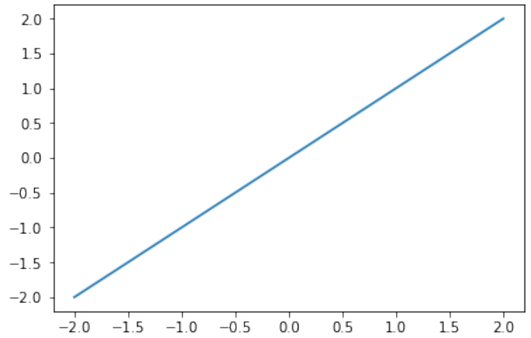
Hyperbolic Tangent
(= a linear transformation of the sigmoid)

$$tanh(x) = 2\sigma(2x) - 1$$



Linear function
(useless as activation function)

$$f(x) = x$$



L1 and L2 Regularization

- We have discusses that L2 Regularization can be used for reducing overfitting
  - -> *Change Regularization and Regularization Rate to observe the effect of Regularization*

L2 Regularization

- We note $|\Theta|^2$ the sum of the square of all parameters Θ_k (this is called the "L2 Norm")

- Then we add this quantity to the loss we want to minimize:

$$Loss = MeanSquaredError = \frac{1}{N} \cdot \sum_i (model(cig_i, bmi_i, ismale_i) - age_i)^2$$

$$Loss = \frac{1}{N} \cdot \sum_i (model(cig_i, bmi_i, ismale_i) - age_i)^2 + \lambda \mid \overrightarrow{\theta} \mid^2$$

- Then we apply Gradient Descent to this new loss

L1 Regularization

- We note $|\Theta|^1$ the sum of the absolute value of all parameters Θ_k (this is called the “L1 Norm”)

- Then we add this quantity to the loss we want to minimize:

$$Loss = MeanSquaredError = \frac{1}{N} \cdot \sum_i (model(cig_i, bmi_i, ismale_i) - age_i)^2$$

$$Loss = \frac{1}{N} \cdot \sum_i (model(cig_i, bmi_i, ismale_i) - age_i)^2 + \lambda \mid \overrightarrow{\theta} \mid$$

- Then we apply Gradient Descent to this new loss

Random restarts

- Because the weights of a neural network are initialized randomly, the result of a training will change every time we reinitialize the network
- Therefore, to obtain best performances, it is common to train a networks several times with different random initializations, and keep the best result
  - *-> Press the reload button to restart a training with new random weight initialization*

Regularization methods for Neural Network

- When we train a network with many neurons, the danger of overfitting is large
- There are a few technics that are very efficient at preventing this:
  - Early Stopping
  - Dropout
  - Weight Decay
  - Stochastic Gradient Descent

Early stopping

- We keep a validation set separate from the training data
- We fix a ***patience*** number (typically patience = 10 or 20)
- During training, if we see no improvement on validation set after ***patience*** measures, we stop the training
  - *Check the evolution of test loss to detect when training should be stopped*

Dropout

- During training, we add random noise to disturb the network
- In practice, we randomly “cut” a certain proportion of connections (typically 10% to 50%)
    - *Check the effect of adding noise*

Stochastic Gradient Descent

- Instead of computing the gradient of the loss for all the training data, we compute it for a subsampled part of the training data
- This is actually done anyway to get faster training
- But it is also beneficial to prevent overfitting even if you could afford to compute the gradient for all the examples at once.
    - *-> change batch size to observe what happen when we compute gradient on more or less examples*

Input features

- We sae that even for a simple model (like linear regression), adding new functions of the input can increase the capacity of the model
  - -> *Add input features and see the effect when there are few neurons*