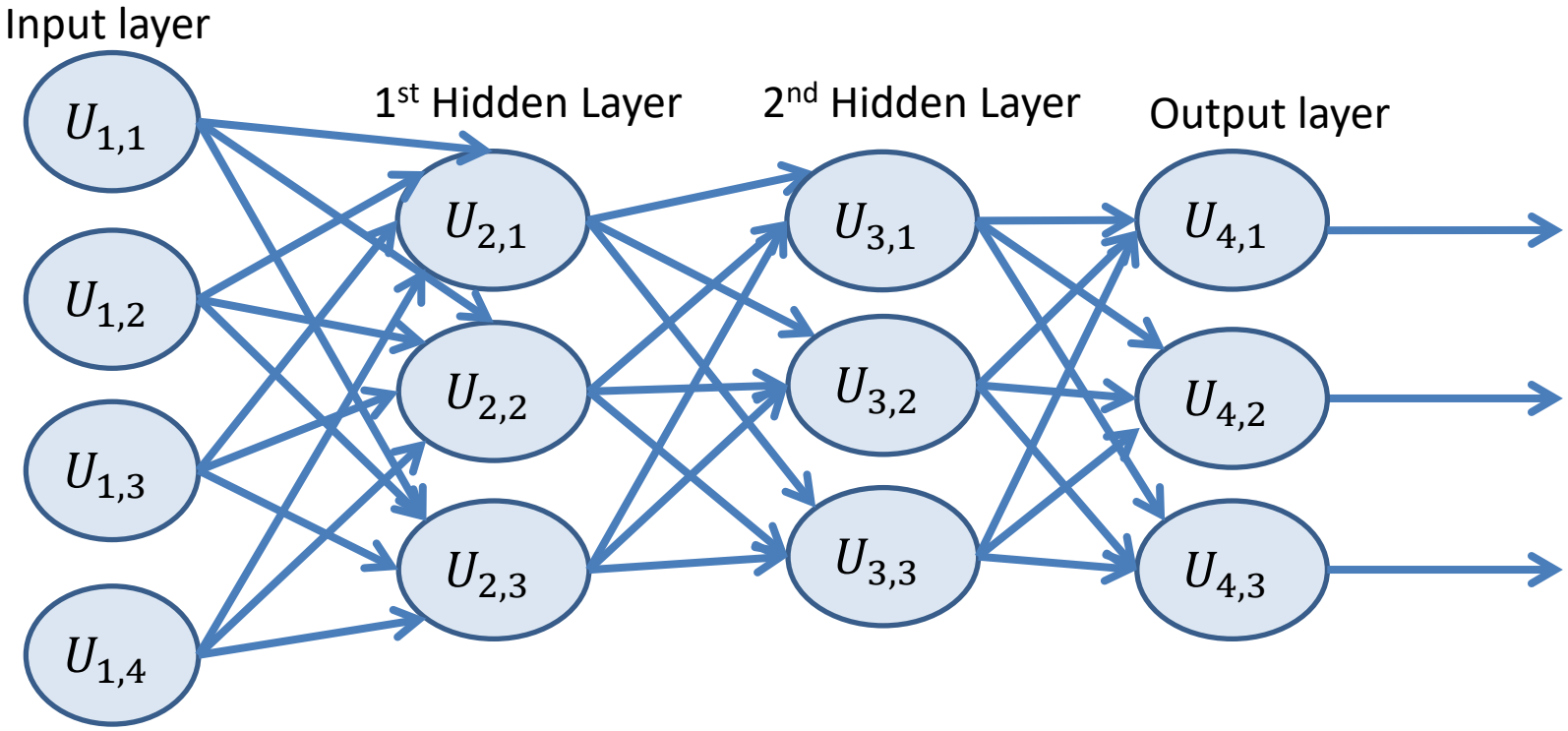
Neural Networks
Part 1 – Introduction, the Inference Module

CSE 4311 – Neural Networks and Deep Learning

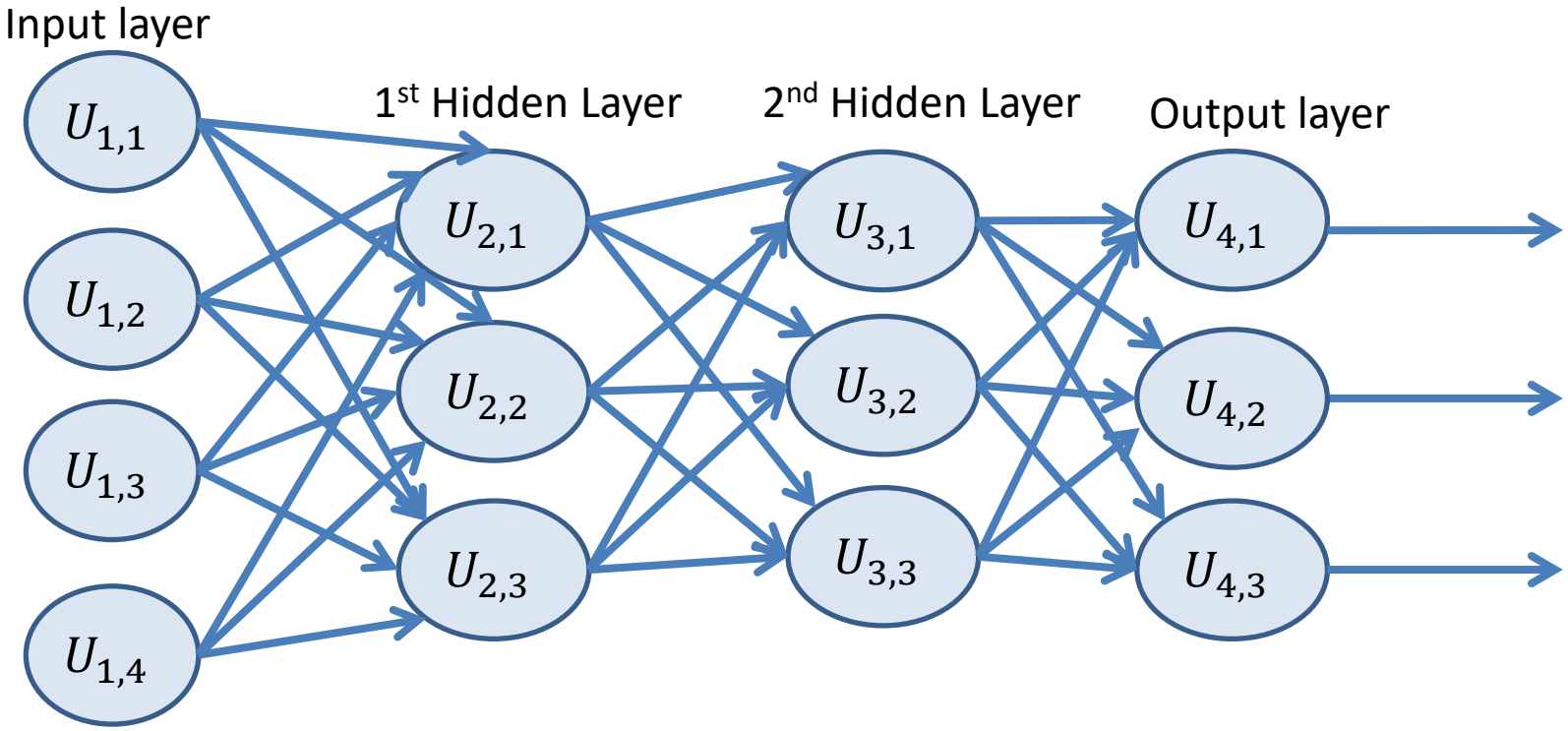
Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

Neural Networks are Graphs



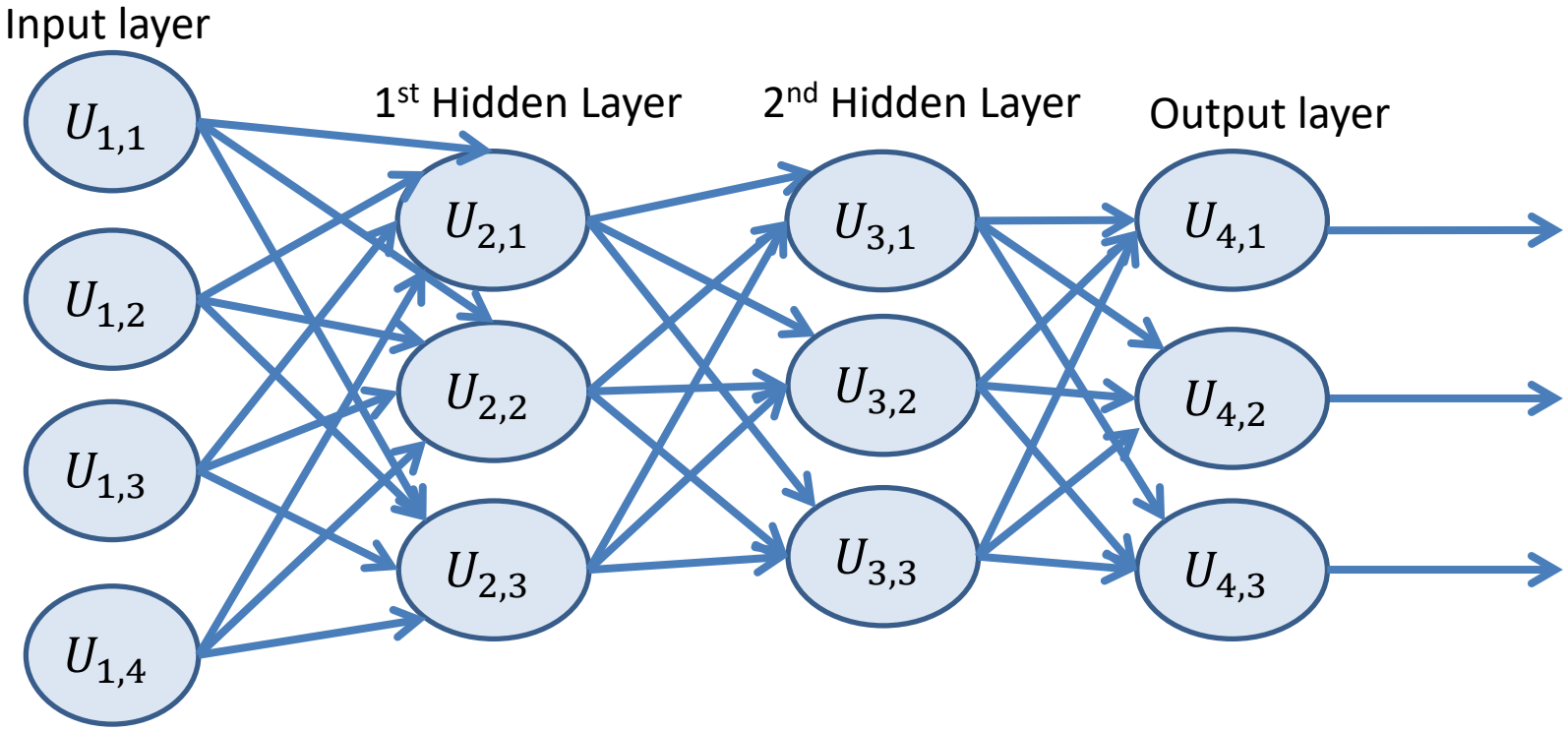
- This is an example of a neural network.
- The drawing will make more sense later.
- What is important at this point is that a neural network is a weighted directed graph.

Neural Networks are Graphs



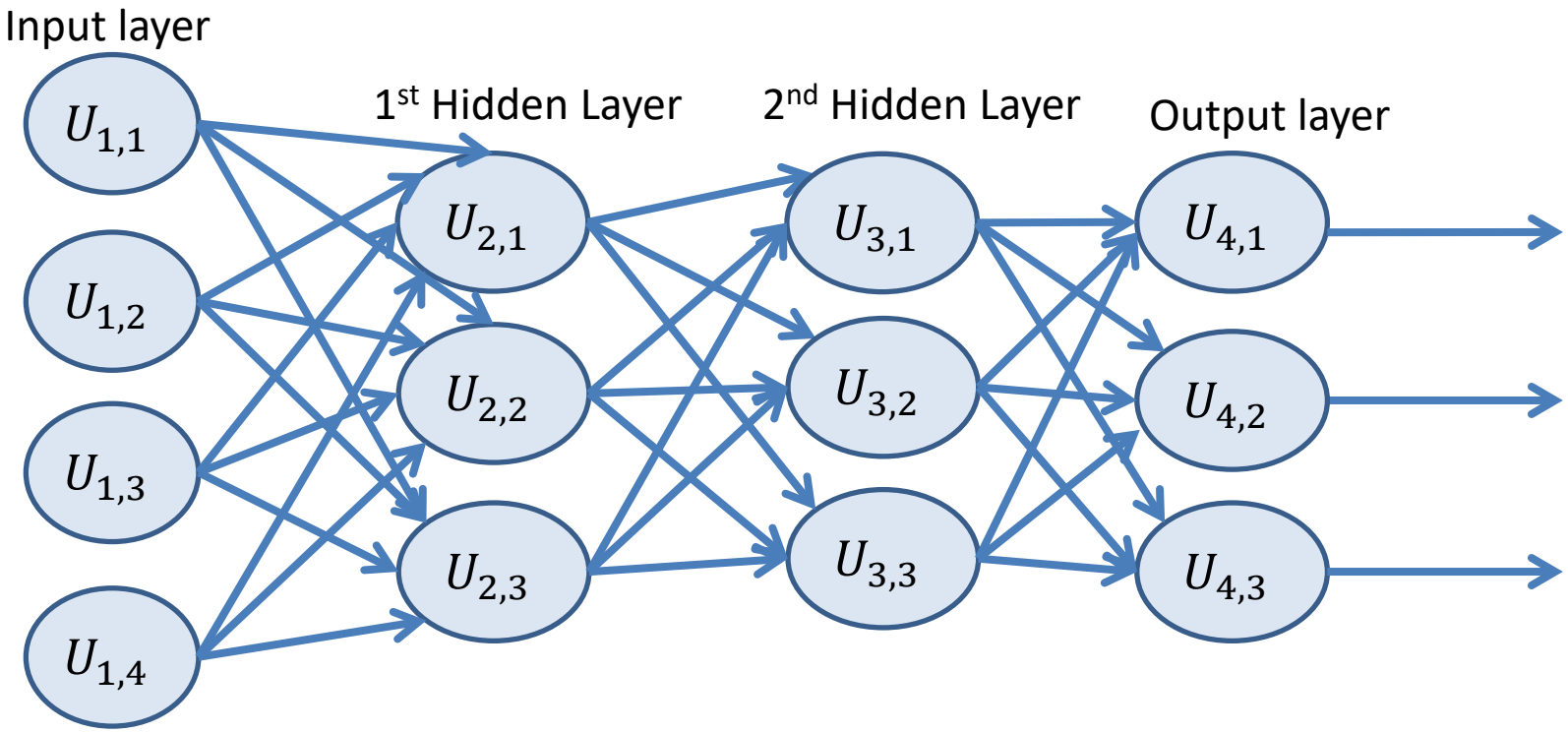
- A directed graph consists of nodes and directed edges.
- Each node (also called **unit**) produces some output.
- Edges go from the output of a unit to the input of another unit.

Input Units and Perceptrons



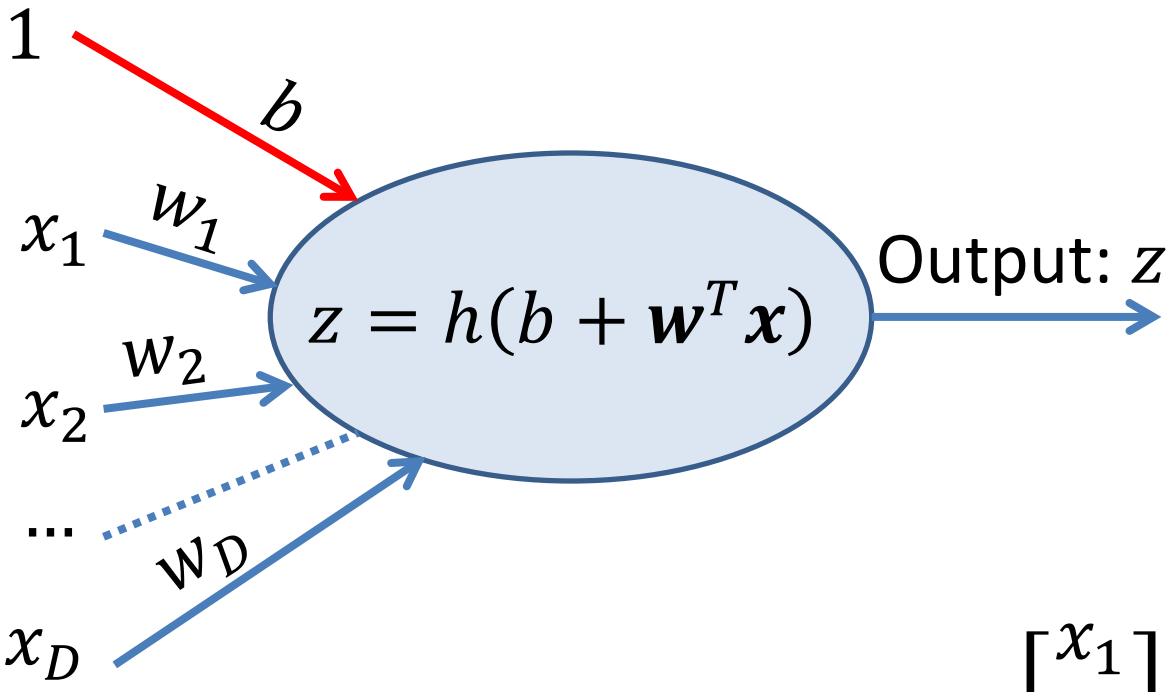
- There are various types of units. The two most simple are:
  - **Input units**. They make up the input layer.
  - **Perceptrons**. They make up the rest of the layers. They are the computational units of the neural network.

Input Units



- Input units are very simple.
  - They represent the input to the neural network.
  - We will define them more formally a bit later, but they are basically a notational convenience.

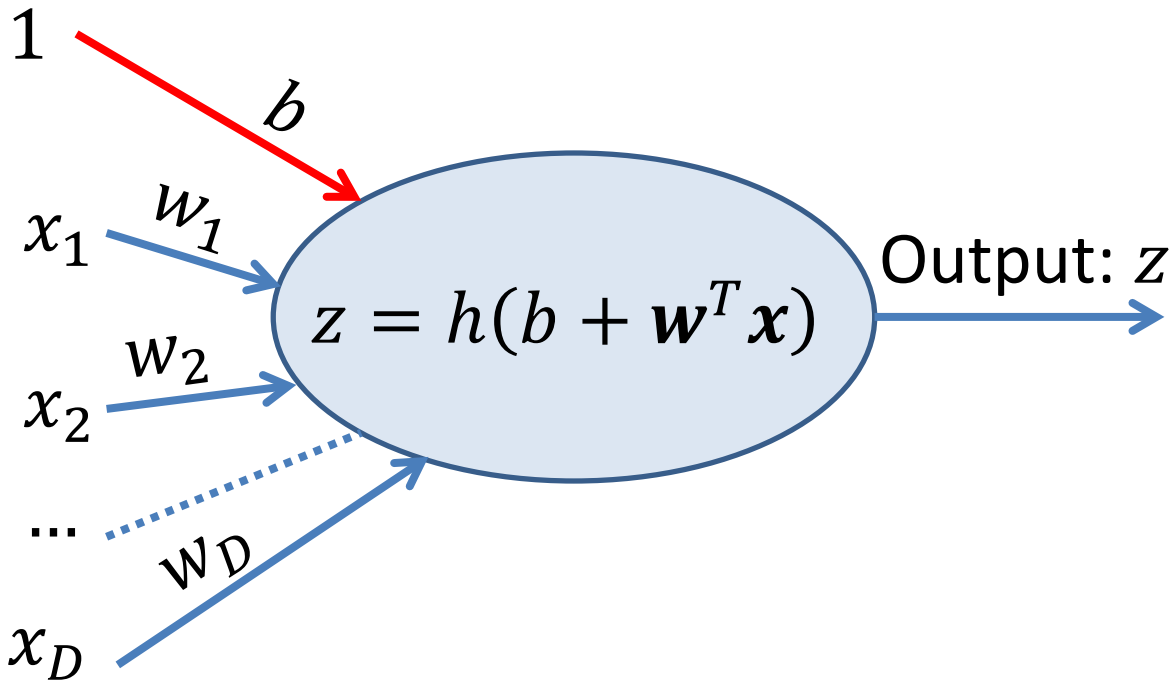
Perceptrons



- A perceptron is a function that maps D-dimensional vectors to real numbers.
- For notational convenience, we add an extra input, called the **bias input**. The bias input is **always equal to 1.**
- b is called the **bias weight**. It is optimized during training.
- $w_1, \ldots, w_D$ are also weights that are optimized during training.

$$\text{input vector } \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \ldots \\ x_D \end{bmatrix}$$

Perceptrons



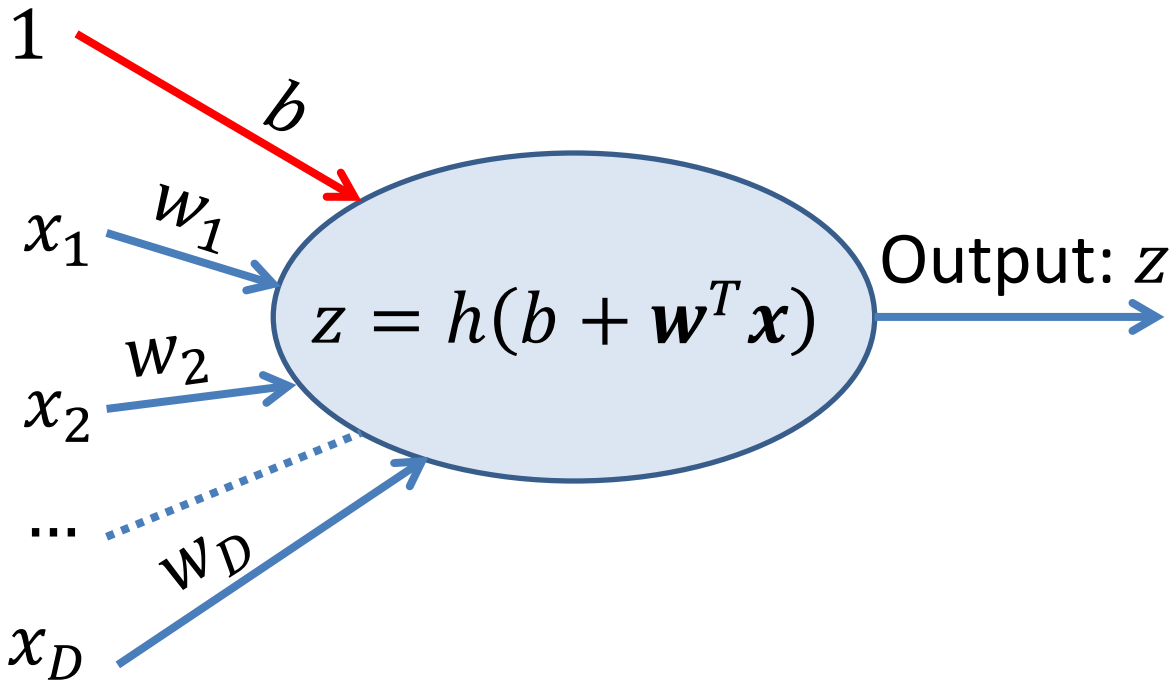
- A perceptron computes its output z in two steps:

Step 1: $a = b + \boldsymbol{w}^T \boldsymbol{x} = b + \sum_{i=1}^{D} (w_i x_i)$

Step 2: $z = h(a)$

- h is called an **activation function**.
- For example, h could be the sigmoid function $\sigma(a) = \frac{1}{1+e^{-a}}$

Perceptrons



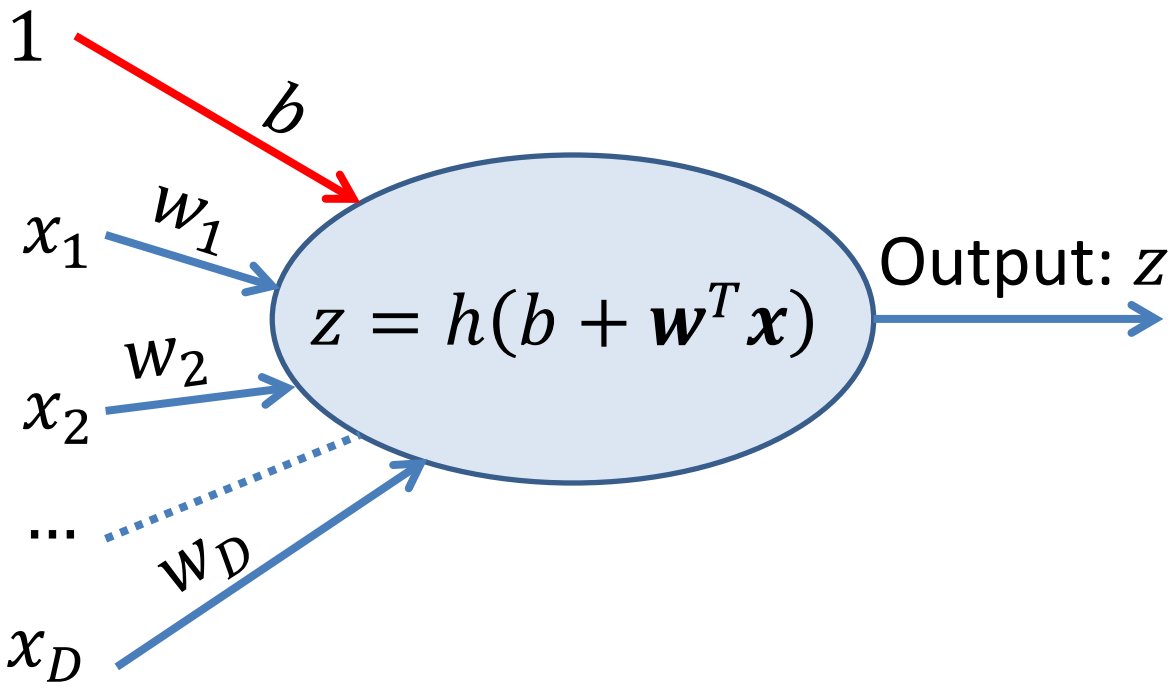
- A perceptron computes its output z in two steps:

Step 1: $a = b + \boldsymbol{w}^T \boldsymbol{x} = b + \sum_{i=1}^{D} (w_i x_i)$

Step 2: $z = h(a)$

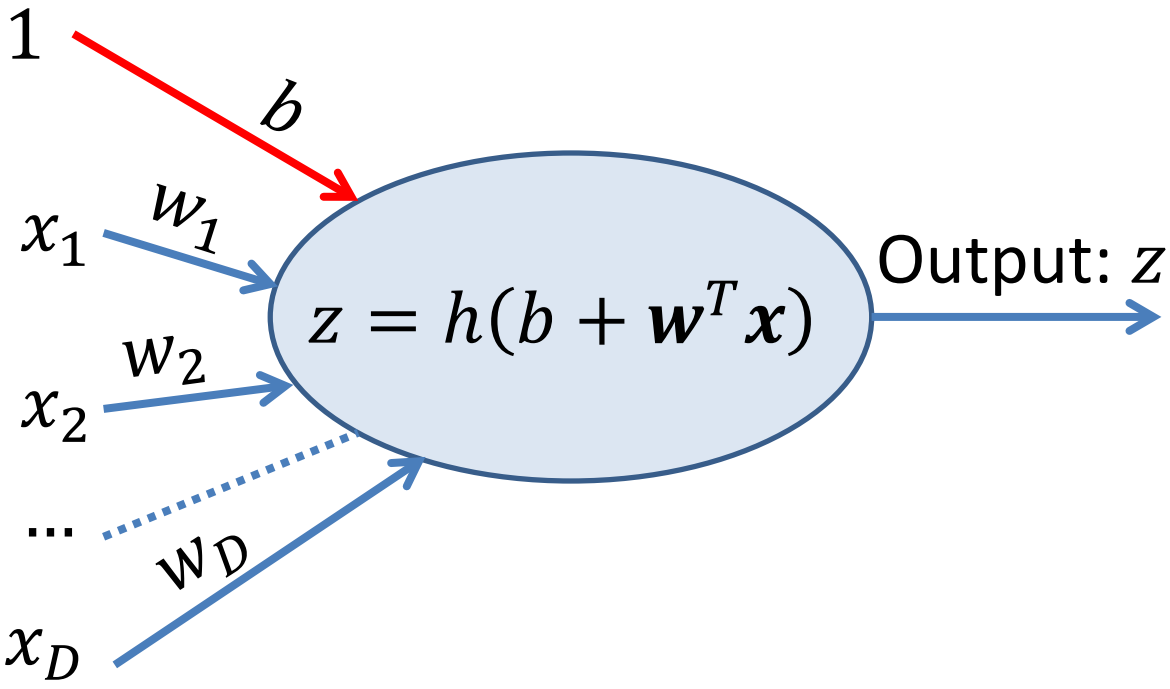
- In a single formula: $z = h\left(b + \sum_{i=1}^{D} (w_i x_i)\right)$

Notation



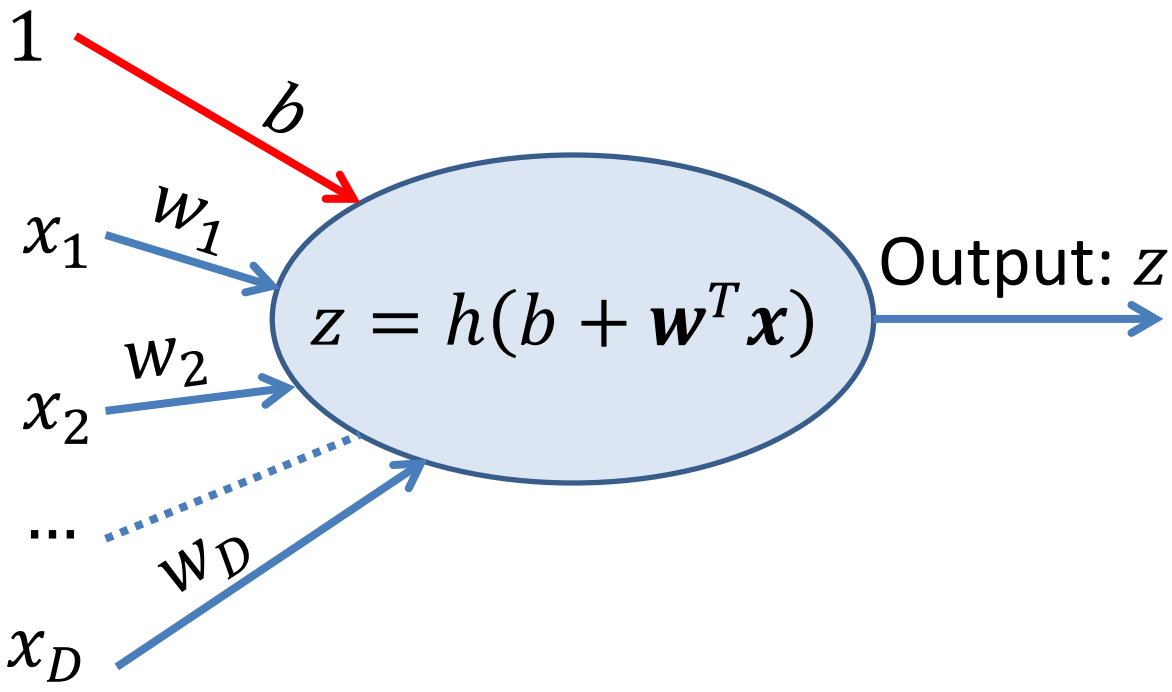
- $\boldsymbol{w}^T \boldsymbol{x}$ is notation that we will be using **a lot** this semester.
  - What does it mean? What is $\boldsymbol{w}$, what is $\boldsymbol{w}^T$, what is $\boldsymbol{x}$?

Notation



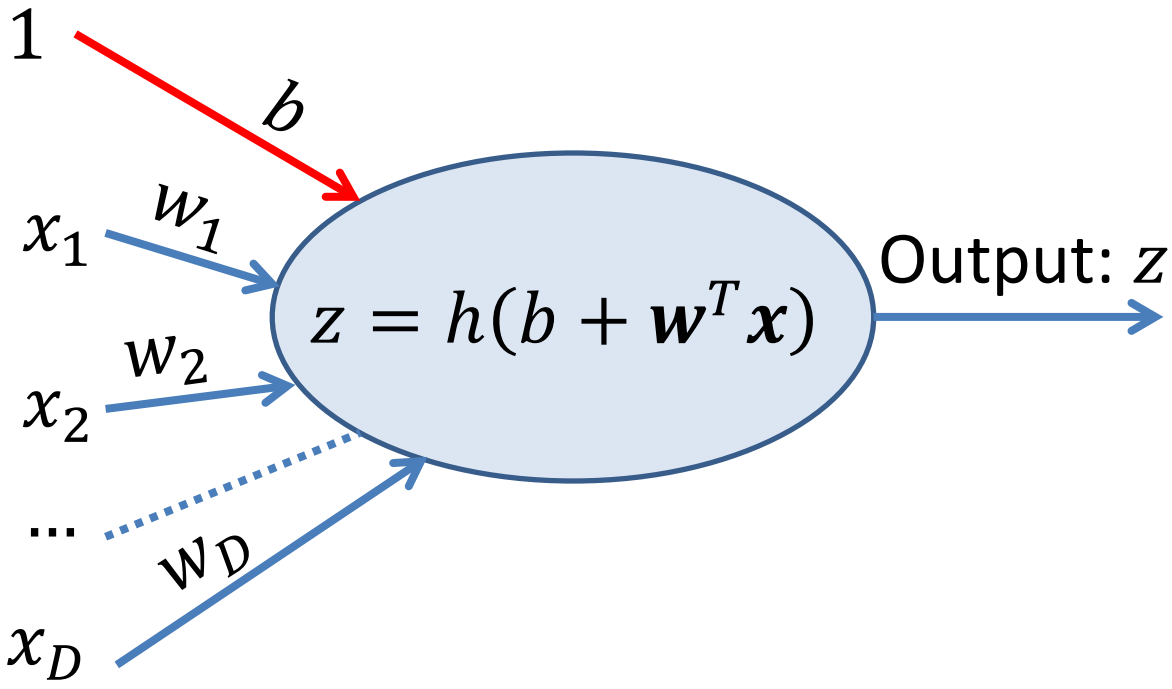
- $\boldsymbol{w}^T \boldsymbol{x}$ is notation that we will be using **a lot** this semester.
  - What does it mean? What is $\boldsymbol{w}$, what is $\boldsymbol{w}^T$, what is $\boldsymbol{x}$?
- $\boldsymbol{w}$ is the vector of weights w_1, …, w_D. It is a column vector.
  - By default, a vector is a column vector, unless specified otherwise.

Notation



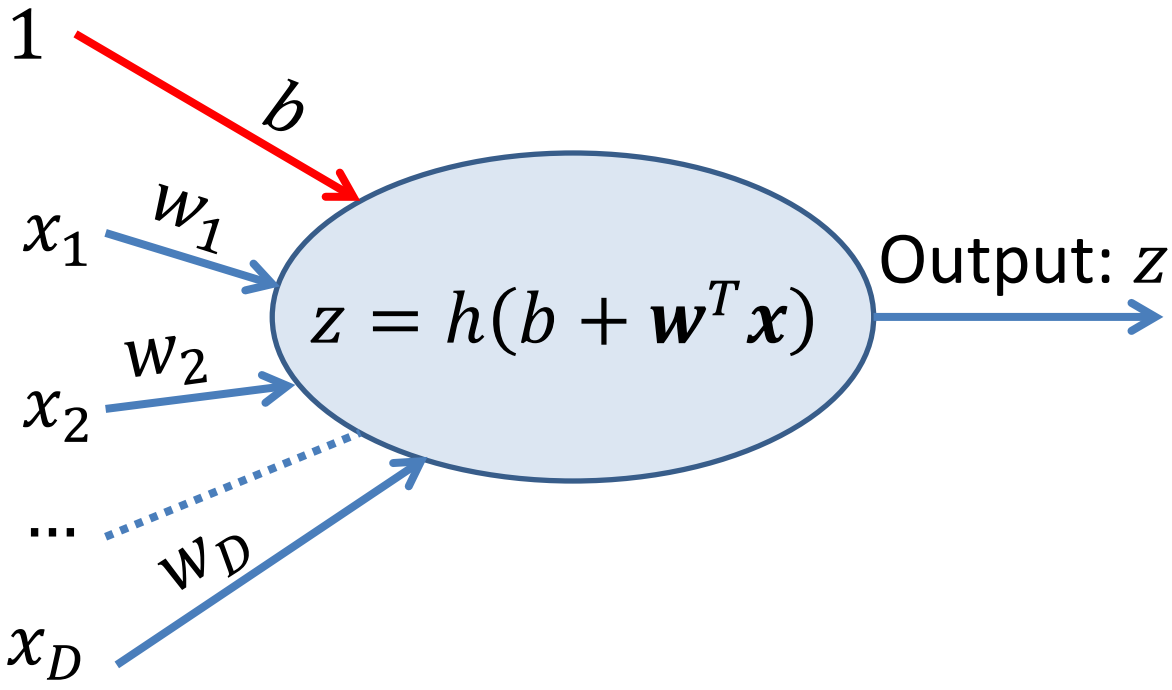
- $\boldsymbol{w}^T \boldsymbol{x}$ is notation that we will be using **a lot** this semester.
  - What does it mean? What is $\boldsymbol{w}$, what is $\boldsymbol{w}^T$, what is $\boldsymbol{x}$?
- $\boldsymbol{w}^T$ is the **transpose** of vector $\boldsymbol{w}$. It is a **row** vector.
- $\boldsymbol{x}$ is the vector of inputs x_1, …, x_D. It is a column vector.
- $\boldsymbol{w}^T \boldsymbol{x}$ is ???

Notation



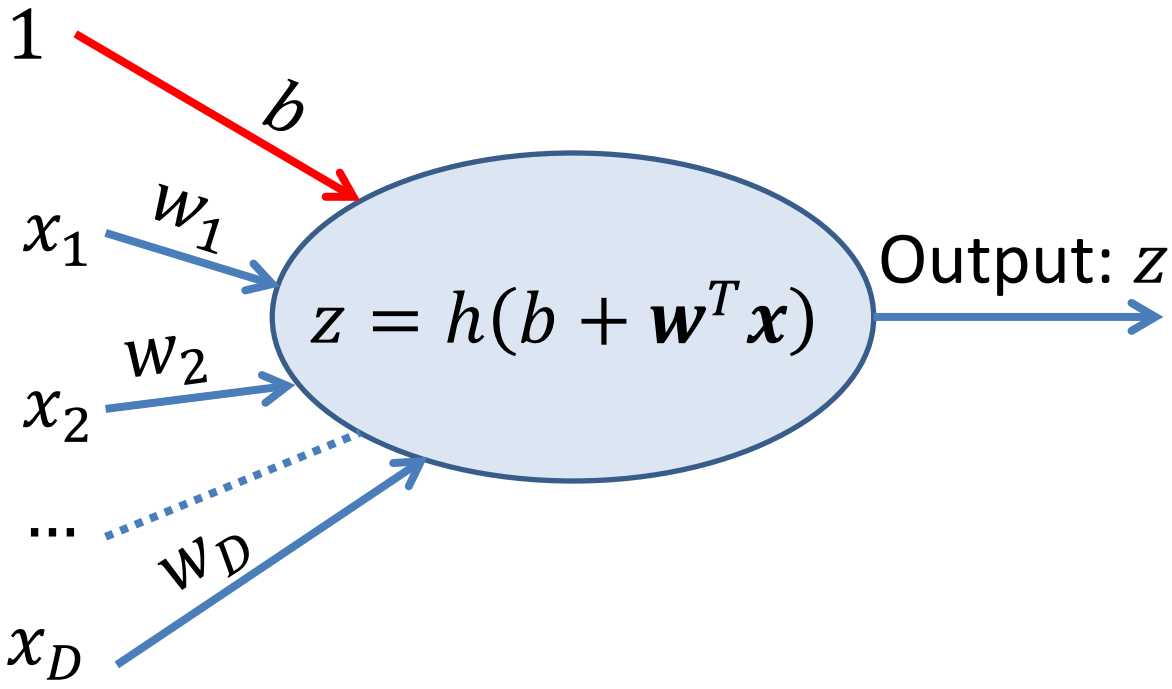
- $\boldsymbol{w}^T$ is the **transpose** of vector $\boldsymbol{w}$. It is a **row** vector.
- $\boldsymbol{x}$ is the vector of inputs x_1, …, x_D. It is a column vector.
- $\boldsymbol{w}^T \boldsymbol{x}$ is matrix multiplication. We multiply $\boldsymbol{w}^T$ by $\boldsymbol{x}$.
  - The result of multiplying a row vector by a column vector is ???

Notation



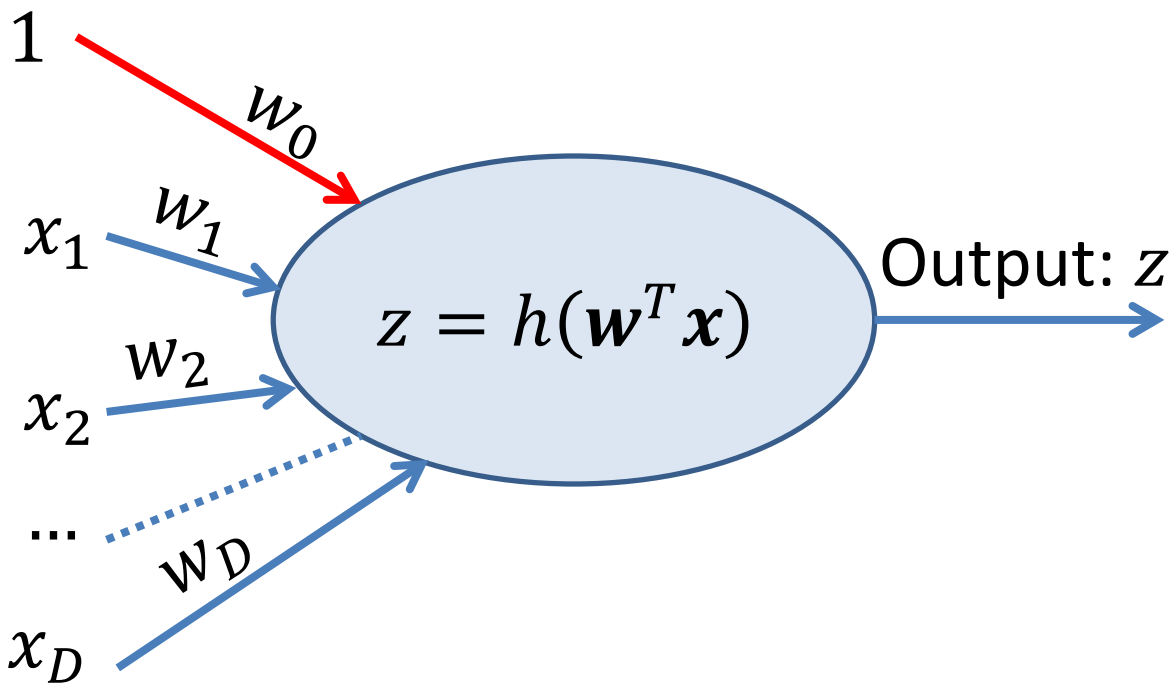
- $\boldsymbol{w}^T$ is the **transpose** of vector $\boldsymbol{w}$. It is a **row** vector.
- $\boldsymbol{x}$ is the vector of inputs x_1, …, x_D. It is a column vector.
- $\boldsymbol{w}^T \boldsymbol{x}$ is matrix multiplication. We multiply $\boldsymbol{w}^T$ by $\boldsymbol{x}$.
  - The result of multiplying a row vector by a column vector is a single number (also called a scalar, also called a 1D vector, also called a 1x1 matrix).

Notation



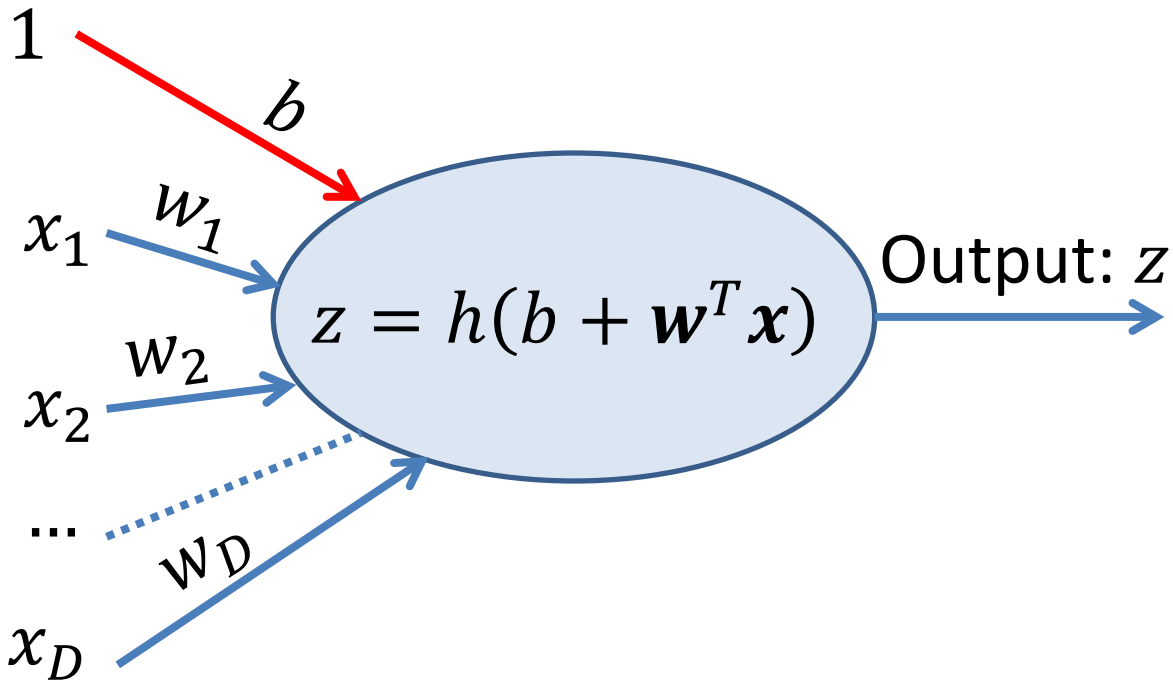
- $\boldsymbol{w}^T$ is the **transpose** of vector $\boldsymbol{w}$. It is a **row** vector.
- $\boldsymbol{x}$ is the vector of inputs x_1, …, x_D. It is a column vector.
- $\boldsymbol{w}^T \boldsymbol{x}$ is matrix multiplication. We multiply $\boldsymbol{w}^T$ by $\boldsymbol{x}$.
  - The result is: $\boldsymbol{w}^T \boldsymbol{x} = \sum_{i=1}^{D} (w_i x_i)$.
- We also call $\boldsymbol{w}^T \boldsymbol{x}$ the **dot product** of $\boldsymbol{w}$ and $\boldsymbol{x}$.
- This is all fundamental linear algebra (course prereq).

Notation for Bias Weight



- There is an alternative representation that we will not use, where b is denoted as w_0, and weight vector $\boldsymbol{w} = (w_0, w_1, w_2, \ldots, w_D)$.
- Then, instead of writing $z = h(b + \boldsymbol{w}^T \boldsymbol{x})$ we can simply write
$$z = h(\boldsymbol{w}^T \boldsymbol{x}).$$
- In our slides, we will denote the bias weight as b and treat it separately from the other weights. That will make life easier later.

Perceptrons



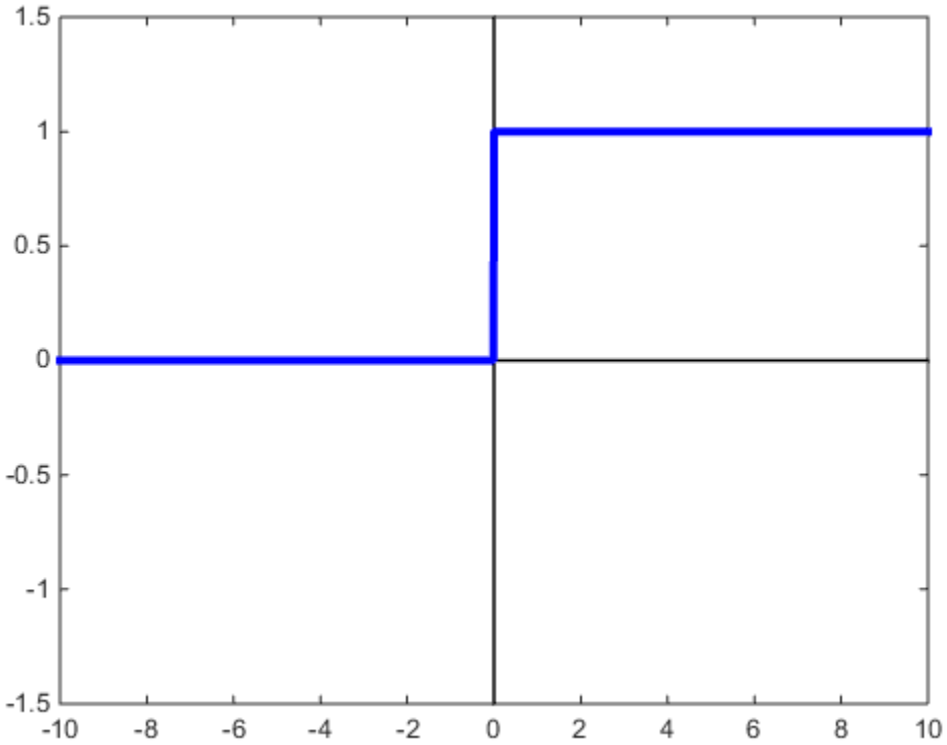
- A perceptron computes its output z in two steps:

Step 1: $a = b + \boldsymbol{w}^T \boldsymbol{x}$

Step 2: $z = h(a)$

- h is called an **activation function**.
- We will use several different activation functions this semester.

Activation Functions

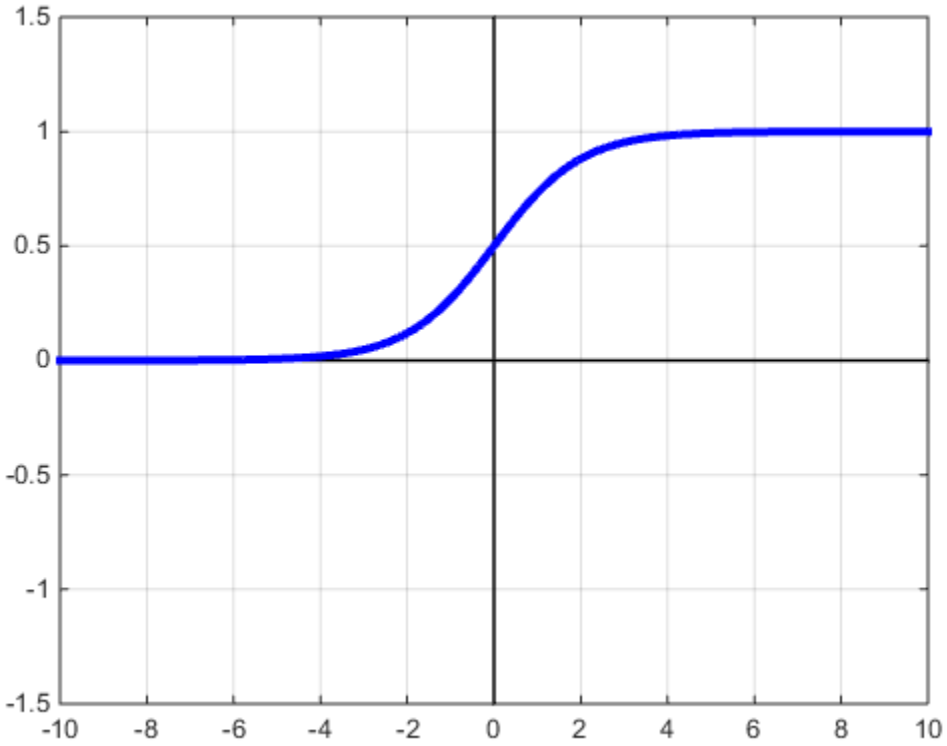


- A perceptron produces output $z = h(b + \boldsymbol{w}^T \boldsymbol{x})$.
- A simple choice for the activation function h: the **step function**.

$$h(a) = \begin{cases} 0, \text{if } a < 0 \\ 1, \text{if } a \geq 0 \end{cases}$$

- The step function is useful for providing some intuitive examples.
- It is **not useful** for actual real-world systems.
  - Reason (will be explained later in detail): it is not differentiable, it does not allow optimization via gradient descent.

Activation Functions

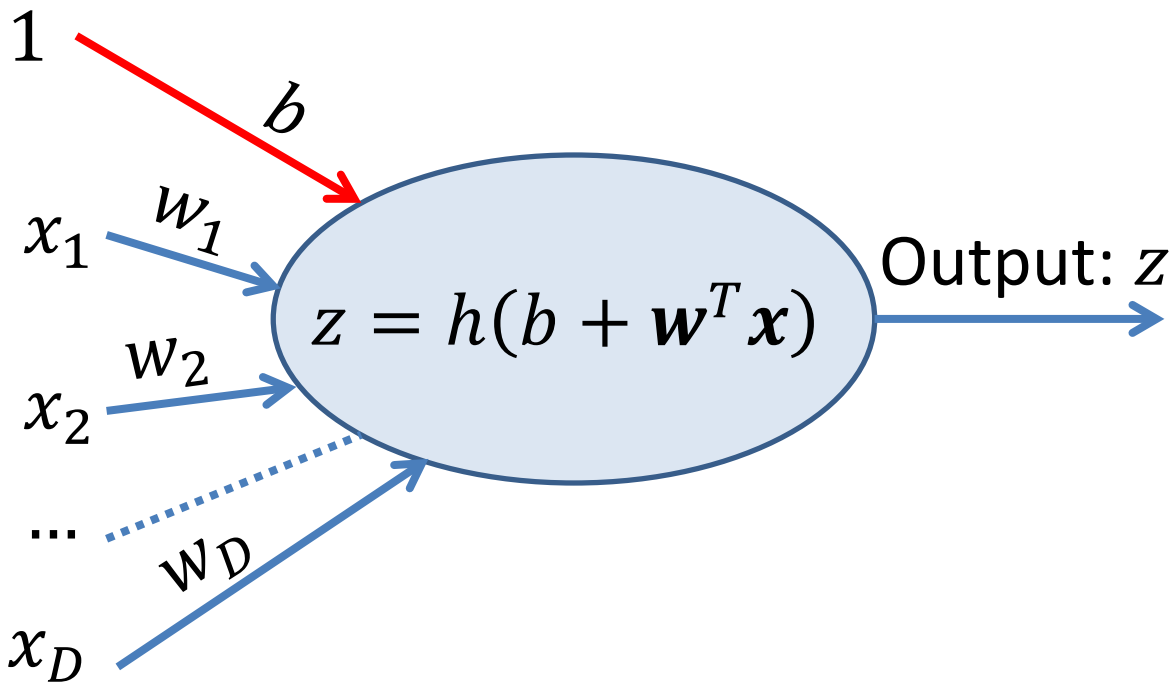


- A perceptron produces output $z = h(b + \boldsymbol{w}^T \boldsymbol{x})$.
- Another choice for the activation function $h(a)$: the **sigmoid function**.

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

- The sigmoid is often used in real-world systems.
- It is a differentiable function, it allows use of gradient descent (will be explained later).

Perceptrons and Neurons



- Perceptrons are inspired by neurons.
  - Neurons are the cells forming the nervous system, and the brain.
  - Neurons somehow sum up their inputs, and if the sum exceeds a threshold, they "fire".
- Since brains are "intelligent", computer scientists have been hoping that perceptron-based systems can be used to model intelligence.

Separate Modules: Training, Inference

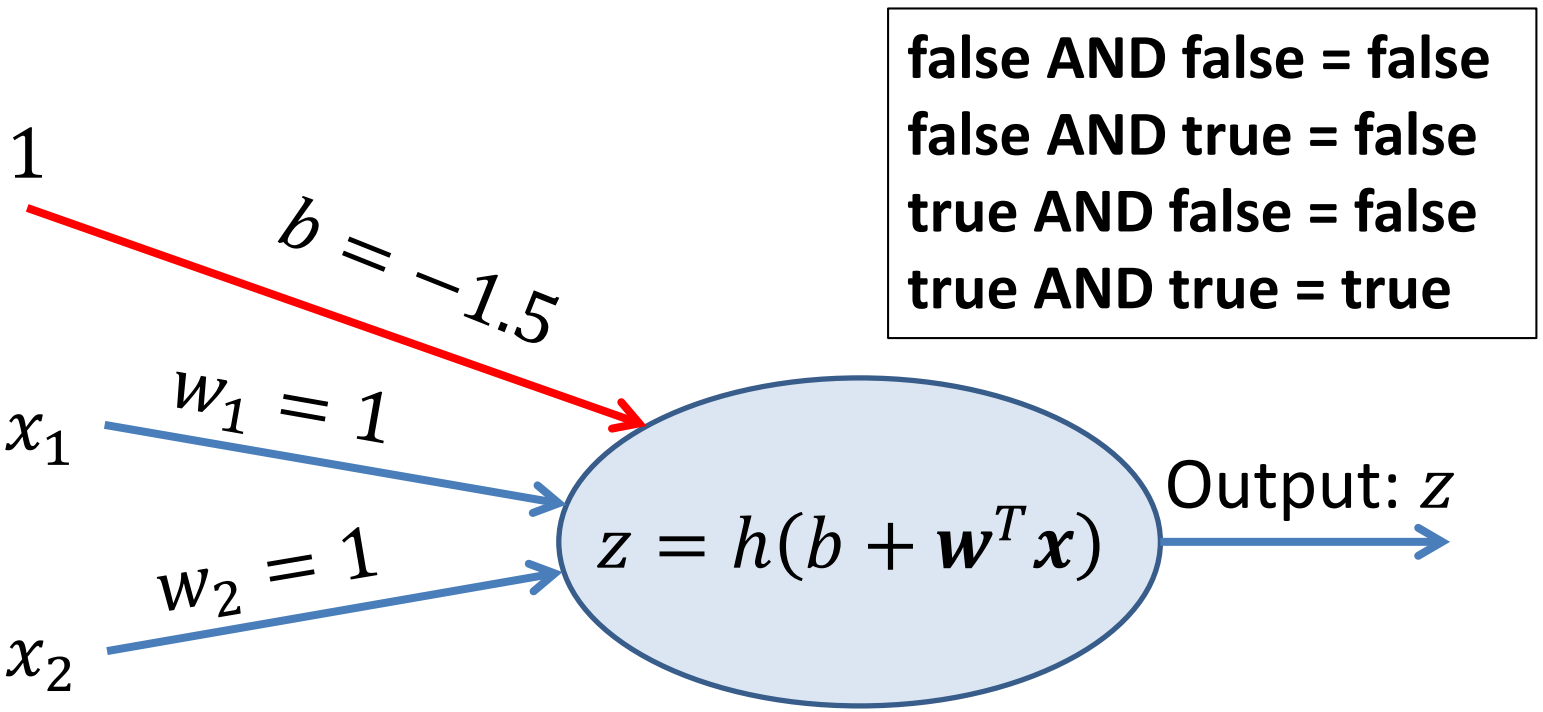
- Neural networks are supervised learning models.
- Typically, implementing a supervised learning model requires implementing two separate modules:
  - A training module, that uses training data to construct the model.
  - An inference module, that applies the model to new data, to recognize the class that the new data belong to.

Separate Modules: Training, Inference

- In the real world, training is done before inference.
  - We cannot use a model before we construct that model.
- For teaching purposes, we start with the inference module. It is more simple to understand and implement.
  - The training module includes the inference module as a subcomponent.
- We will start by seeing some simple examples of perceptrons and neural networks.
  - We just want to see how to apply them to test data.
  - We will NOT worry (for now) about training them.
- Still, just as a preview: the job of training is to assign values to the weights.
  - At inference time, the weights are fixed.

Example: The AND Perceptron

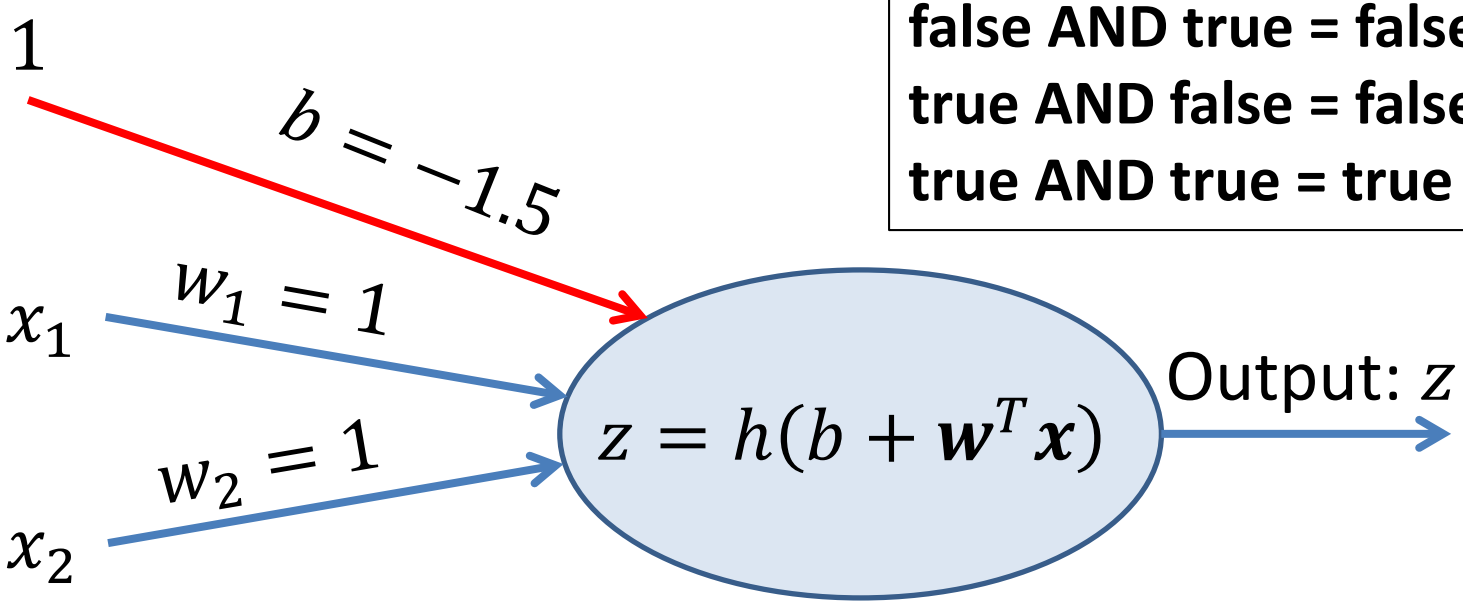
- Suppose we use the **step function** for activation.
- Suppose boolean value **false** is represented as number 0.
- Suppose boolean value **true** is represented as number 1.
- Then, the perceptron below computes the boolean AND function:



Example: The AND Perceptron

- Verification: If $x_1 = 0$ and $x_2 = 0$:
  - $b + \boldsymbol{w}^T \boldsymbol{x} = -1.5 + 1 * 0 + 1 * 0 = -1.5.$
  - $h(b + \boldsymbol{w}^T \boldsymbol{x}) = \mathrm{h}(-1.5) = 0.$
- Corresponds to case **false AND false = false**.

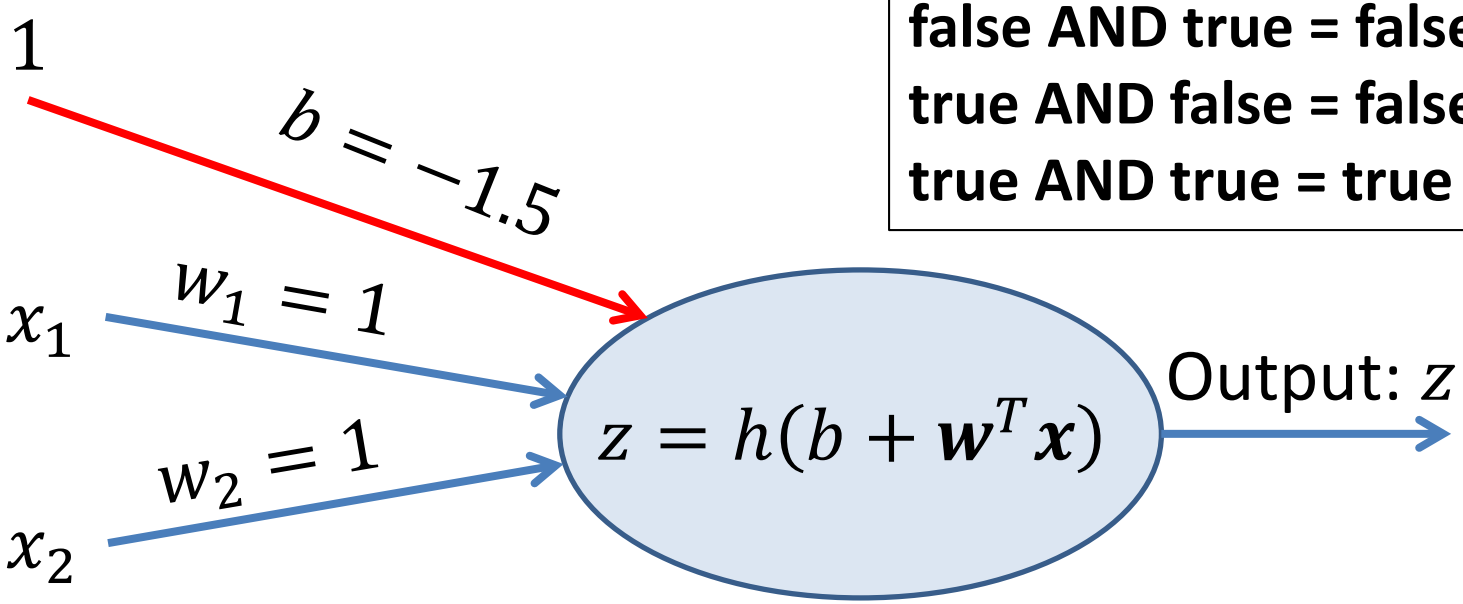
false AND false = false
false AND true = false
true AND false = false
true AND true = true



Example: The AND Perceptron

- Verification: If $x_1 = 0$ and $x_2 = 1$:
  - $b + \boldsymbol{w}^T \boldsymbol{x} = -1.5 + 1 * 0 + 1 * 1 = -0.5.$
  - $h(b + \boldsymbol{w}^T \boldsymbol{x}) = \mathrm{h}(-0.5) = 0.$
- Corresponds to case **false AND true = false**.

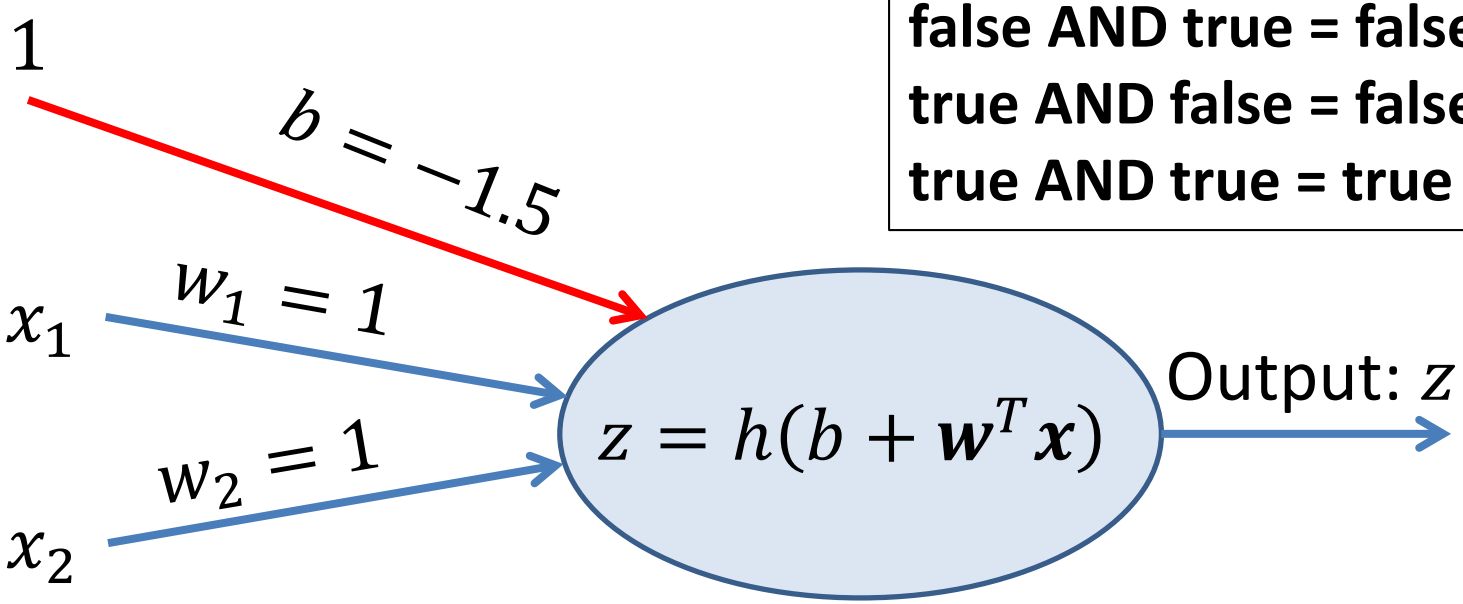
false AND false = false
false AND true = false
true AND false = false
true AND true = true



Example: The AND Perceptron

- Verification: If $x_1 = 1$ and $x_2 = 0$:
  - $b + \boldsymbol{w}^T \boldsymbol{x} = -1.5 + 1 * 1 + 1 * 0 = -0.5.$
  - $h(b + \boldsymbol{w}^T \boldsymbol{x}) = \mathrm{h}(-0.5) = 0.$
- Corresponds to case **true AND false = false**.

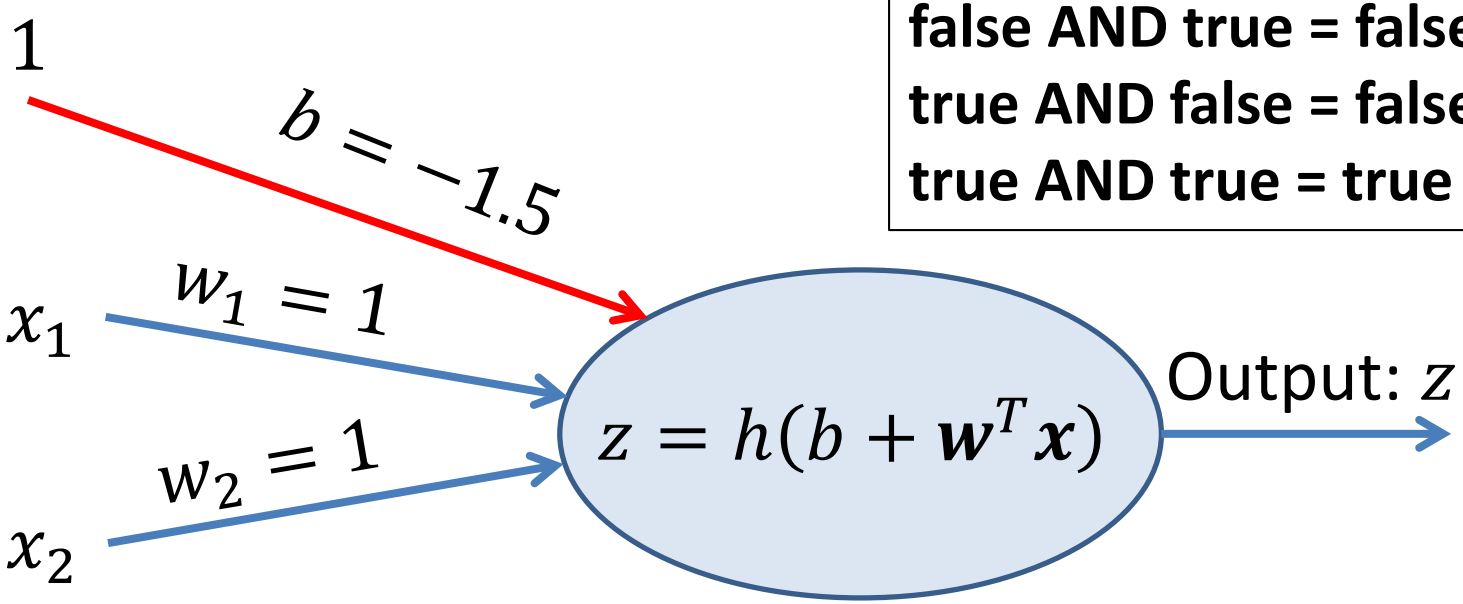
false AND false = false
false AND true = false
true AND false = false
true AND true = true



Example: The AND Perceptron

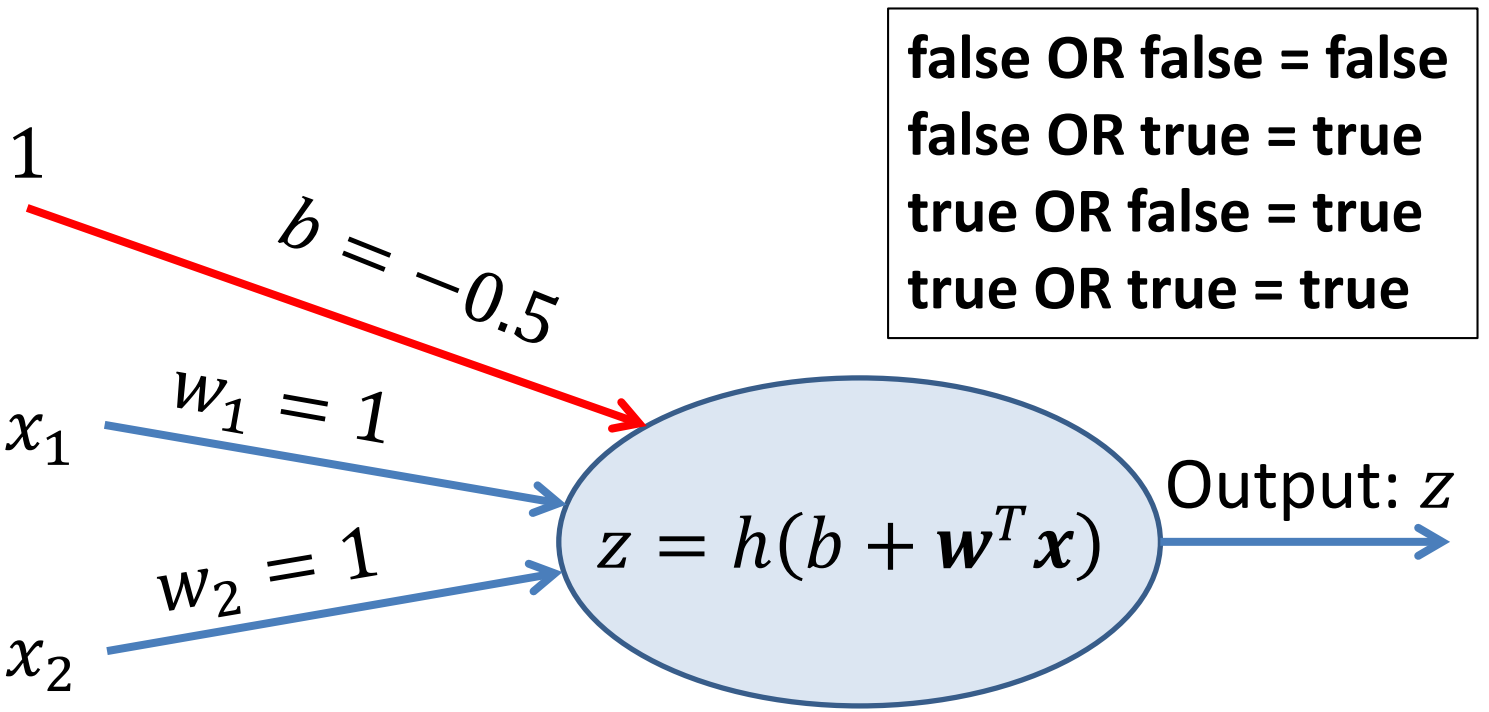
- Verification: If $x_1 = 1$ and $x_2 = 1$:
  - $b + \boldsymbol{w}^T \boldsymbol{x} = -1.5 + 1 * 1 + 1 * 1 = 0.5.$
  - $h(b + \boldsymbol{w}^T \boldsymbol{x}) = \mathrm{h}(0.5) = 1.$
- Corresponds to case **true AND true = true**.

false AND false = false
false AND true = false
true AND false = false
true AND true = true



Example: The OR Perceptron

- Suppose we use the **step function** for activation.
- Suppose boolean value **false** is represented as number 0.
- Suppose boolean value **true** is represented as number 1.
- Then, the perceptron below computes the boolean OR function:

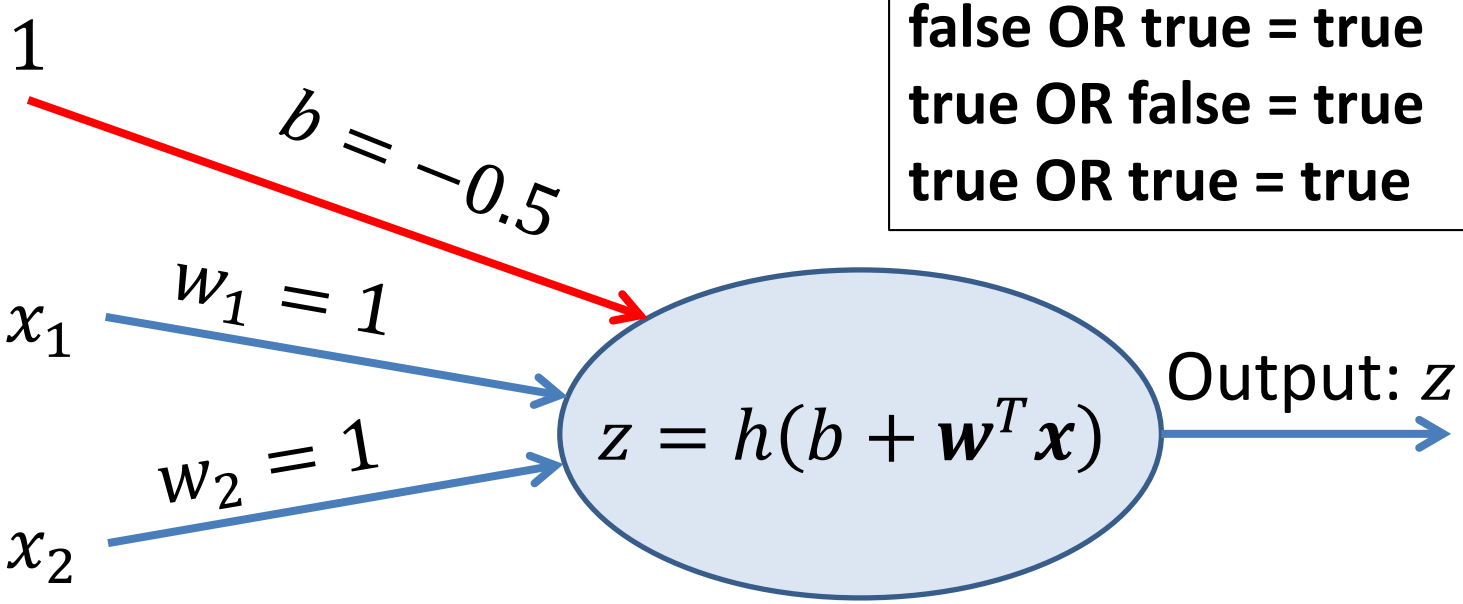


false OR false = false
false OR true = true
true OR false = true
true OR true = true

Example: The OR Perceptron

- Verification: If $x_1 = 0$ and $x_2 = 0$:
  - $b + \boldsymbol{w}^T \boldsymbol{x} = -0.5 + 1 * 0 + 1 * 0 = -0.5.$
  - $h(b + \boldsymbol{w}^T \boldsymbol{x}) = \mathrm{h}(-0.5) = 0.$
- Corresponds to case **false OR false = false**.

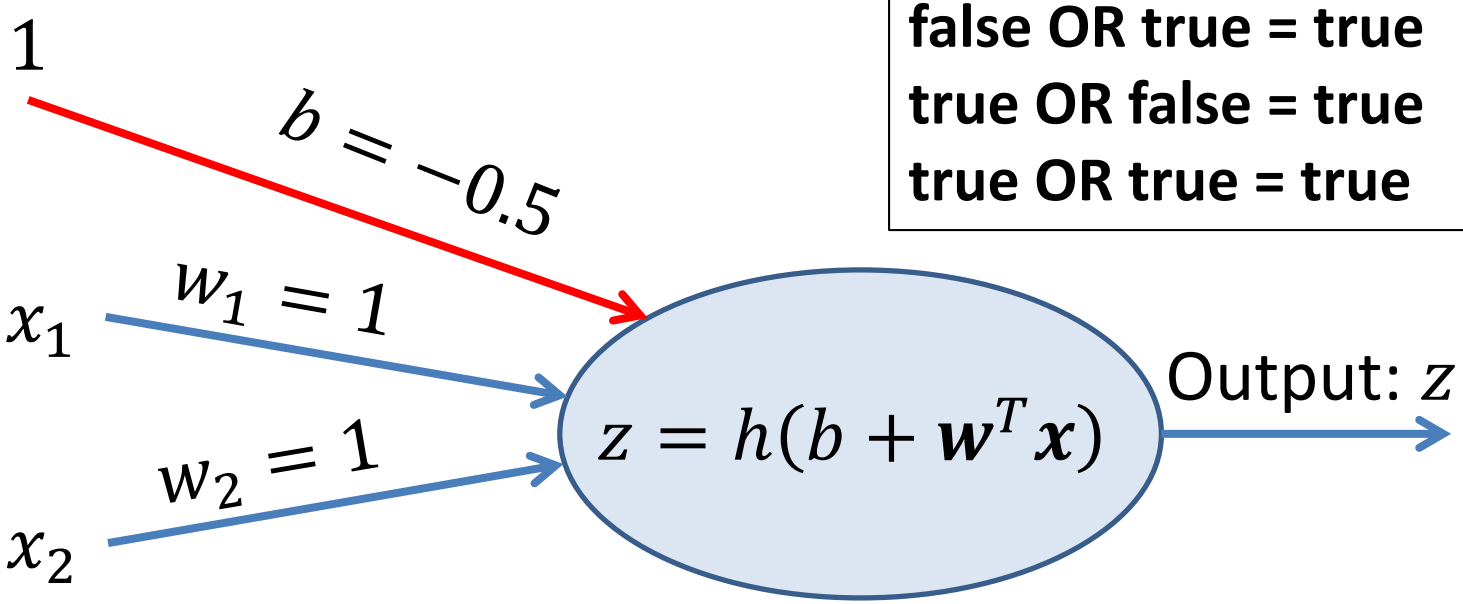
false OR false = false
false OR true = true
true OR false = true
true OR true = true



Example: The OR Perceptron

- Verification: If $x_1 = 0$ and $x_2 = 1$:
  - $b + \boldsymbol{w}^T \boldsymbol{x} = -0.5 + 1 * 0 + 1 * 1 = 0.5.$
  - $h(b + \boldsymbol{w}^T \boldsymbol{x}) = \mathrm{h}(0.5) = 1.$
- Corresponds to case **false OR true = true**.

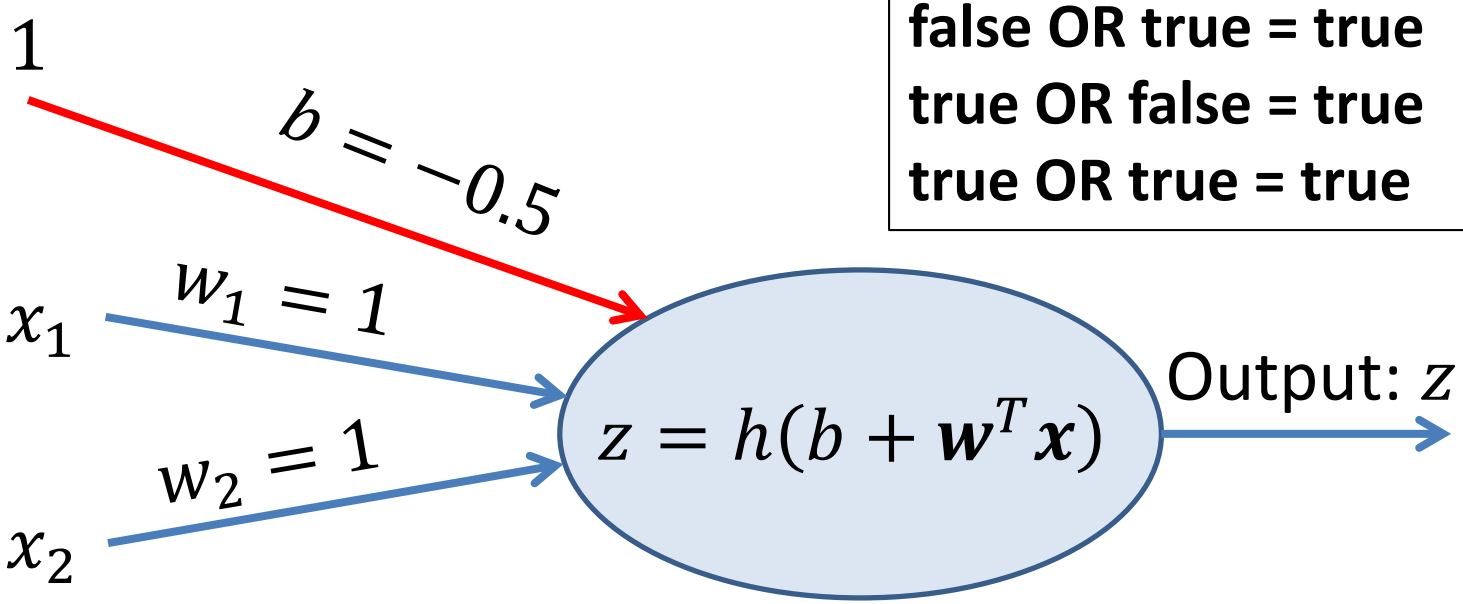
false OR false = false
false OR true = true
true OR false = true
true OR true = true



Example: The OR Perceptron

- Verification: If $x_1 = 1$ and $x_2 = 0$:
  - $b + \boldsymbol{w}^T \boldsymbol{x} = -0.5 + 1 * 1 + 1 * 0 = 0.5.$
  - $h(b + \boldsymbol{w}^T \boldsymbol{x}) = \mathrm{h}(0.5) = 1.$
- Corresponds to case **true OR false = true**.

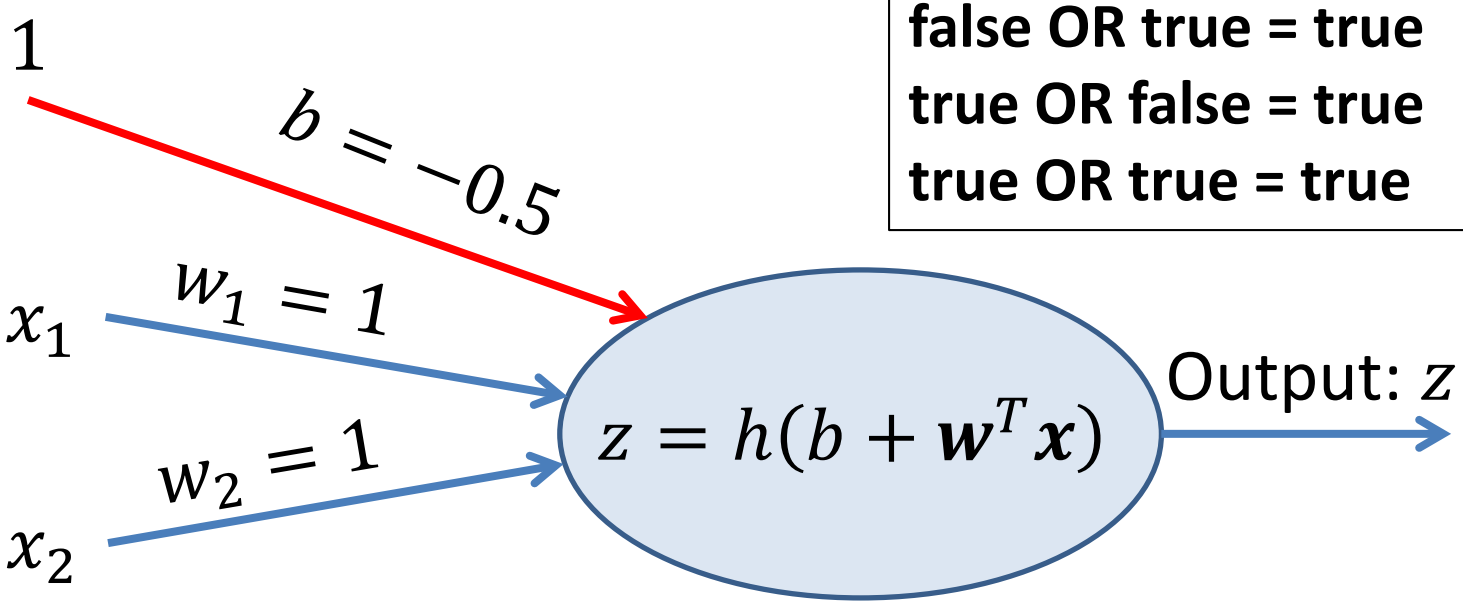
false OR false = false
false OR true = true
true OR false = true
true OR true = true



Example: The OR Perceptron

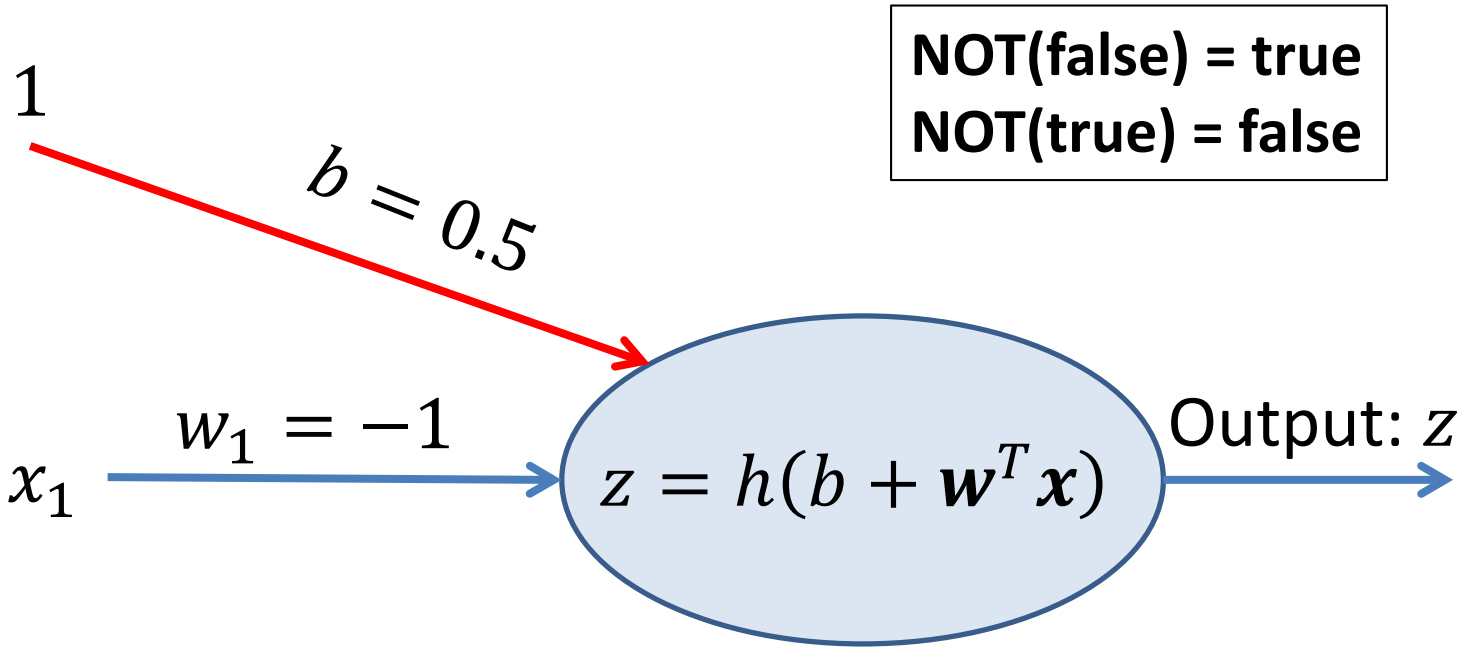
- Verification: If $x_1 = 1$ and $x_2 = 1$:
  - $b + \boldsymbol{w}^T \boldsymbol{x} = -0.5 + 1 * 1 + 1 * 1 = 1.5.$
  - $h(b + \boldsymbol{w}^T \boldsymbol{x}) = \mathrm{h}(1.5) = 1.$
- Corresponds to case **true OR true = true**.

false OR false = false
false OR true = true
true OR false = true
true OR true = true



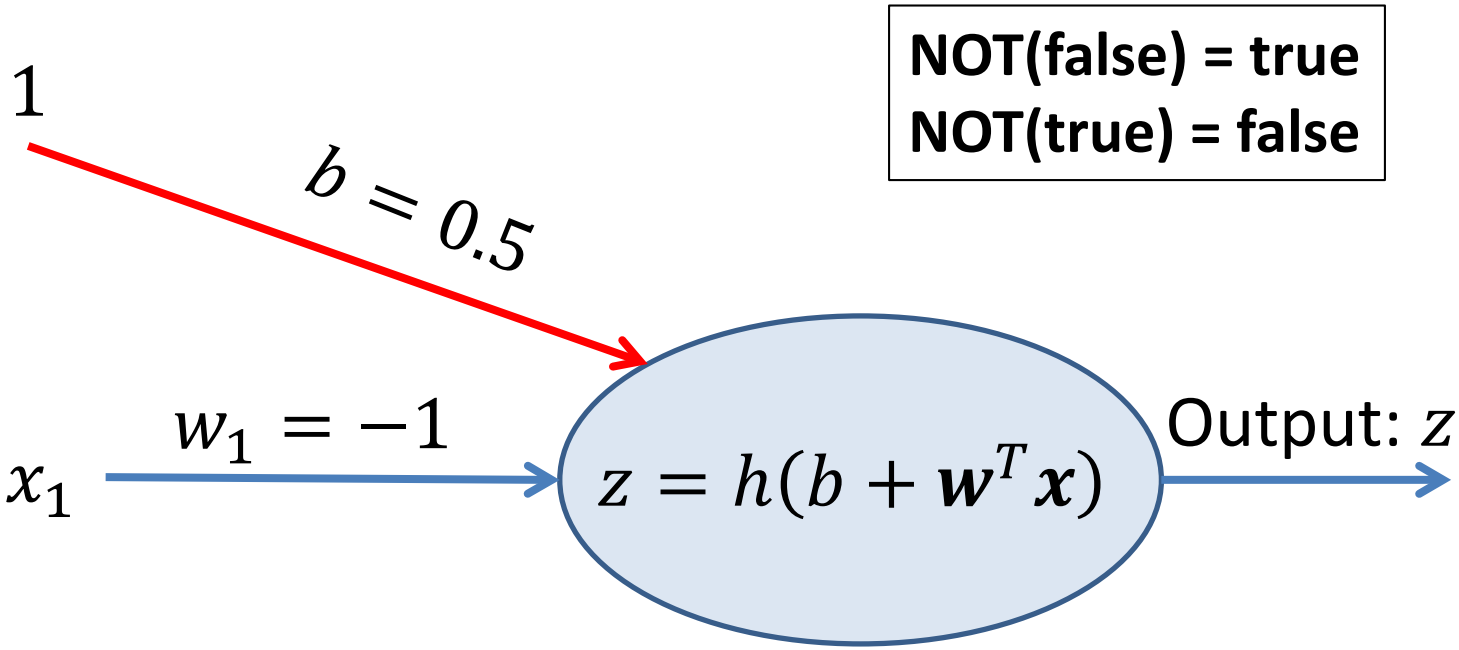
Example: The NOT Perceptron

- Suppose we use the **step function** for activation.
- Suppose boolean value **false** is represented as number 0.
- Suppose boolean value **true** is represented as number 1.
- Then, the perceptron below computes the boolean NOT function:



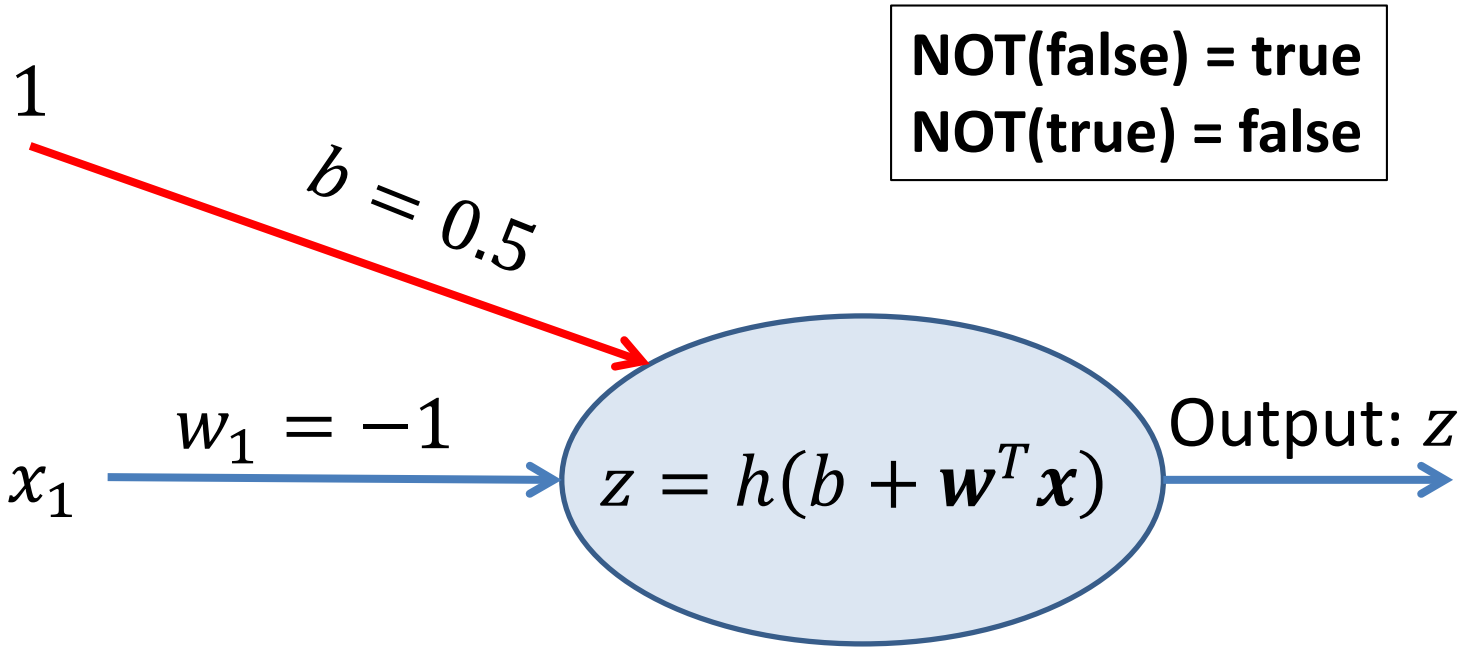
Example: The NOT Perceptron

- Verification: If $x_1 = 0$:
  - $b + \boldsymbol{w}^T \boldsymbol{x} = 0.5 - 1 * 0 = 0.5.$
  - $h(b + \boldsymbol{w}^T \boldsymbol{x}) = \mathrm{h}(0.5) = 1.$
- Corresponds to case **NOT(false) = true**.



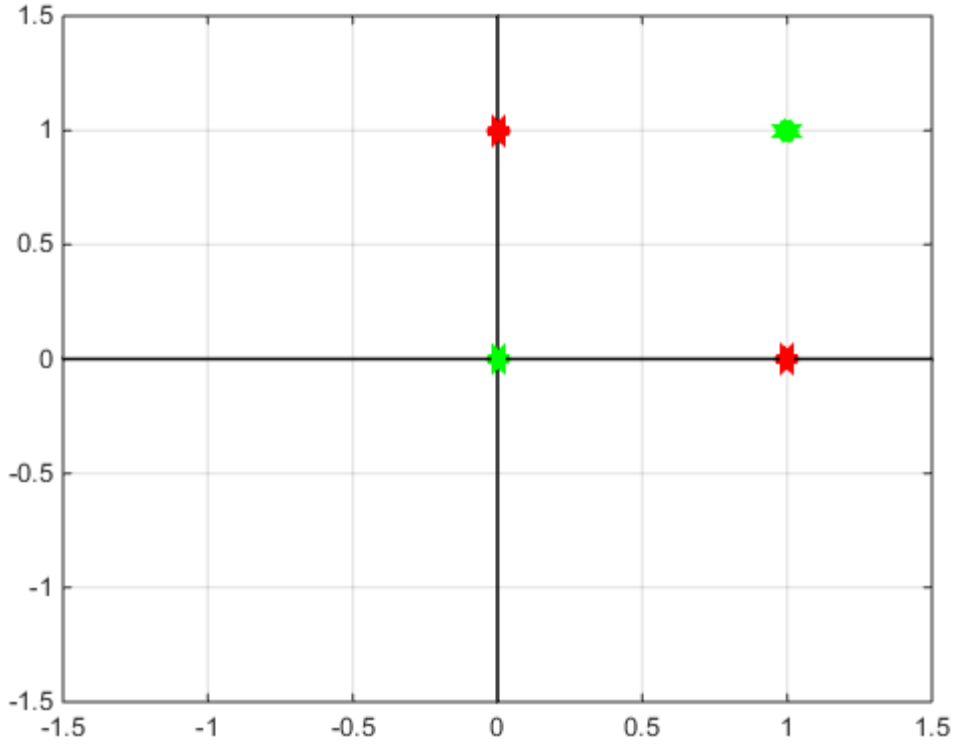
Example: The NOT Perceptron

- Verification: If $x_1 = 1$:
  - $b + \boldsymbol{w}^T \boldsymbol{x} = 0.5 - 1 * 1 = -0.5.$
  - $h(b + \boldsymbol{w}^T \boldsymbol{x}) = \mathrm{h}(-0.5) = 0.$
- Corresponds to case **NOT(true) = false**.



The XOR Function

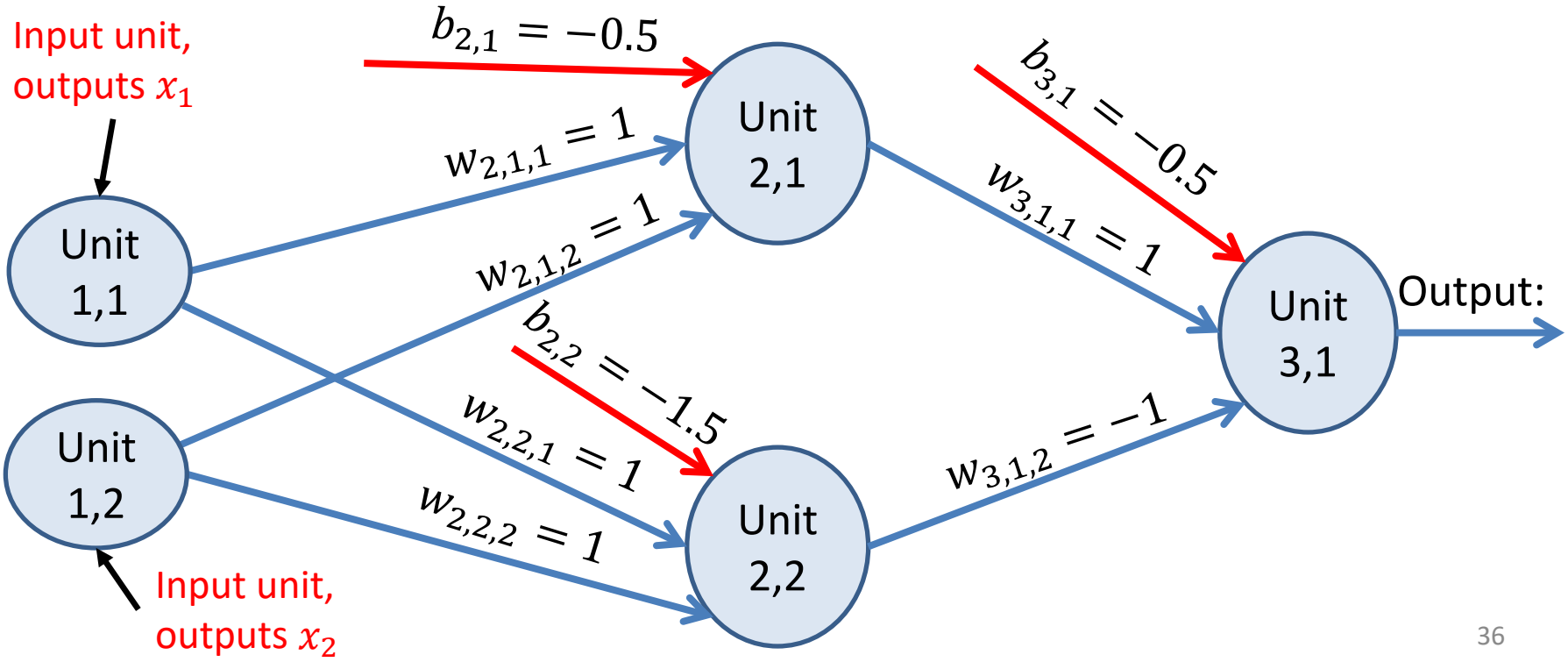
false XOR false = false
false XOR true = true
true XOR false = true
true XOR true = false



- As before, **false** is 0, **true** is 1.
- The figure shows the four input points of the XOR function.
  - red corresponds to output value **true**.
  - green corresponds to output value **false**.
- The two classes (true and false) are not linearly separable.
  - This means that we cannot separate them with a straight line.
- It can be proven that no perceptron can compute the XOR function (because the dot product is a linear operation).

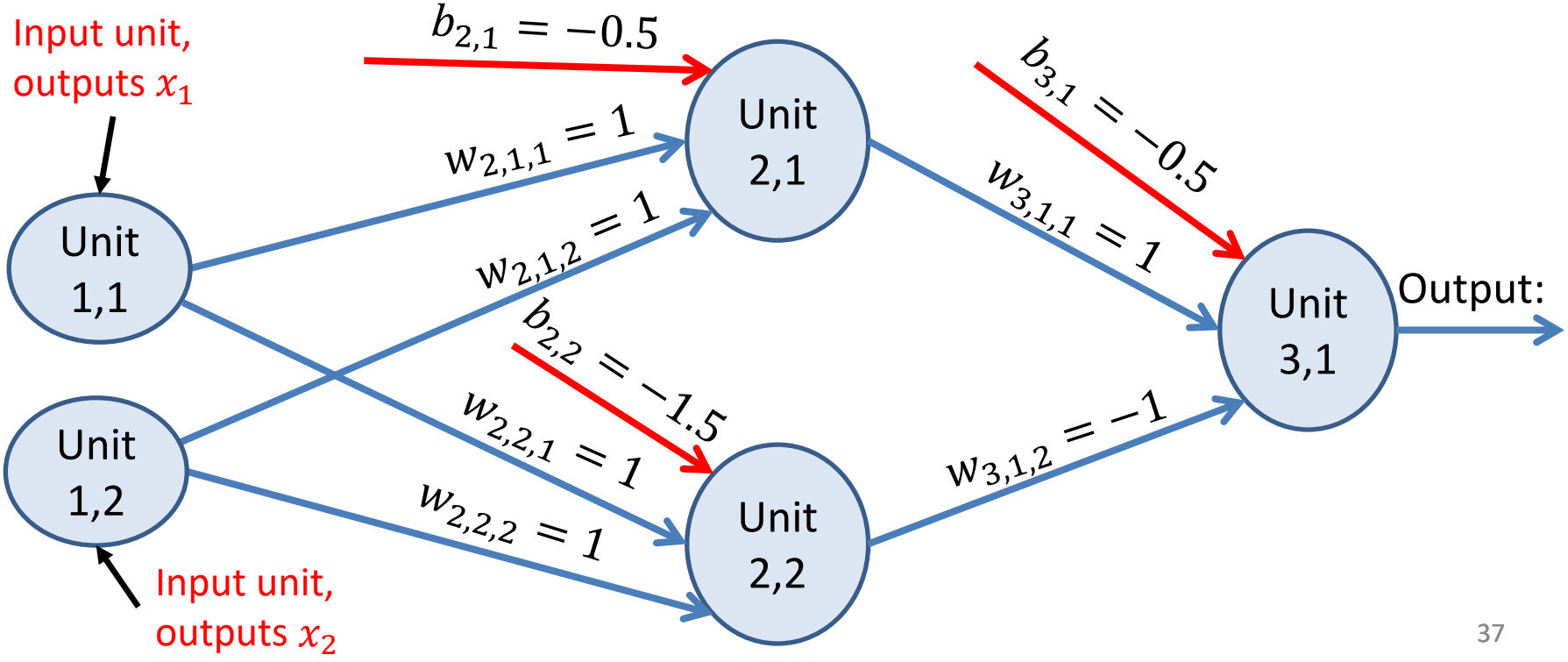
Our First Neural Network: XOR

- A neural network is built using perceptrons as building blocks.
- The inputs to some perceptrons are outputs of other perceptrons.
- Here is an example neural network computing the XOR function.



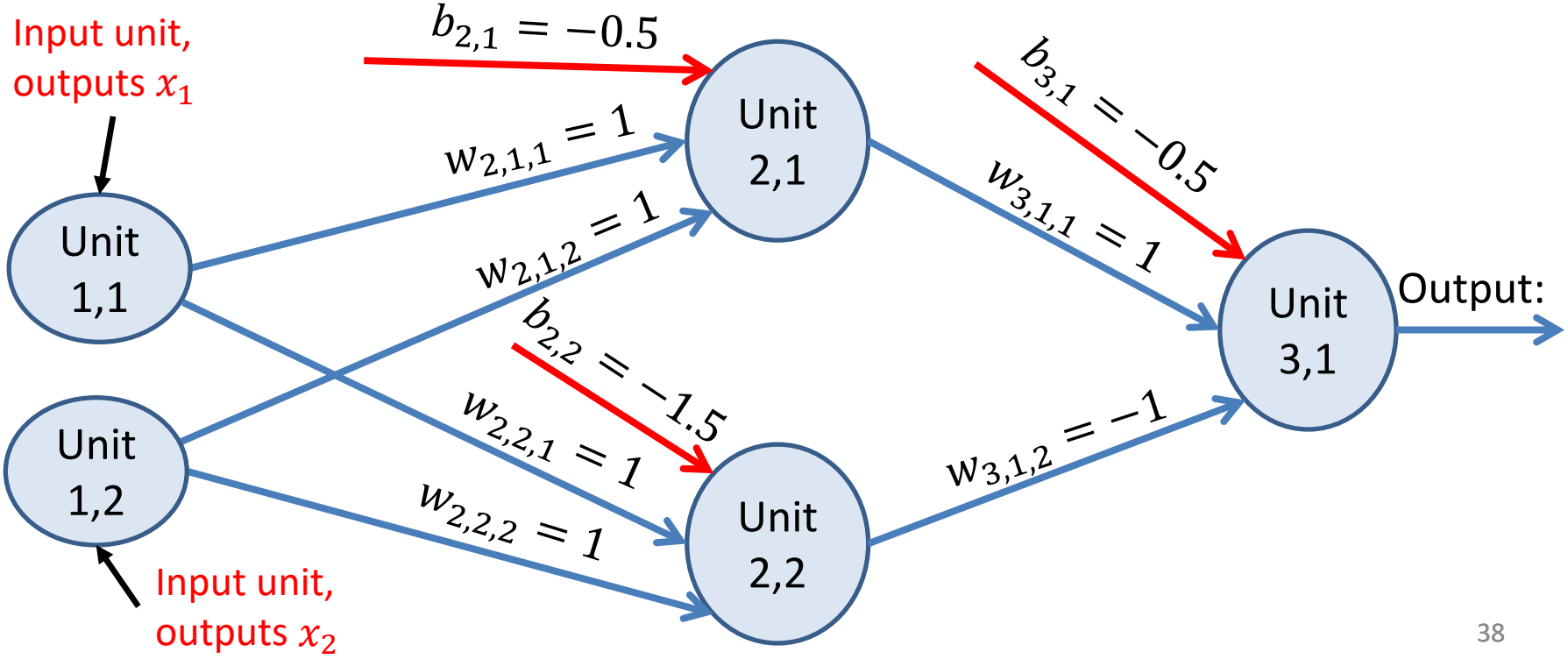
Our First Neural Network: XOR

- Terminology: inputs and perceptrons are all called "units".
- Units are grouped in layers: layer 1 (input), layer 2, layer 3 (output).
- The input layer just represents the inputs to the network.
  - There are two inputs: x_1 and x_2.



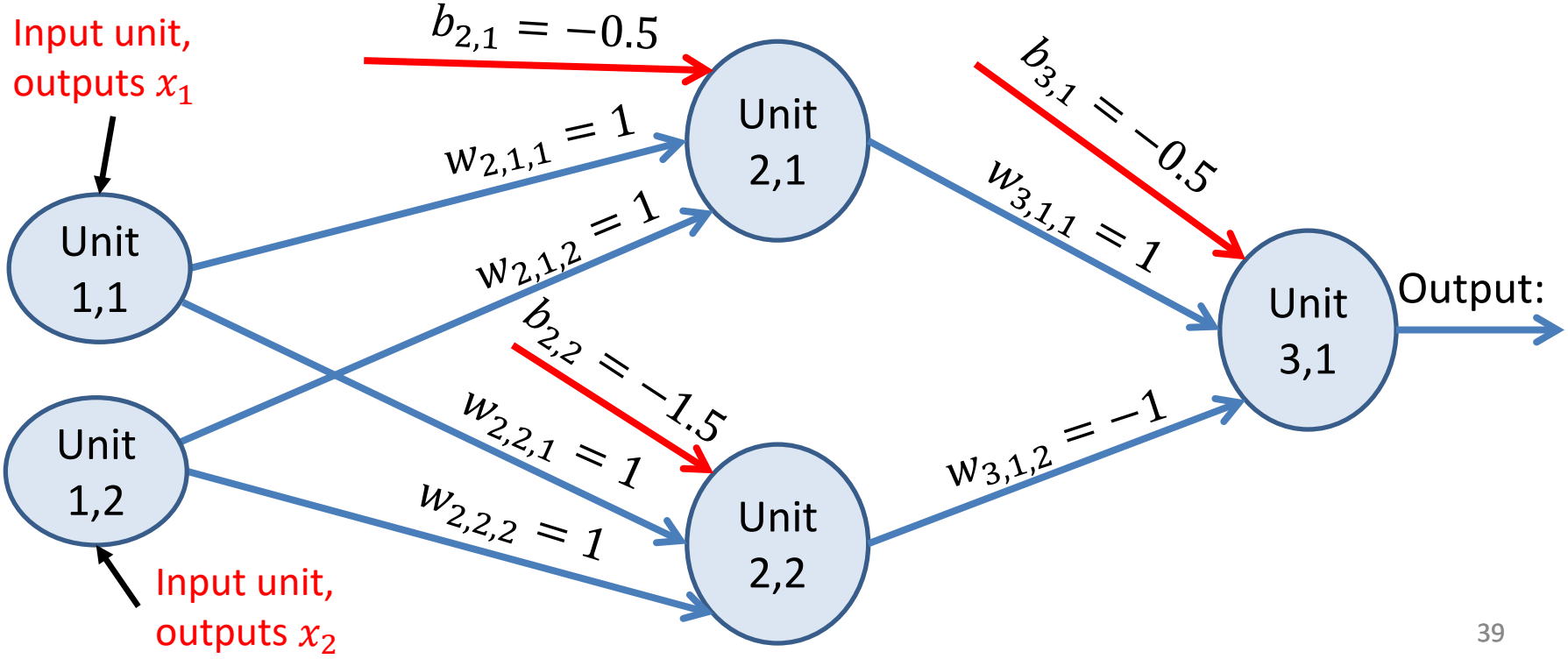
Our First Neural Network: XOR

- Such networks are called **layered** networks, more details later.
- Each unit is indexed by two numbers (layer index, unit index).
- Each bias weight b is indexed by the same two numbers as its unit.
- Each weight w is indexed by three numbers (layer, unit, weight).



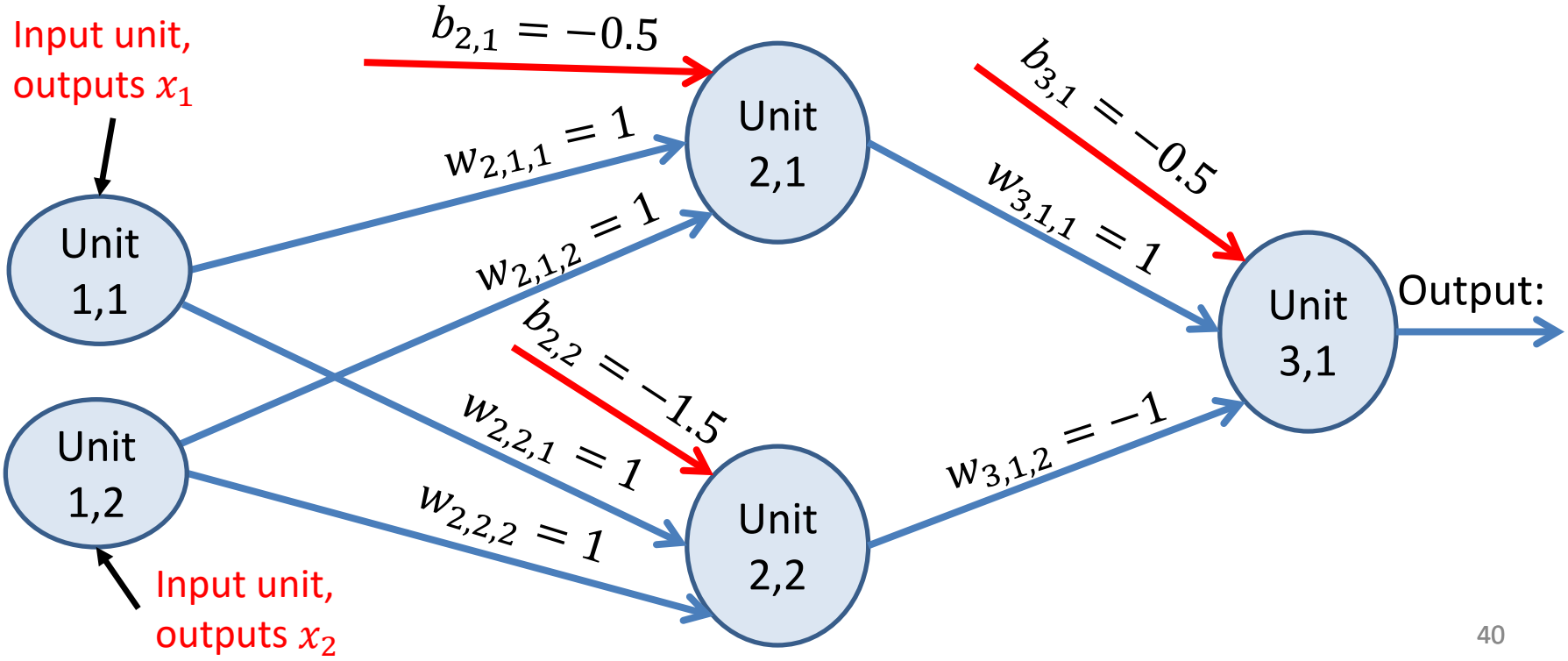
Our First Neural Network: XOR

- Note: every weight is associated with two units: it connects the output of a unit with an input of another unit.
  - Which of the two units do we use to index the weight?



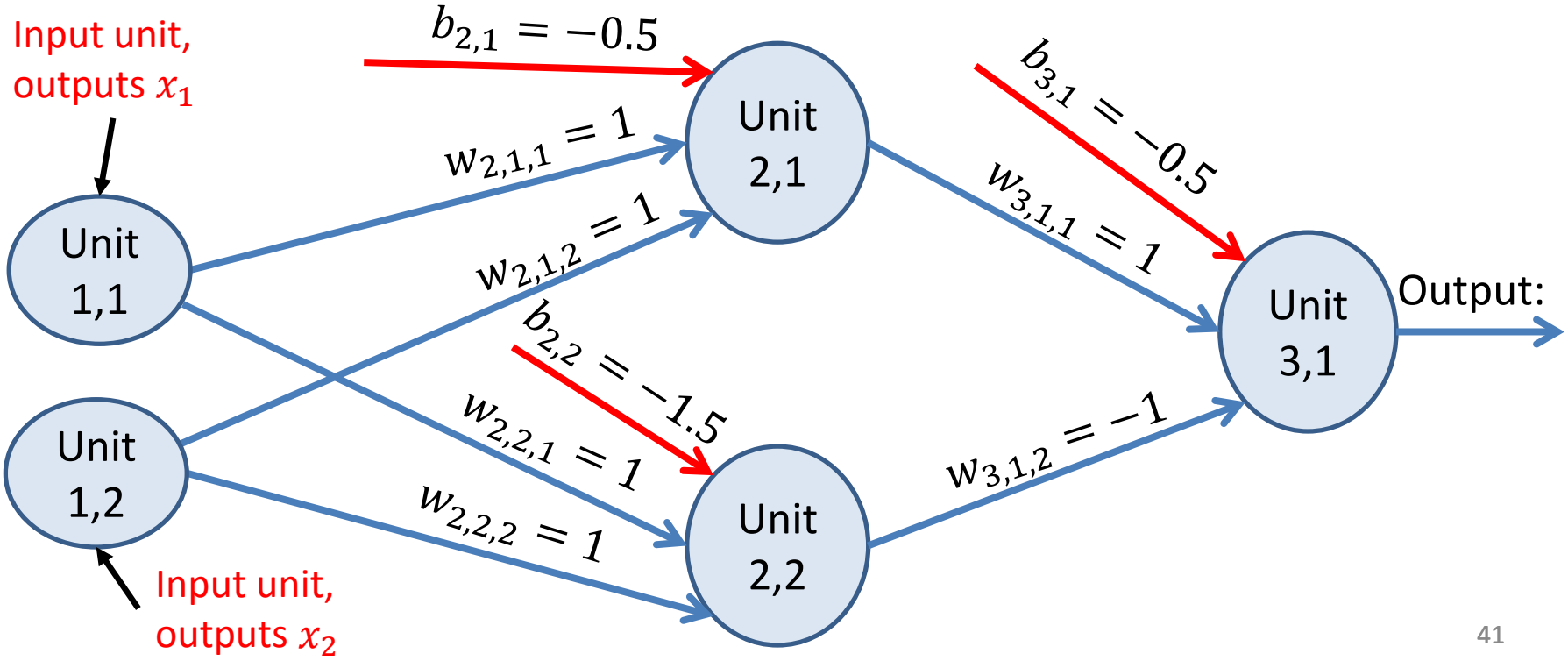
Our First Neural Network: XOR

- To index a weight w, we use the layer number and unit number of the unit for which w is an **incoming weight**.
- Weights incoming to unit l, i are indexed as $l, i, j,$ where j ranges from 1 to the number of incoming weights for unit l, i.



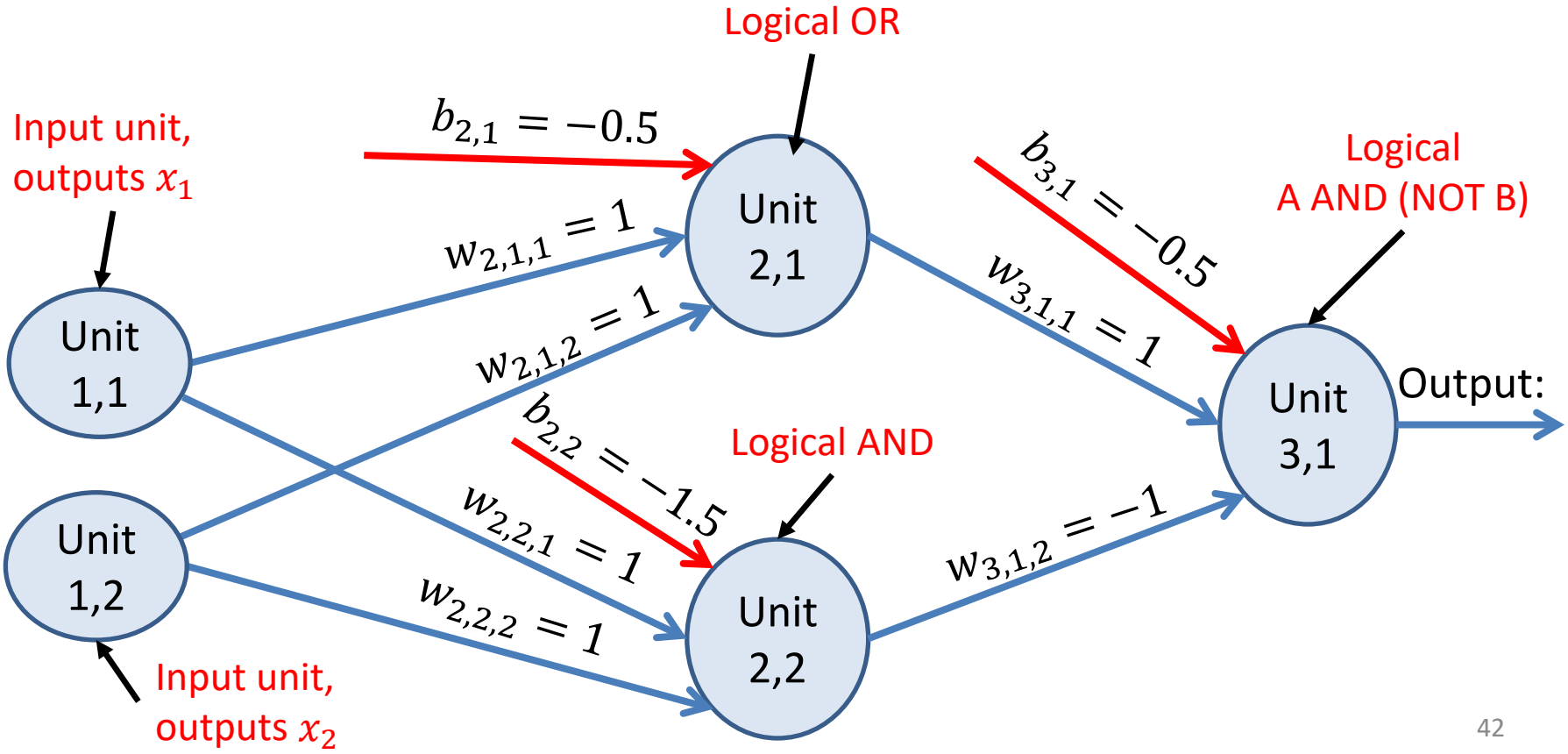
Our First Neural Network: XOR

- Weights incoming to unit l, i are indexed as $l, i, j,$ where j ranges from 1 to the number of incoming weights for unit l, i.
- Since the input layer (which is layer 1) has no incoming weights, there are no weights indexed as $w_{1,i,j}$.



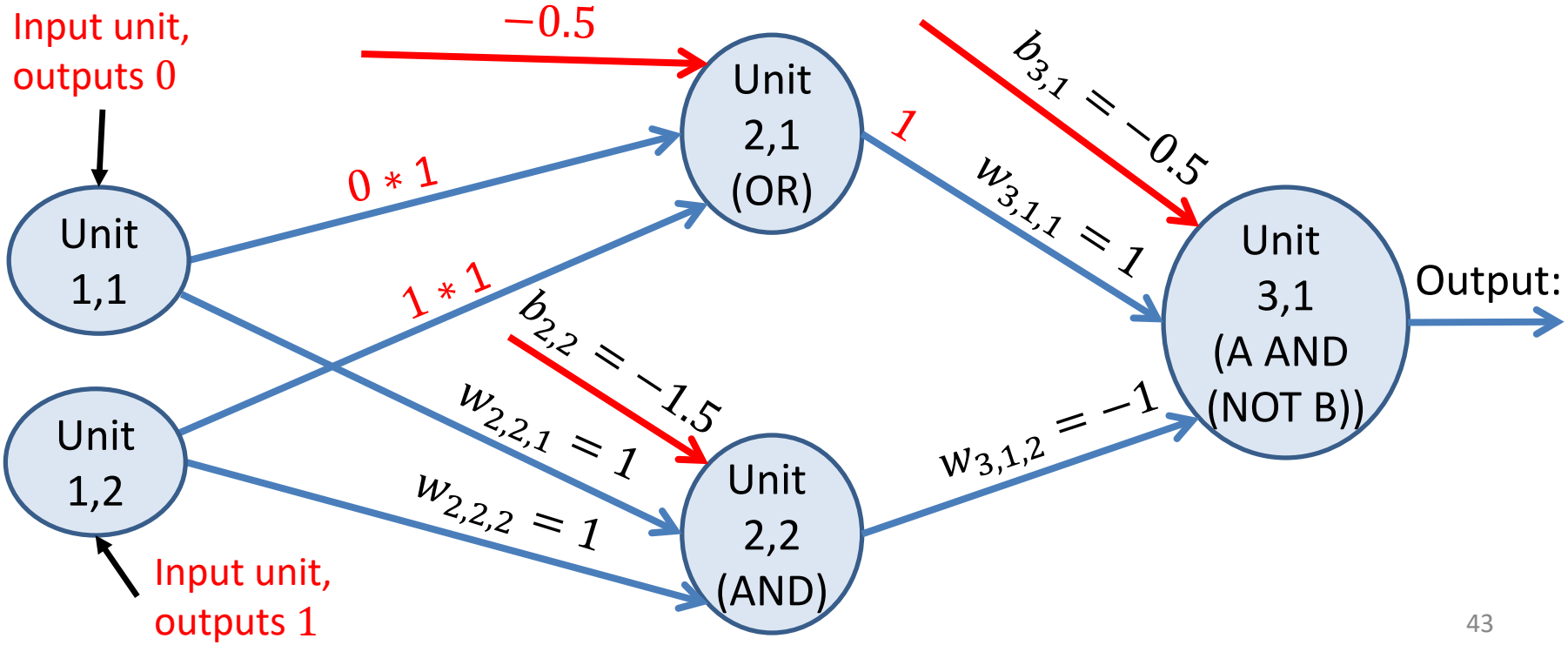
Our First Neural Network: XOR

- The XOR network shows how individual perceptrons can be combined to perform more complicated functions.



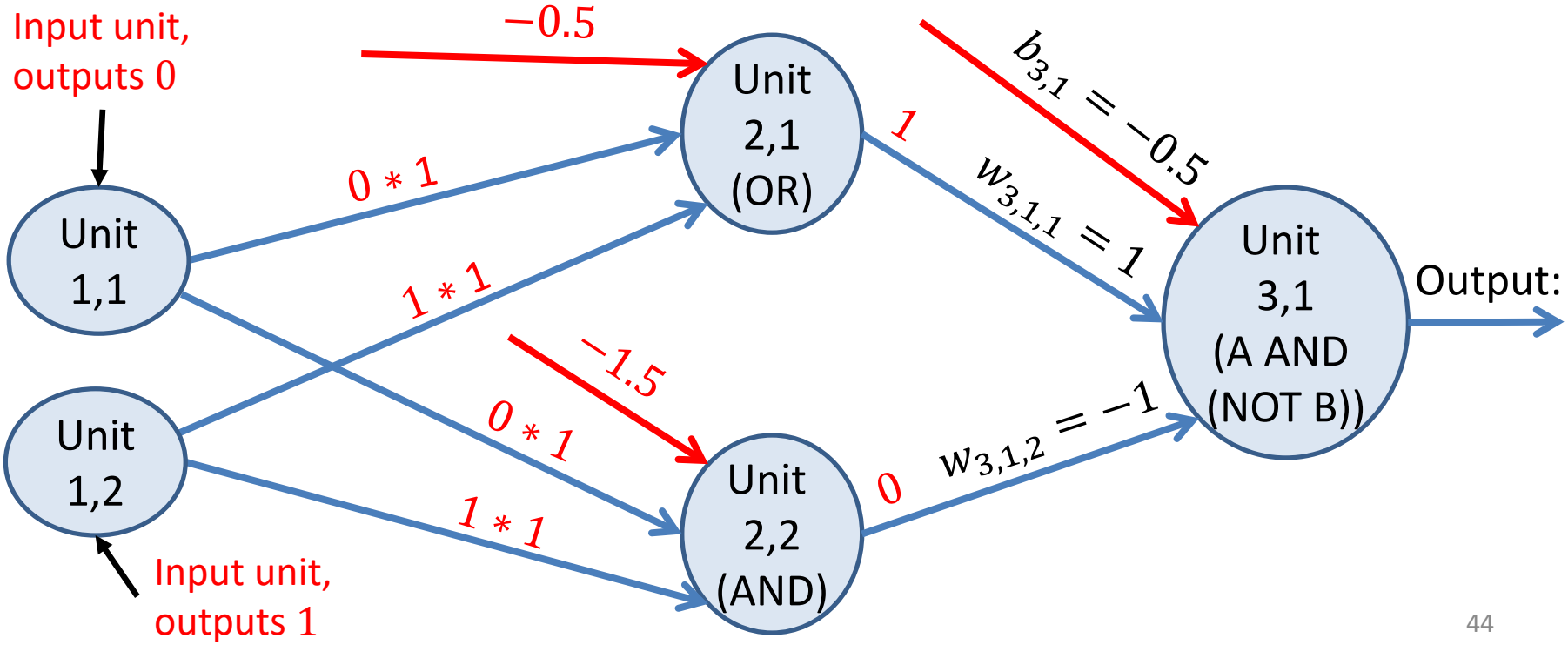
Computing the Output: An Example

- Suppose that $x_1 = 0, x_2 = 1$ (corresponding to **false** XOR **true**).
- For Unit 2,1, which performs a logical OR:
  - The output is $h(-0.5 + 0 * 1 + 1 * 1) = h(0.5)$.
  - Assuming that h is the step function, $h(0.5) = 1$, so Unit 2,1 outputs 1.



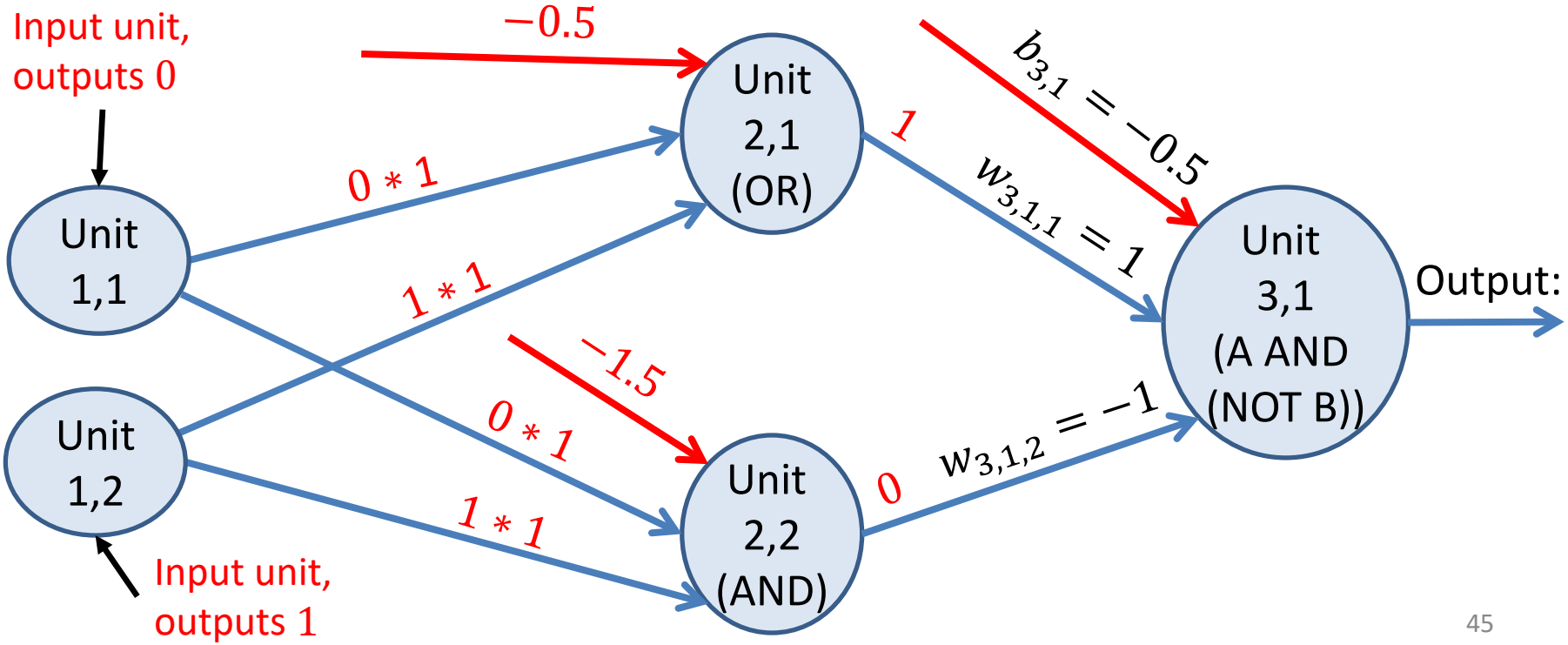
Computing the Output: An Example

- Suppose that $x_1 = 0, x_2 = 1$ (corresponding to **false** XOR **true**).
- For Unit 2,2, which performs a logical AND:
  - The output is $h(-1.5 + 0 * 1 + 1 * 1) = h(-0.5)$.
  - Since h is the step function, $h(-0.5) = 0$, so Unit 2,2 outputs 0.



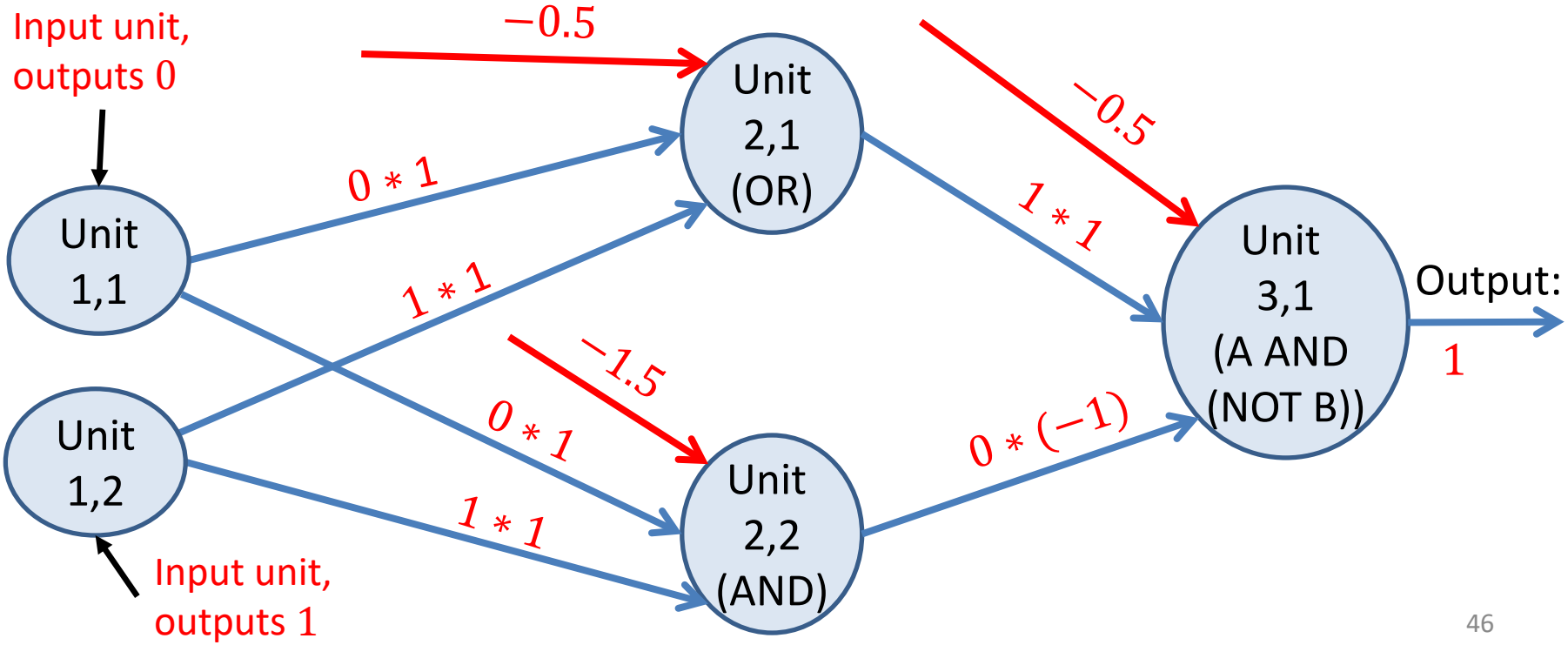
Computing the Output: An Example

- Suppose that $x_1 = 0, x_2 = 1$ (corresponding to **false** XOR **true**).
- Unit 3,1 is the **output unit**, computing the A AND (NOT B) function:
  - One input is the output of the OR unit, which is 1.
  - The other input is the output of the AND unit, which is 0.



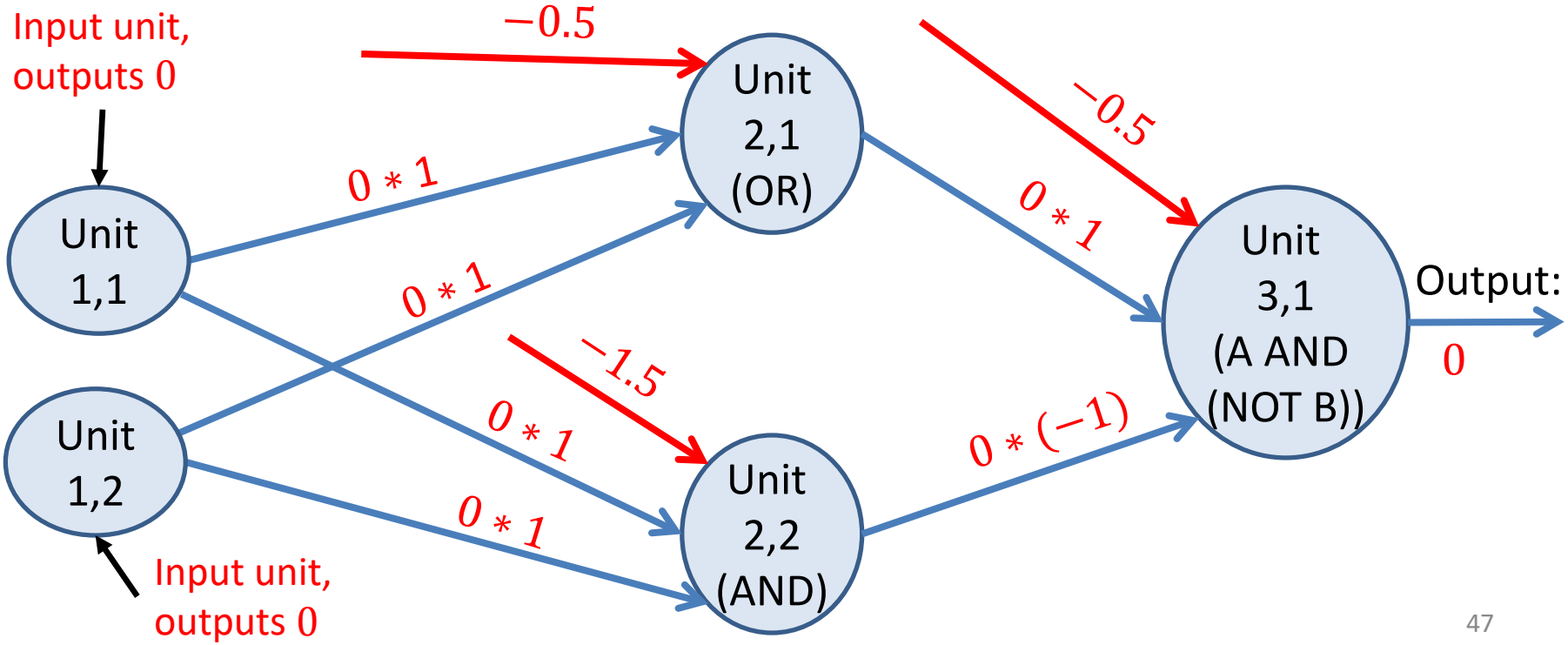
Computing the Output: An Example

- Suppose that $x_1 = 0, x_2 = 1$ (corresponding to **false** XOR **true**).
- For the output unit (computing the A AND (NOT B) function):
  - The output is $h(-0.5 + 1 * 1 + 0 * (-1)) = h(0.5)$.
  - Since h is the step function, $h(0.5) = 1$, so Unit 3,1 outputs 1.



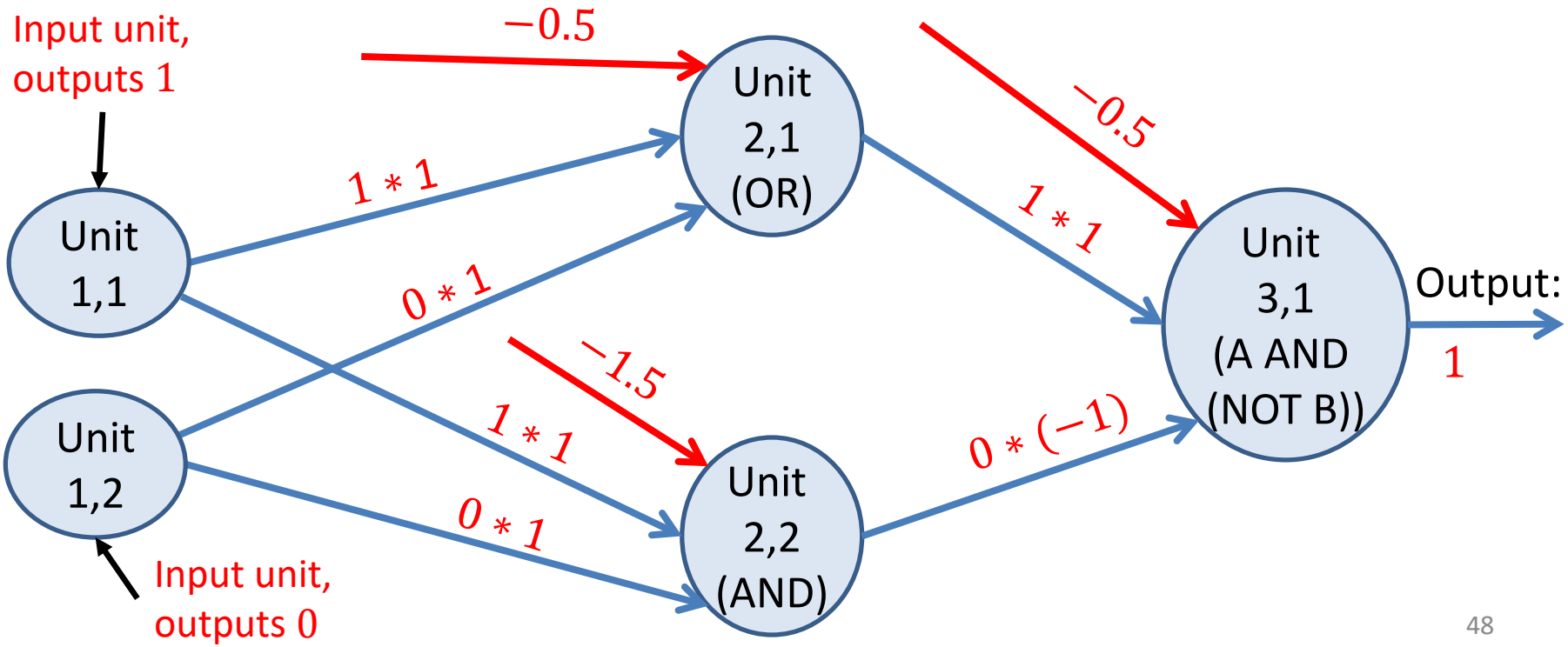
Verifying the XOR Network

- We can follow the same process to compute the output of this network for the other three cases.
  - Here we consider the case where $x_1 = 0, x_2 = 0$ (corresponding to **false** XOR **false**).
  - The output is 0, as it should be.



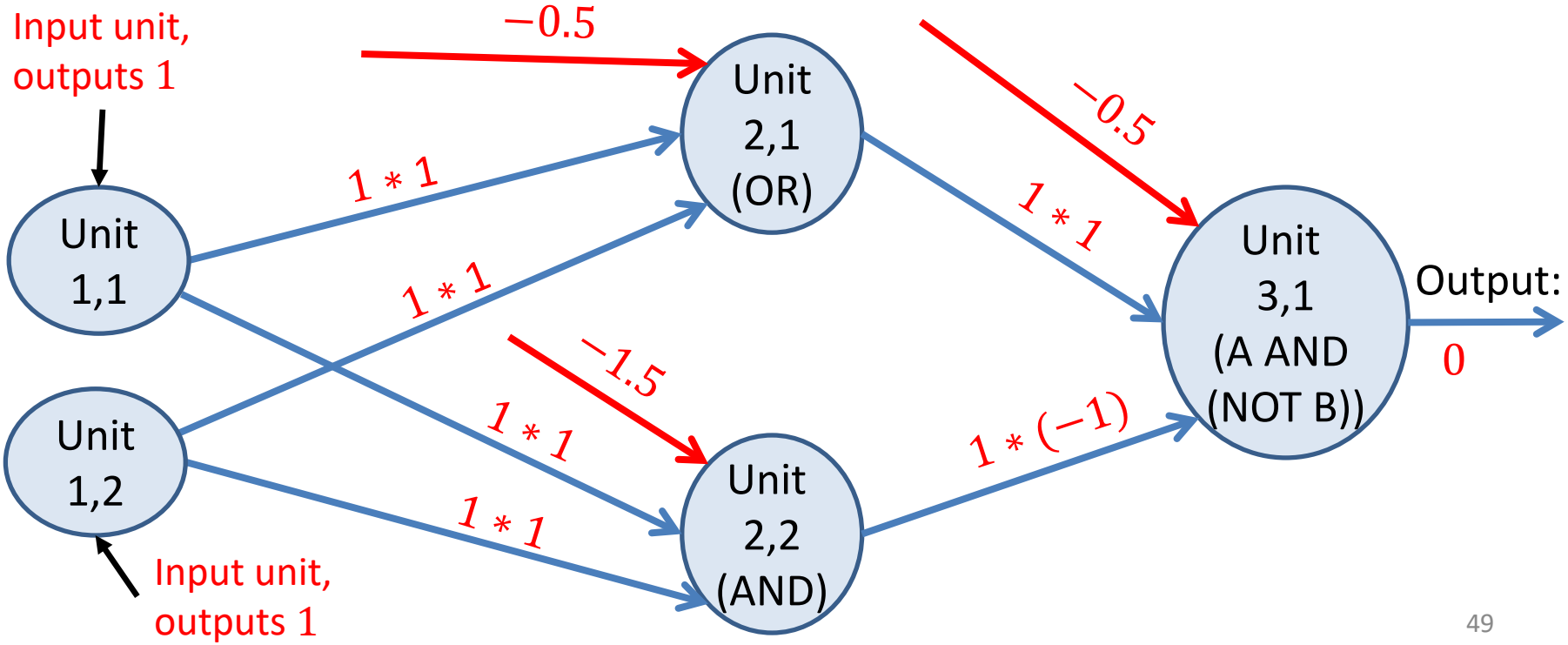
Verifying the XOR Network

- We can follow the same process to compute the output of this network for the other three cases.
  - Here we consider the case where $x_1 = 1, x_2 = 0$ (corresponding to **true** XOR **false**).
  - The output is 1, as it should be.



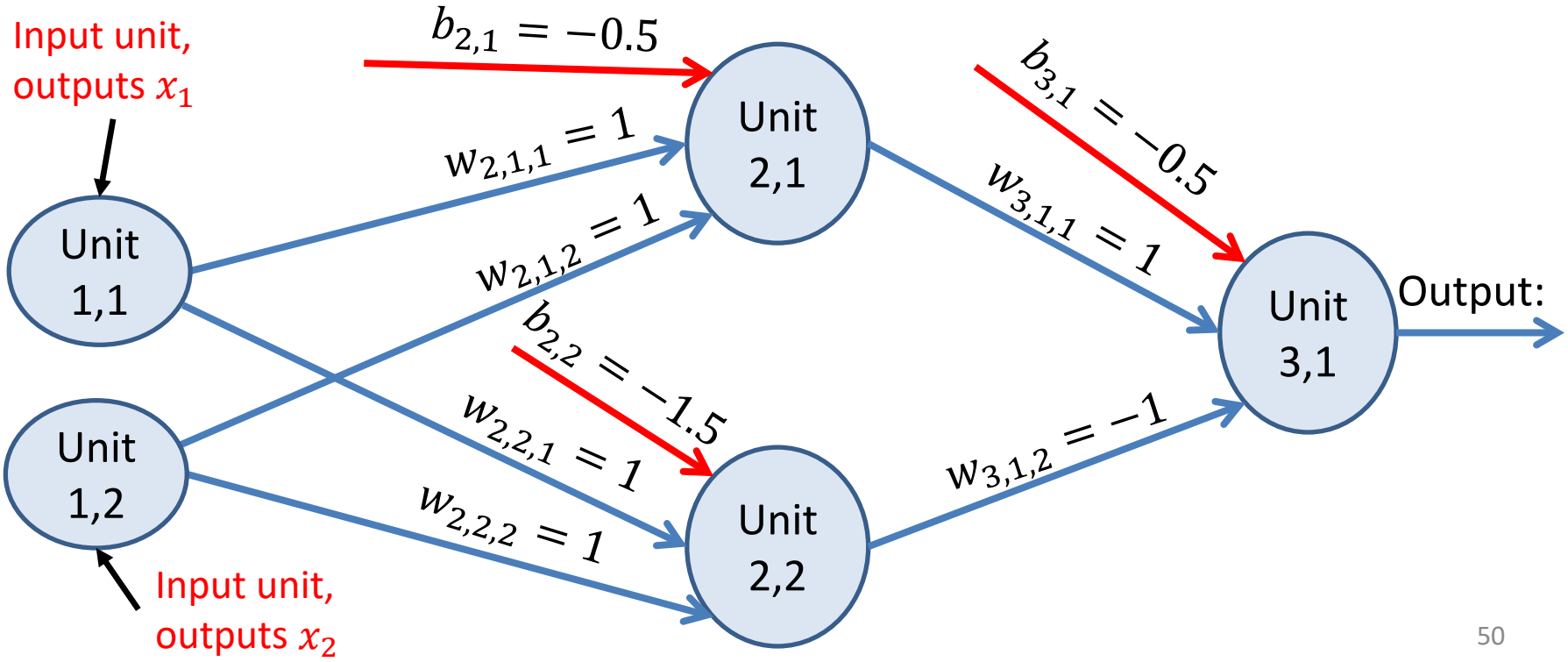
Verifying the XOR Network

- We can follow the same process to compute the output of this network for the other three cases.
  - Here we consider the case where $x_1 = 1, x_2 = 1$ (corresponding to **true** XOR **true**).
  - The output is 0, as it should be.



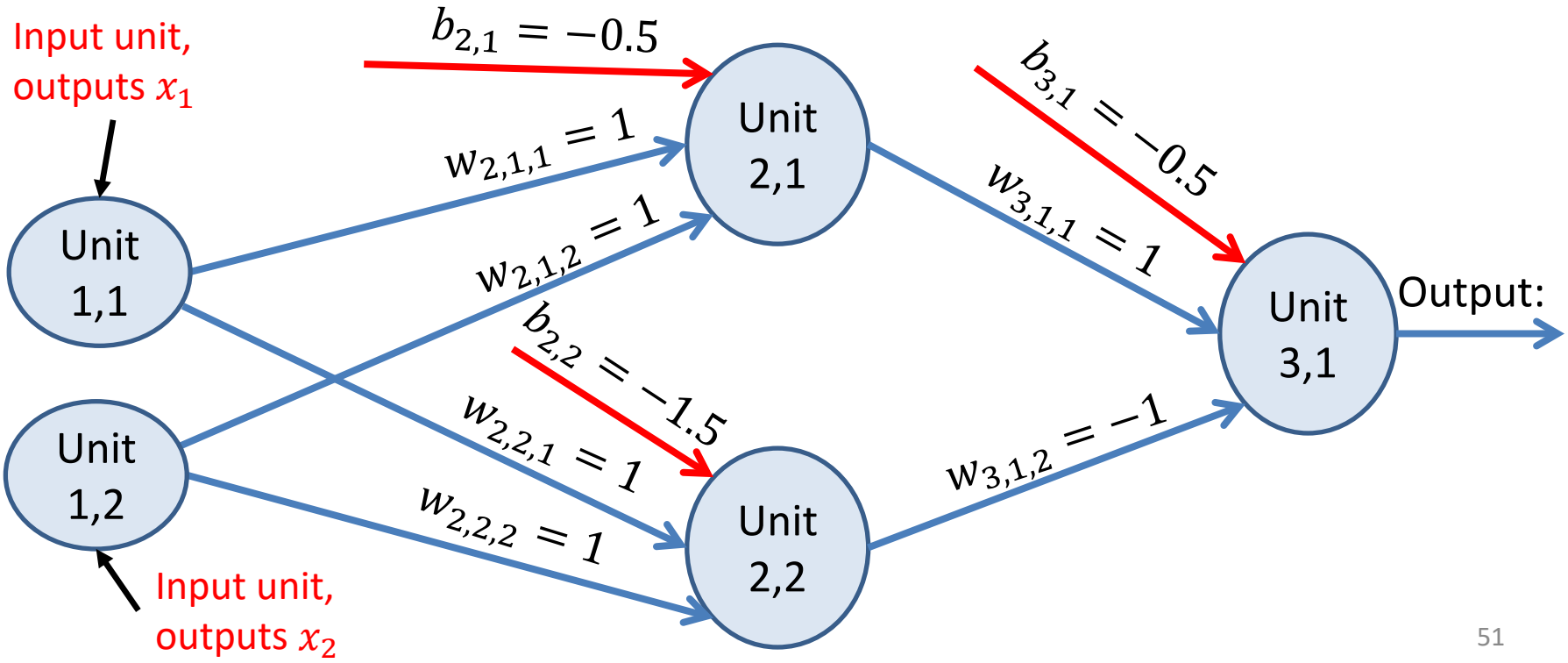
Neural Networks

- Our XOR neural network consists of five units:
  - Two input units, that just represent the two inputs to the network.
  - Three perceptrons.



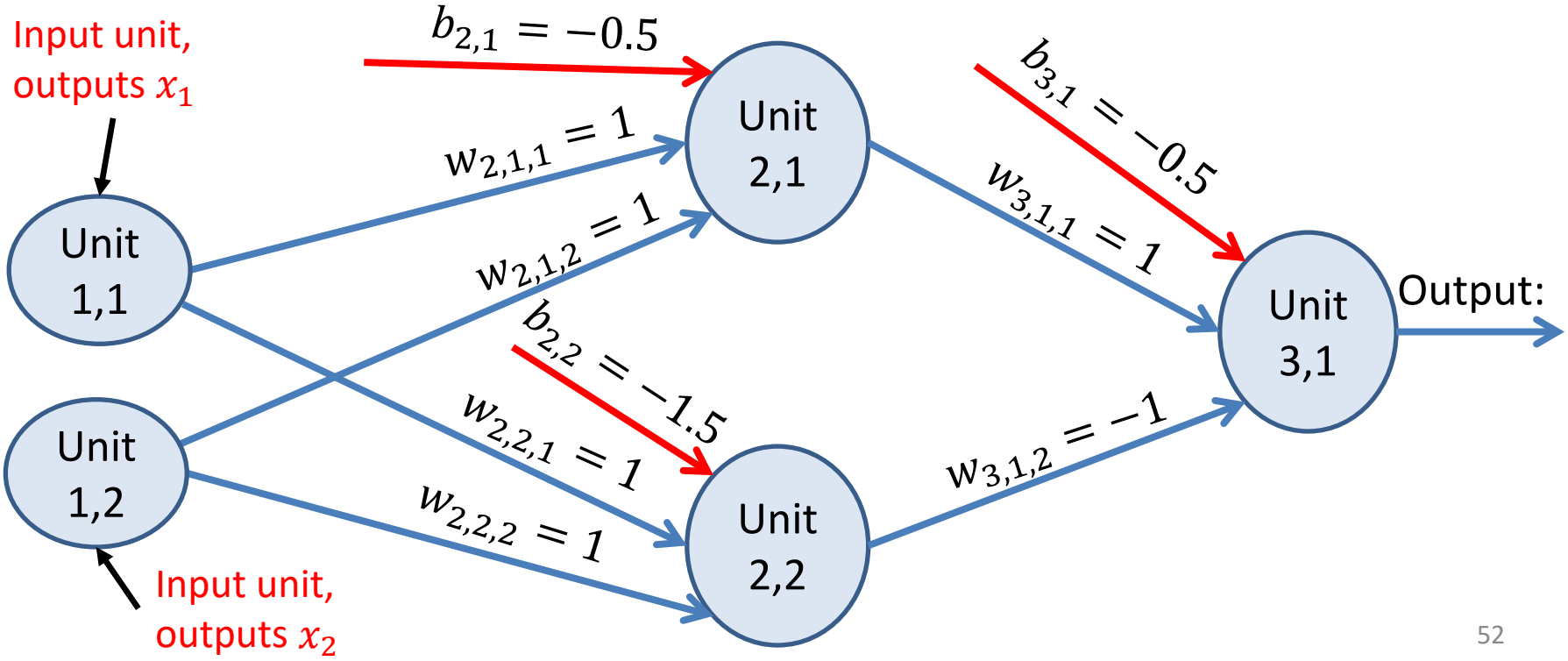
Neural Network Layers

- Usually, as in the XOR example, neural networks are organized into layers.
- The **input layer** is the initial layer of input units (units 1,1 and 1,2 in our example).
- The output layer is at the end (unit 3,1 in our example).
- Zero, one or more hidden layers can be between the input and output layers.



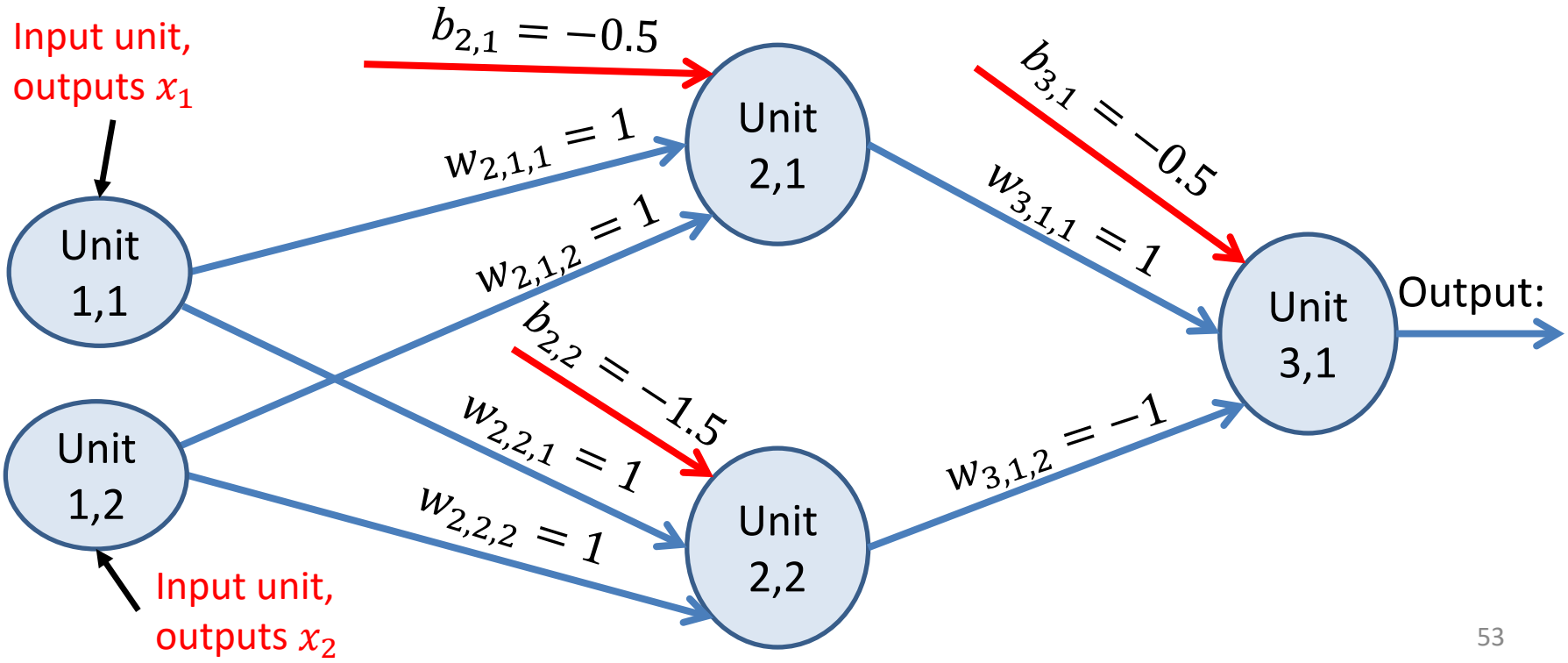
Neural Network Layers

- There is only one hidden layer in our example, containing units 2,1 and 2,2.
- Each hidden layer's inputs are outputs from the previous layer.
- Each hidden layer's outputs are inputs to the next layer.
- The first hidden layer's inputs come from the input layer.
- The last hidden layer's outputs are inputs to the output layer.



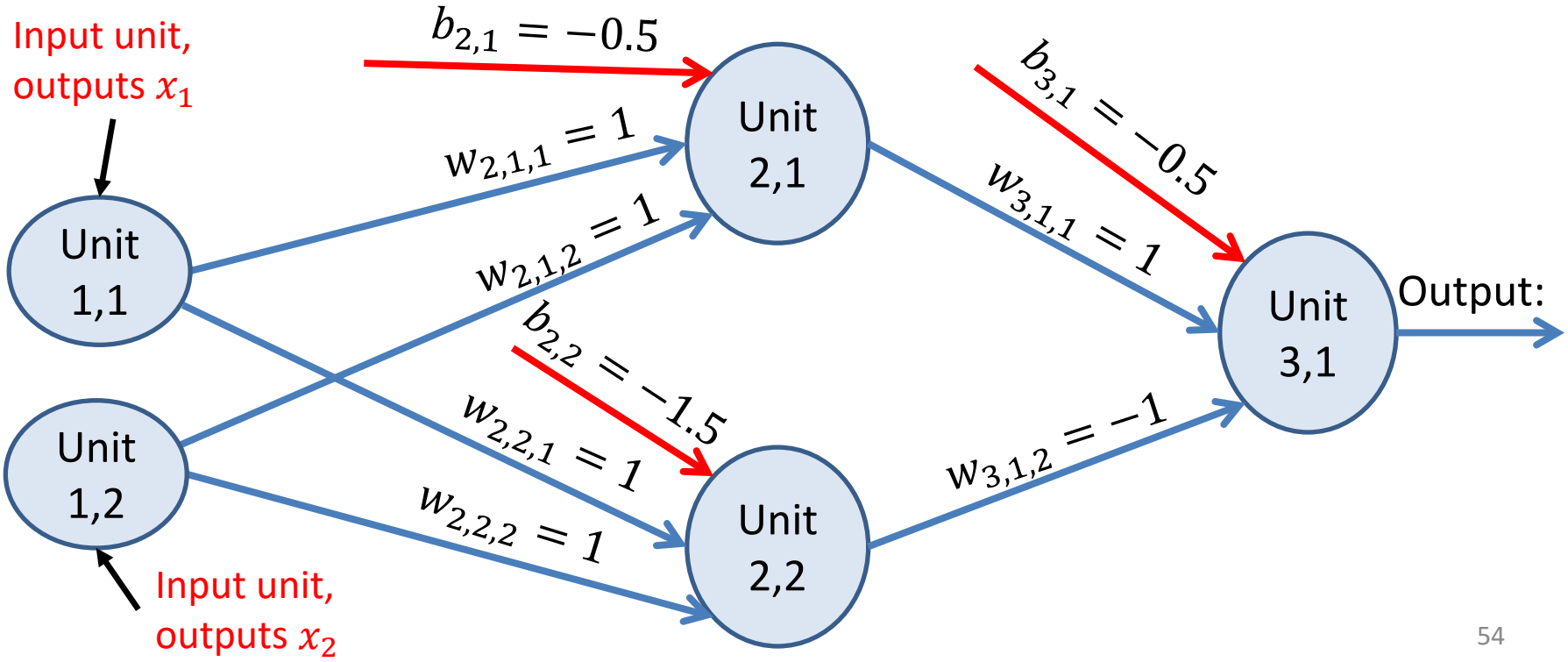
Feedforward Networks

- Feedforward networks are networks where there are no directed loops.
- If there are no loops, the output of a unit cannot (directly or indirectly) influence its input.
- There are some varieties of neural networks that are not feedforward or layered. For start, we focus on layered feedforward networks.



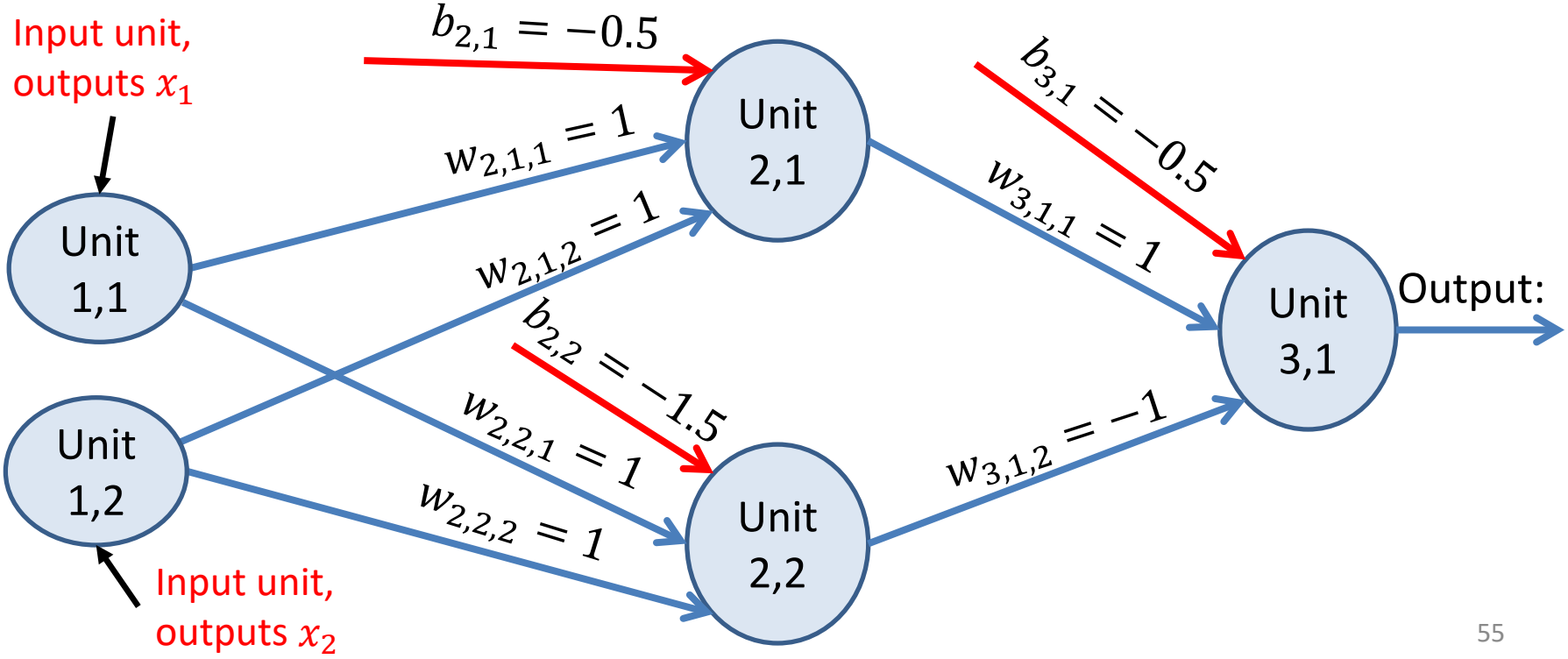
Computing the Output

- Notation: L is the number of layers. Layer 1 is the input layer, layer L is the output layer.
- The outputs of the units of layer 1 are simply the inputs to the network.
- For (layer $l = 2; l \leq L; l = l + 1$):
  - Compute the outputs of layer l, given the outputs of layer $l - 1$.



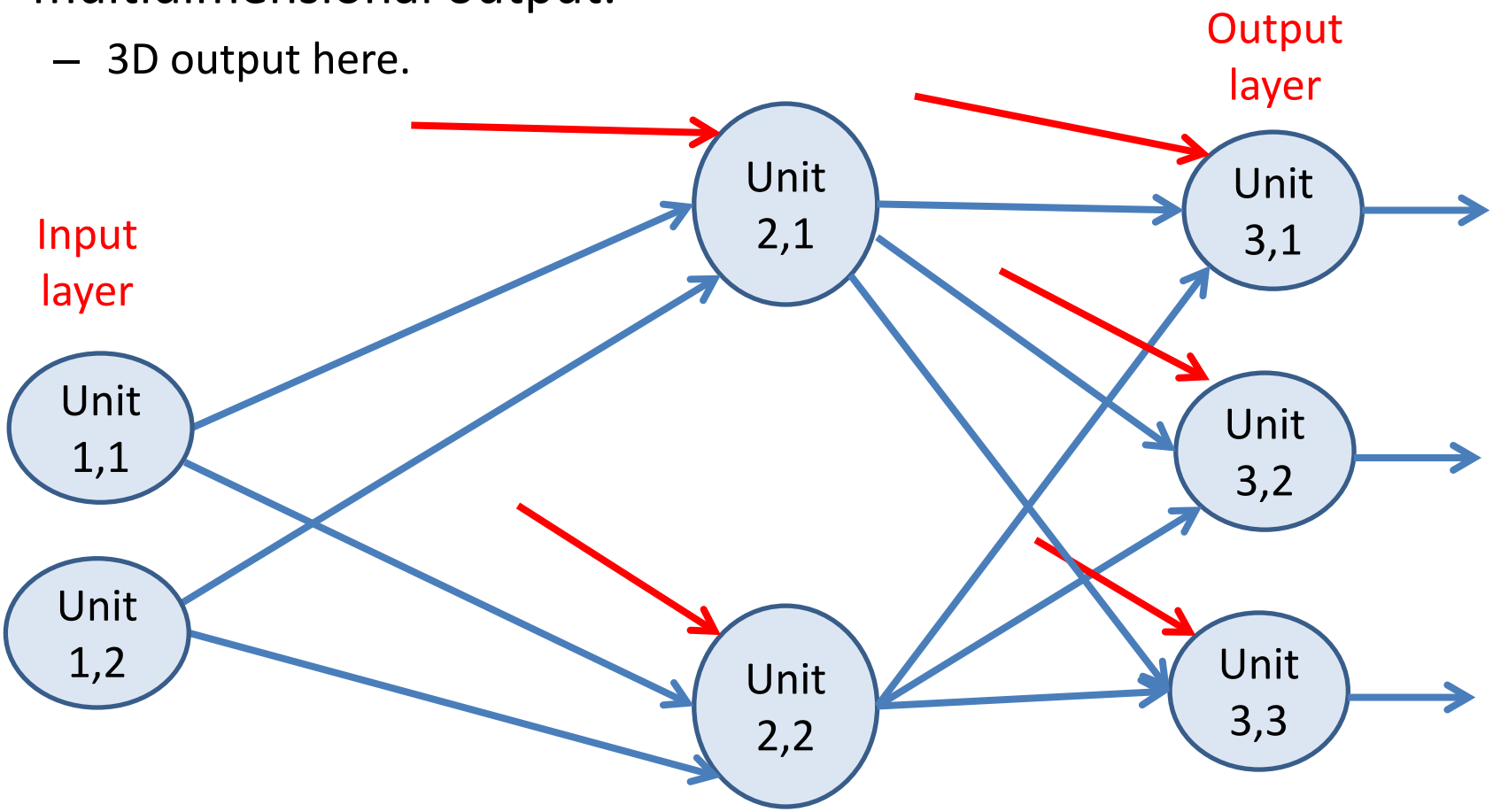
Computing the Output

- To compute the outputs of layer l (where $l > 1$), we simply need to compute the output of each perceptron belonging to layer l .
  - For each such perceptron, its inputs are coming from outputs of units at layer $l - 1$, which we have already computed.
  - Remember, we compute layer outputs in **increasing order of l**.



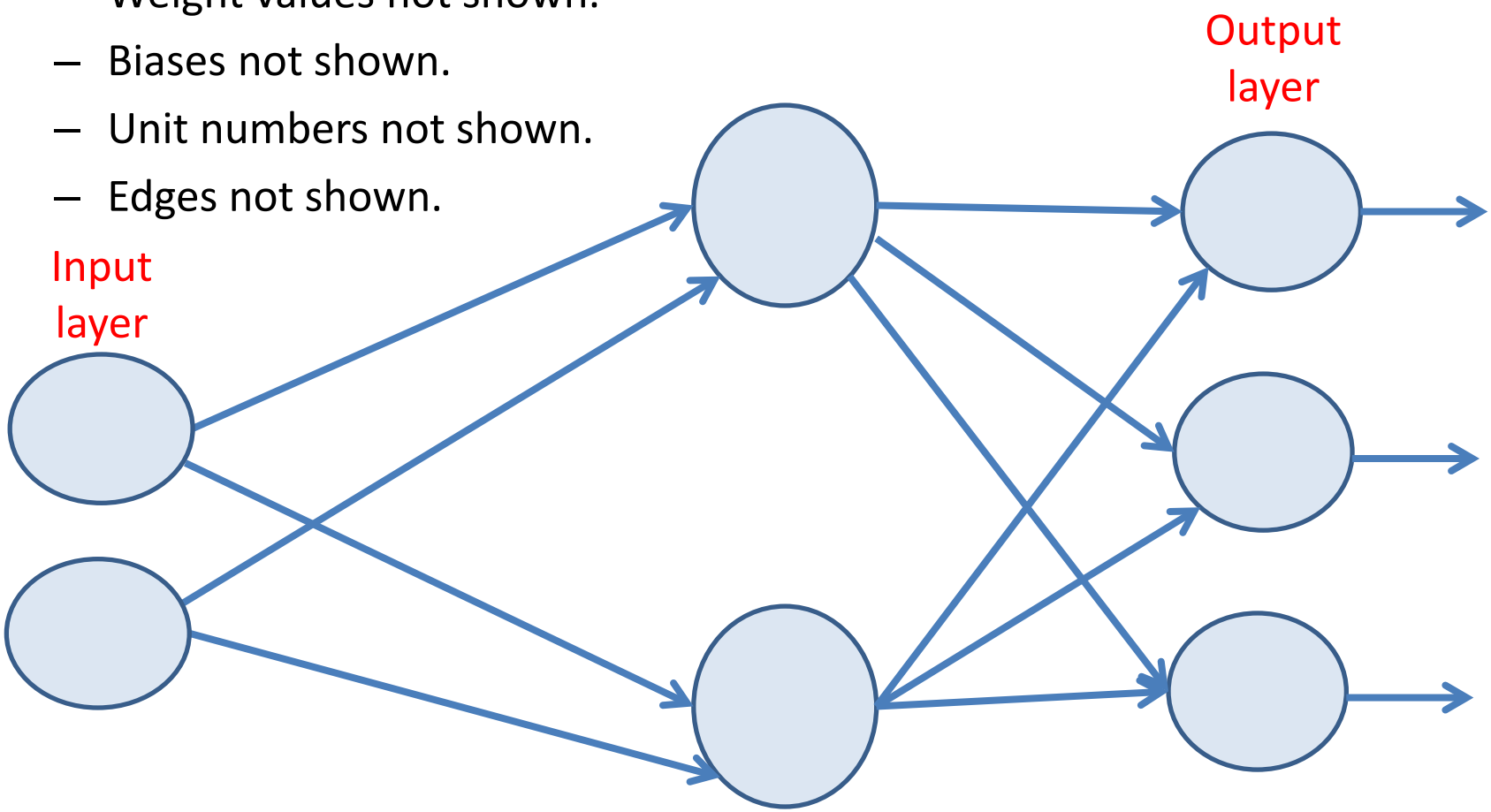
Multiple Output Units

- The output layer can have multiple units.
- This simply corresponds to the neural network producing a multidimensional output.
  - 3D output here.

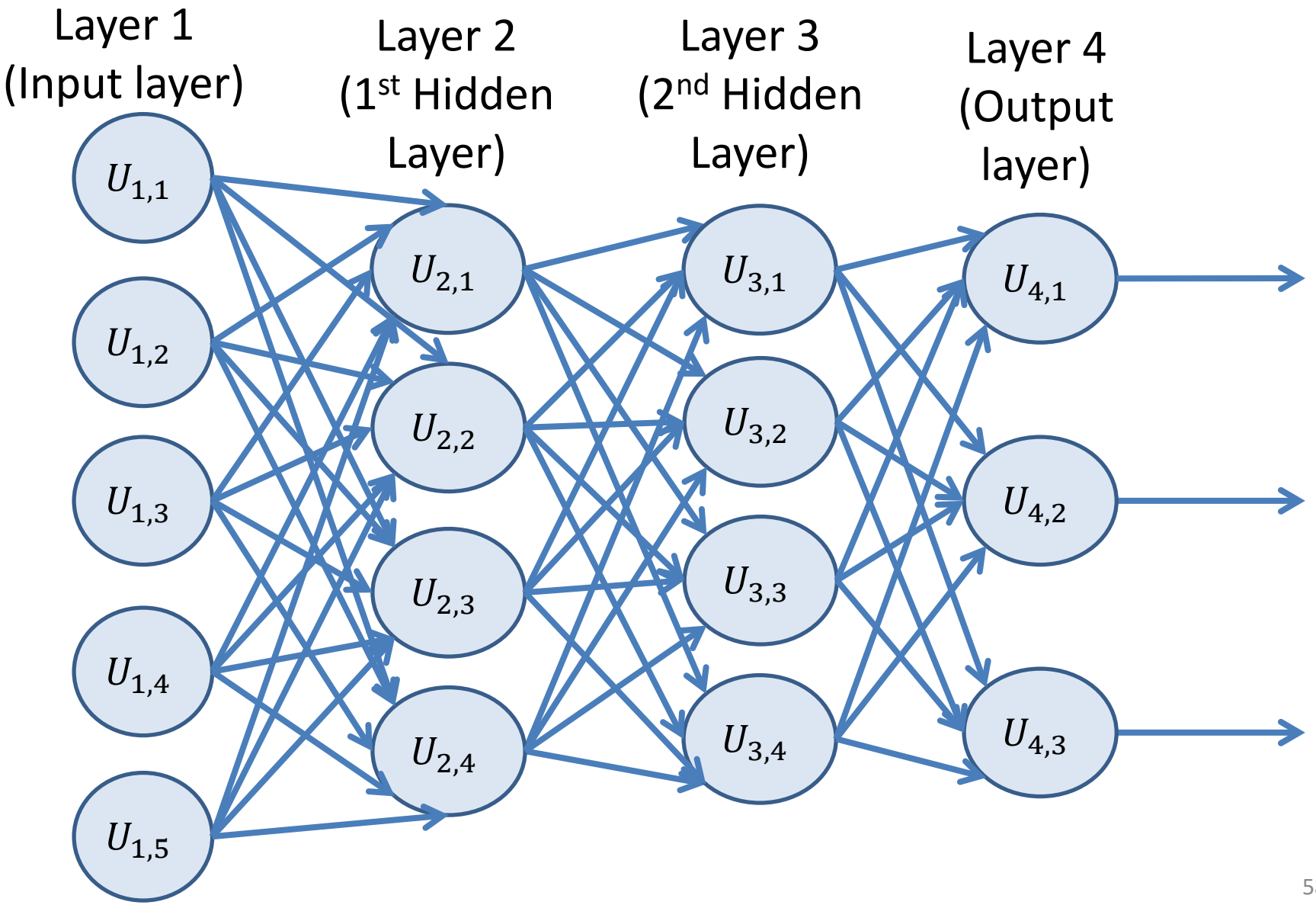


Simplified Drawings

- Drawings of larger neural networks can get very complicated.
- Various conventions can be used to simplify the picture.
  - Weight values not shown.
  - Biases not shown.
  - Unit numbers not shown.
  - Edges not shown.



A More Complicated Network



Fully Connected Layers

- An important property of a layer is how its units are connected to the previous layer.
  - Obviously, this is not applicable to the input layer, that has no previous layer.
- The most simple type (and most expensive computationally) is a fully connected layer.
  - Every unit in this layer is connected to every unit in the previous layer.
- At first, we will work with fully connected layers.
  - This will be the type that you will implement first.
- Then we will talk about other types of layers.
  - Convolutional, max-pooling, LSTM are examples of types of layers that we will study and use in later assignments.

What Neural Networks Can Compute

- An individual perceptron is a linear classifier.
  - The weights of the perceptron define a linear boundary between two classes.
- Layered feedforward neural networks with one hidden layer can compute any continuous function.
- Layered feedforward neural networks with two hidden layers can compute any mathematical function.
- This has been known for decades, and is one reason scientists have been optimistic about the potential of neural networks to model intelligent systems.
- Another reason is the analogy between neural networks and biological brains, which have been a standard of intelligence we are still trying to achieve.
- There is only one catch: How do we find the right weights?

Finding the Right Weights

- The next topic will be how to train a neural network.
- To define a neural network we need to specify two things:
  - The topology.
    - layers.
    - nodes per layer.
    - how nodes in one layer are connected to nodes in the next layers.
  - The weights of the edges.
- We typically come up with the topology manually.
  - Would be great to learn it automatically, but current methods do not work very well.
  - So, typically, topology specifications are hyperparameters.
- The goal of training is to find good values for the weights.