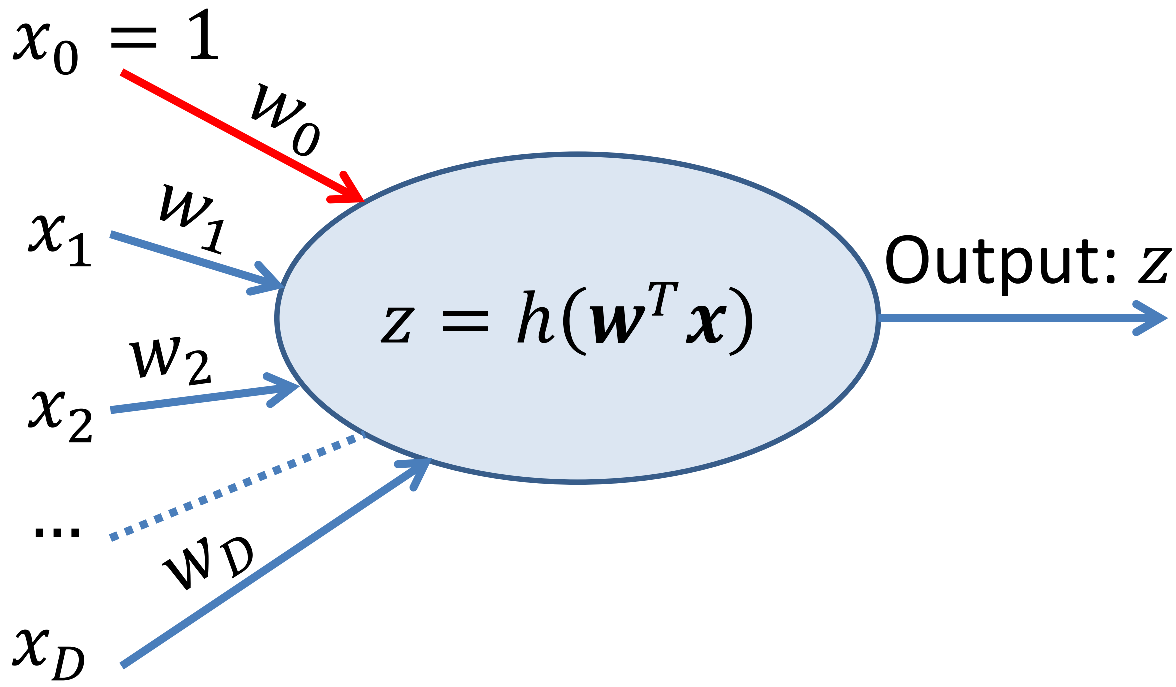
CSE 6367: Computer Vision

Convolutional Neural Networks

Slide Courtesy: Dr. Vassilis Athitsos
CSE@UTA

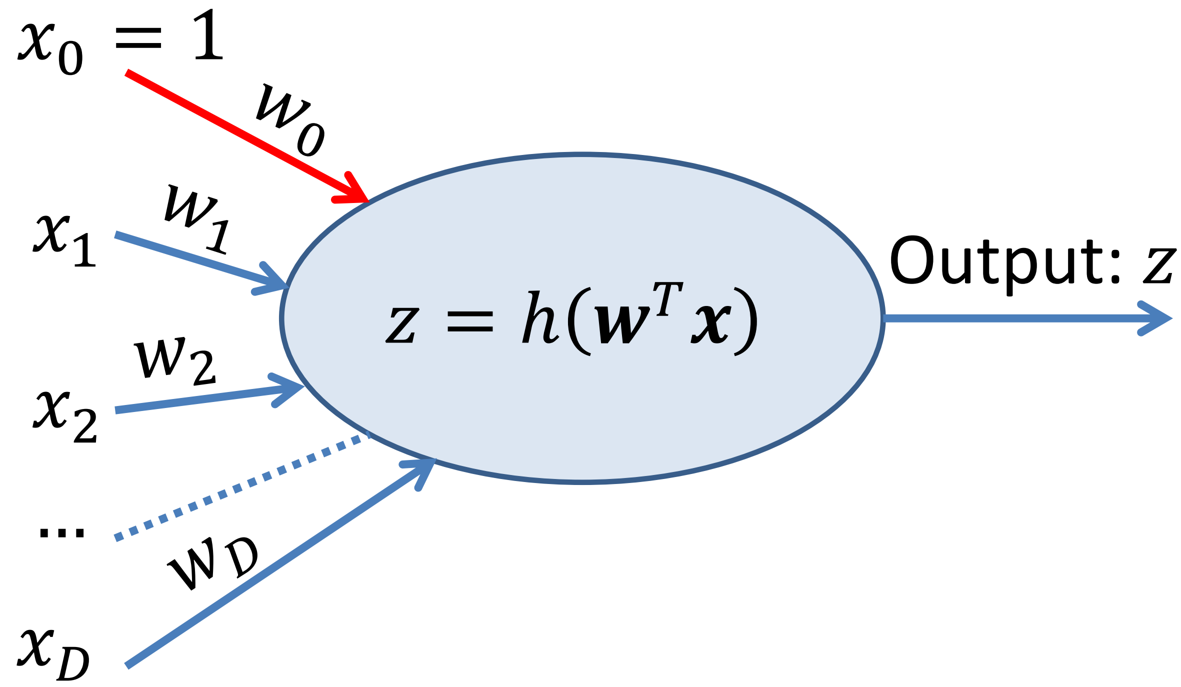
Perceptrons



- A perceptron is a function that maps D-dimensional vectors to real numbers.
- For notational convenience, we add a zero-th dimension to every input vector, that is always equal to 1.
- x_0 is called the **bias input**. It is **always equal to 1.**
- w_0 is called the **bias weight**. It is optimized during training.

$$\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \dots \\ x_D \end{bmatrix}$$

Perceptrons



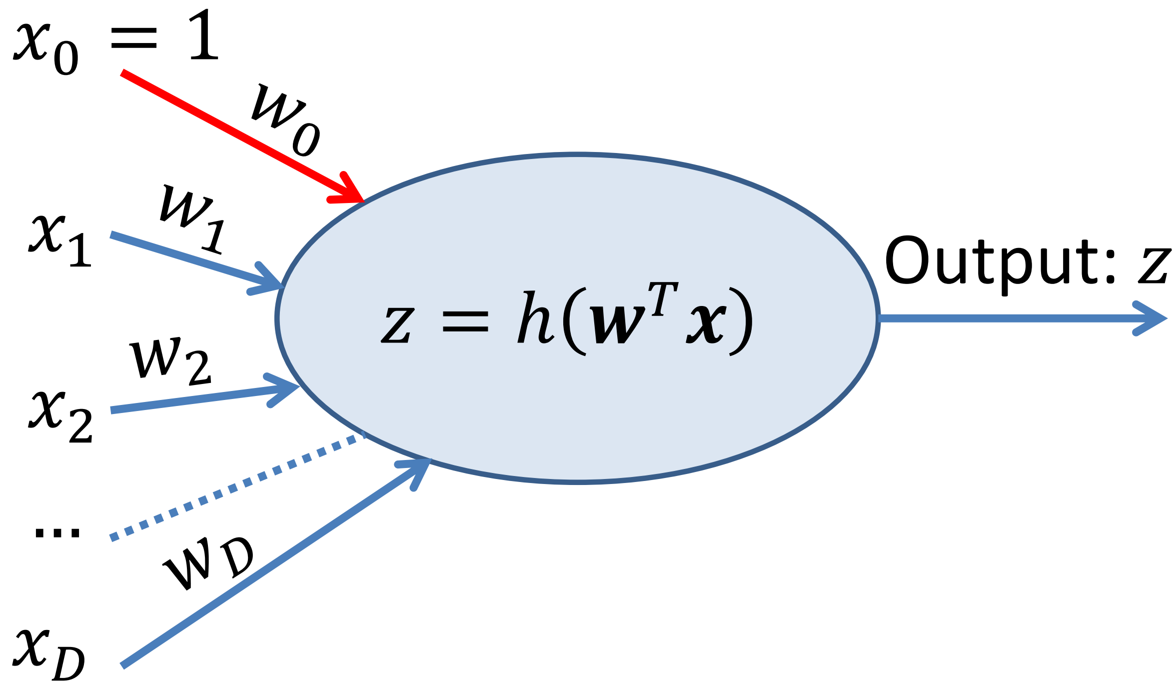
- A perceptron computes its output z in two steps:

First step: $a = \boldsymbol{w}^T\boldsymbol{x} = \sum_{i=0}^{D}(w_i x_i)$

Second step: $z = h(a)$

- In a single formula: $z = h\left(\sum_{i=0}^{D}(w_i x_i)\right)$

Perceptrons



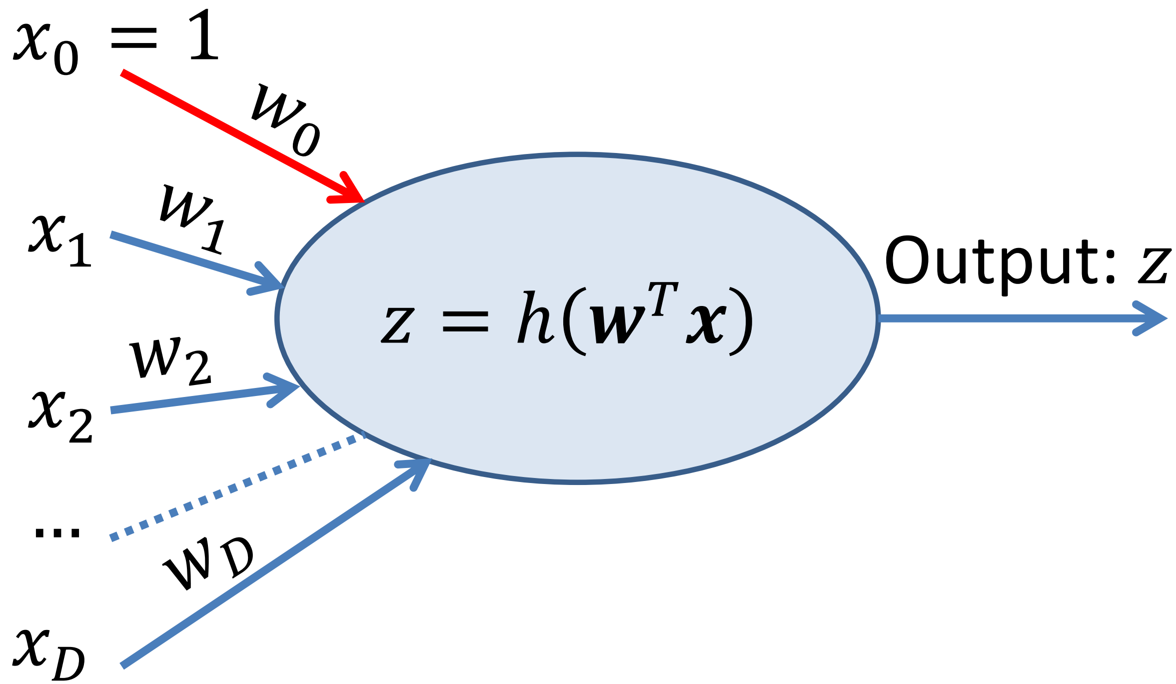
- A perceptron computes its output z in two steps:

First step: $a = \boldsymbol{w}^T \boldsymbol{x} = \sum_{i=0}^{D}(w_i x_i)$

Second step: $z = h(a)$

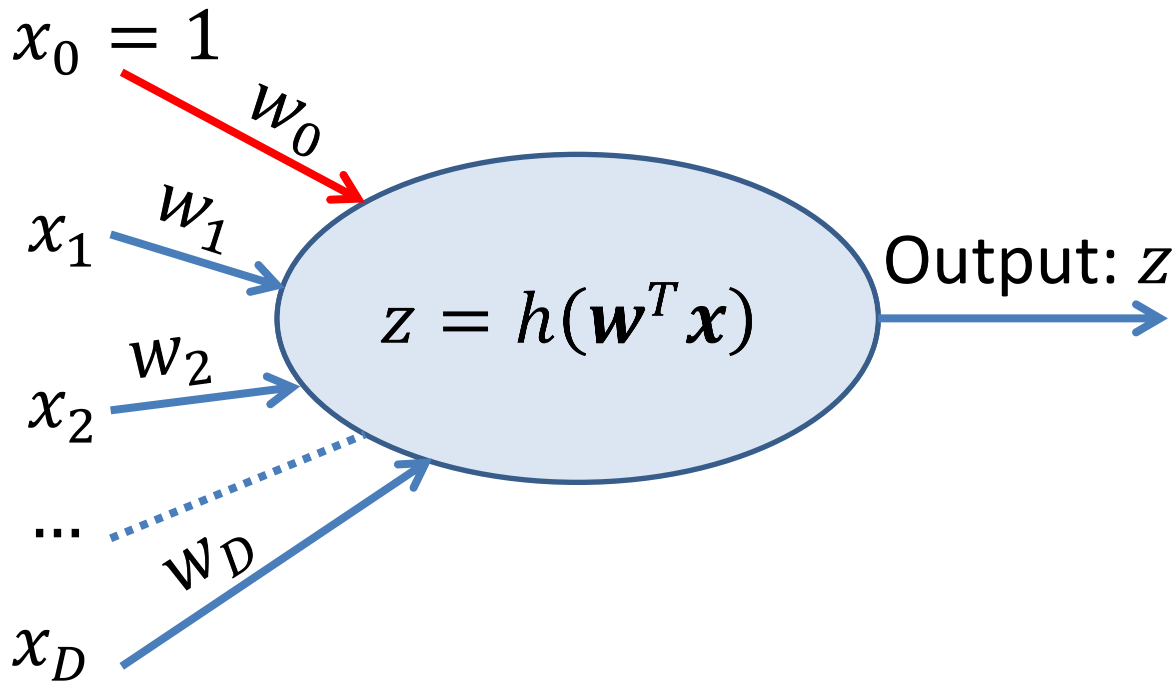
- h is called an **activation function**.
- For example, h could be the sigmoidal function $\sigma(a) = \frac{1}{1+e^{-a}}$

Perceptrons



- We have seen perceptrons before, we just did not call them perceptrons.
- For example, logistic regression produces a classifier function $\mathrm{y}(\mathbf{x}) = \sigma\left(\boldsymbol{w}^{\mathrm{T}} \varphi(\boldsymbol{x})\right)$.
- If we set $\varphi(\boldsymbol{x}) = \boldsymbol{x}$ and $h = \sigma$, then $\mathrm{y}(\mathbf{x})$ is a perceptron.

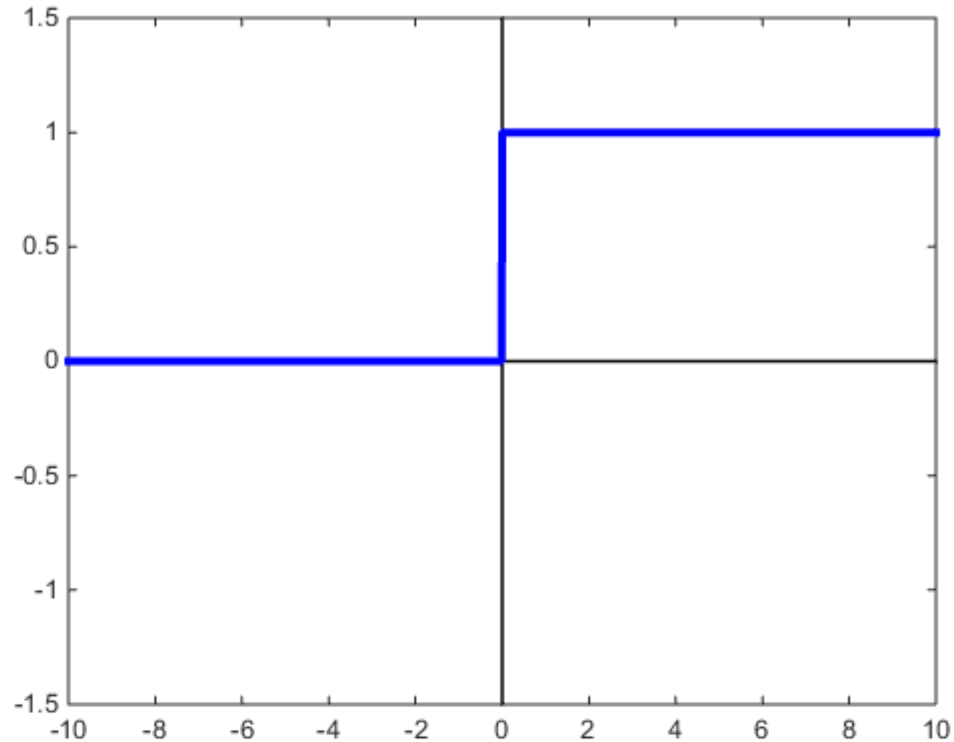
Perceptrons and Neurons



- 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.

Activation Functions

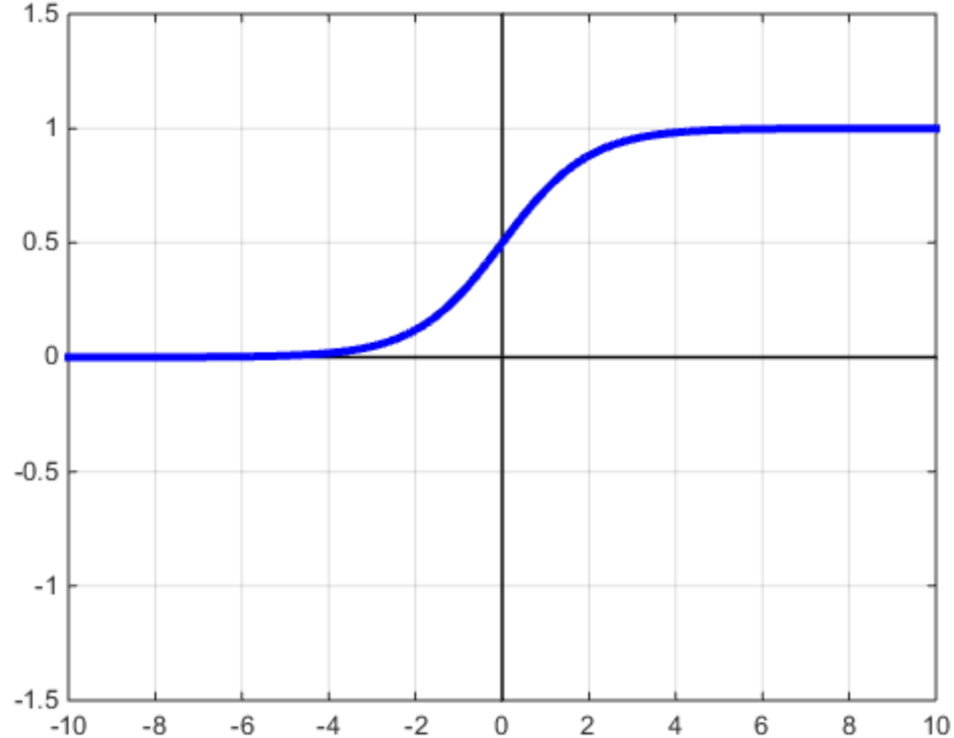
- A perceptron produces output $z = h(\boldsymbol{w}^T \boldsymbol{x})$.
- One choice for the activation function h: the **step function**.



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

- The step function is useful for providing some intuitive examples.
- It is not useful for actual real-world systems.
  – Reason: it is not differentiable, it does not allow optimization via gradient descent.

Activation Functions



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

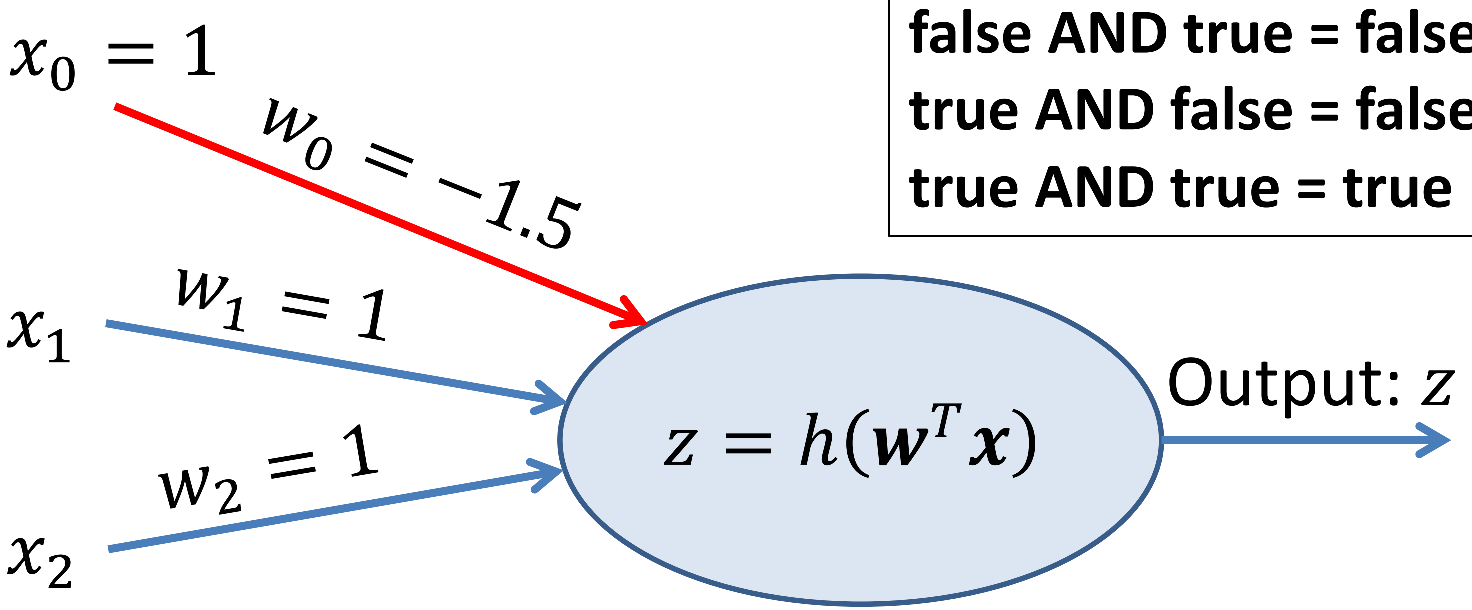
$$\sigma(a) = \frac{1}{1+e^{-a}}$$

- The sigmoidal is often used in real-world systems.
- It is a differentiable function, it allows use of gradient descent.

Example: The AND Perceptron

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

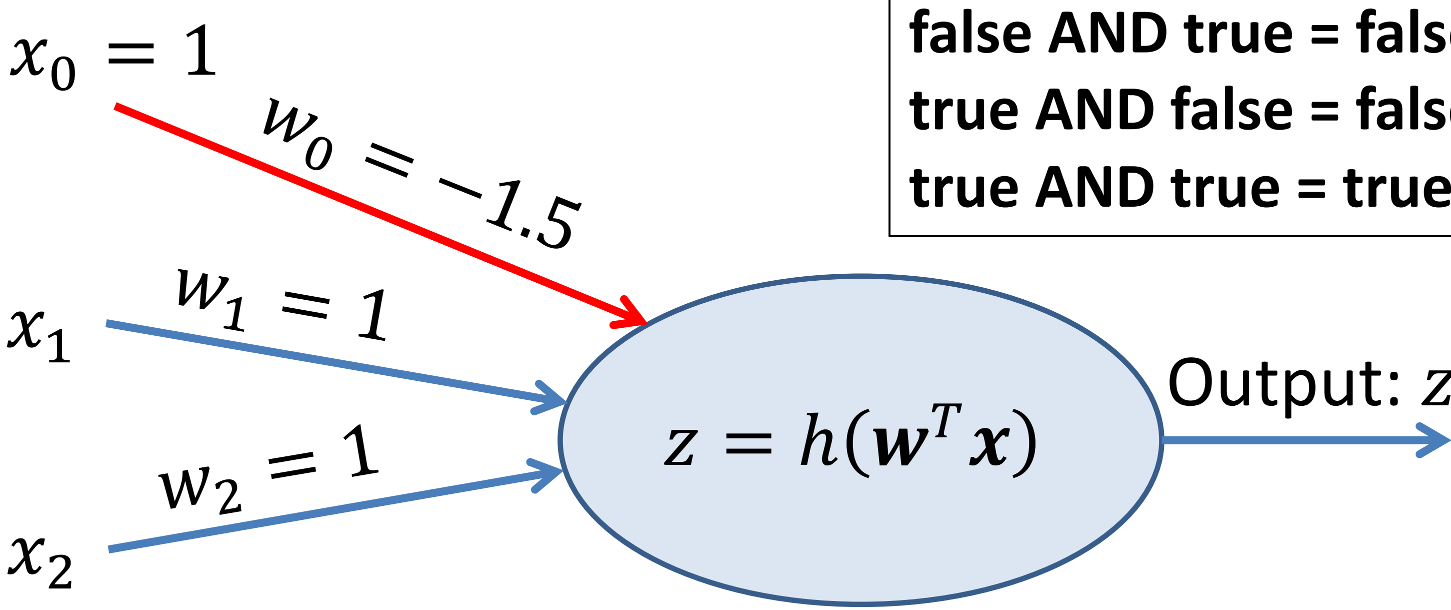
false AND false = false
false AND true = false
true AND false = false
true AND true = true



Example: The AND Perceptron

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

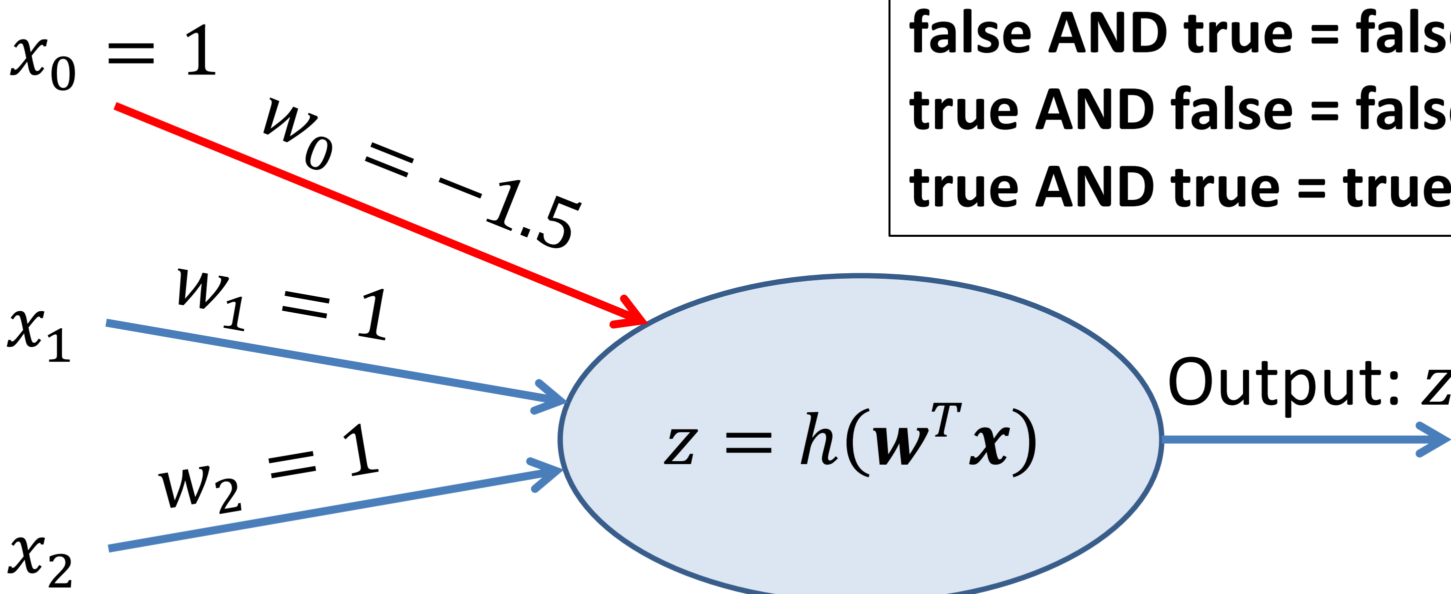
false AND false = false
false AND true = false
true AND false = false
true AND true = true



Example: The AND Perceptron

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

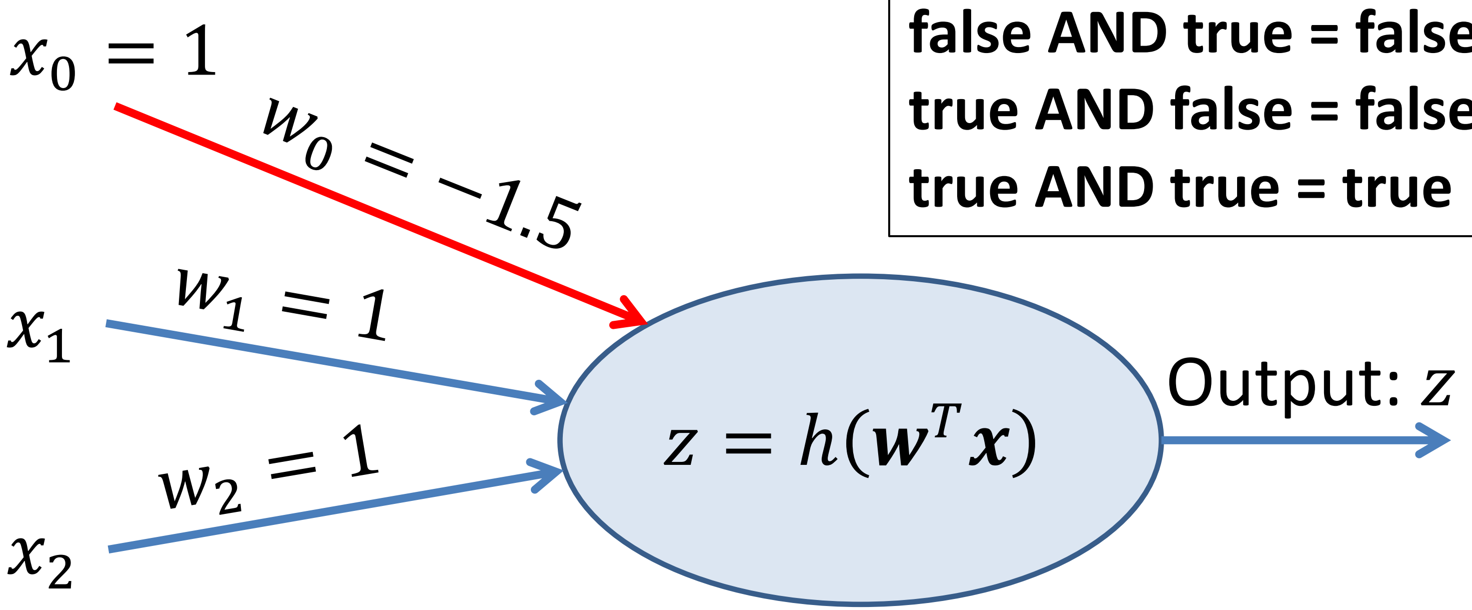
false AND false = false
false AND true = false
true AND false = false
true AND true = true



Example: The AND Perceptron

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

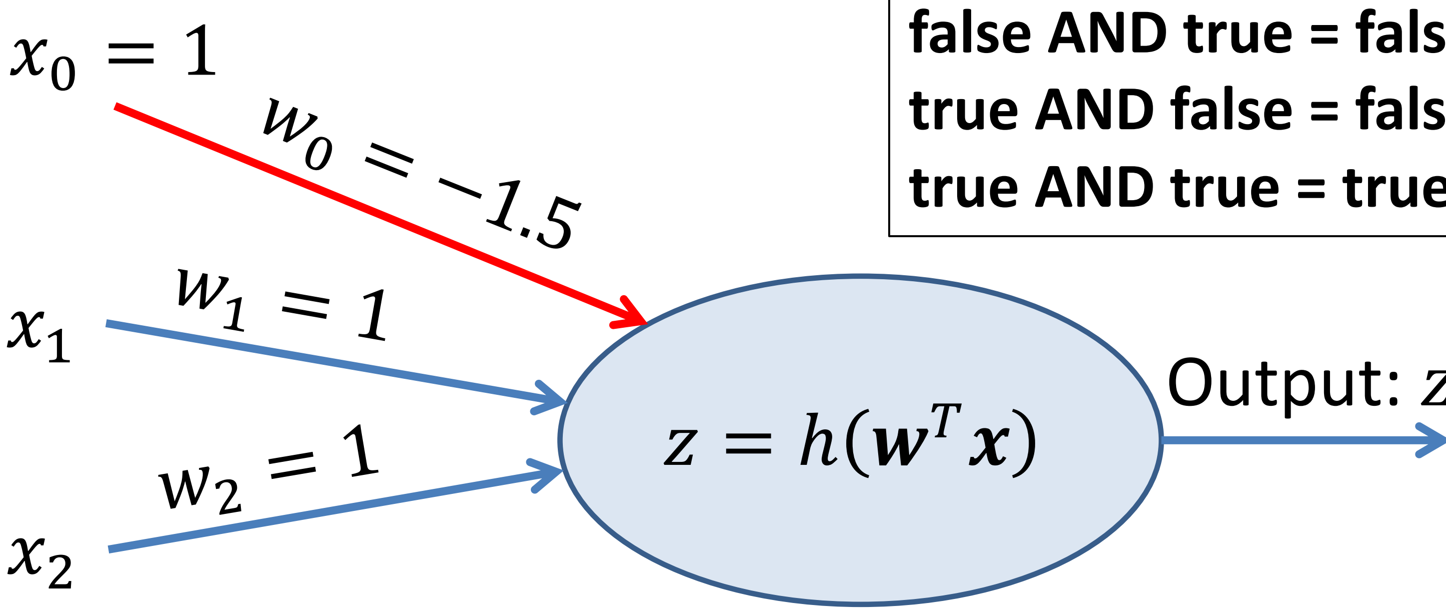
false AND false = false
false AND true = false
true AND false = false
true AND true = true



Example: The AND Perceptron

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

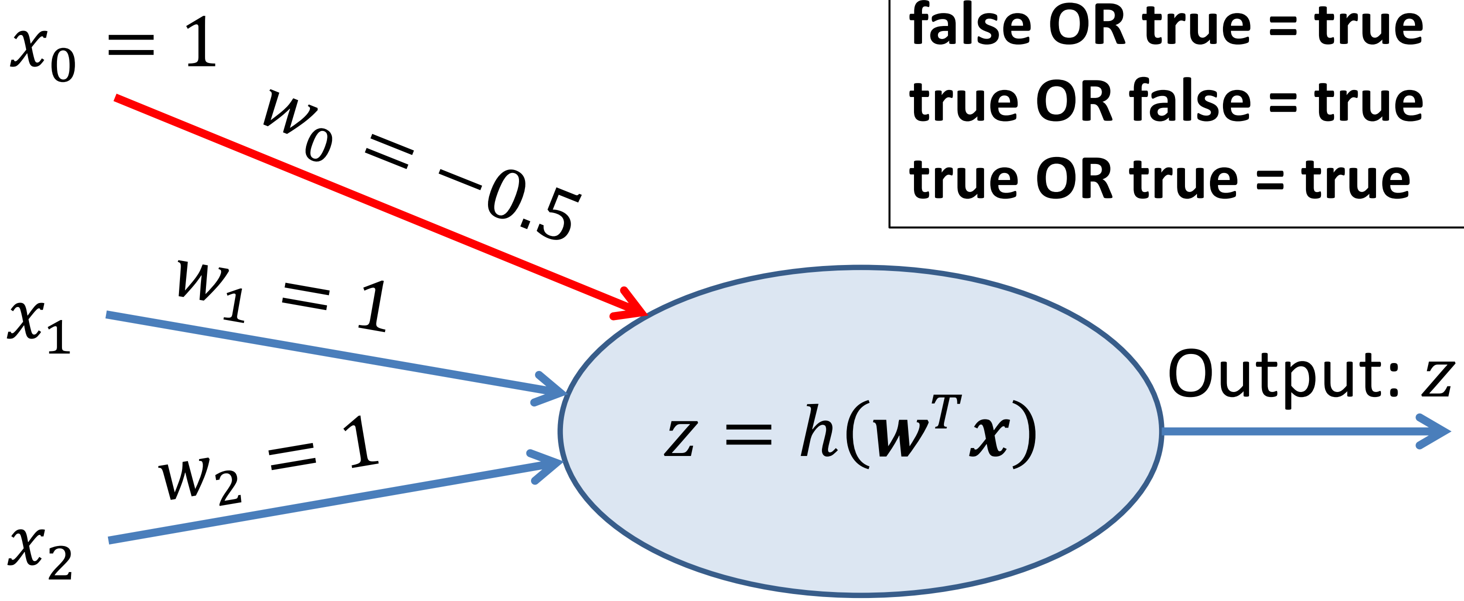
false AND false = false
false AND true = false
true AND false = false
true AND true = true



Example: The OR Perceptron

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

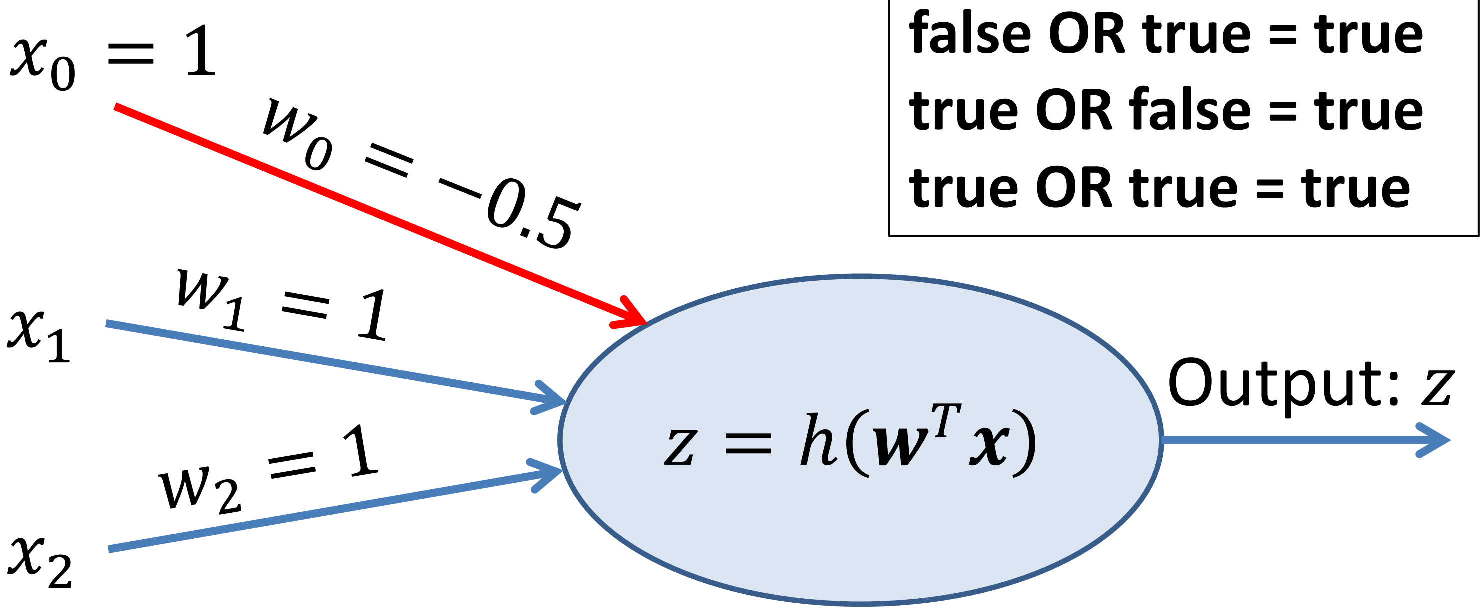
false OR false = false
false OR true = true
true OR false = true
true OR true = true



Example: The OR Perceptron

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

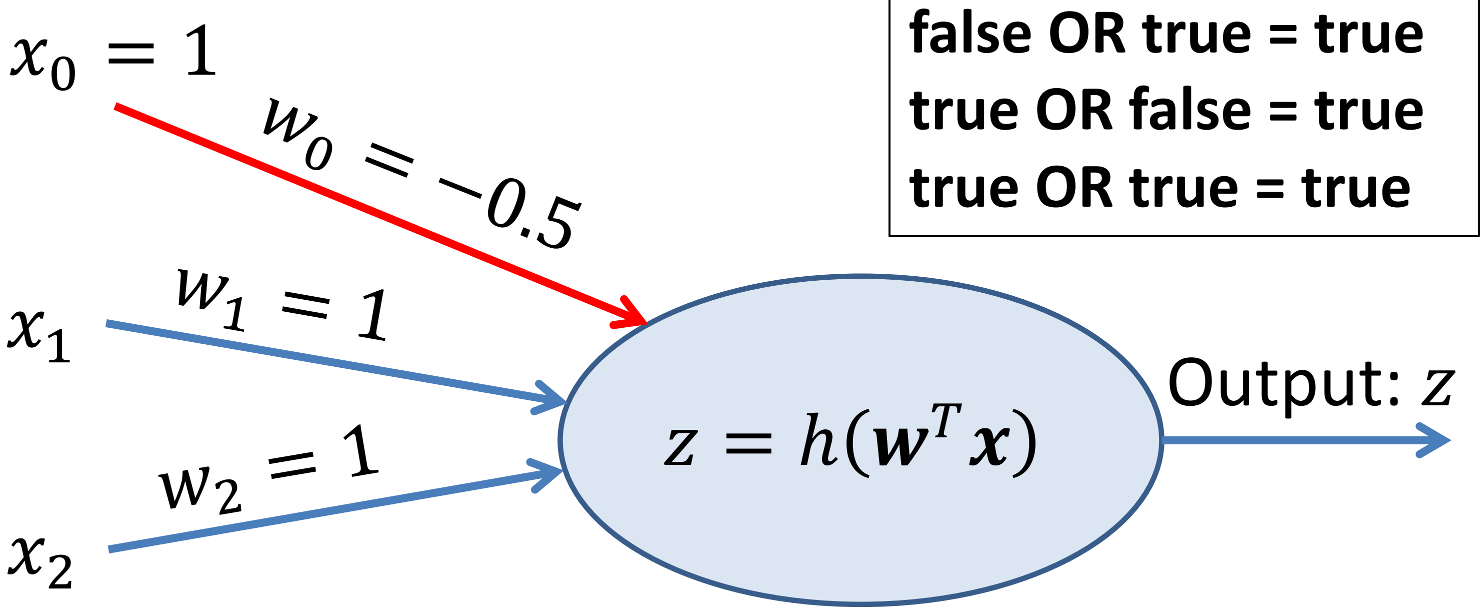
false OR false = false
false OR true = true
true OR false = true
true OR true = true



Example: The OR Perceptron

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

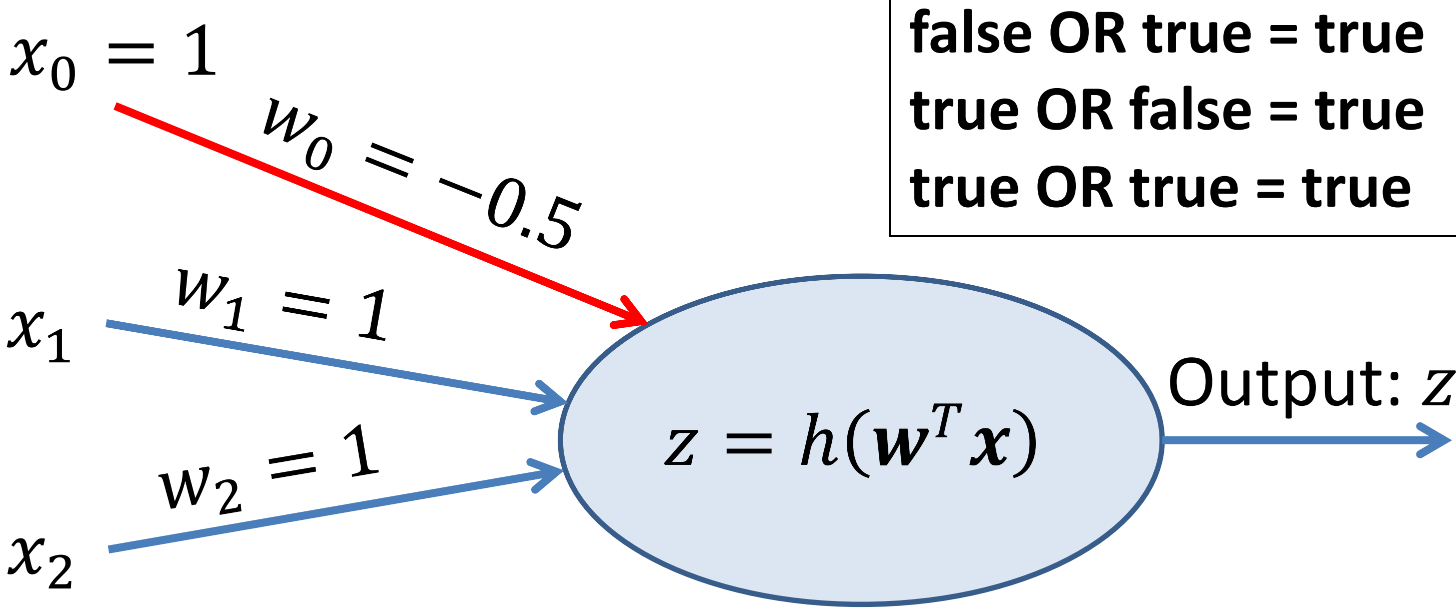
false OR false = false
false OR true = true
true OR false = true
true OR true = true



Example: The OR Perceptron

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

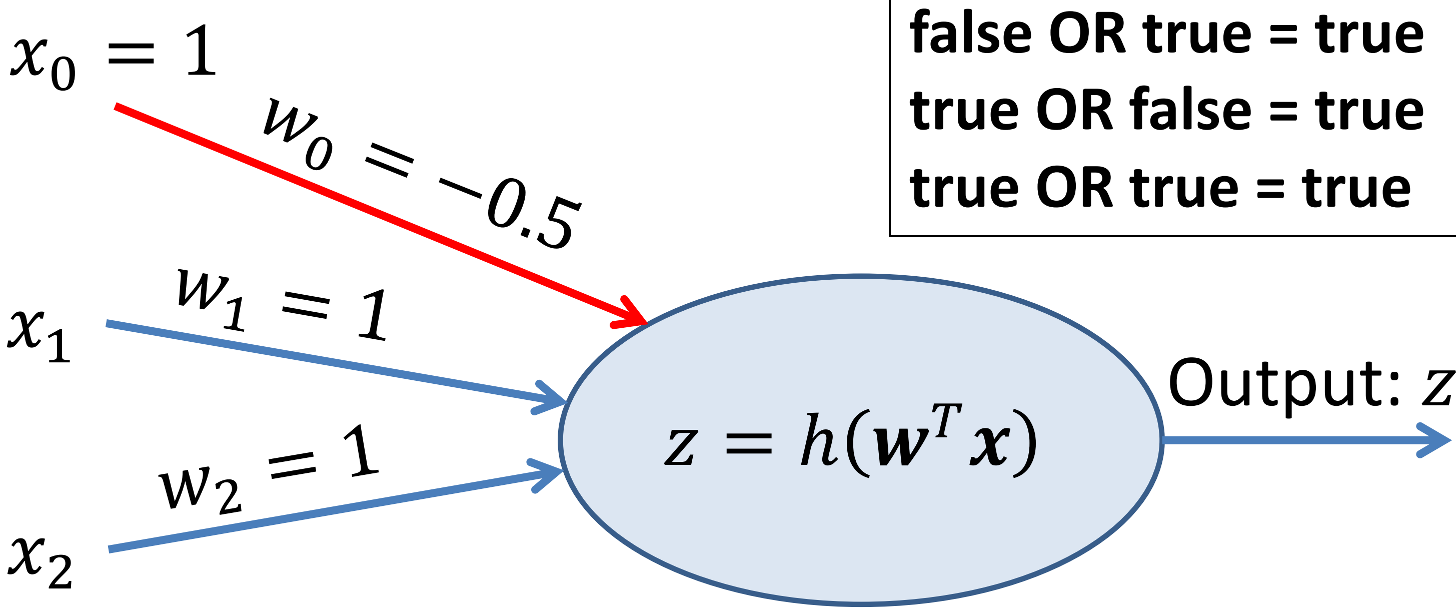
false OR false = false
false OR true = true
true OR false = true
true OR true = true



Example: The OR Perceptron

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

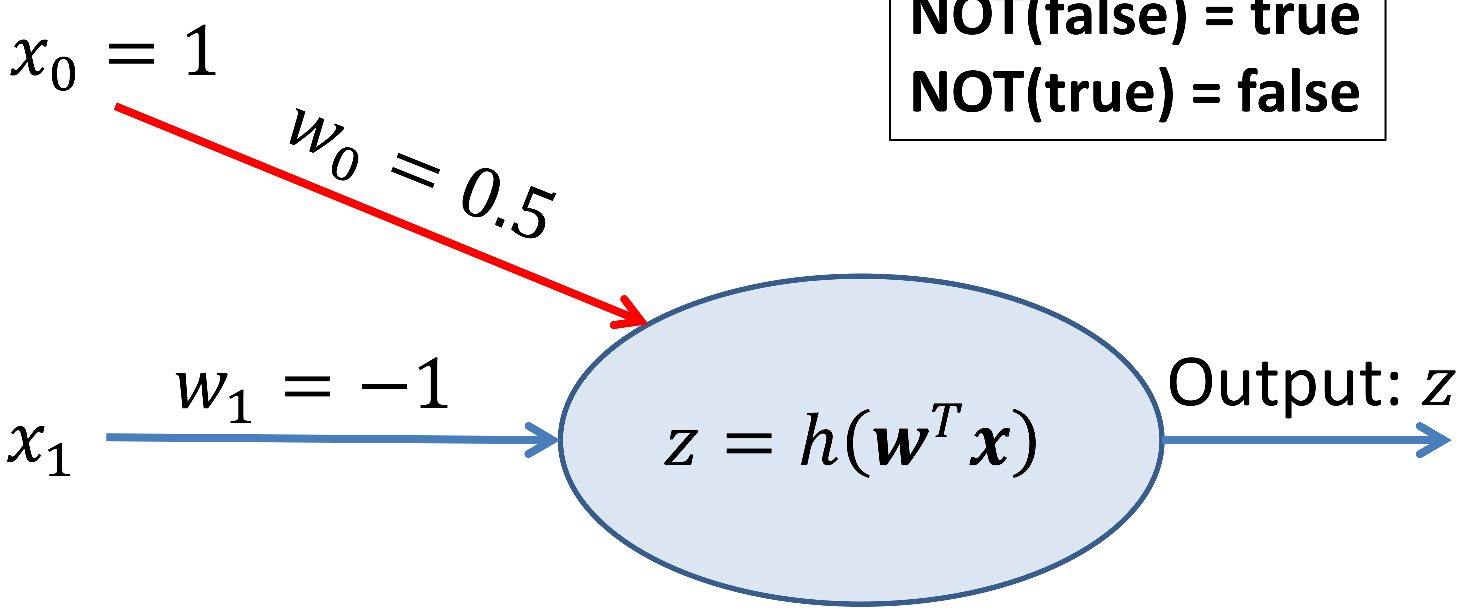
false OR false = false
false OR true = true
true OR false = true
true OR true = true



Example: The NOT Perceptron

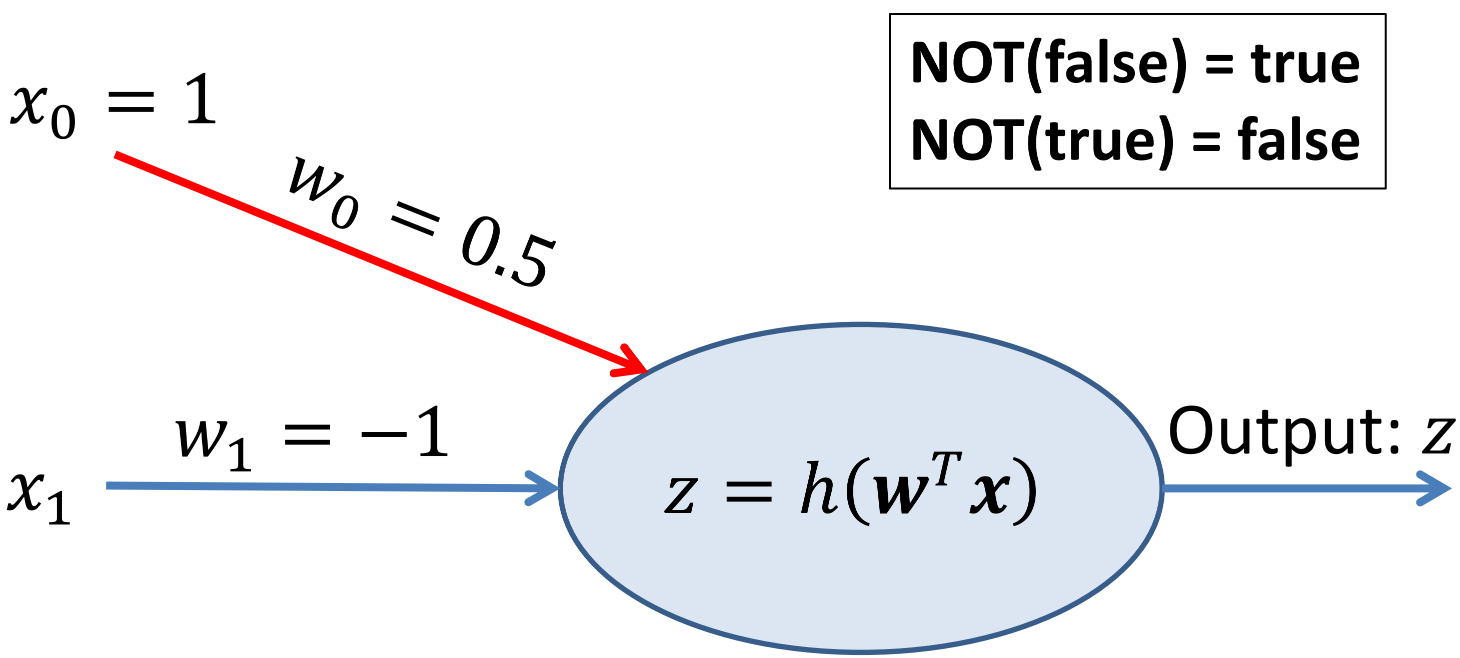
- Suppose we use the **step function** for activation.
- Suppose boolean value **false** is represented as number 0.
- Suppose boolean value tru**e** is represented as number 1.
- Then, the perceptron below computes the boolean NOT function:

NOT(false) = true
NOT(true) = false



Example: The NOT Perceptron

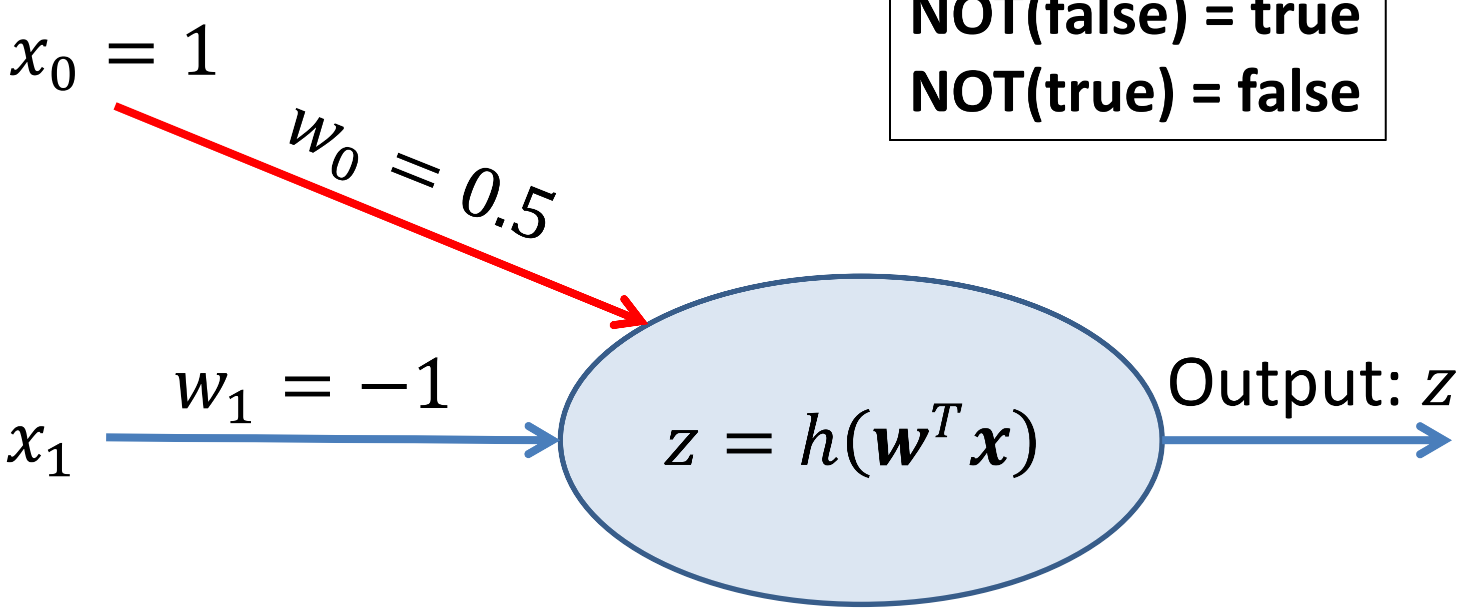
- Verification: If $x_1 = 0$:
  - $\boldsymbol{w}^T\boldsymbol{x} = 0.5 - 1 * 0 = 0.5.$
  - $h(\boldsymbol{w}^T\boldsymbol{x}) = \mathrm{h}(0.5) = 1.$
- Corresponds to case **NOT(false) = true**.



Example: The NOT Perceptron

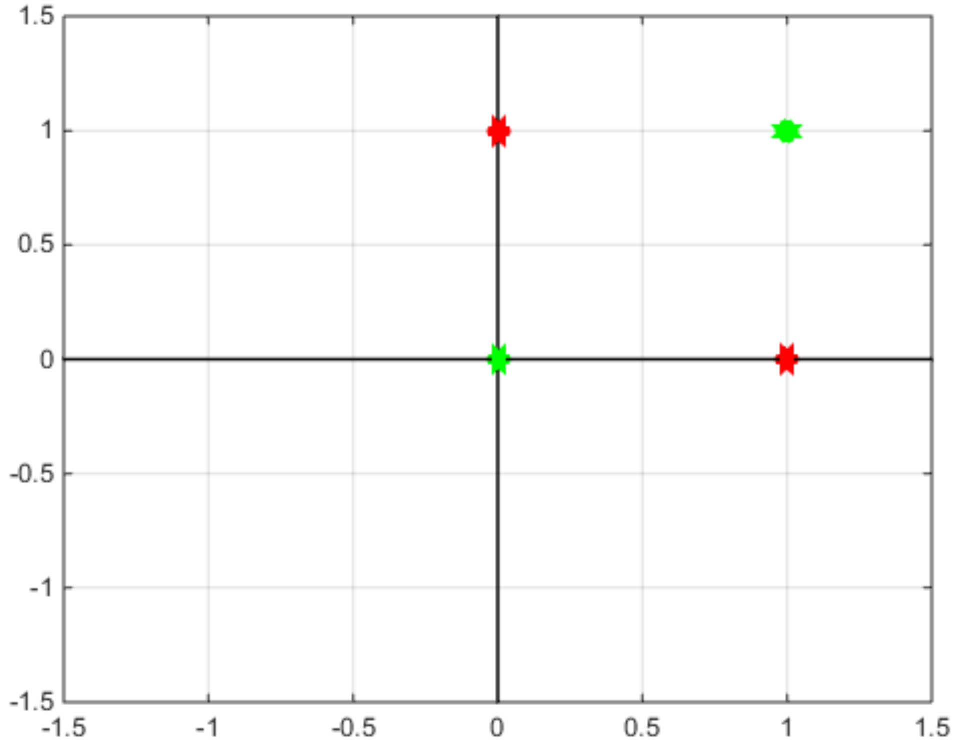
- Verification: If $x_1 = 1$:
  - $\boldsymbol{w}^T\boldsymbol{x} = 0.5 - 1 * 1 = -0.5.$
  - $h(\boldsymbol{w}^T\boldsymbol{x}) = \mathrm{h}(-0.5) = 0.$
- Corresponds to case **NOT(true) = false**.

NOT(false) = true
NOT(true) = false



The XOR Function

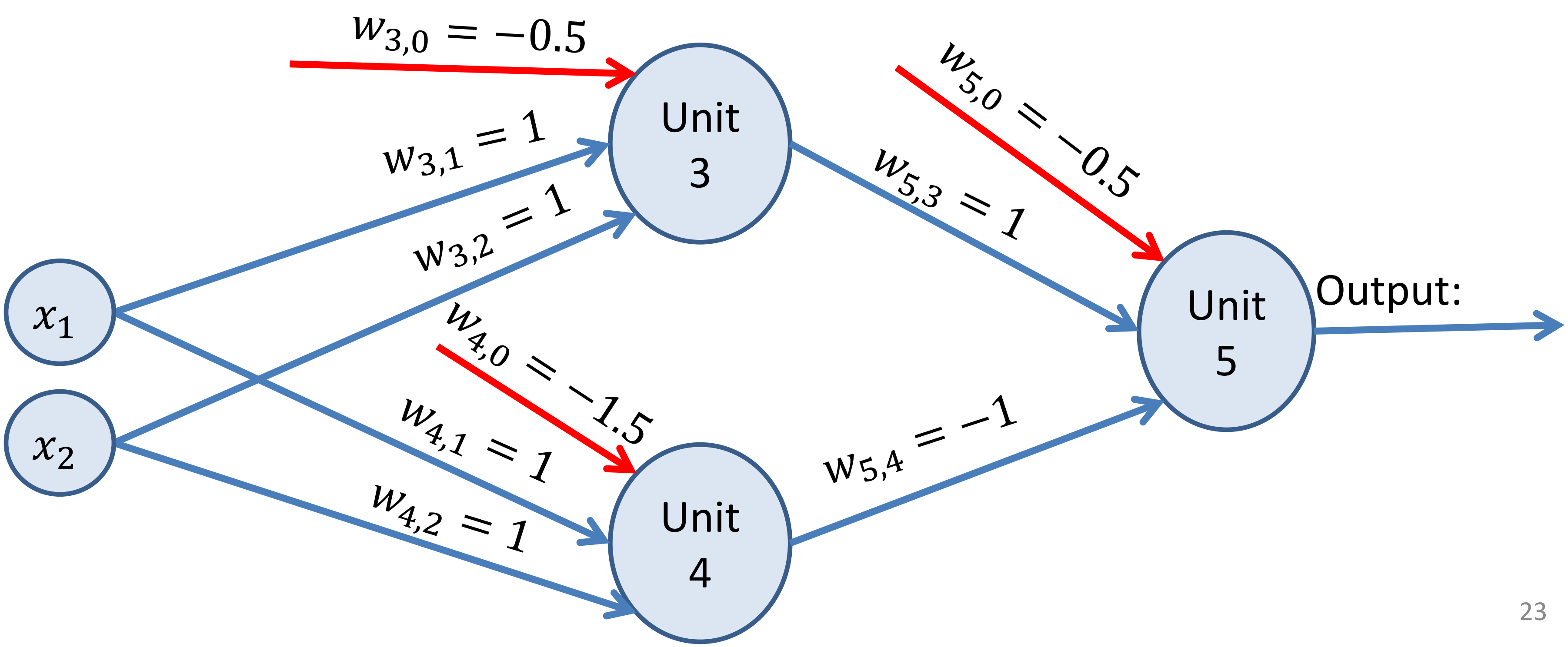
false XOR false = false
false XOR true = true
true XOR false = true
true XOR true = false



- As before, we represent **false** with 0 and **true** with 1.
- The figure shows the four input points of the XOR function.
  - green corresponds to output value **true**.
  - red corresponds to output value **false**.
- The two classes (true and false) are not linearly separable.
- Therefore, no perceptron can compute the XOR function.

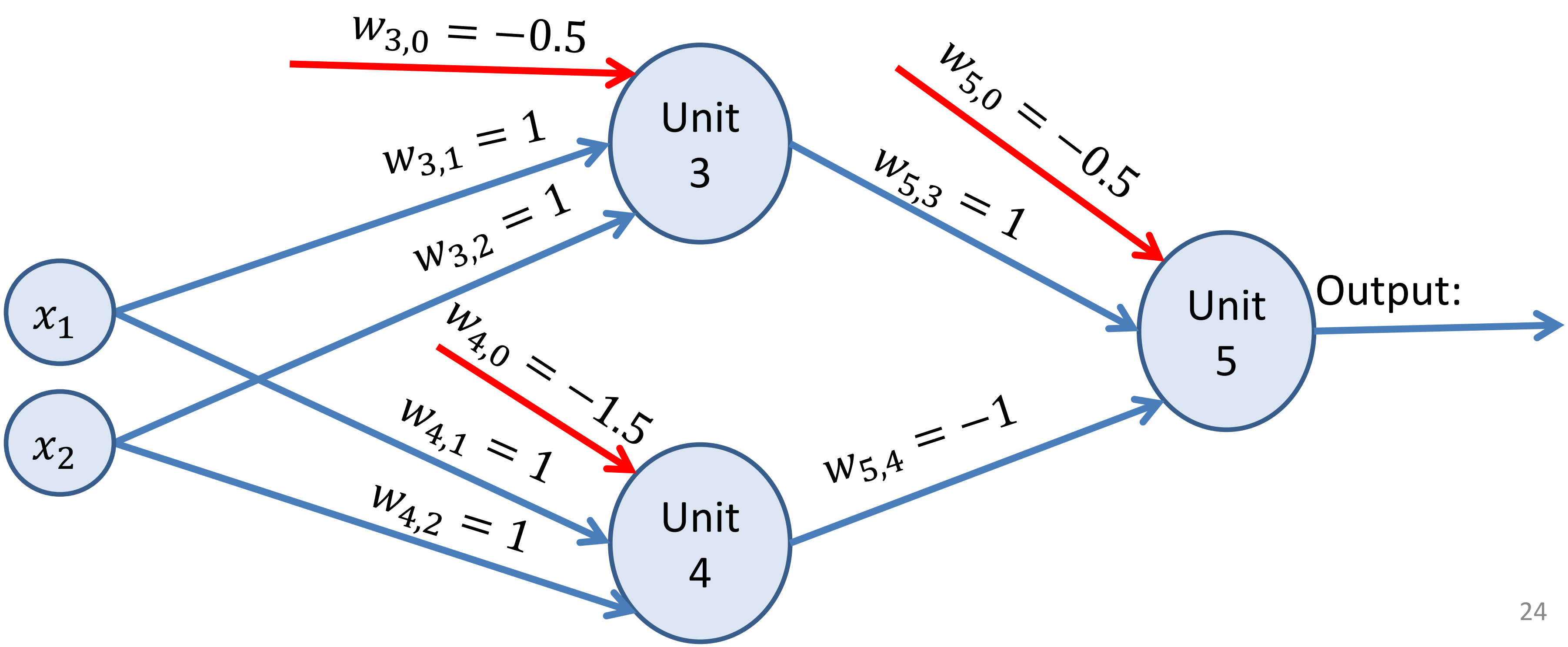
Our First Neural Network: XOR

- 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.



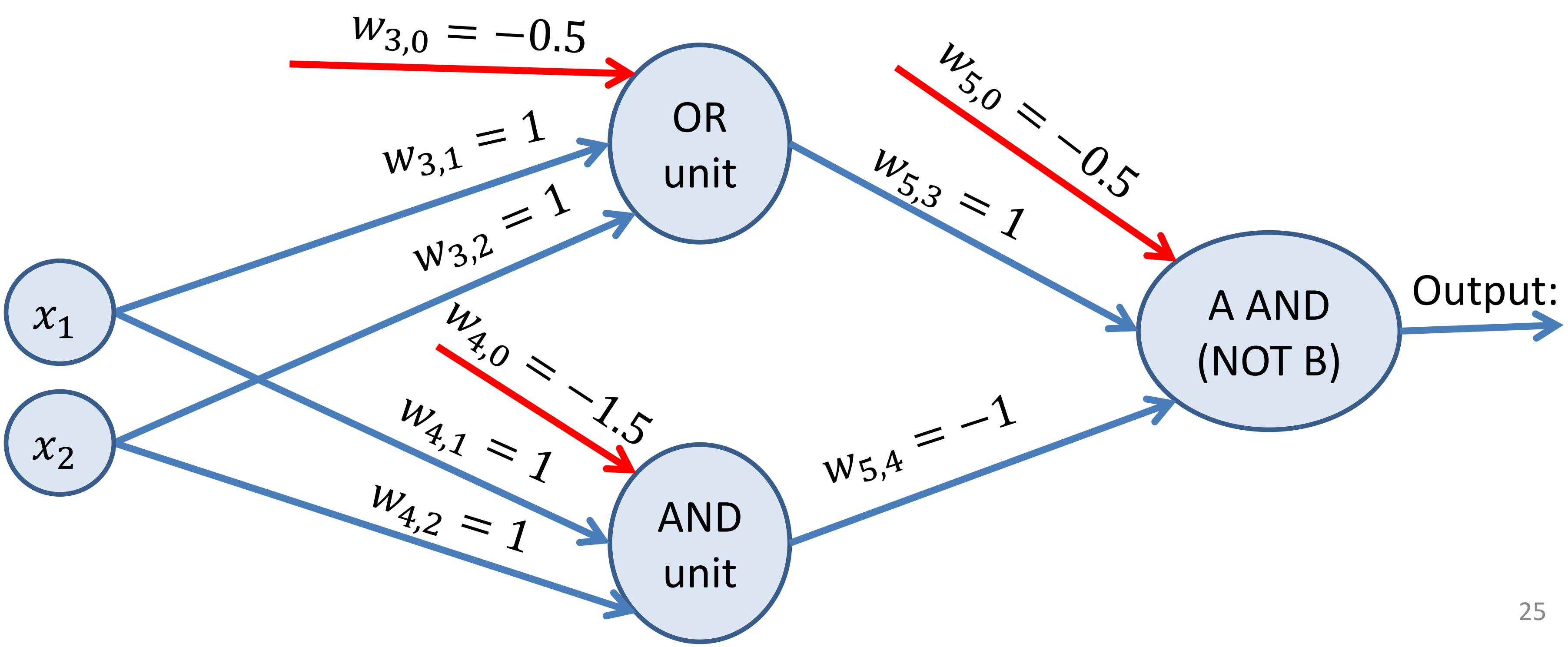
Our First Neural Network: XOR

- To simplify the picture, we do not show the bias input anymore.
  - We just show the bias weights $w_{j,0}$.
- Besides the bias input, there are two inputs: x_1, x_2.



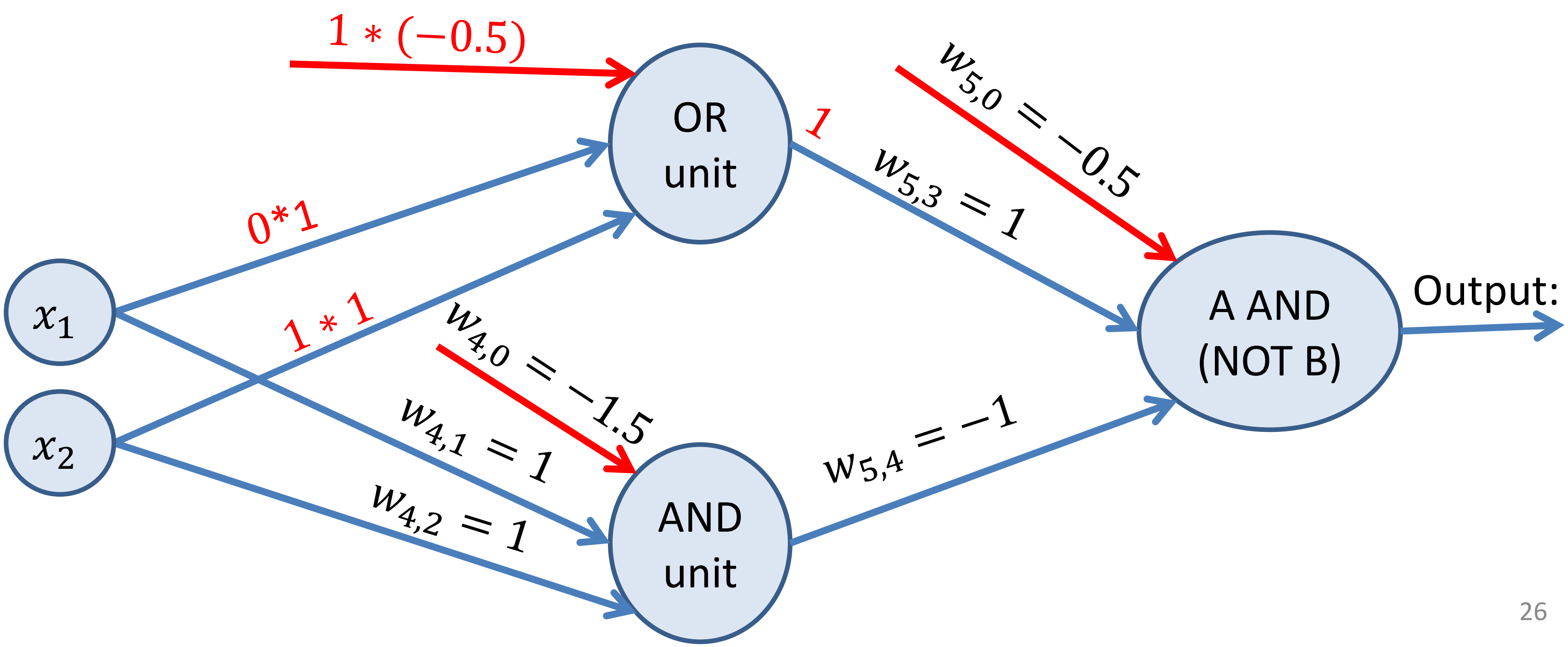
Our First Neural Network: XOR

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



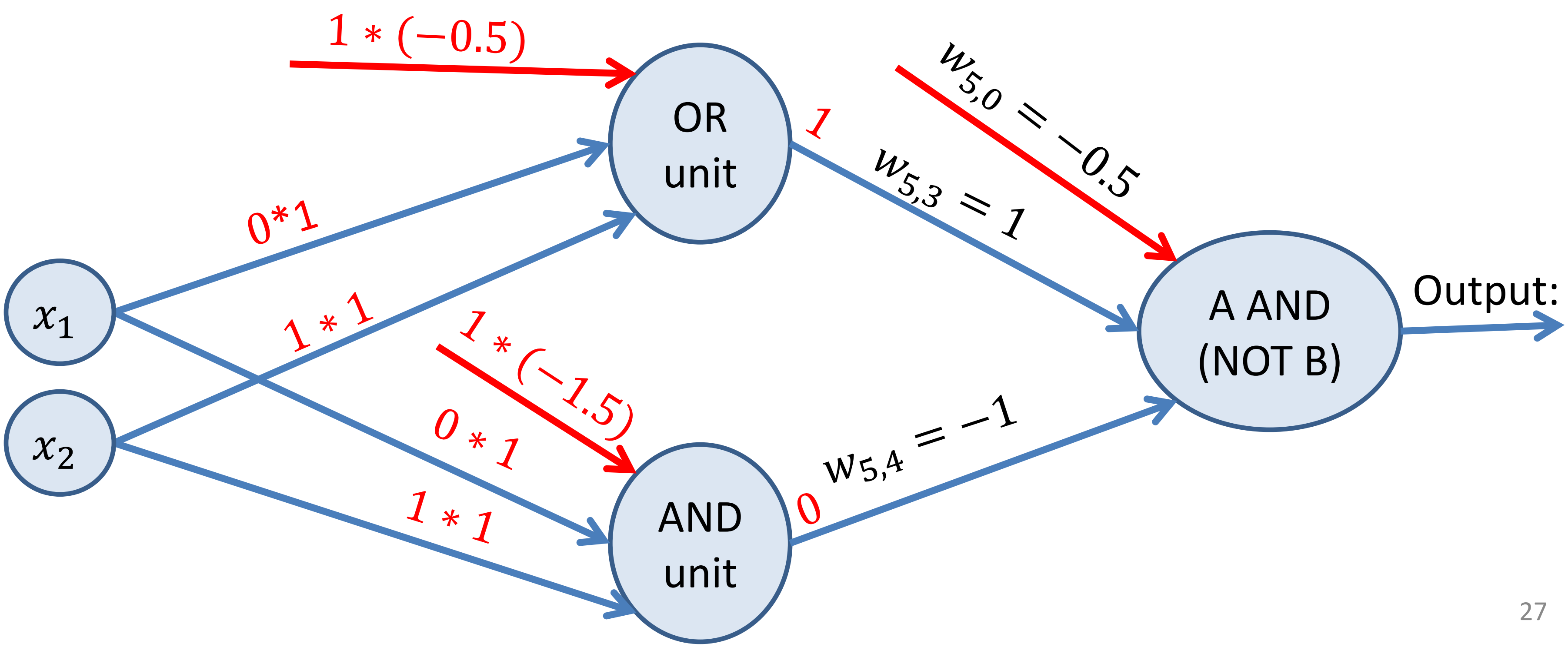
Computing the Output: An Example

- Suppose that $x_1 = 0, x_2 = 1$ (corresponding to **false** XOR **true**).
- For the OR unit:
  - The dot product is: $1 * (-0.5) + 0 * 1 + 1 * 1 = 0.5$.
  - The activation function (assuming a step function) outputs 1.



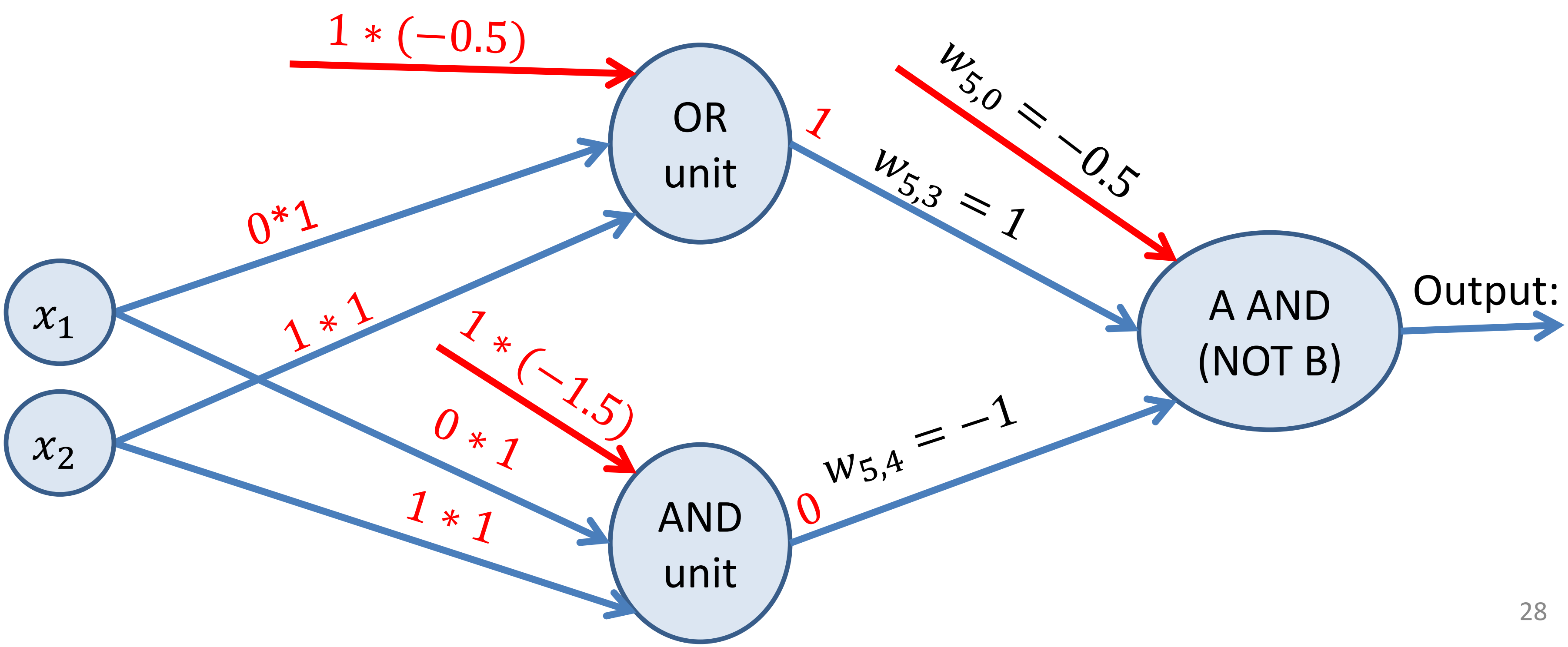
Computing the Output: An Example

- Suppose that $x_1 = 0, x_2 = 1$ (corresponding to **false** XOR **true**).
- For the AND unit:
  - The dot product is: $1 * (-1.5) + 0 * 1 + 1 * 1 = -0.5$.
  - The activation function (assuming a step function) outputs 0.



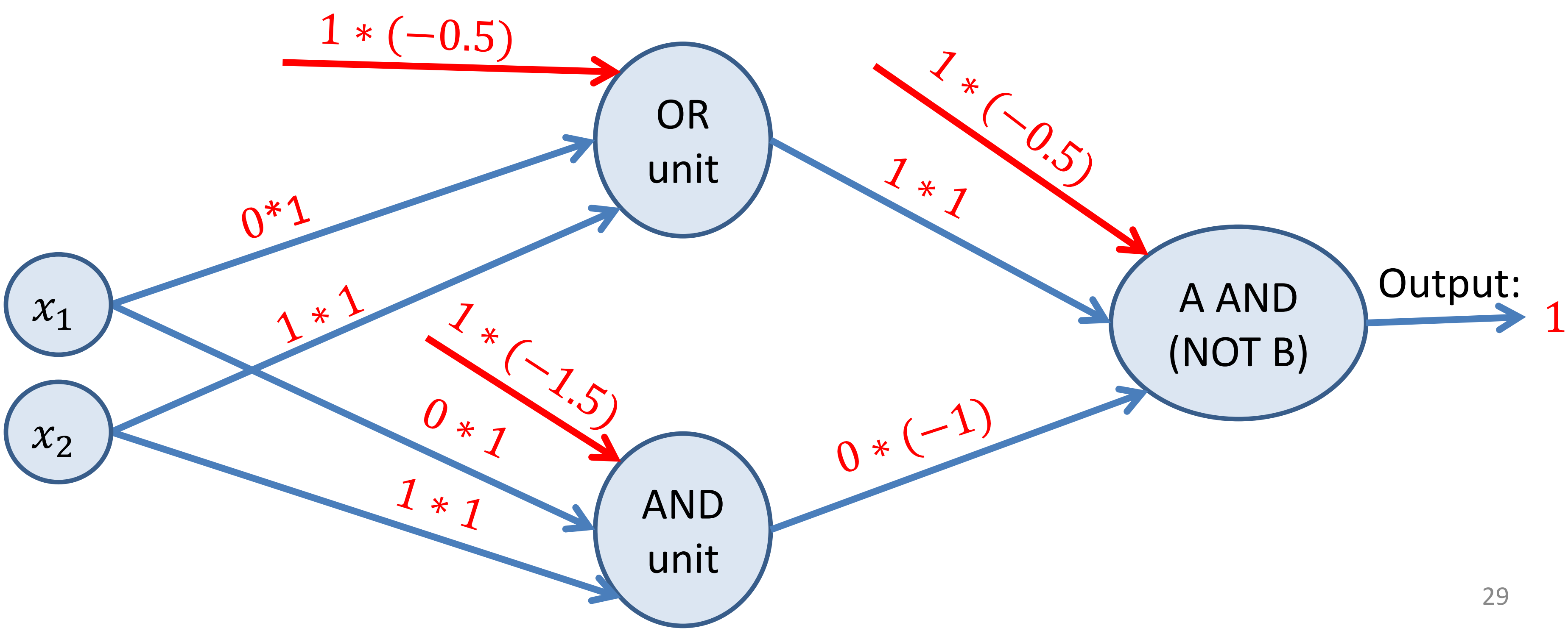
Computing the Output: An Example

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



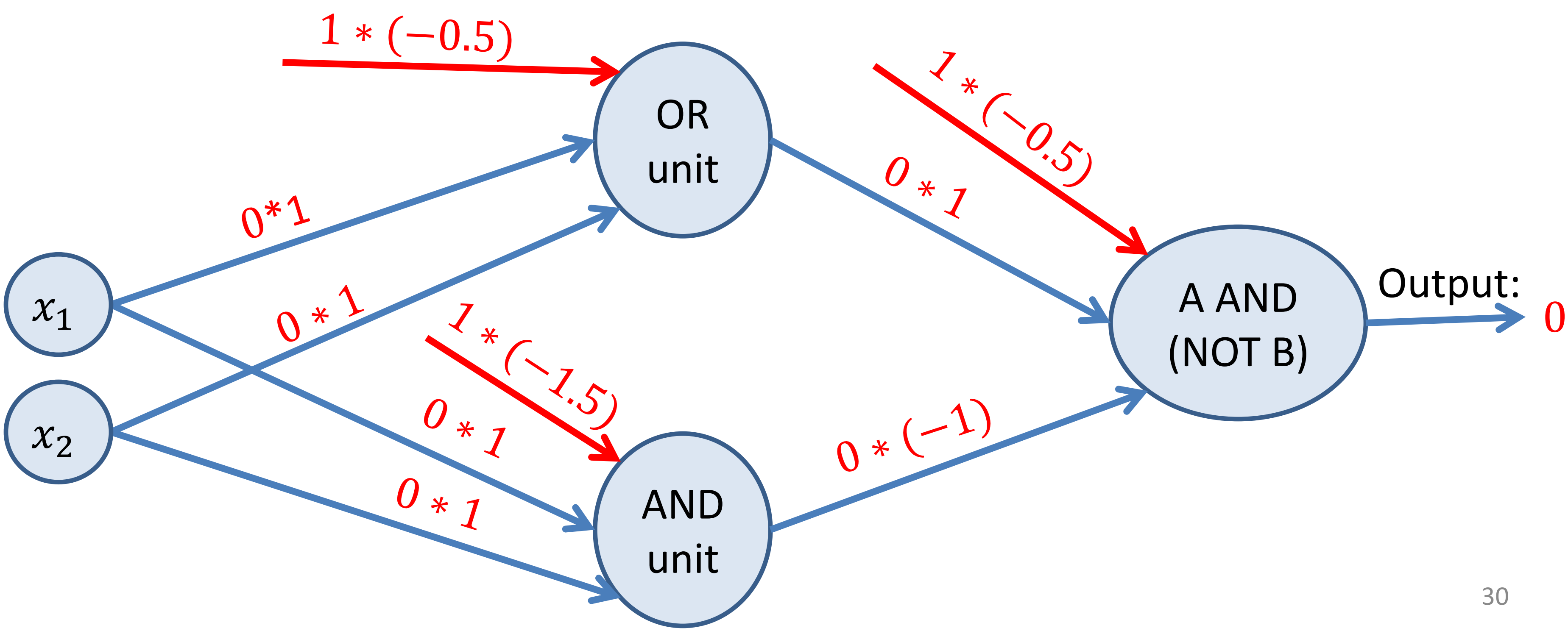
Computing the Output: An Example

- 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 dot product is: $1 * (-0.5) * 1 + 1 * 1 + 1 * 0 = 0.5$.
  - The activation function (assuming a step function) outputs 1.



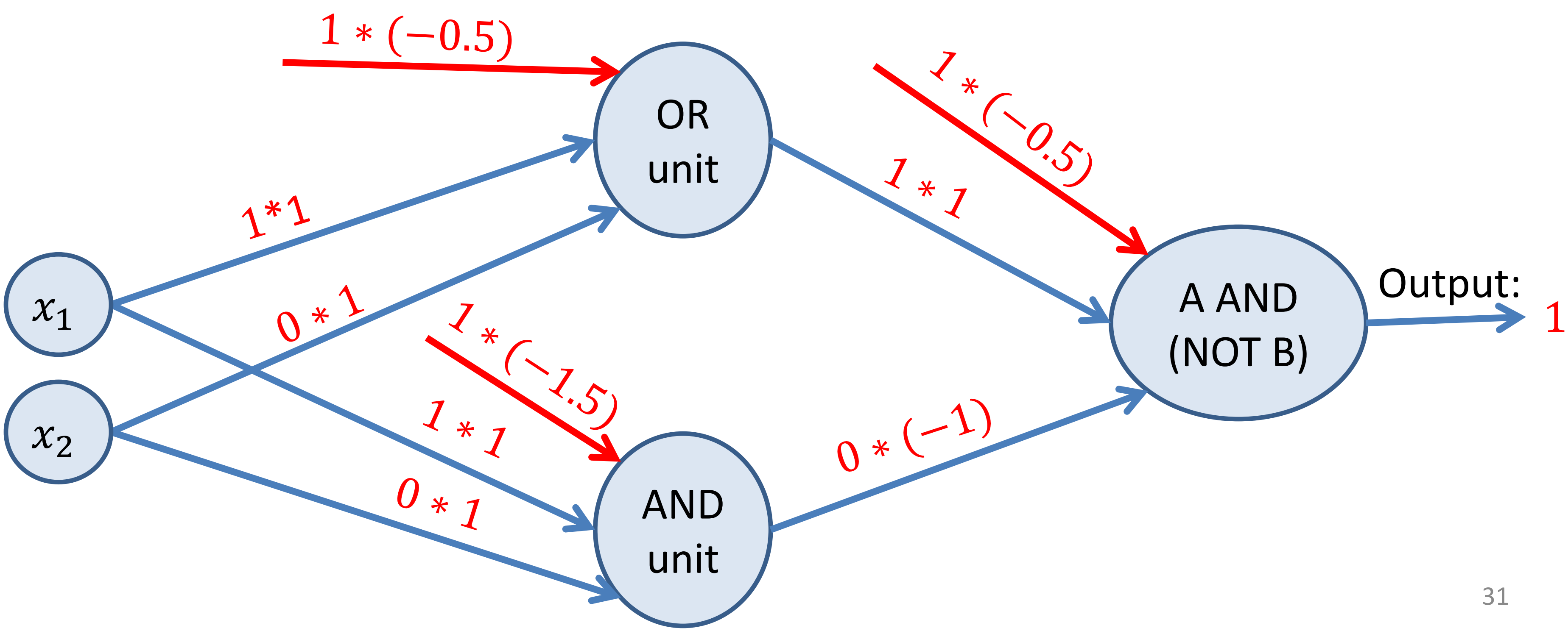
Verifying the XOR Network

- 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.



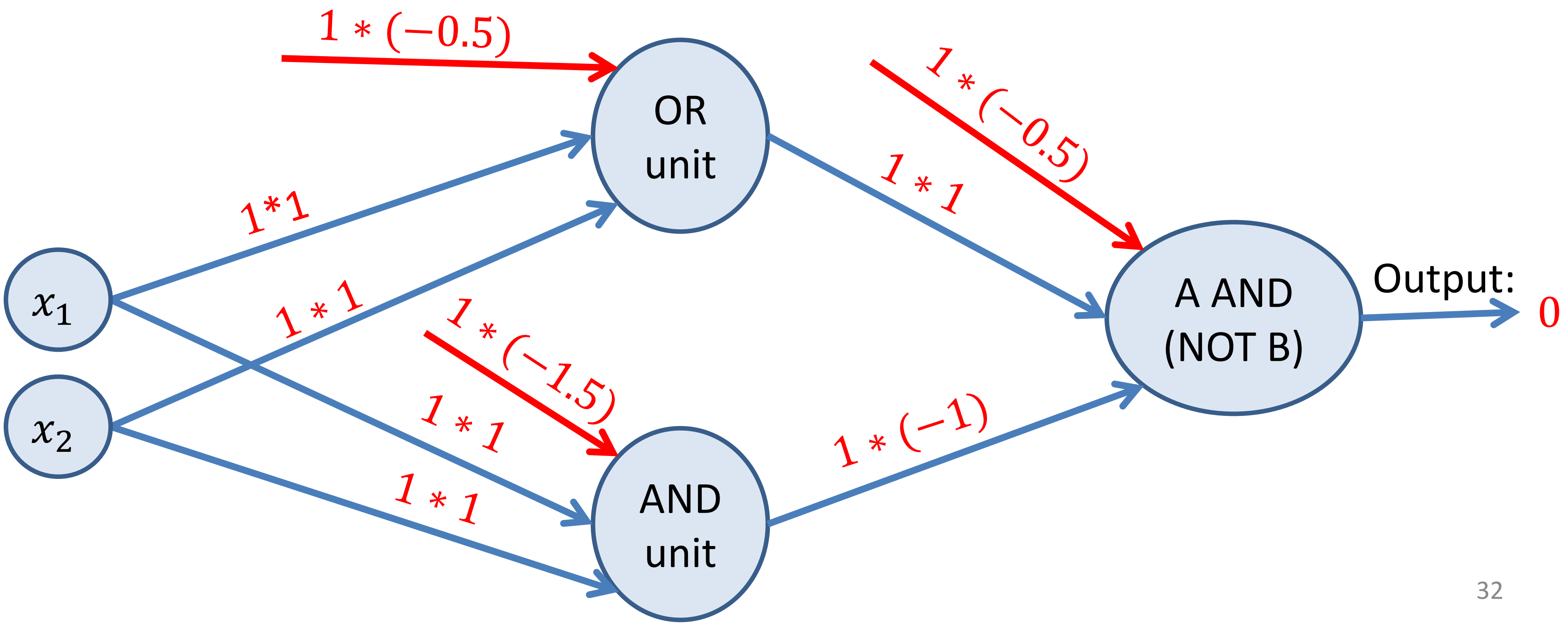
Verifying the XOR Network

- 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.



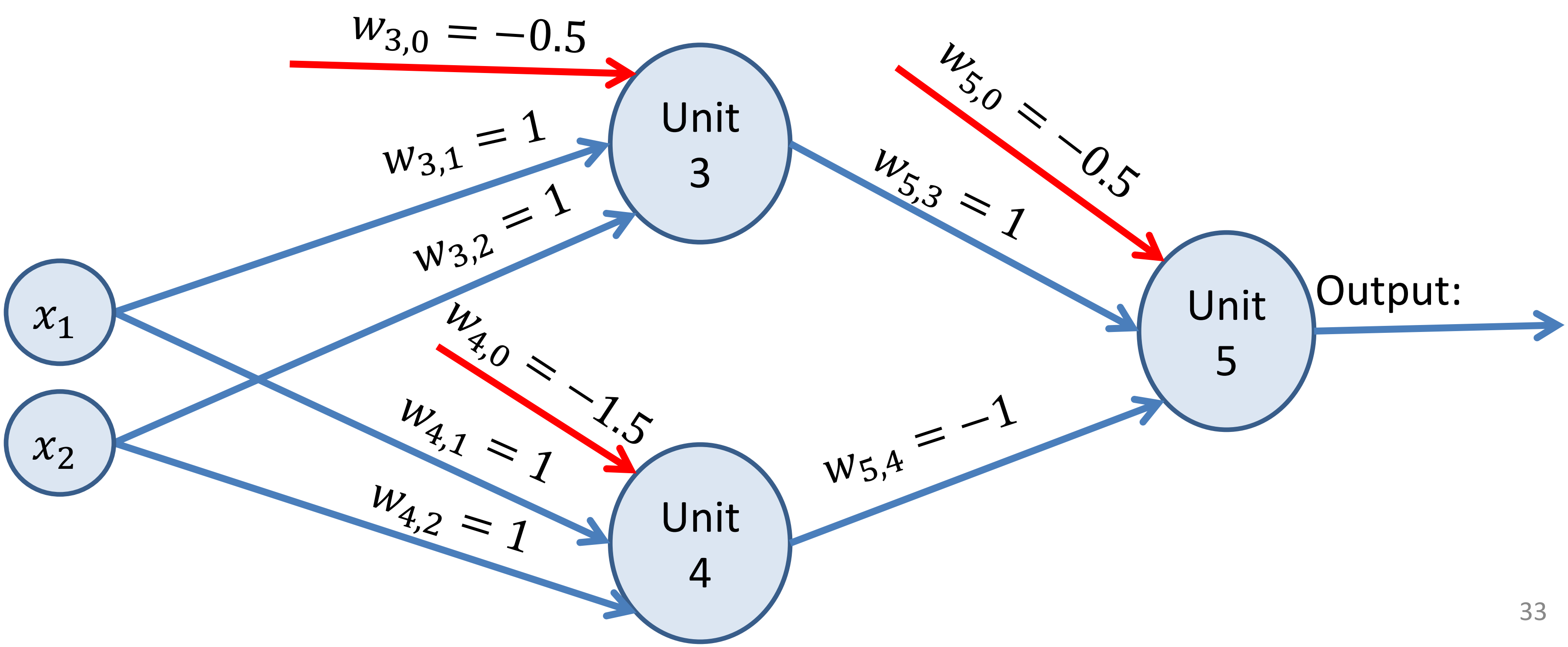
Verifying the XOR Network

- 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.



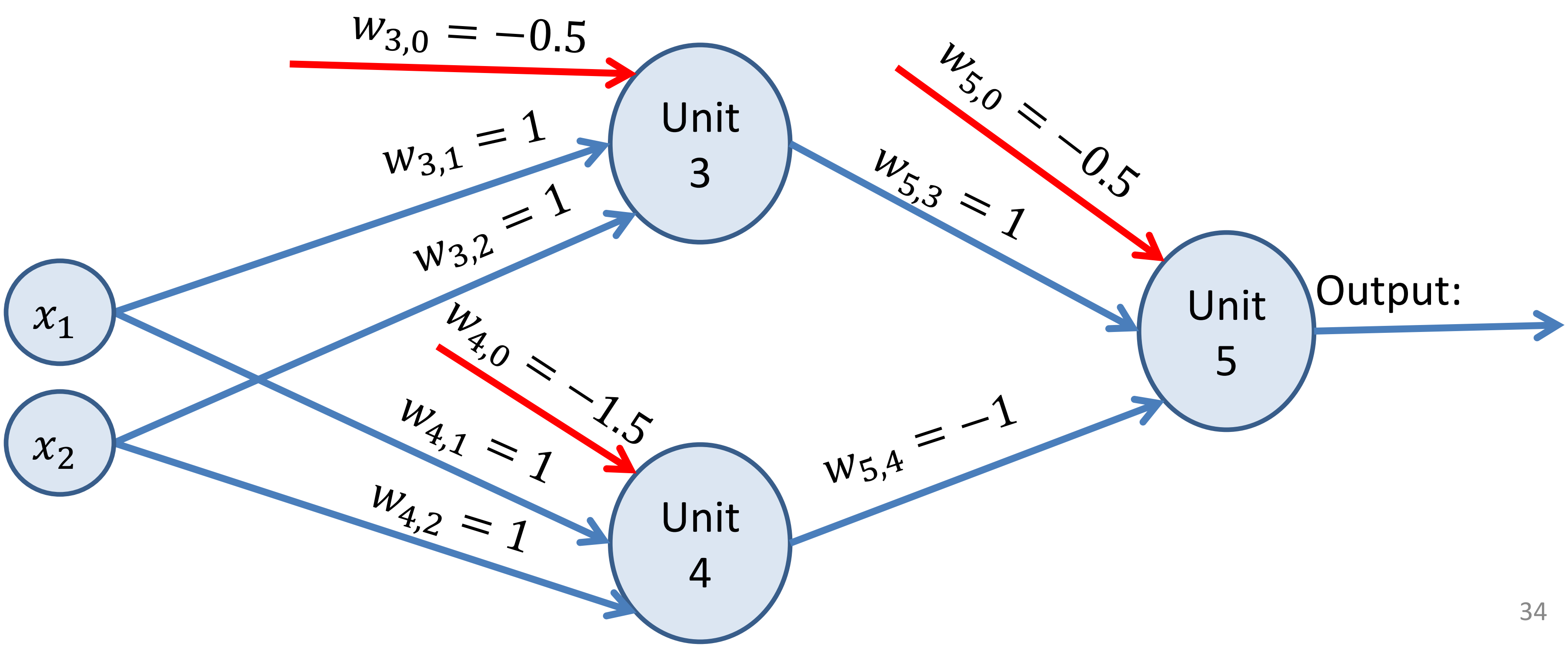
Neural Networks

- This neural network example consists of six units:
  - Three input units (including the not-shown bias input).
  - Three perceptrons.
- Yes, in the notation we will be using, inputs count as units.



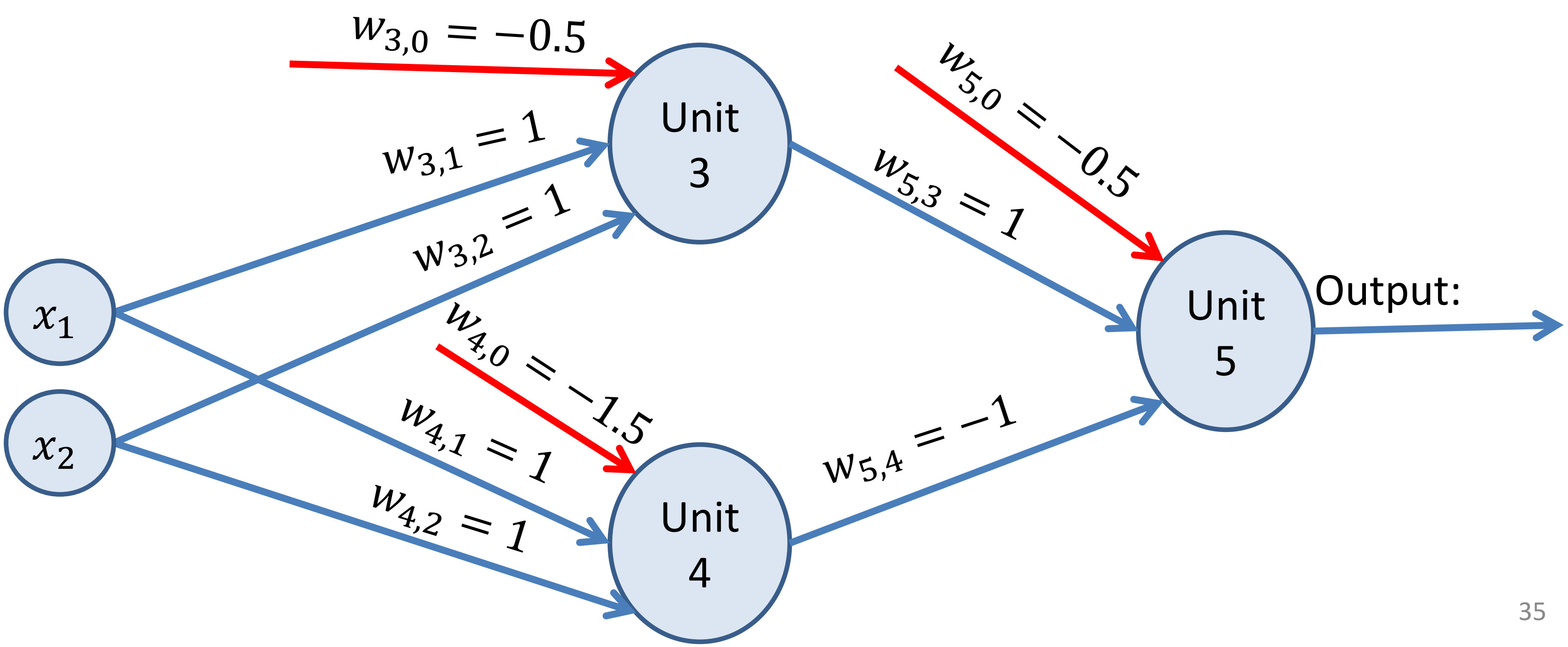
Neural Networks

- Weights are denoted as w_{ji}.
  - Weight w_{ji} belongs to the edge that connects the **output** of unit i with an **input** of unit j.
- Units $0, 1, \ldots, D$ are the **input units** (units 0, 1, 2 in this example).



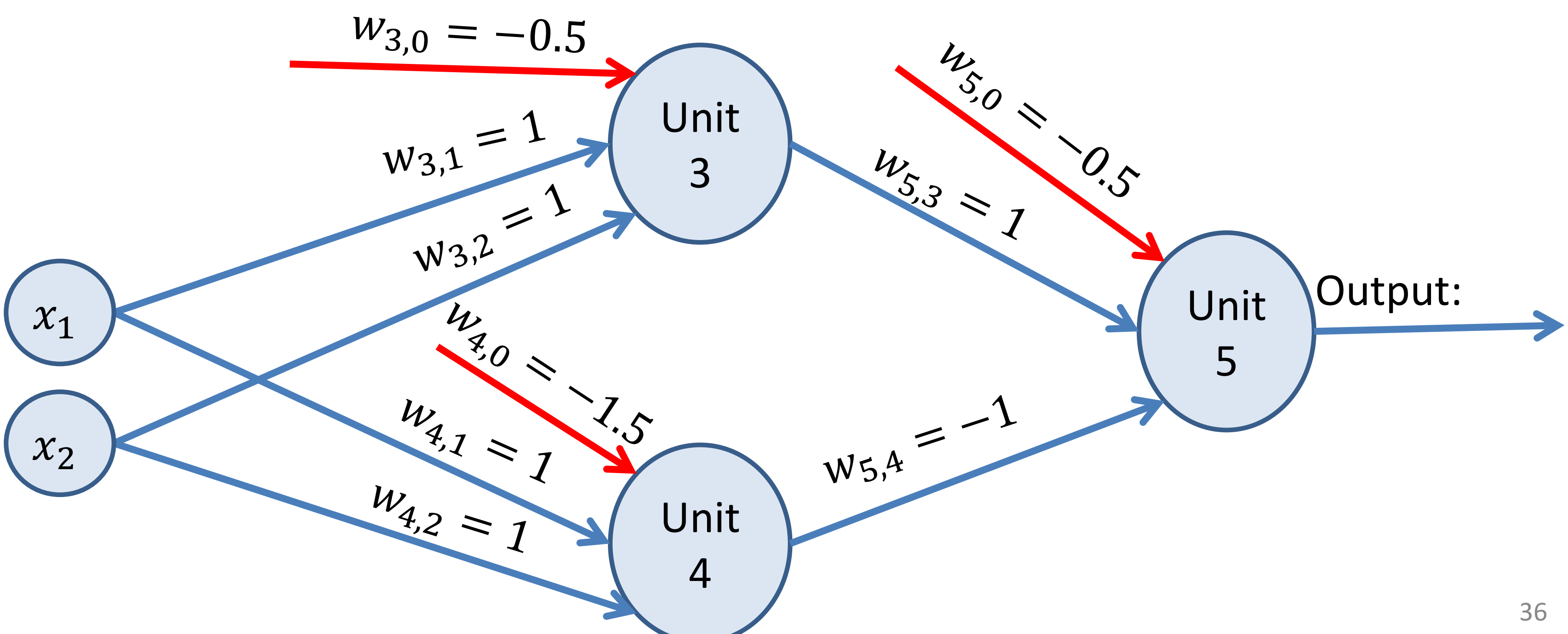
Neural Network Layers

- Oftentimes, neural networks are organized into layers.
- The **input layer** is the initial layer of input units (units 0, 1, 2 in our example).
- The output layer is at the end (unit 5 in our example).
- Zero, one or more hidden layers can be between the input and output layers.



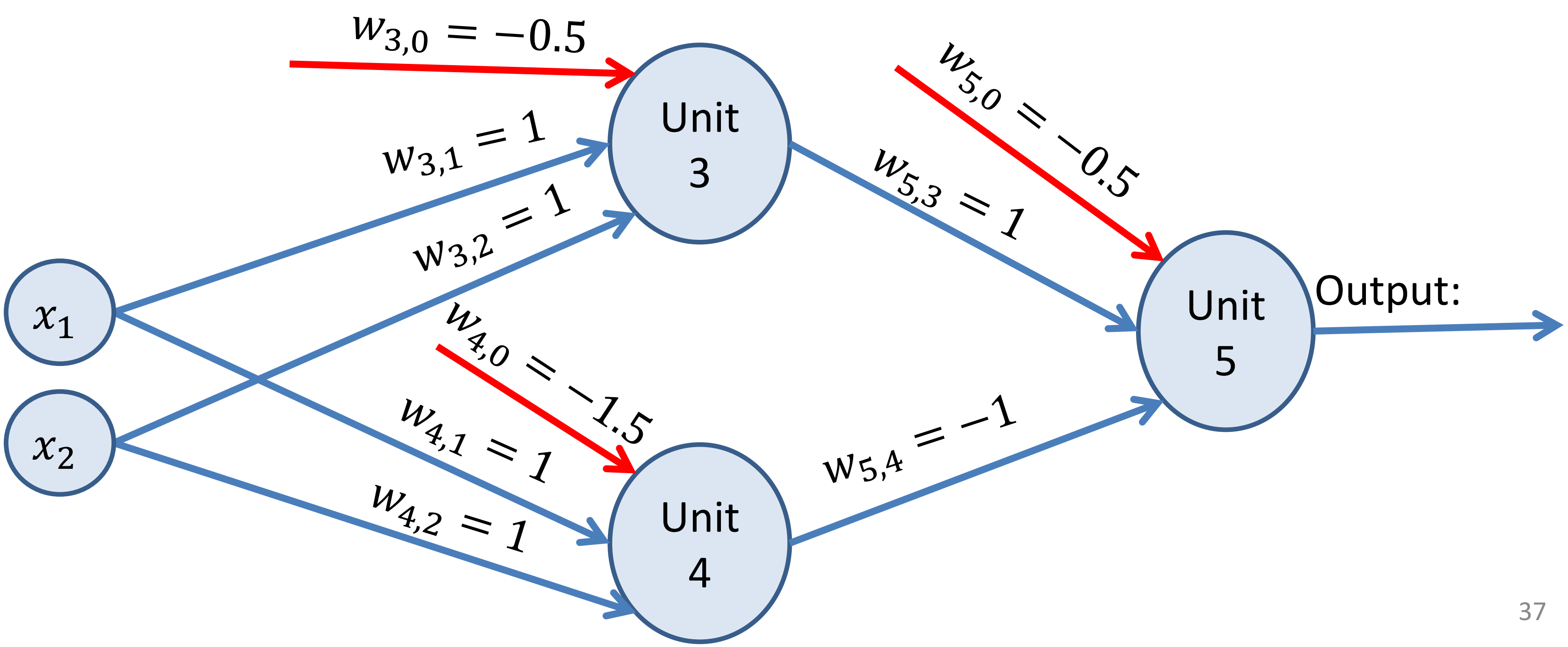
Neural Network Layers

- There is only one hidden layer in our example, containing units 4 and 5.
- 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.



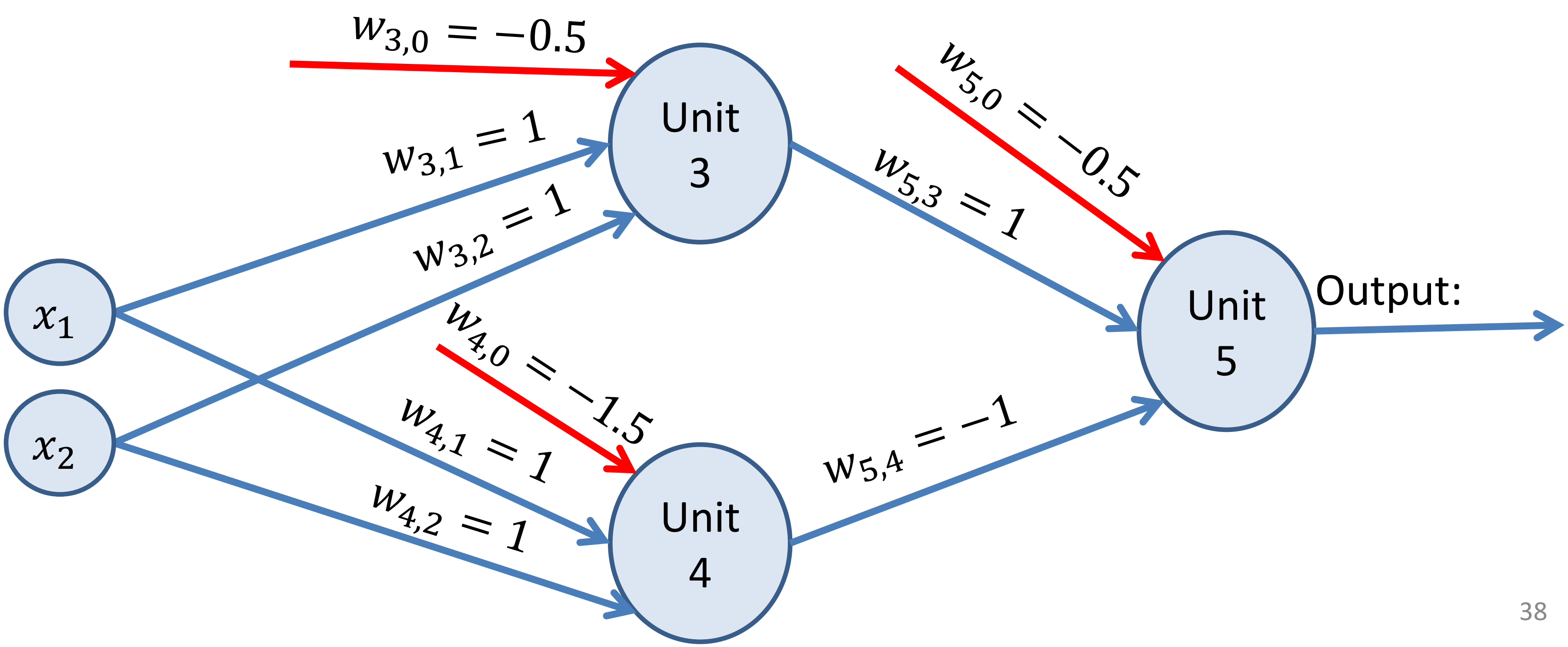
Feedforward Networks

- Feedforward networks are networks where there are no directed loops.
- If there are no loops, the output of a neuron cannot (directly or indirectly) influence its input.
- While there are varieties of neural networks that are not feedforward or layered, our main focus will be layered feedforward networks.



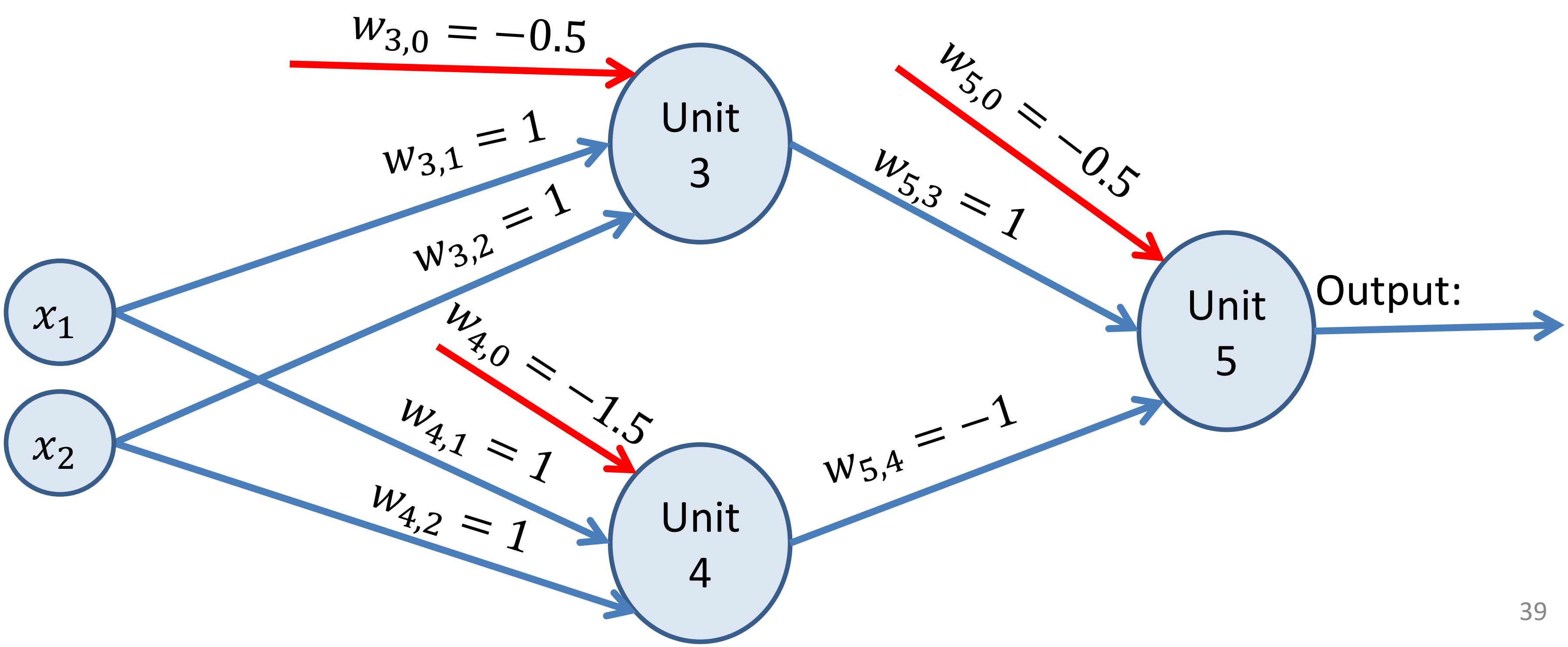
Computing the Output

- Notation: L is the number of layers.
  - Layer 1 is the input layer, layer L is the output layer.
- Given values for the input units, output is computed as follows:
- For $(l = 2; l \leq L; l = l + 1)$:
  - 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 perceptrons at layer $l - 1$.
  - Remember, we compute layer outputs in increasing order of l.

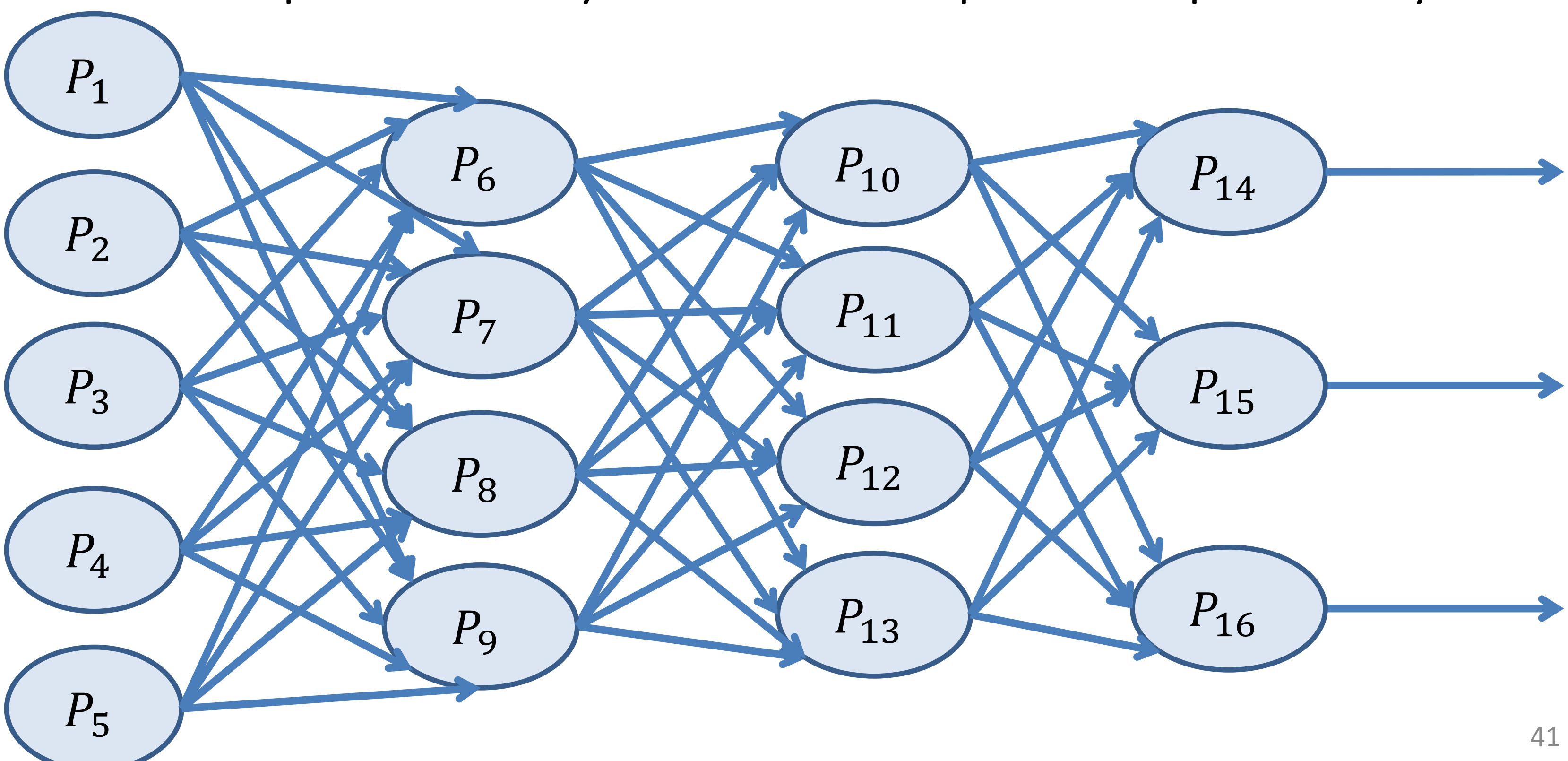


What Neural Networks Can Compute

- 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?

Layered Feedforward Networks

- Neural networks are graphs.
- In layered feedforward networks, the units are organized in layers.
  - The inputs in each layer come from outputs in the previous layer.



Design Choices

- Some important design choices in a neural network:
  - Number of layers.
    - Too many: slow to train, risk of overfitting.

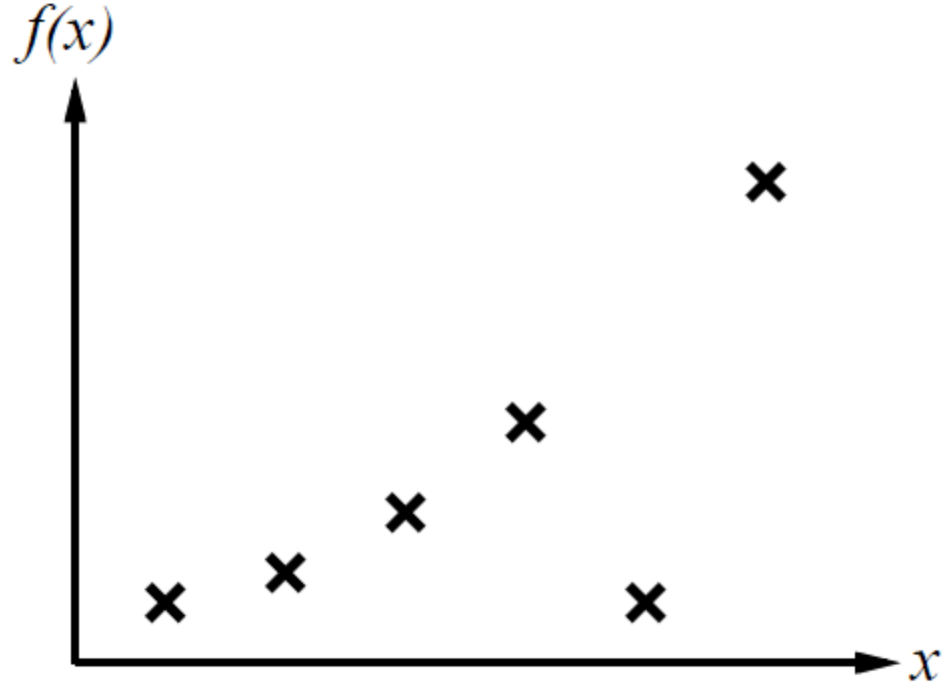
Design Choices

- Some important design choices in a neural network:
  - Number of layers.
    - Too many: slow to train, risk of overfitting.
- Here we need a necessary parenthesis: what is overfitting?

Overfitting

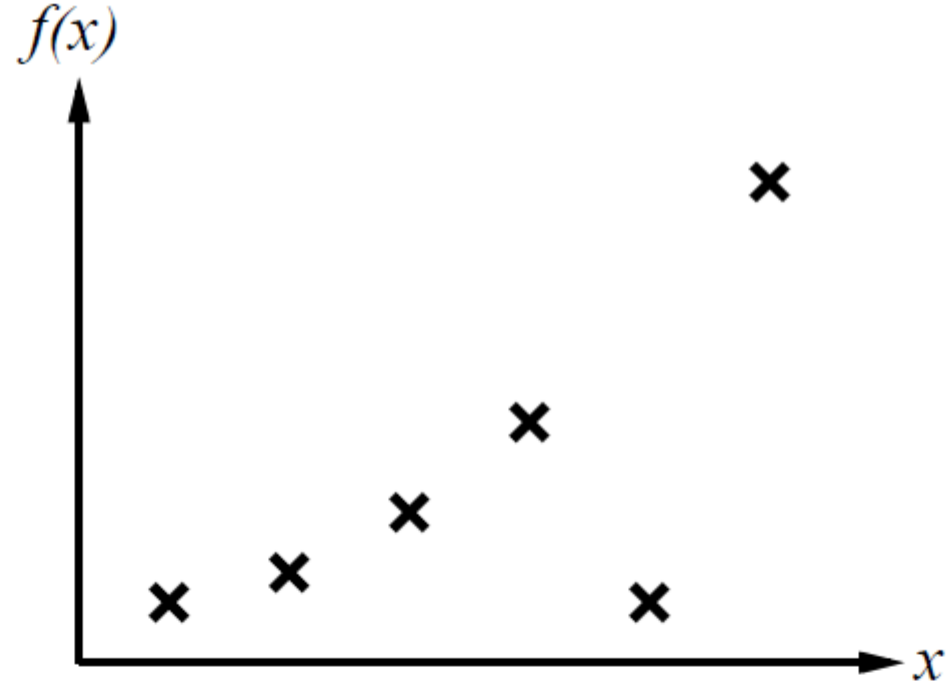
- Some important design choices in a neural network:
  - Number of layers.
    - Too many: slow to train, risk of overfitting.
- What is overfitting?
  - Overfitting is the commonly occurring phenomenon in machine learning where, after training, we get a model that works much better on the training data than on test data.
- Why does overfitting happen?
  - If our model has too many degrees of freedom, fitting the training data very well does not automatically imply that it will fit test data very well.

A Simple Learning Task



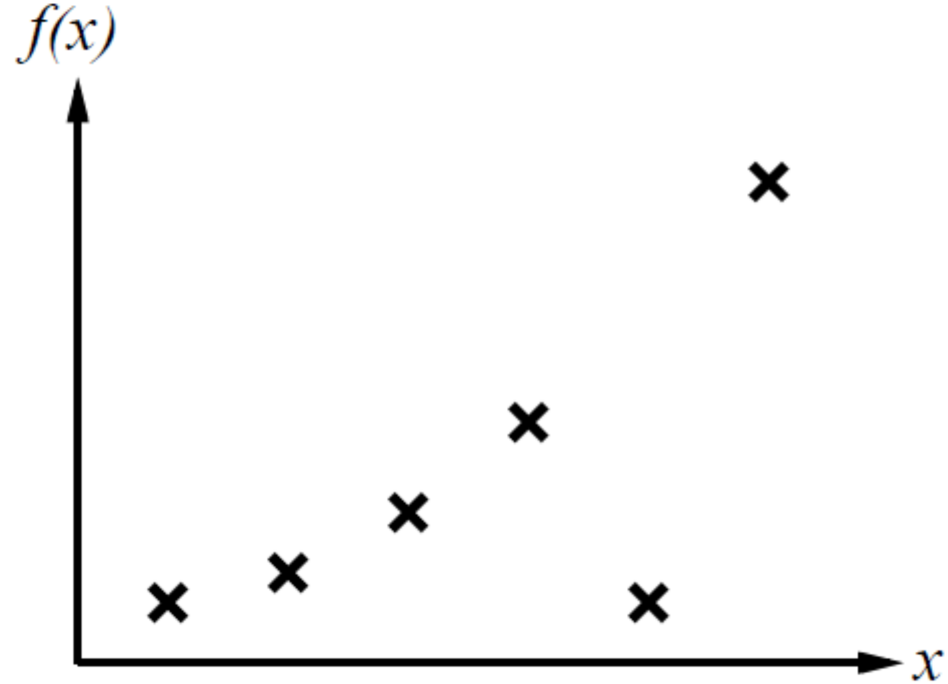
- This is a toy regression example
  - Source. S. Russell and P. Norvig, "Artificial Intelligence: A Modern Approach".
- Here, the input is a single real number.
- The output is also a real number.
- So, our target function F_{true} is a function from the reals to the reals.
  - Usually patterns are much more complex.
  - In this example it is easy to visualize training examples and learned functions.

A Simple Learning Task



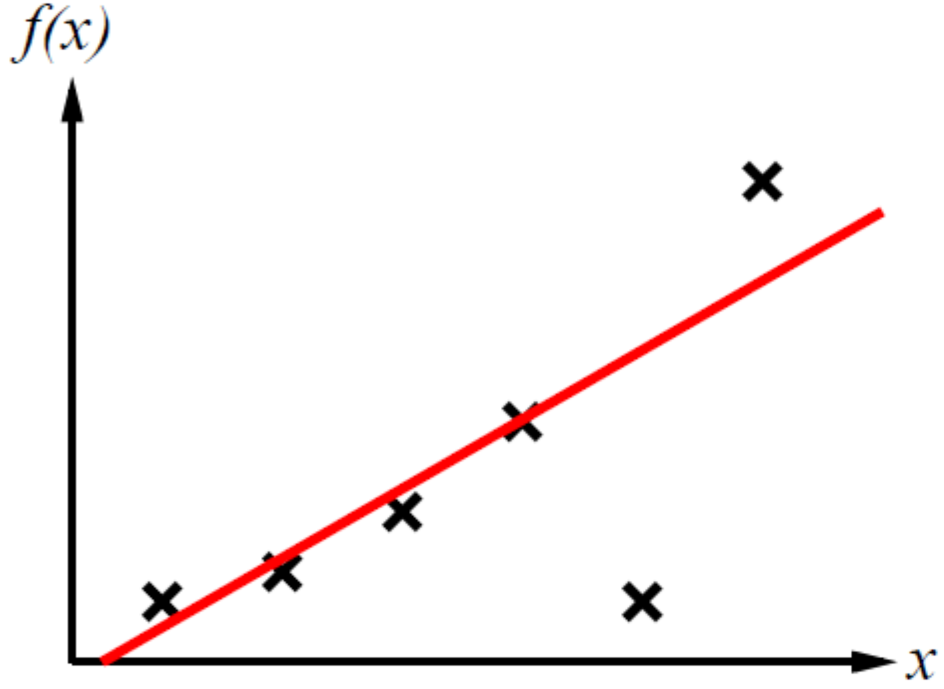
- Each training example is denoted as (x_n, t_n), where:
  - x_n is the example input.
  - t_n is the desired output (also called target output).
- Each example (x_n, t_n) is marked with $\times$ on the figure.
  - x_n corresponds to the x-axis.
  - t_n corresponds to the y-axis.
- Based on the figure, what do you think F_{true} looks like?

A Simple Learning Task



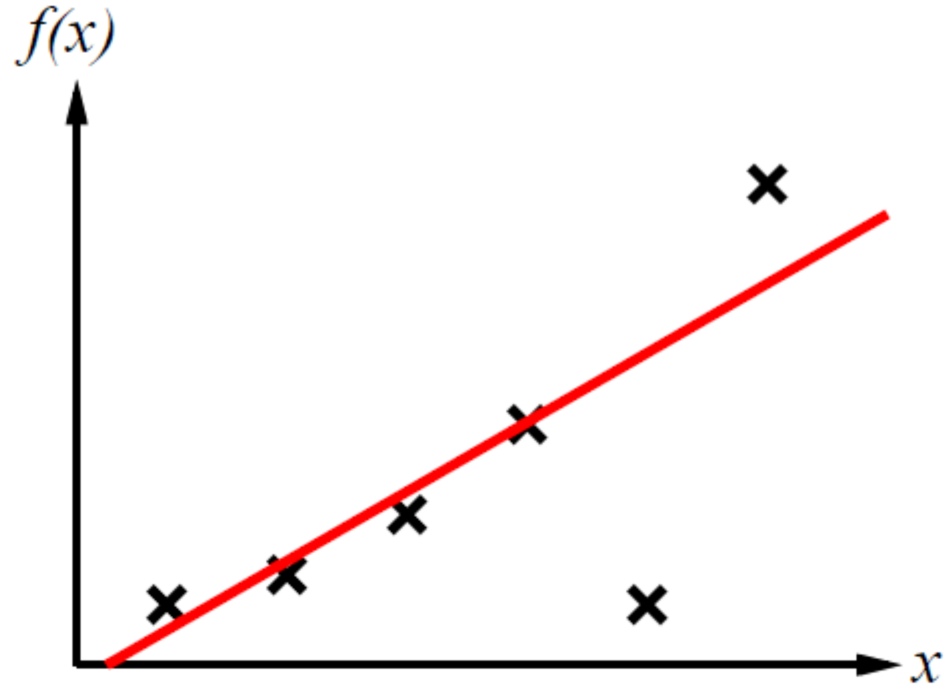
- Different people may give different answers as to what F_{true} may look like.
- That shows the challenge in supervised learning: we can find some plausible functions, but:
  - How do we know which one of them is correct?
  - Given many choices for the function, how can we evaluate each choice?

A Simple Learning Task



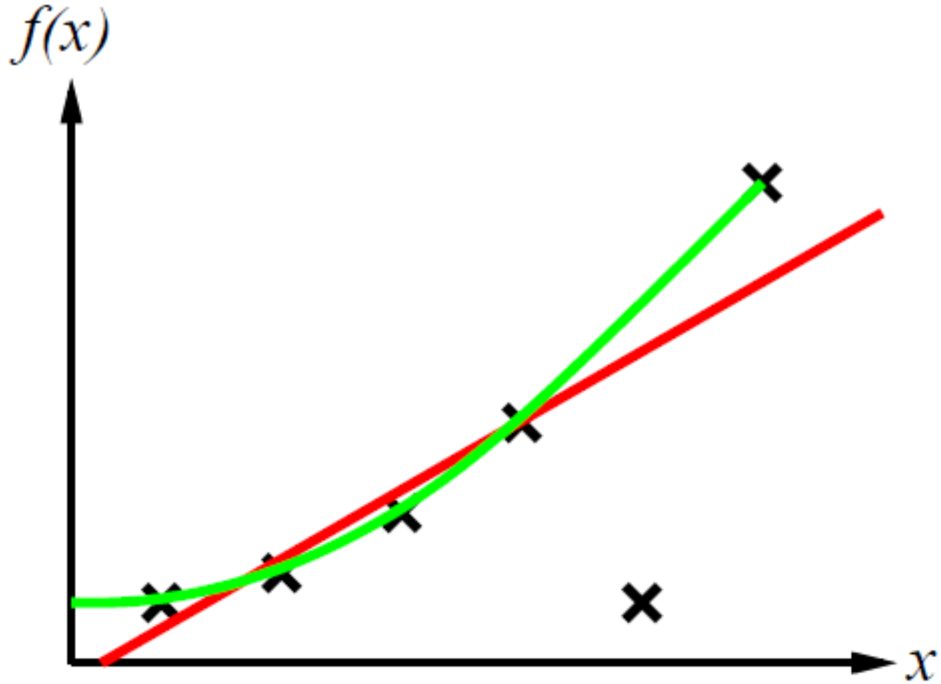
- Here is one possible function F.
- Can anyone guess how it was obtained?

A Simple Learning Task



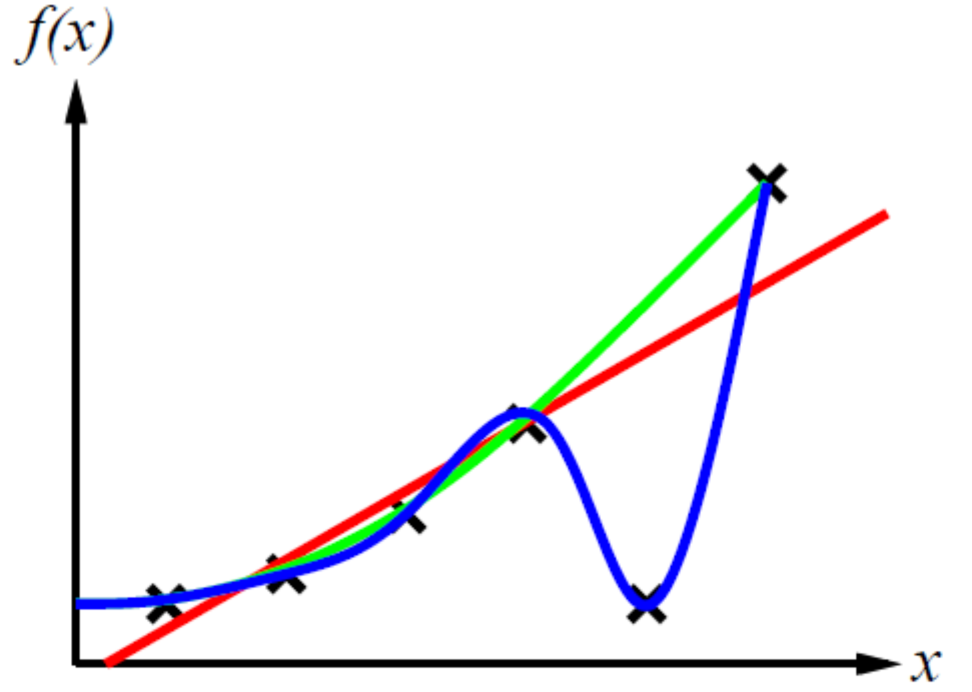
- Here is one possible function F.
- Can anyone guess how it was obtained?
- It was obtained by fitting a line to the training data.

Fitting Polynomials



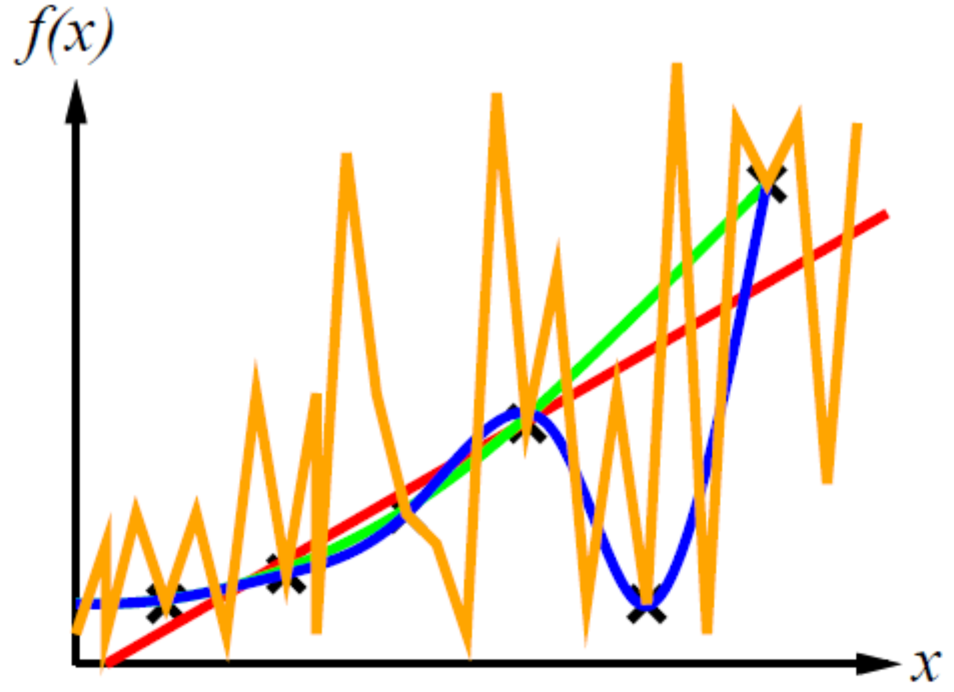
- Here we see another possible function F, shown in green.
- It looks like a quadratic function (second degree polynomial).
- It fits all the data perfectly, except for one.

Fitting Polynomials



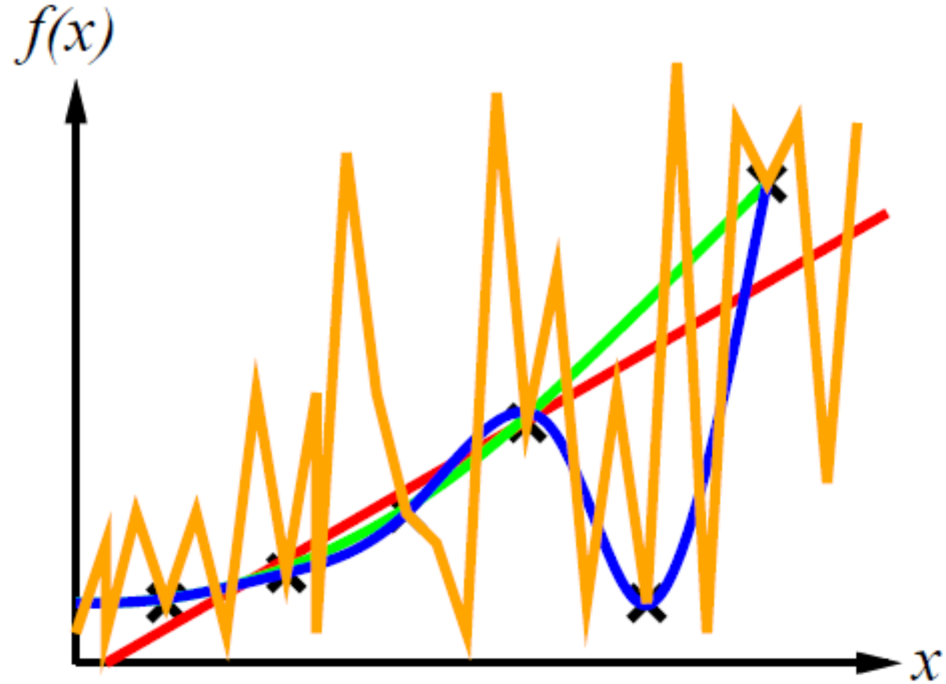
- Here we see a third possible function F, shown in blue.
- It looks like a cubic degree polynomial.
- It fits all the data perfectly.

More Complicated Solutions



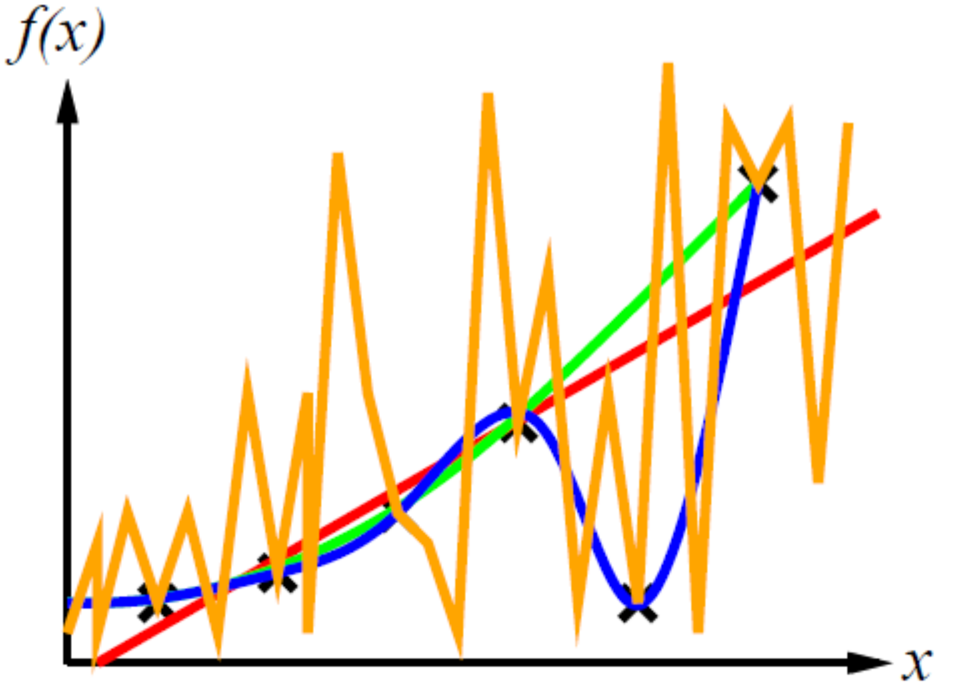
- Here we see a fourth possible function F, shown in orange.
- It zig-zags a lot.
- It fits all the data perfectly.

The Model Selection Problem



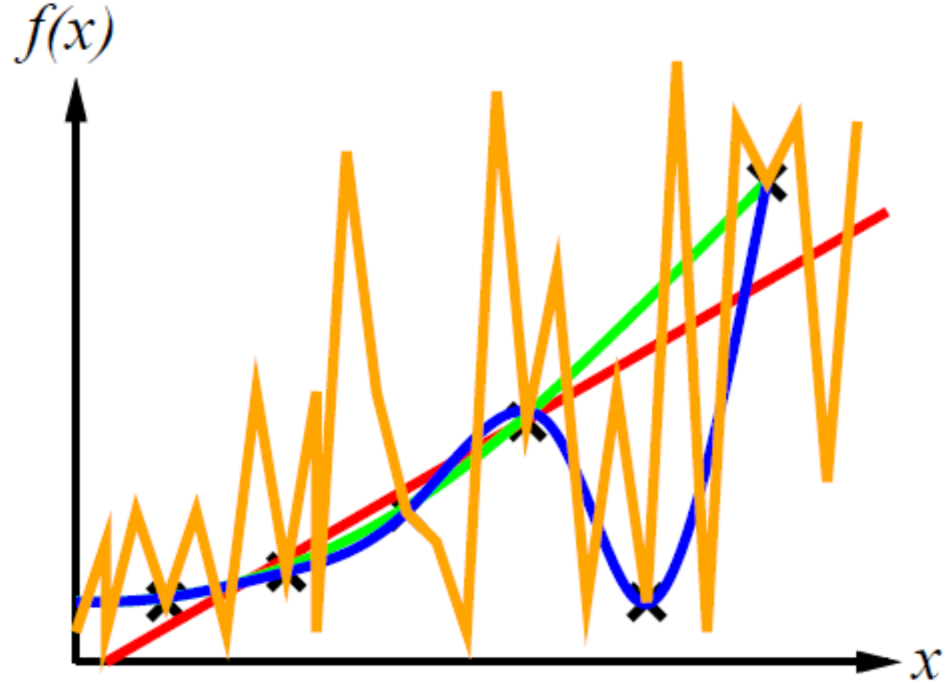
- Overall, we can come up with an infinite number of possible functions here.
- The question is, how do we choose which one is best?
- Or, an easier version, how do we choose a good one.
- This is called the **model selection problem**: out of an infinite number of possible **models** for our data, we must choose one.

The Model Selection Problem



- An easier version of the model selection problem: given a model (i.e., a function modeling our data), how can we measure how good this model is?
- **What are your thoughts on this?**

Using Training Error

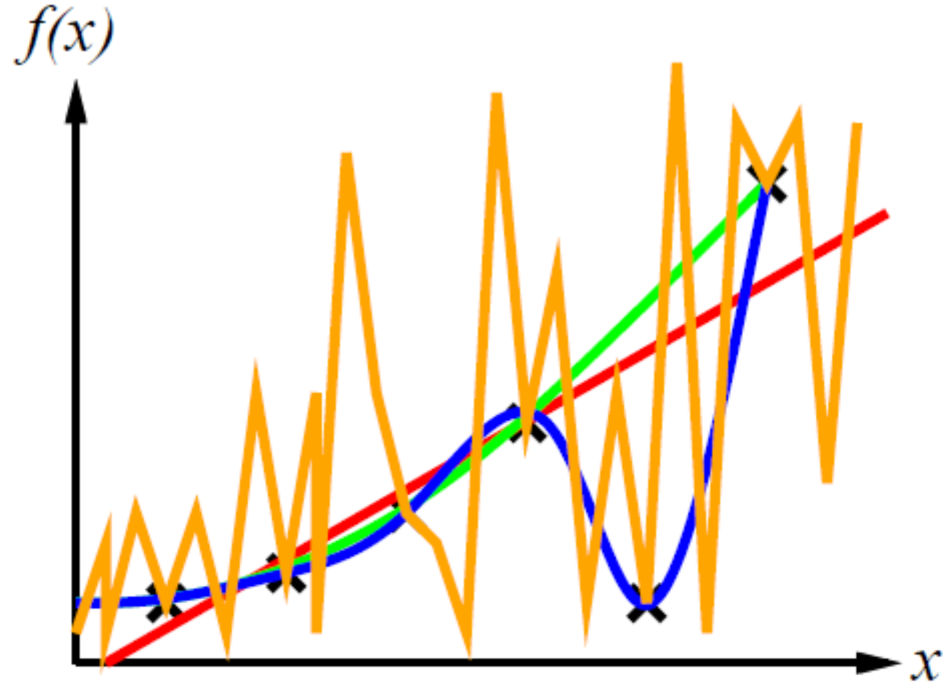


- One naïve solution is to evaluate solutions based on **training error**.
- For any function F, its training error can be measured as a sum of squared errors over training patterns x_n:

$$\sum_n (t_n - F(x_n))^2$$

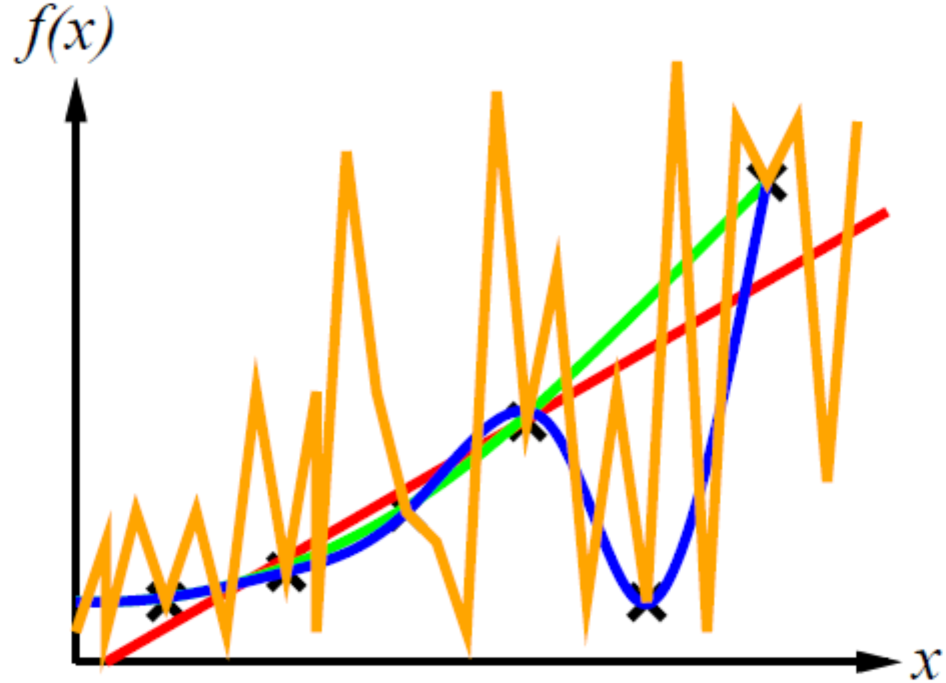
- What are the pitfalls of choosing the "best" function based on training error?

Using Training Error



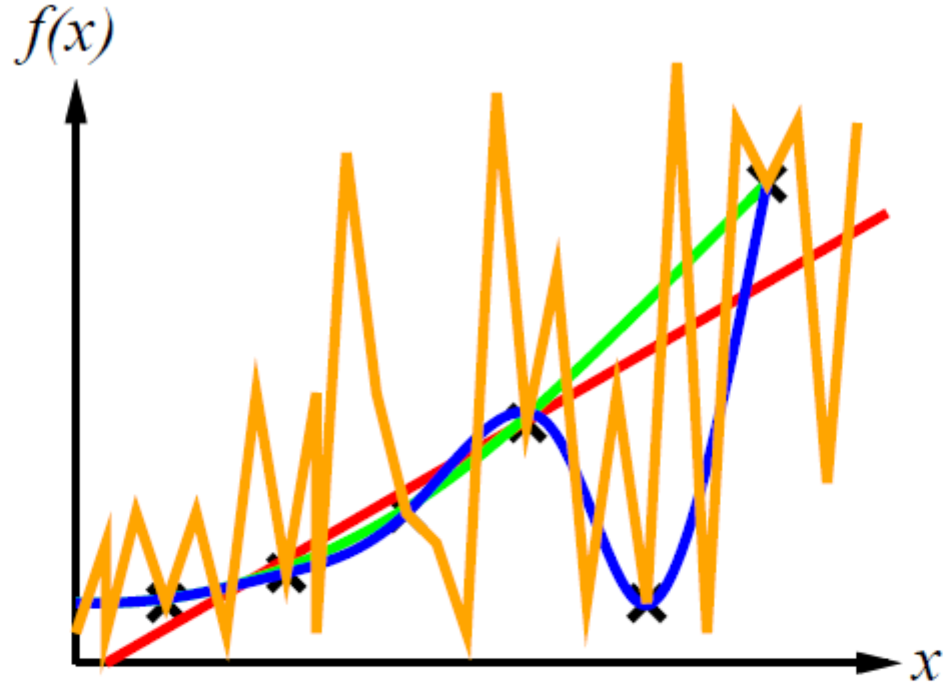
- What are the pitfalls of choosing the “best” function based on training error?
- The zig-zagging orange function comes out as “perfect”: its training error is zero.
- As a human, would you find more reasonable the orange function or the blue function (cubic polynomial)?
  - They both have zero training error.

Using Training Error



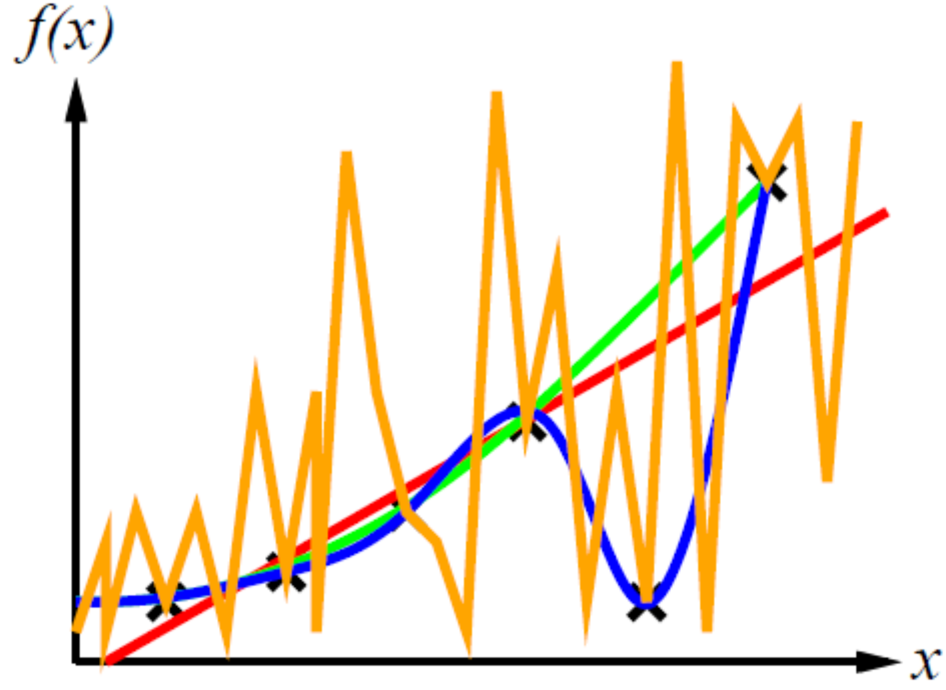
- What are the pitfalls of choosing the “best” function based on training error?
- The zig-zagging orange function comes out as “perfect”: its training error is zero.
- As a human, would you find more reasonable the orange function or the blue function (cubic polynomial)?
  - They both have zero training error.
  - However, the zig-zagging function looks pretty arbitrary.

Ochham's Razor



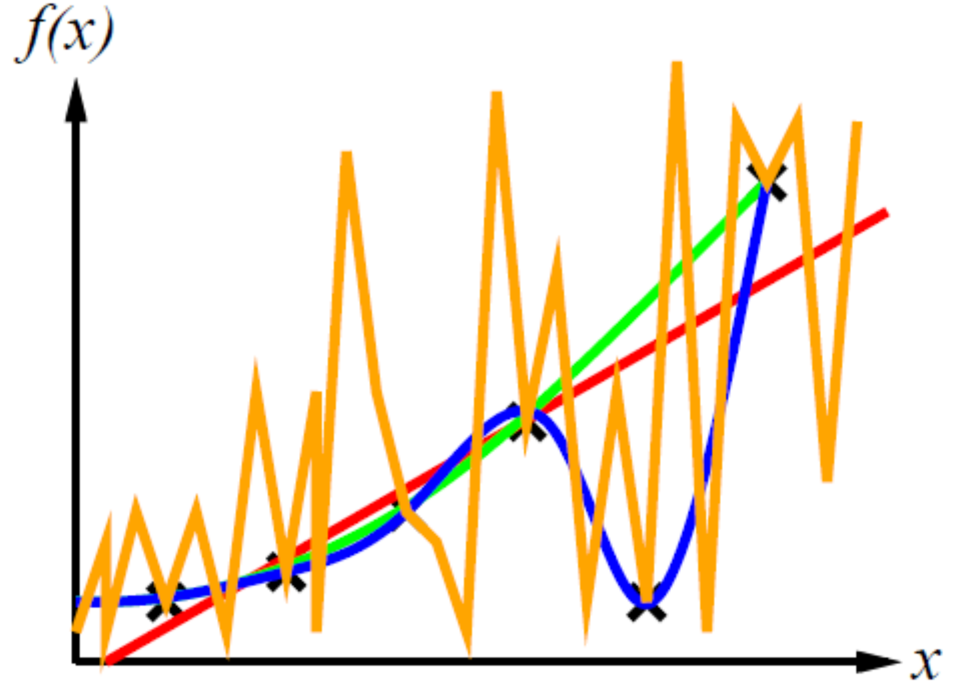
- Ockham's razor: given two equally good explanations, choose the more simple one.
  - This is an old philosophical principle (Ockham lived in the 14th century).
- Based on that, we prefer a cubic polynomial over a crazy zig-zagging function, because it is more simple, and they both have zero training error.

Simplicity vs. Training Error



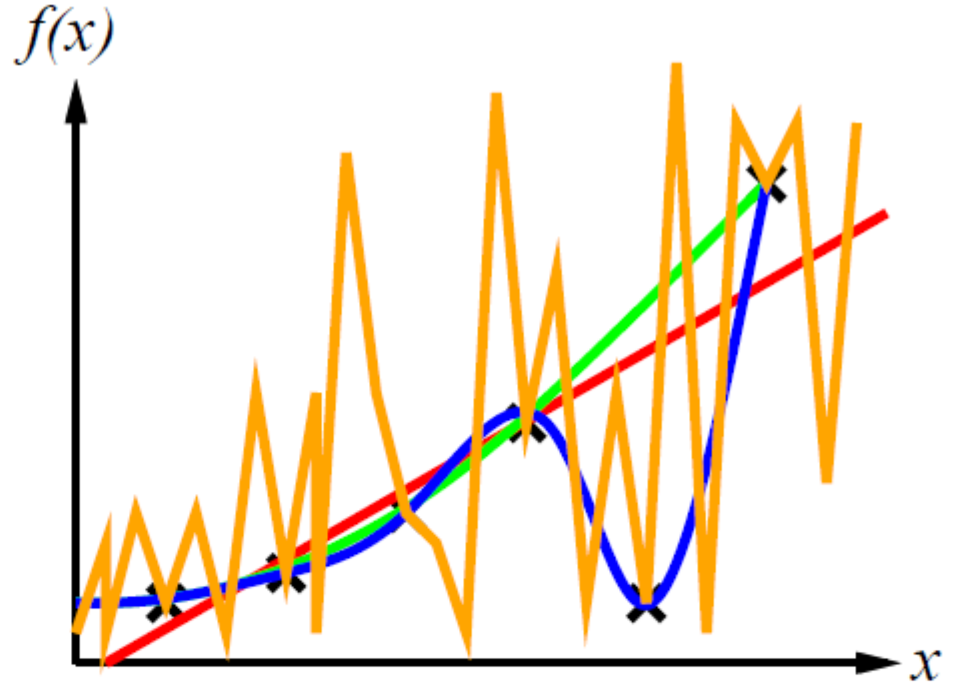
- However, real life is more complicated.
- What if none of the functions have zero training error?
- How do we weigh simplicity versus training error?

Simplicity vs. Training Error



- However, real life is more complicated.
- What if none of the functions have zero training error?
- How do we weigh simplicity versus training error?
- There is no standard or straightforward solution to this.
- There exist many machine learning algorithms. Each corresponds to a different approach for resolving the trade-off between simplicity and training error.

Using a Validation Set



- We can use a **validation** set of examples, to detect overfitting.
- Validation objects are not used for training.
- Suppose that model A is more simple, and model B is more complicated.
- Suppose that model B gives lower training error than model A, and model A gives lower validation error than model B.
  - Then, model B overfits the training data, and we should use model A instead.

Design Choices

- Some important design choices in a neural network:
  - Number of layers.
    - Too many: slow to train, risk of overfitting.

Design Choices

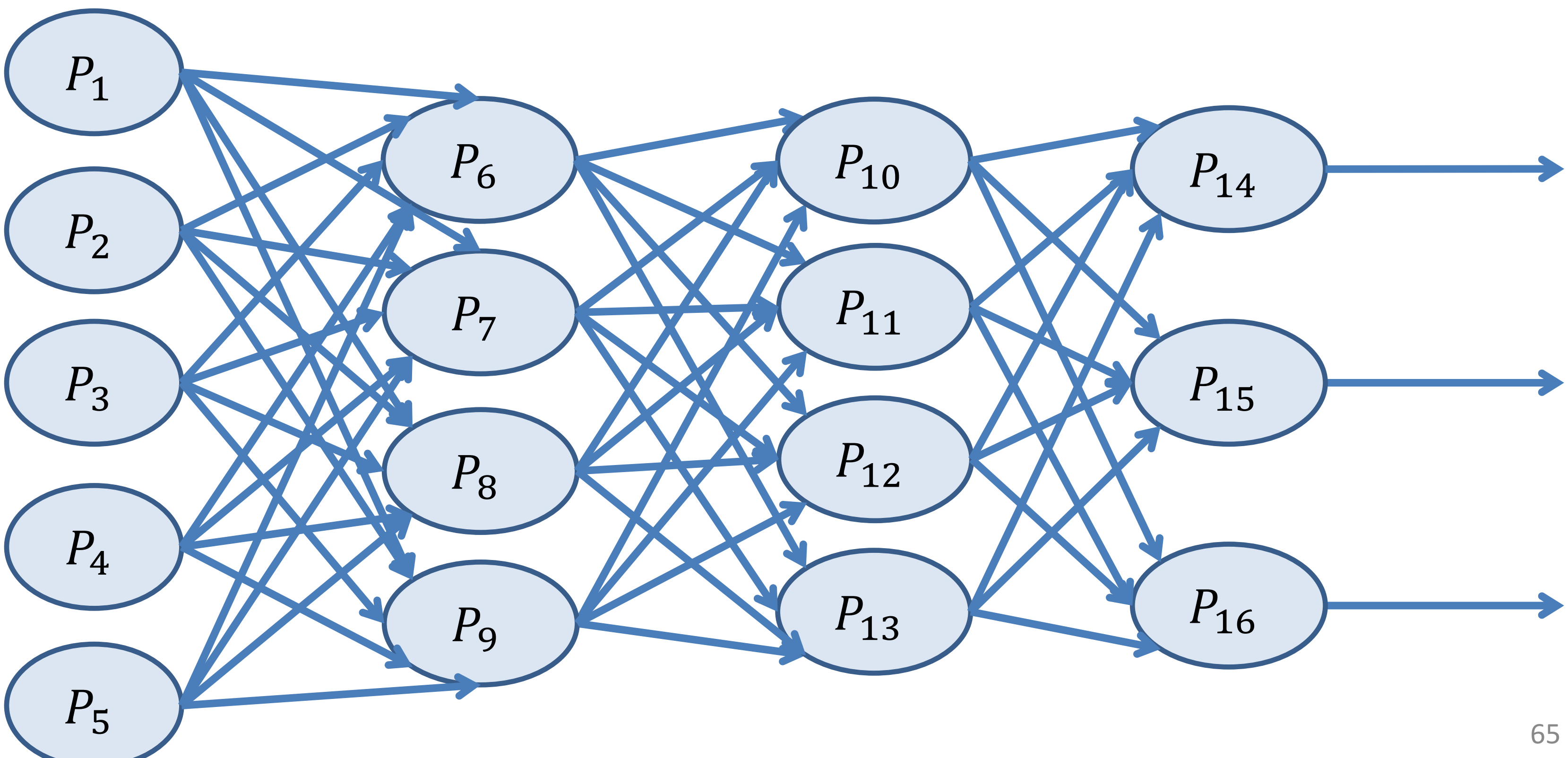
- Some important design choices in a neural network:
  - Number of layers.
    - Too many: slow to train, risk of overfitting.
    - Too few: may be too simple to learn an accurate model.

Design Choices

- Some important design choices in a neural network:
  - Number of layers.
    - Too many: slow to train, risk of overfitting.
    - Too few: may be too simple to learn an accurate model.
  - Number of units per layer.
    - We have a separate choice for each layer.
    - Too many: slow to train, dangers of overfitting.
    - Too few: may be too simple to learn an accurate model.
    - We have no choice for input layer (number of units = number of dimensions of input vectors) and output layer (number of units = number of classes).
  - Connectivity: what outputs are connected to what inputs?

Fully Connected Layer

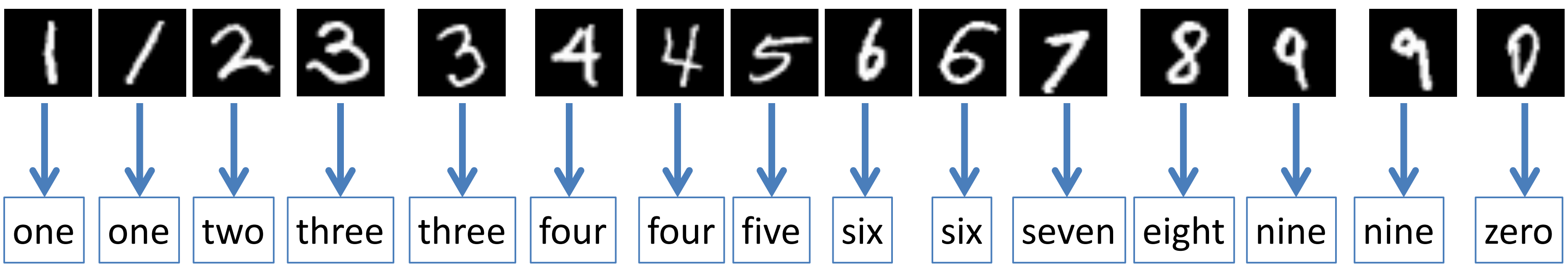
- A fully connected layer is a layer where every unit has as inputs ALL the outputs of the previous layer.
- In the picture, all layers (except the first one) are fully connected.



Example of Fully Connected Layer

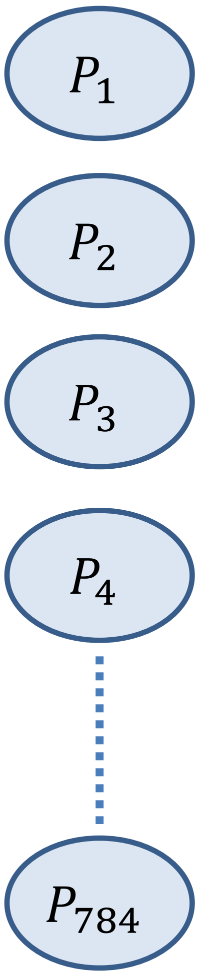
- Consider the MNIST dataset.
- Each image is of size 28x28.
  - Each image is a 784-dimensional vector.

example inputs



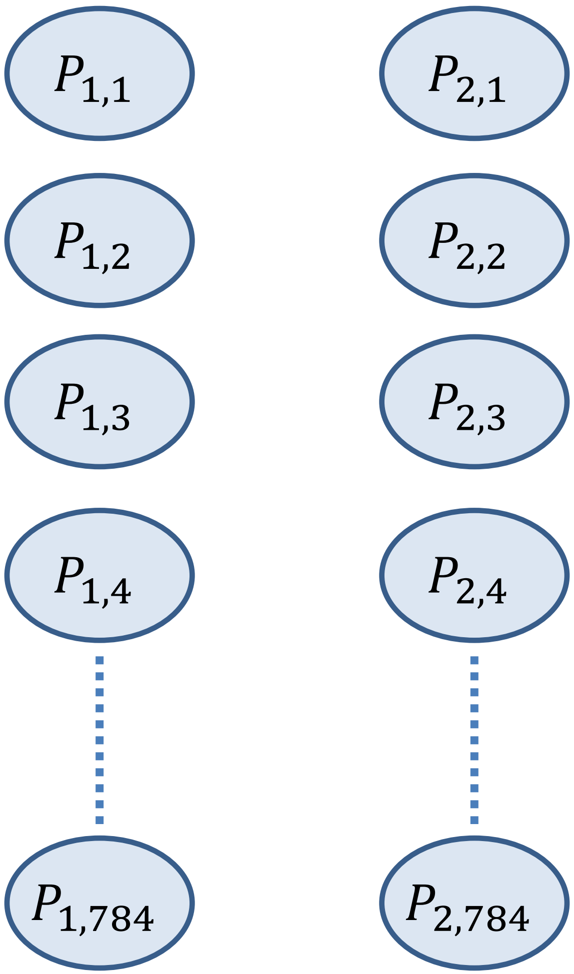
desired outputs (class labels)

Example of Fully Connected Layer



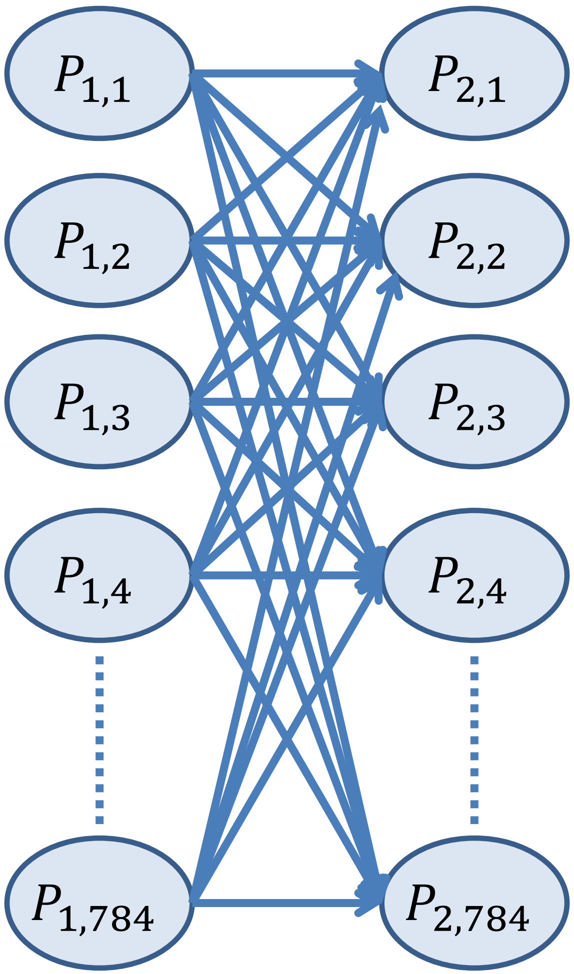
- Thus, the input layer has 784 units.

Example of Fully Connected Layer



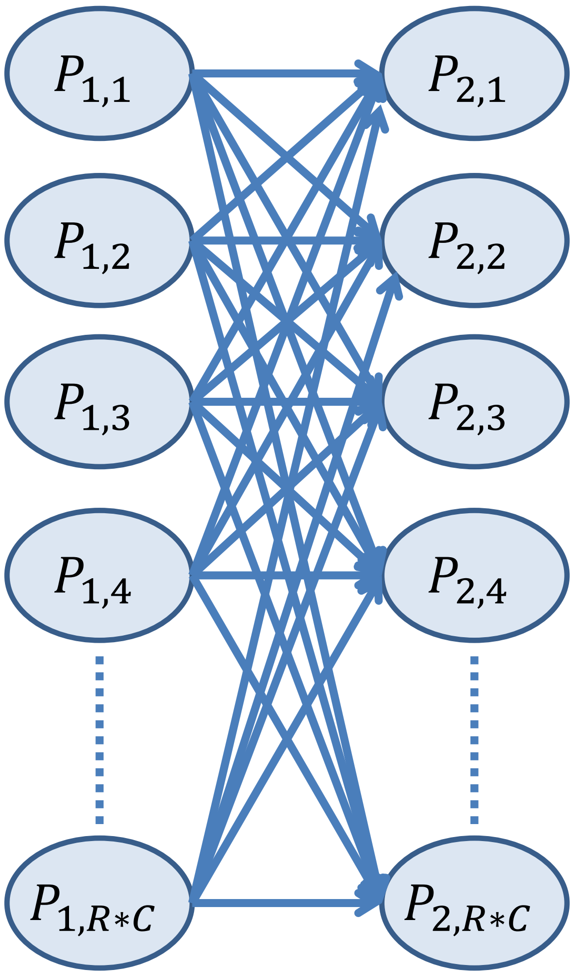
- Thus, the input layer has 784 units.
- What about the next layer?
  - How many units? There is no standard way to decide. Let's pick 784, to match the input layer.

Example of Fully Connected Layer



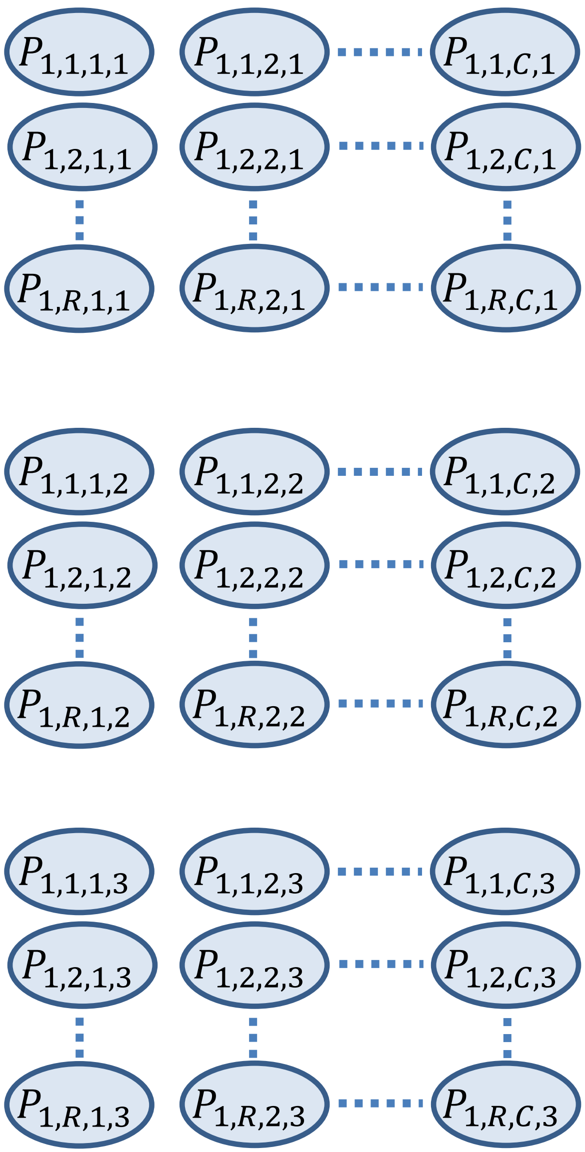
- Thus, the input layer has 784 units.
- What about the next layer?
  - How many units? There is no standard way to decide. Let's pick 784, to match the input layer.
- Connectivity? Let's make it fully connected.
- How many weights do we need to learn?
  - 784*784 = 614,656 weights.

Example of Fully Connected Layer



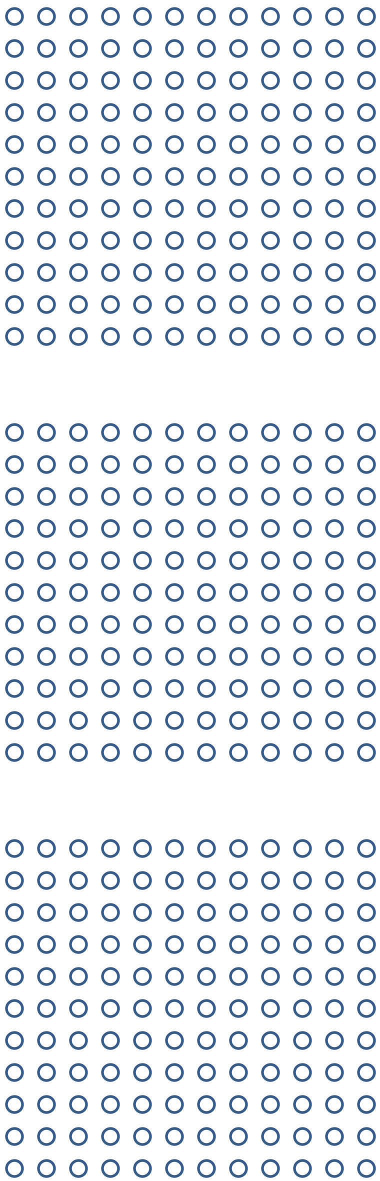
- For larger images, let's say 400x500 pixels, the number of weights could be enormous.
  - It would be 400*500*400*500, which is 40 billion weights.
- Such a layer would require:
  - Large storage to store the weights.
  - Long time to train.
  - Long time to classify an input image.

Visualizing the Input Layer in 3D



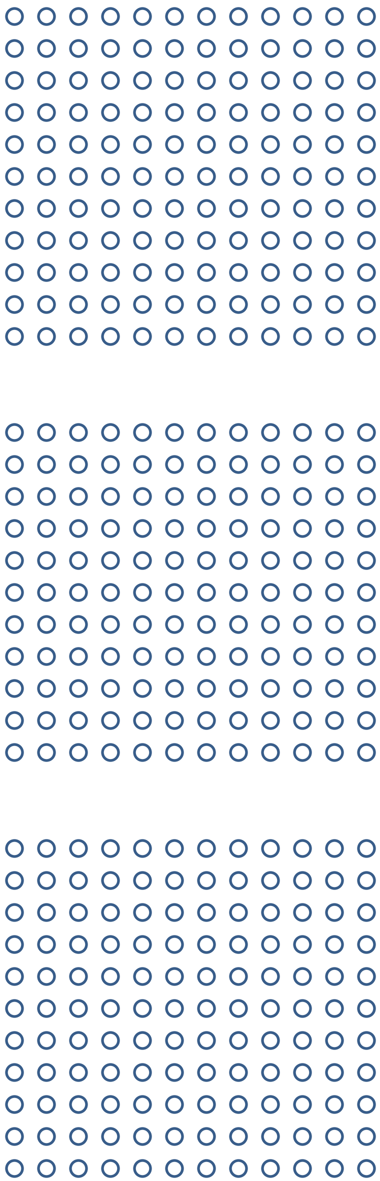
- To describe a convolutional layer, we need to change our notation a bit.
- The input layer will be a 3D array, to better reflect the fact that the input is an RGB image (of size R*C*3).
- So, what you see on the left is a single layer: the input layer, with R*C*3 units.
  - Unit $P_{1,i,j,k}$ is the unit corresponding to layer 1 (the input layer), row i, column j, channel k.

Visualizing the Input Layer in 3D



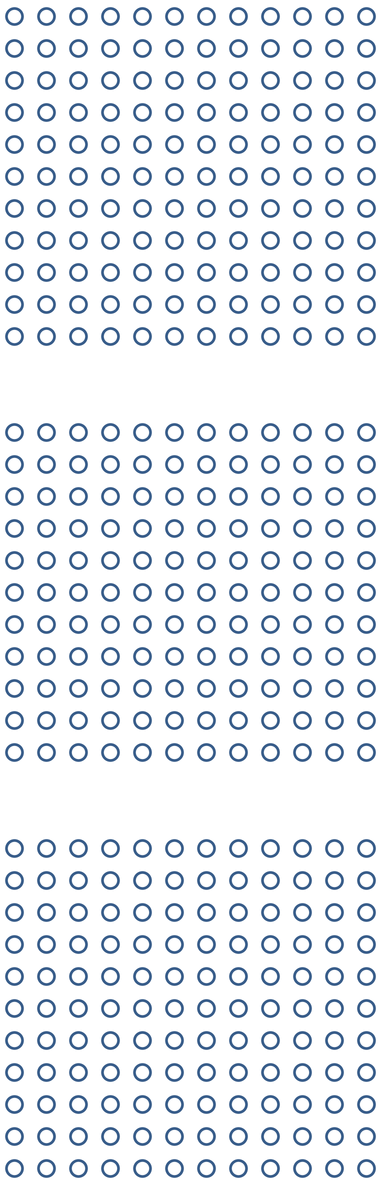
- Sometimes it is useful to visualize a larger number of units.
- In that case, we just show them as dots, or little circles, or something of that sort…
- So, the image on the left still shows the input layer:
  - R*C*3 units, corresponding to an RGB image with R rows and C columns.

Visualizing the Input Layer in 3D



- Before we define convolutional layers in their general forms, we will see a specific example.
- In this example, the convolutional layer will be the second layer, right after the input layer.
- The convolutional layer is based on (but not identical to) a convolution operation.

Example of a Convolutional Layer

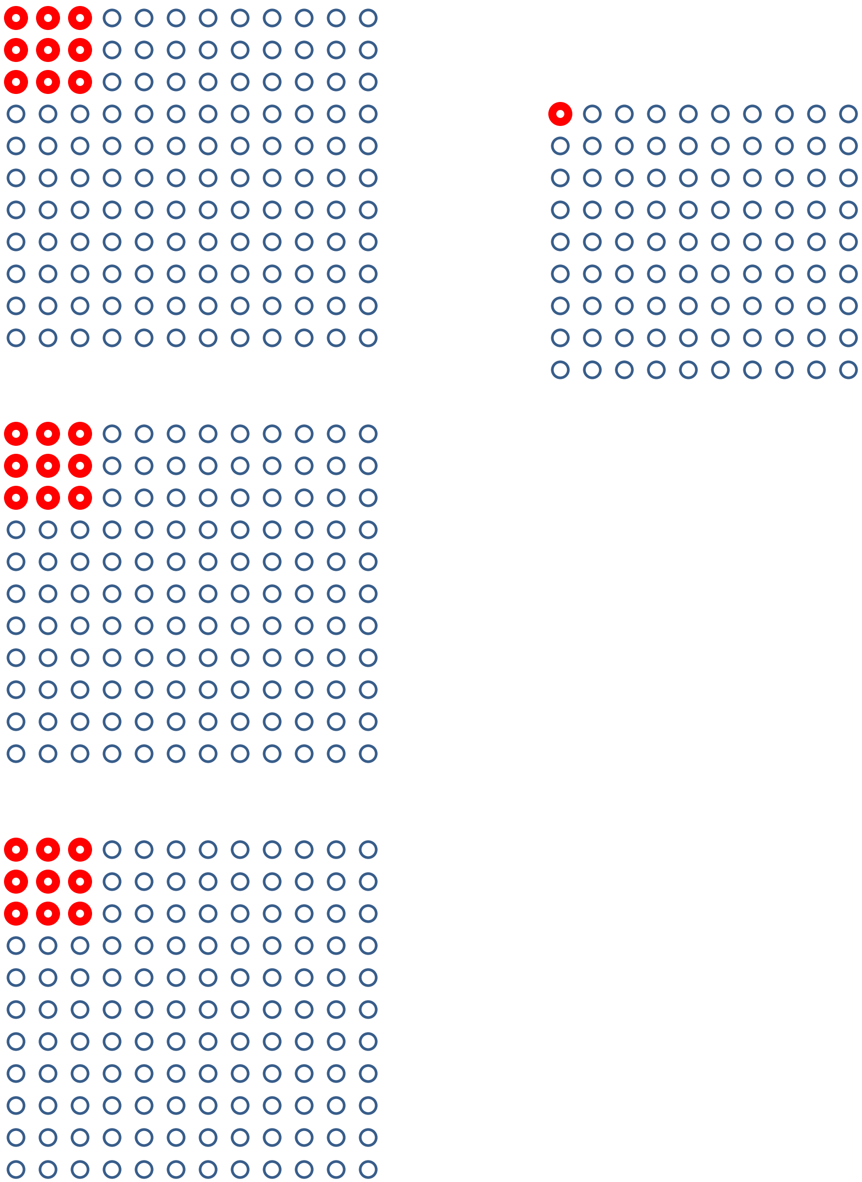


- A convolution is defined by an M*N*C array of weights.
- So far we have seen 2D convolutions, applied to grayscale images.
  - To define a 2D convolution, we need an M*N array of weights.
- We can extend this idea to convolutions applied to an RGB image, by using M*N*3 filters.
- In our example, we will use M=3 and N=3.
- Let's see what happens with a single such filter.

Example of a Convolutional Layer

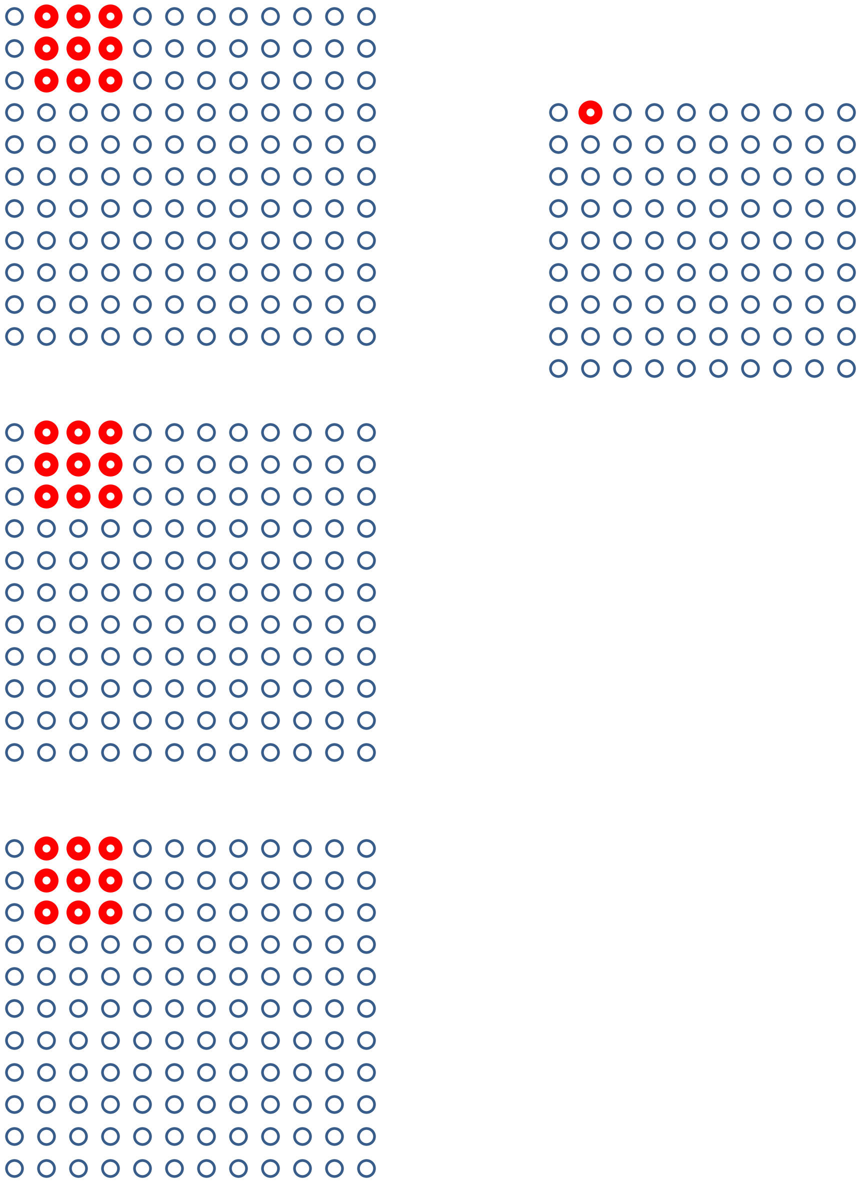
- We are using a 3*3*3 filter.
- In the result, we will throw away boundary values, where part of the filter does not align with any image pixels.
- So, if the image has R rows and C columns, the result will have R-2 rows and C-2 columns.

Example of a Convolutional Layer



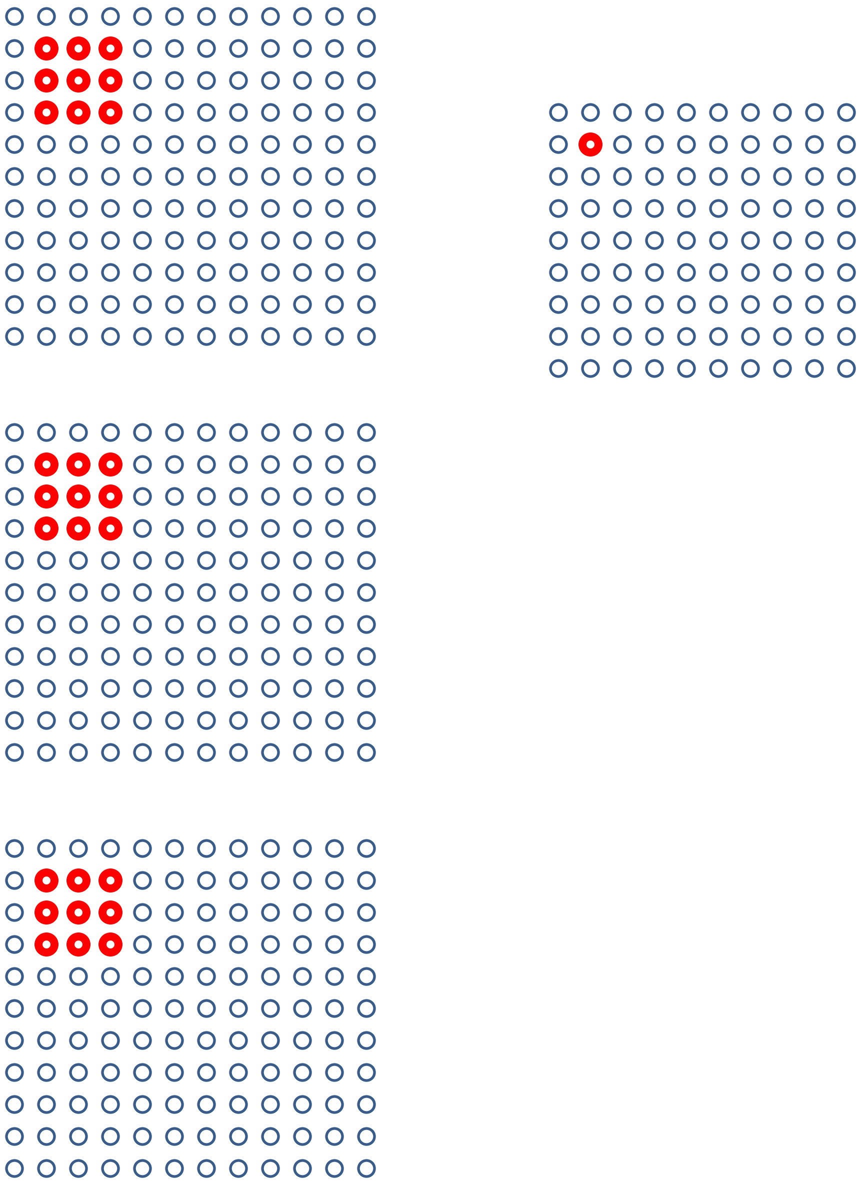
- The convolution result at location (1,1) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of a Convolutional Layer



- Similarly, the convolution result at location (1,2) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of a Convolutional Layer



- Similarly, the convolution result at location (2,2) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: ??x??x??

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

- How do we compute result(1,1)?

Example With Numbers

Input image: 7x9x3

58	**17**	**75**	9	51	54	21	18	91
6	**65**	**19**	93	52	36	31	23	98
24	**74**	**69**	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	**99**	**34**	78	4	86	49	69	73
24	**4**	**68**	72	75	81	17	5	15
49	**89**	**14**	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	**74**	**31**	13	20	38	39	99	82
20	**27**	**71**	99	37	20	59	74	27
49	**43**	**67**	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849						

- How do we compute result(1,1)?
- We must sum up all these values:

58*5	17*4	75*1
6*2	65*1	19*2
24*6	74*9	69*7

53*7	99*2	34*5
24*6	4*1	68*5
49*9	89*3	14*6

38*9	74*8	31*5
20*6	27*4	71*3
49*4	43*3	67*2

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	???					

- How do we compute result(1,2)?

Example With Numbers

Input image: 7x9x3

58	**17**	**75**	**9**	51	54	21	18	91
6	**65**	**19**	**93**	52	36	31	23	98
24	**74**	**69**	**78**	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	**99**	**34**	**78**	4	86	49	69	73
24	**4**	**68**	**72**	75	81	17	5	15
49	**89**	**14**	**91**	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	**74**	**31**	**13**	20	38	39	99	82
20	**27**	**71**	**99**	37	20	59	74	27
49	**43**	**67**	**18**	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	**7463**					

- How do we compute result(1,2)?
- We must sum up all these values:

17*5	75*4	9*1
65*2	19*1	93*2
74*6	69*9	78*7

99*7	34*2	78*5
4*6	68*1	72*5
89*9	14*3	91*6

74*9	31*8	13*5
27*6	71*4	99*3
43*4	67*3	18*2

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	7463	6193	7261	6177	6309	6484
5580	7161	7311	7902	6699	5329	5375
6690	7301	6211	6464	7450	5865	6553
6826	7003	6740	6804	7072	5875	5869
6917	6605	6838	6247	6121	5410	6179

- This way we can compute all values in the result.

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	7463	6193	7261	6177	6309	6484
5580	7161	7311	7902	6699	5329	5375
6690	7301	6211	6464	7450	5865	6553
6826	7003	6740	6804	7072	5875	5869
6917	6605	6838	6247	6121	5410	6179

- How do these operations correspond to a layer in a neural network?

Example With Numbers

Input image: 7x9x3

58	**17**	**75**	9	51	54	21	18	91
6	**65**	**19**	93	52	36	31	23	98
24	**74**	**69**	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	**99**	**34**	78	4	86	49	69	73
24	**4**	**68**	72	75	81	17	5	15
49	**89**	**14**	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	**74**	**31**	13	20	38	39	99	82
20	**27**	**71**	99	37	20	59	74	27
49	**43**	**67**	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849						

- Consider the value at position (1,1) of the result.
- How do we obtain that value?
- It is a dot product between a subset of the input numbers, and a specific set of weights.
- This like having a perceptron.
  - What are the inputs of the perceptron?
  - What are the weights of the perceptron?

Example With Numbers

Input image: 7x9x3

58	**17**	**75**	9	51	54	21	18	91
6	**65**	**19**	93	52	36	31	23	98
24	**74**	**69**	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	**99**	**34**	78	4	86	49	69	73
24	**4**	**68**	72	75	81	17	5	15
49	**89**	**14**	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	**74**	**31**	13	20	38	39	99	82
20	**27**	**71**	99	37	20	59	74	27
49	**43**	**67**	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849						

- Consider the value at position (1,1) of the result.
- How do we obtain that value?
- It is a dot product between a subset of the input numbers, and a specific set of weights.
- This like having a perceptron.
  - Perceptron inputs are the red values
  - Perceptron weights are the filter values.

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	7463	6193	7261	6177	6309	6484
5580	7161	7311	7902	6699	5329	5375
6690	7301	6211	6464	7450	5865	6553
6826	7003	6740	6804	7072	5875	5869
6917	6605	6838	6247	6121	5410	6179

- So, the convolution operation can be thought of as defining the second layer of a neural network.
  - Each unit in that layer is connected to 27 units in the input layer.
  - **ALL UNITS HAVE IDENTICAL WEIGHTS,** specified by the values in the 3x3x3 filter.

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	7463	6193	7261	6177	6309	6484
5580	7161	7311	7902	6699	5329	5375
6690	7301	6211	6464	7450	5865	6553
6826	7003	6740	6804	7072	5875	5869
6917	6605	6838	6247	6121	5410	6179

- However, there is an important difference between a convolution operation and the output of a neural network layer.
  - The convolution corresponds to taking, for each perceptron, the dot product between inputs and weights.
  - Does a perceptron output just that dot product?

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	7463	6193	7261	6177	6309	6484
5580	7161	7311	7902	6699	5329	5375
6690	7301	6211	6464	7450	5865	6553
6826	7003	6740	6804	7072	5875	5869
6917	6605	6838	6247	6121	5410	6179

- However, there is an important difference between a convolution operation and the output of a neural network layer.
  - The convolution corresponds to taking, for each perceptron, the dot product between inputs and weights.
  - The output of a perceptron is obtained by applying an activation function to the dot product.

ReLU Activation

- ReLU stands for “Rectified Linear Unit”.
- ReLU is a commonly used activation function for convolutional layers.
- The definition is simple: $h(x) = \max(x, 0)$
  - If the dot product between inputs and weights is negative, the perceptron outputs 0.
  - If the dot product is positive, the perceptron outputs the dot product itself.

Convolutional Layers

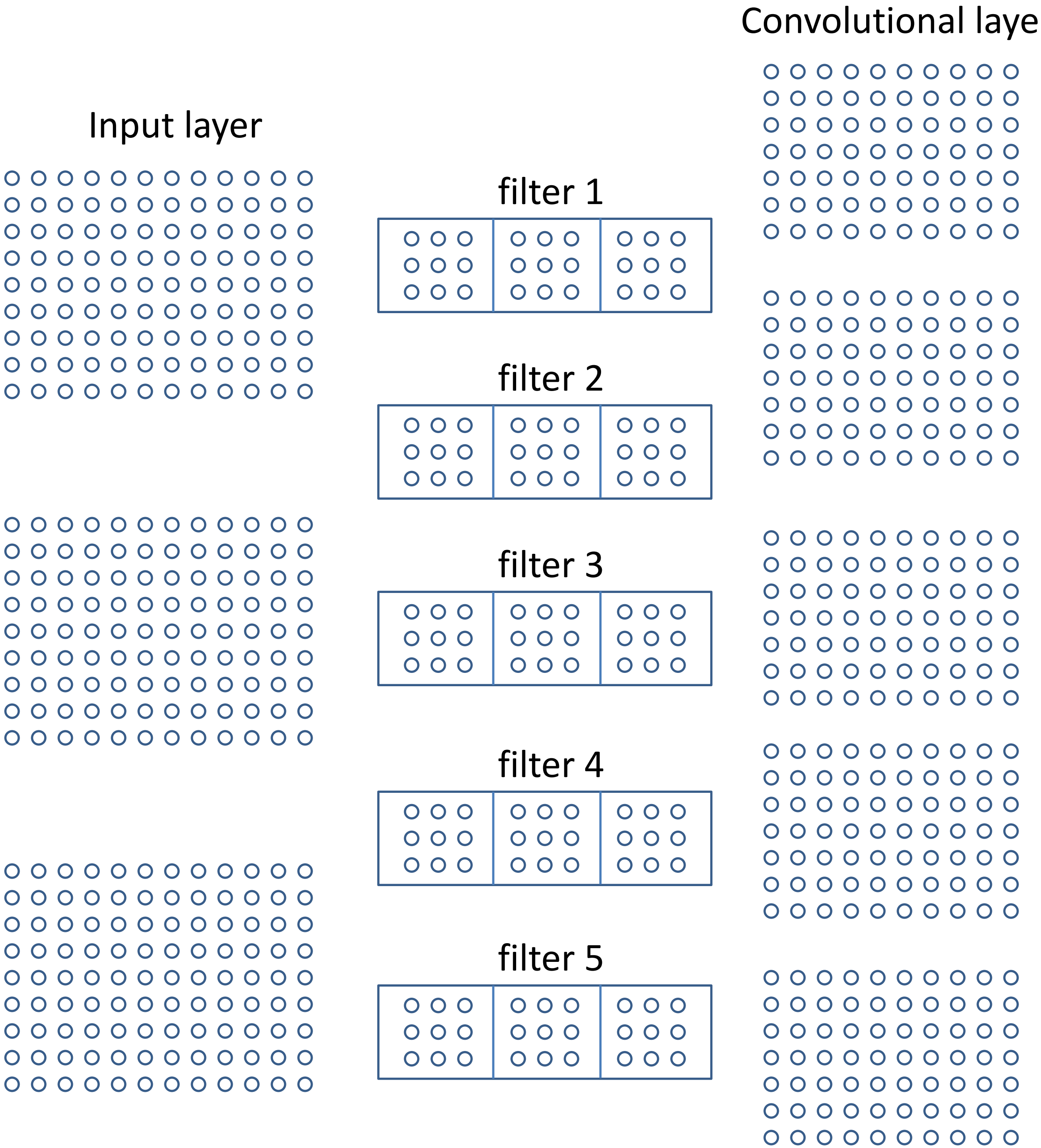
- So, a convolution can be thought of as specifying a neural network layer.
- The filter values specify the weights of every single perceptron in the layer.
- However, remember that **<u>the output of a convolutional layer is not the result of the convolution</u>**.
  - Each dot product passes through an activation function (usually ReLU) in order to produce an output.

Applying Multiple Convolutions

- In our previous example:
  - We started with a 3-channel input image (an RGB image).
    - This defined a 3D array of input units, of dimensions R*C*3.
  - We applied a single 3x3x3 convolutional filter.
  - We obtained a 1-channel output.
    - This defined a 2D array of units in the convolutional layer.
- We can apply multiple convolutions at the same time.
  - We can apply K different 3x3x3 convolutional filters.
  - This leads to a K-channel output.
    - This defines a 3D array of units in the convolutional layer, of dimensions (R-2)*(C-2)*K.
    - Each 2D slice of that array corresponds to a single 3x3x3 filter.

Example of Multiple Convolutions

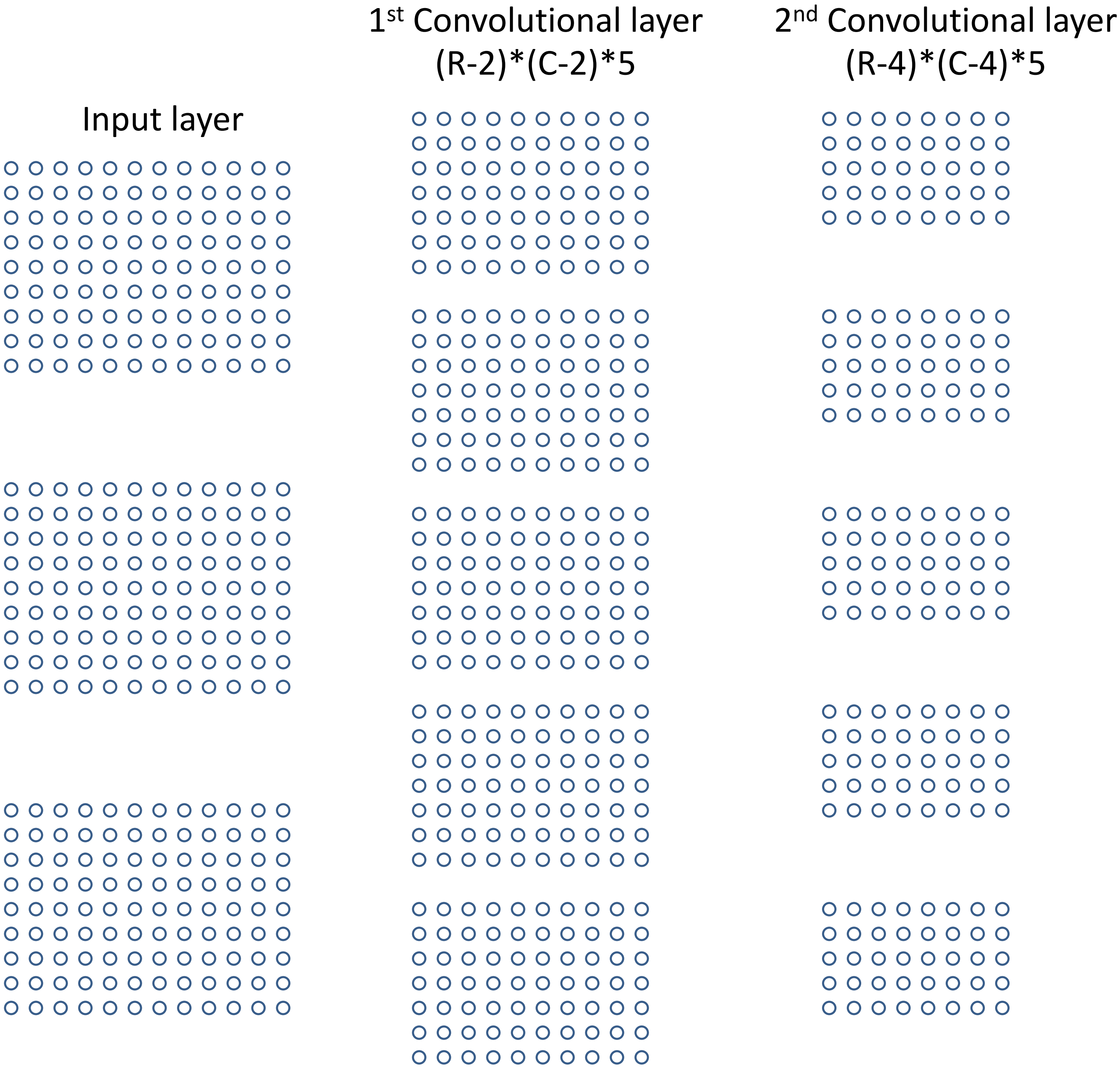
- In our previous example:
  - We started with a 3-channel input image (an RGB image).
    - This defined a 3D array of input units, of dimensions R*C*3.
  - We applied a single 3x3x3 convolutional filter.
  - We obtained a 1-channel output.
    - This defined a 2D array of units in the convolutional layer.
- We can apply multiple convolutions at the same time.
  - We can apply K different 3x3x3 convolutional filters.
  - This leads to a K-channel output.
    - This defines a 3D array of units in the convolutional layer, of dimensions (R-2)*(C-2)*K.
    - Each 2D slice of that array corresponds to a single 3x3x3 filter.



- R*C*3 input units (one unit for each pixel value).
- (R-2)*(C-2)*5 units in the convolutional layer.
- The convolutional layer consists of five 2D arrays of units.
- Each such 2D array corresponds to a 3x3x3 filter.

Multiple Convolutional Layers

- In our previous example:
  - We started with a 3-channel input image (an RGB image).
    - This defined a 3D array of input units, of dimensions R*C*3.
  - We applied 5 3x3x3 convolutional filters.
  - We obtained a 5-channel output.
    - This defined a convolutional layer with (R-2)*(C-2)*5 units.
- We can put a new convolutional layer after the previous one.
  - We can define K 3x3x5 filters.
    - K is whatever we want.
    - 5 is there to match the number of channels in the previous layer.
  - We end up with a layer of size (R-4)*(C-4)*K units.

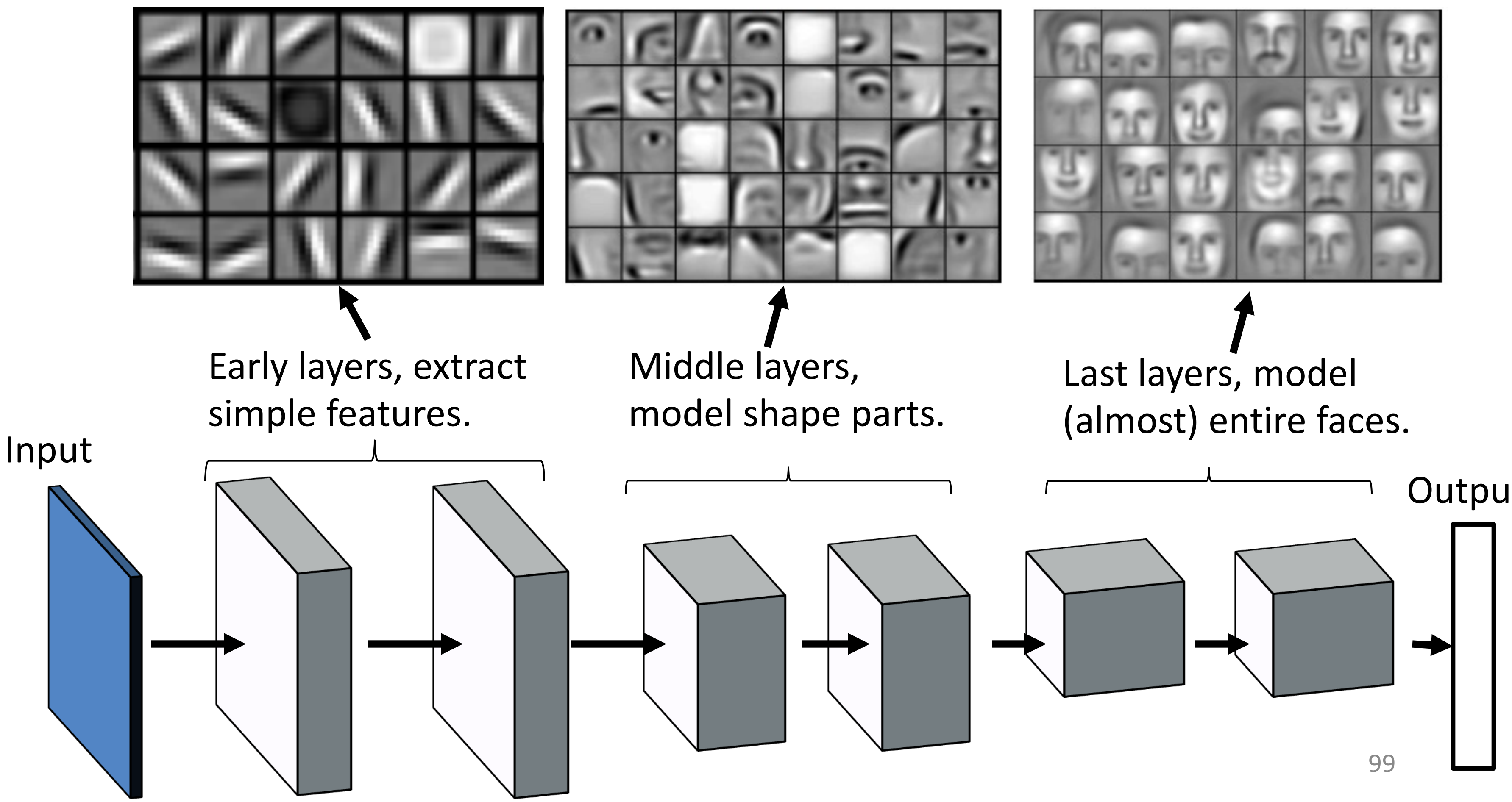

1st Convolutional layer
(R-2)*(C-2)*5
2nd Convolutional layer
(R-4)*(C-4)*5
Input layer

Multiple Convolutional Layers

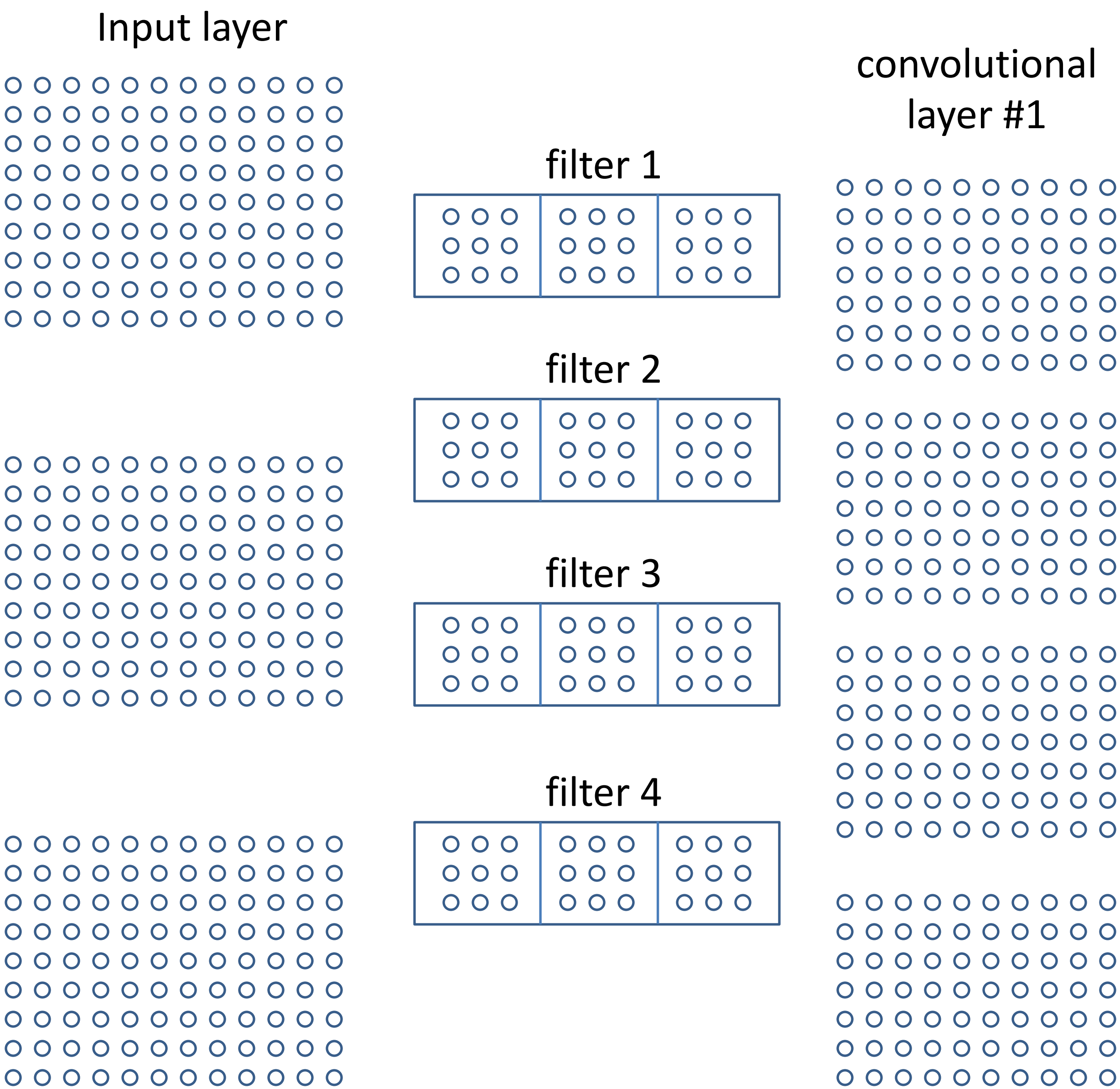
- Overall, we can put as many convolutional layers in sequence as we want.

CNN Visualization

- Visualization of a deep neural network trained with face images.
  - Credit for feature visualizations: [Lee09] Honglak Lee, Roger Grosse, Rajesh Ranganath and Andrew Y. Ng., ICML 2009.

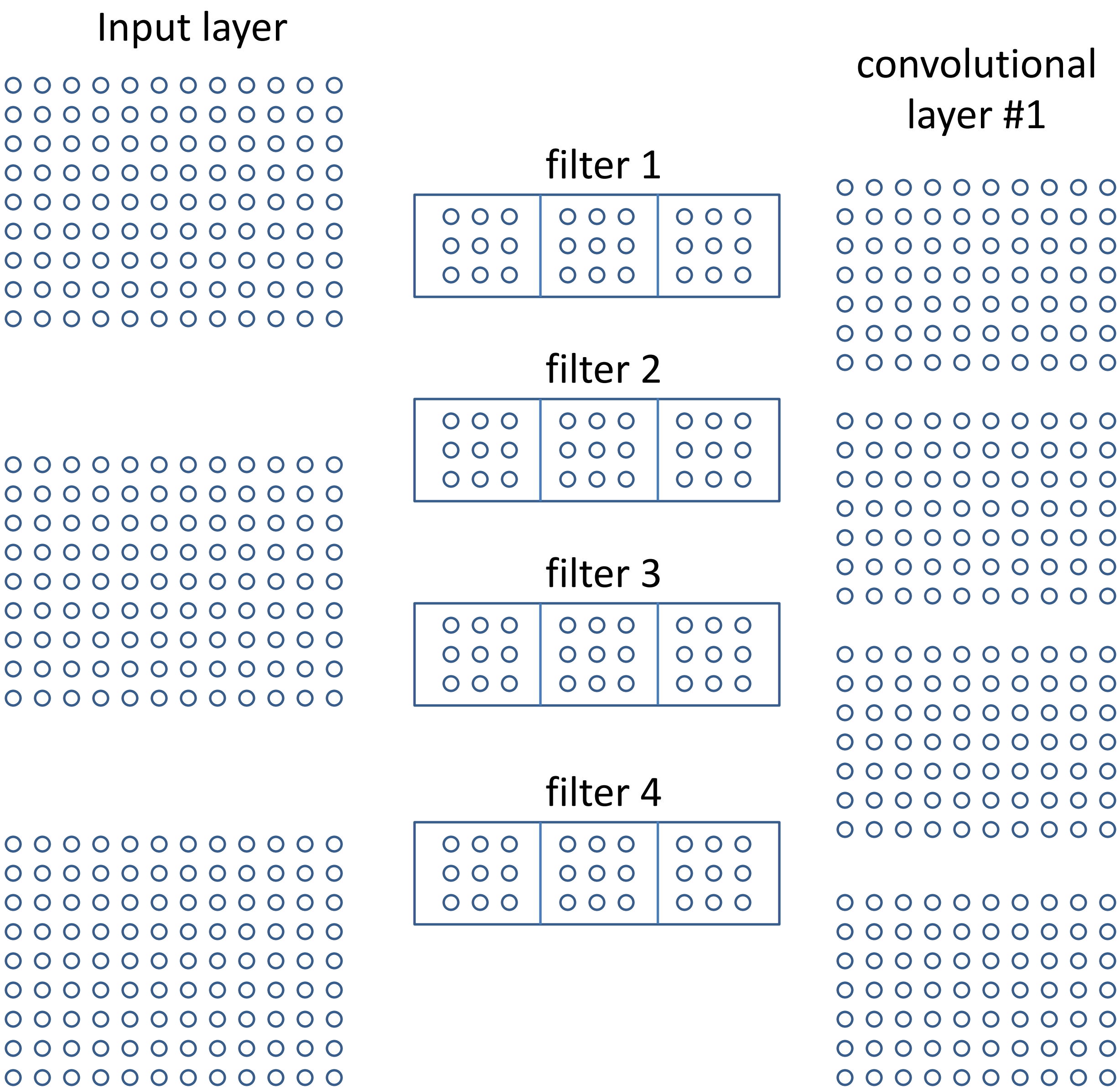


Convolutional Vs. Fully Connected



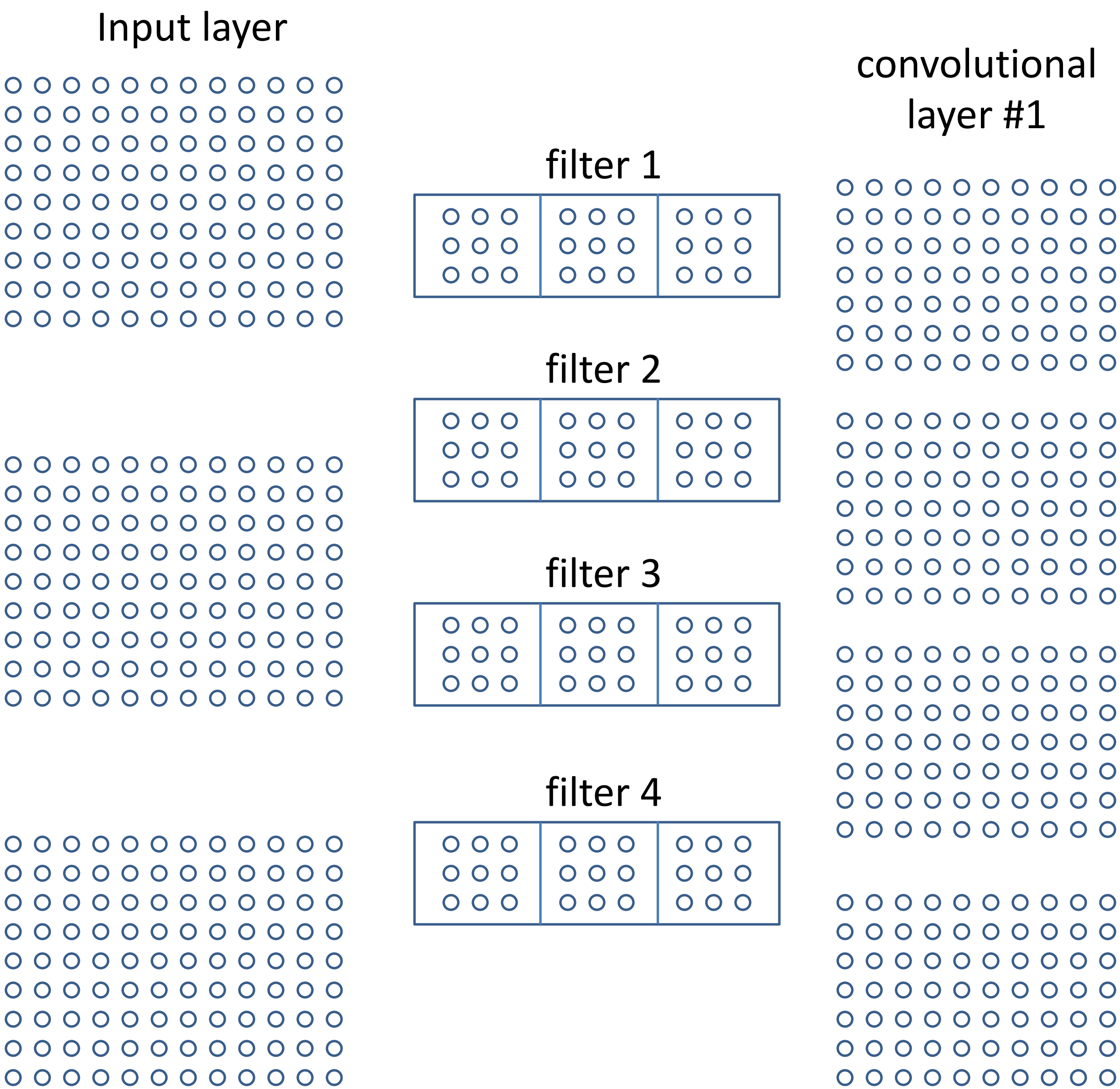
- R*C*3 input units (one unit for each pixel value).
- (R-2)*(C-2)*4 units in the convolutional layer.
- How many weights do we need to learn for this layer?
- How many weights would we need to learn for a fully-connected layer with the same number of units?

Convolutional Vs. Fully Connected



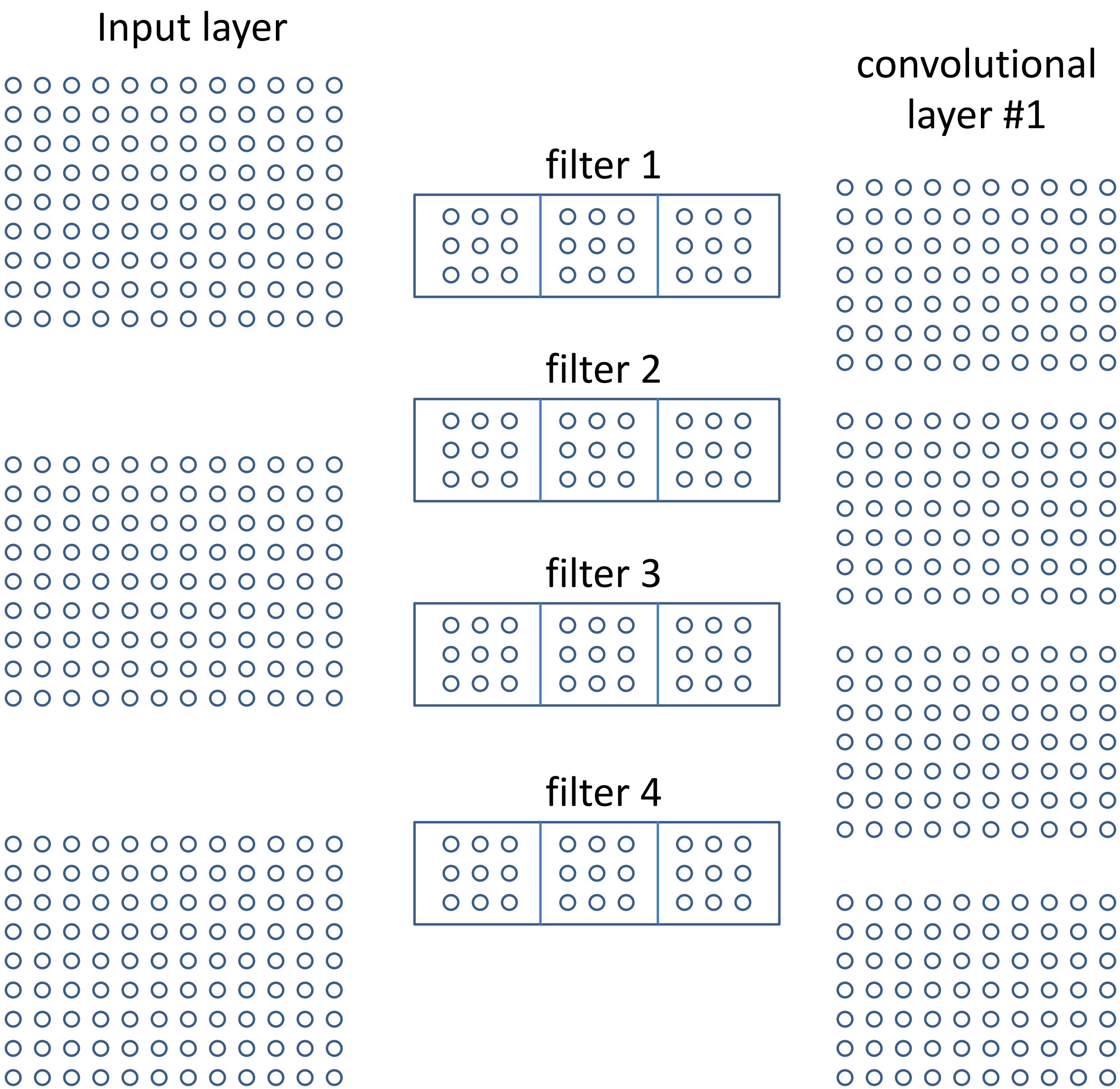
- R*C*3 input units (one unit for each pixel value).
- (R-2)*(C-2)*4 units in the convolutional layer.
- The convolutional layer requires learning four filters.
- Each filter has 3x3x3=27 weights.
- So, in total we need to learn 4*27 = 108 weights.

Convolutional Vs. Fully Connected



- R*C*3 input units (one unit for each pixel value).
- (R-2)*(C-2)*4 units in the convolutional layer.
- For a fully connected layer, there is an edge connecting each input unit to each fully connected layer unit.
- Number of edges: R*C*3*(R-2)*(C-2)*4.
- We need to learn that many weights.
- For a 400x500 image: 475 billion weights.

Convolutional Vs. Fully Connected



- So, for the convolutional layer we need to learn 108 weights.
- For a similarly sized fully connected layer we would need 475 billion weights.
- Thus, convolutional layers are much more practical to use in terms of:
  - Storage
  - training time
  - runtime on test data
  - amount of training data required.

Max Pooling Layer

- A max pooling layer is a layer where every unit corresponds to a 2x2 (or 3x3) neighborhood in the previous layer, and the output is the maximum value in that neighborhood.

Max Pooling Example

Input image: 7x7x3

58	17	75	9	51	54	21
6	65	19	93	52	36	31
24	74	69	78	82	94	48
36	65	19	49	80	88	24
83	46	37	44	65	56	85
2	55	63	45	38	63	20
5	30	79	31	82	59	23

53	99	34	78	4	86	49
24	4	68	72	75	81	17
49	89	14	91	51	58	98
63	92	73	90	48	19	72
68	80	11	34	91	24	51
40	10	66	70	61	89	48
37	27	50	20	62	3	6

38	74	31	13	20	38	39
20	27	71	99	37	20	59
49	43	67	18	47	43	26
34	55	54	4	99	49	30
96	95	70	57	16	13	62
93	42	67	89	86	59	27
6	99	18	67	65	23	83

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Convolutional Layer Output: 5x5x1

5849	7463	6193	7261	6177
5580	7161	7311	7902	6699
6690	7301	6211	6464	7450
6826	7003	6740	6804	7072
6917	6605	6838	6247	6121

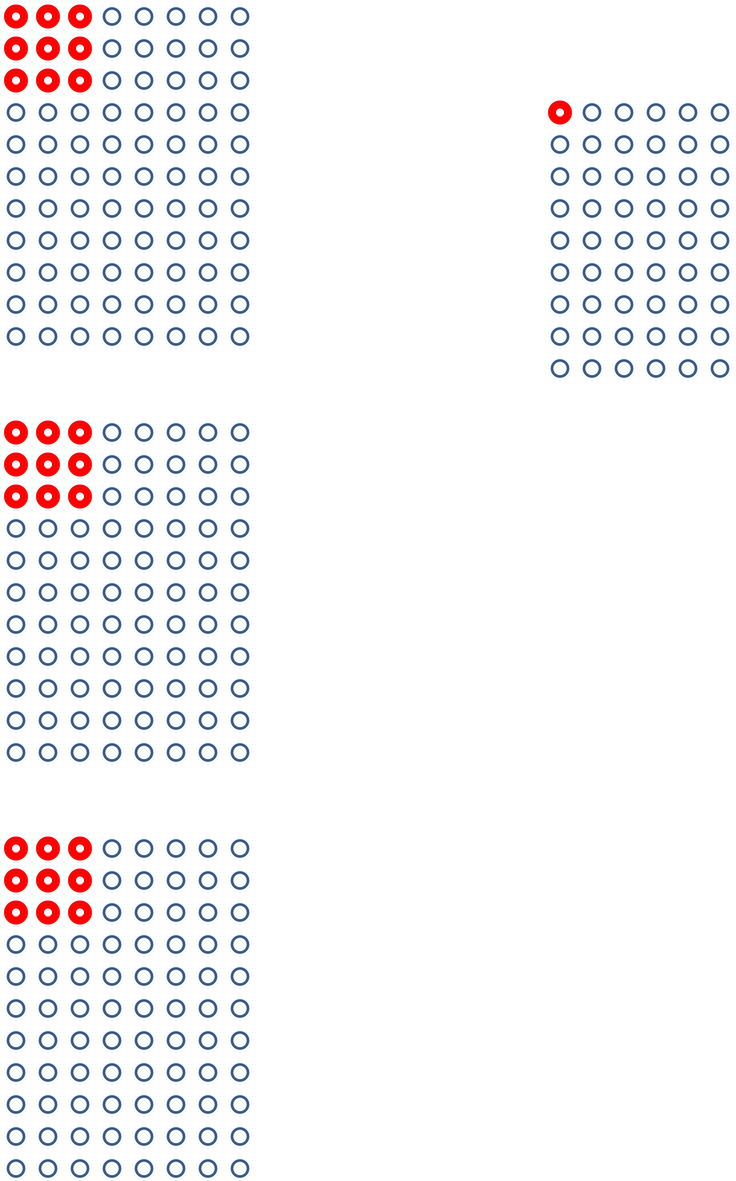
Max Pool Layer Output: 3x3x1

7463	7902	6699
7301	6804	7450
6917	6838	6121

Stride

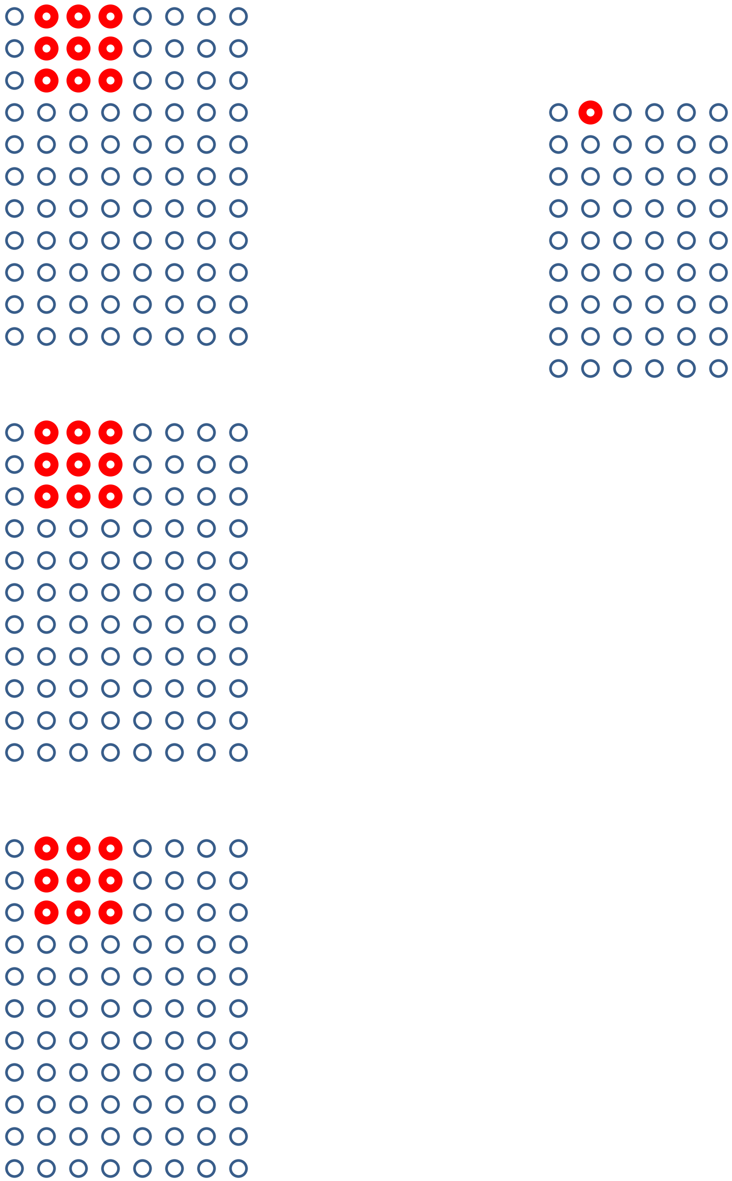
- For each convolutional layer, we can pick a **stride** *S.*
- *S* is a single number, that influences the number of outputs of the convolutional layer.
  – It determines how many locations to shift the filter to the left or to the bottom to compute the next value of the convolution result.
- Stride = 1: we shift the filter one location at a time, so no rows or columns are skipped.
- Stride = 2: we shift the filter two locations at a time. Half the rows and half the columns are skipped.
  – This means that the convolutional layer will have about half the number of rows and columns of the previous layer.

Example of Stride 1



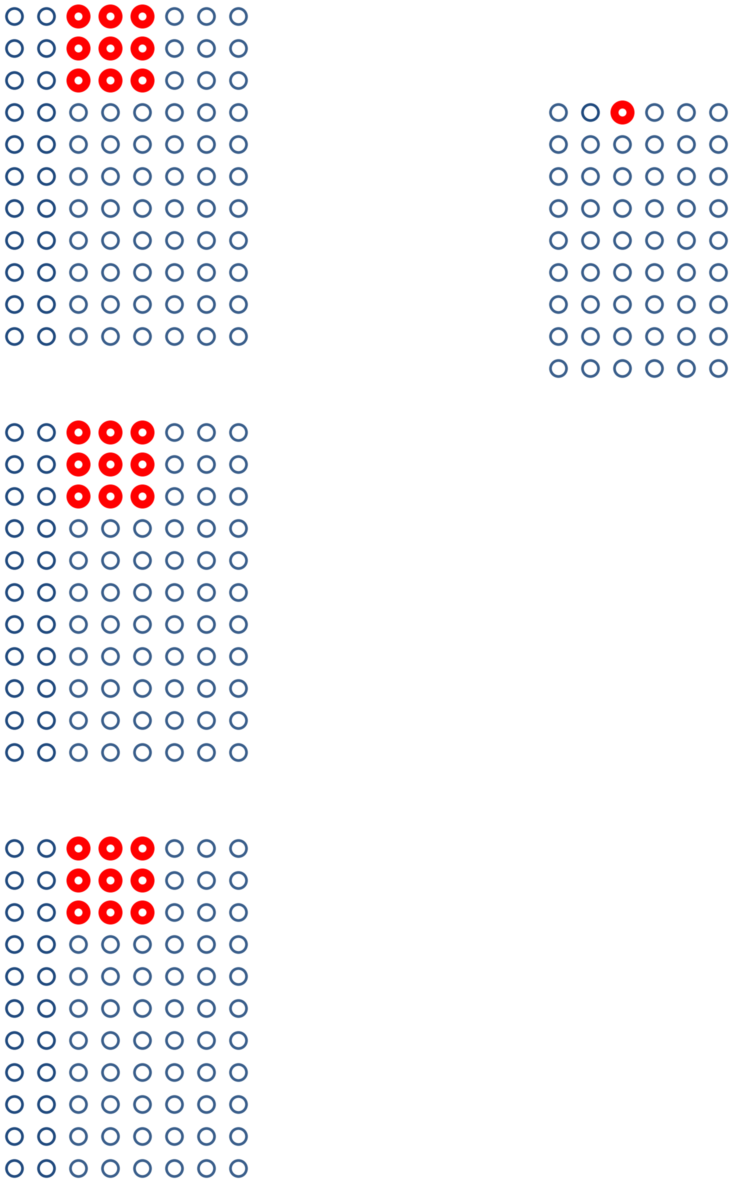
- The convolution result at location (1,1) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of Stride 1



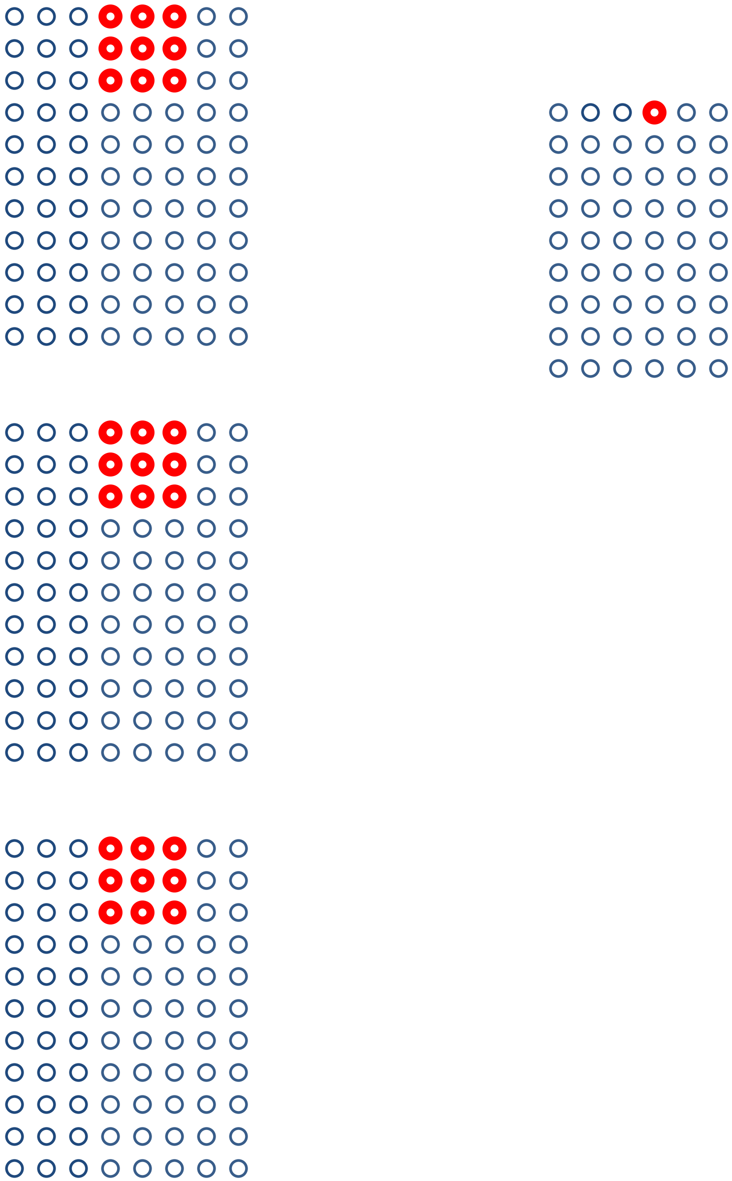
- The convolution result at location (1,2) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of Stride 1



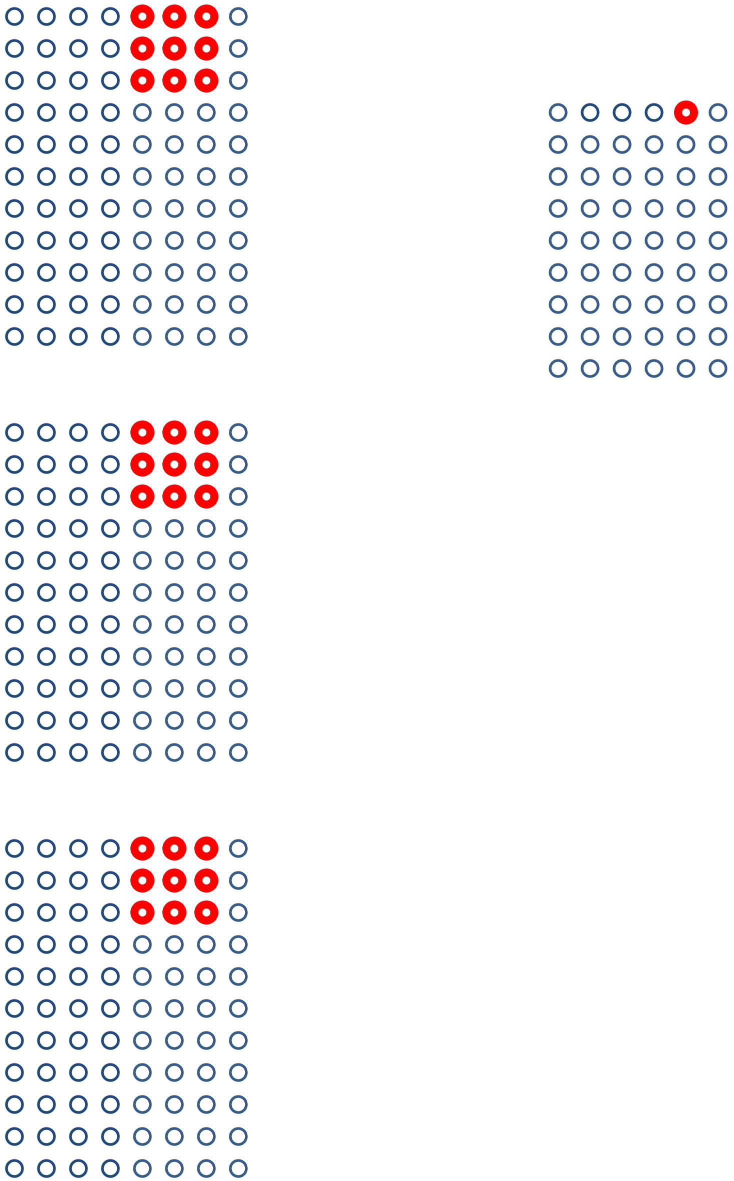
- The convolution result at location (1,3) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of Stride 1



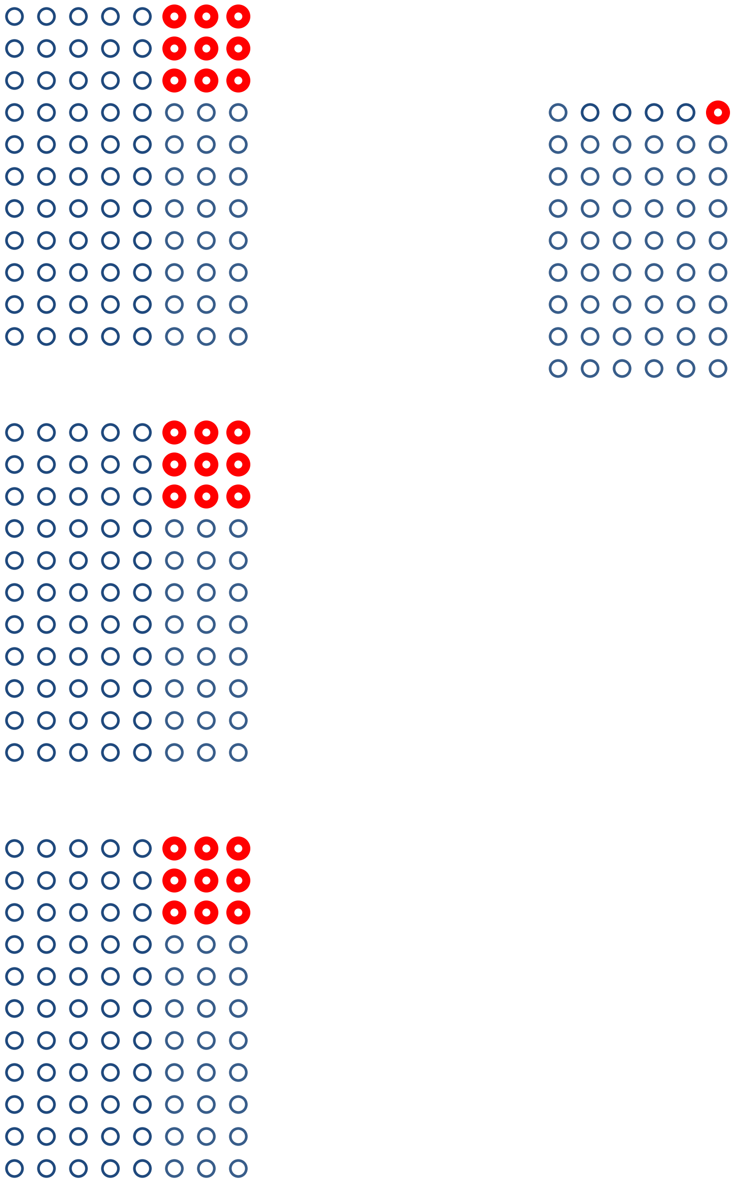
- The convolution result at location (1,4) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of Stride 1



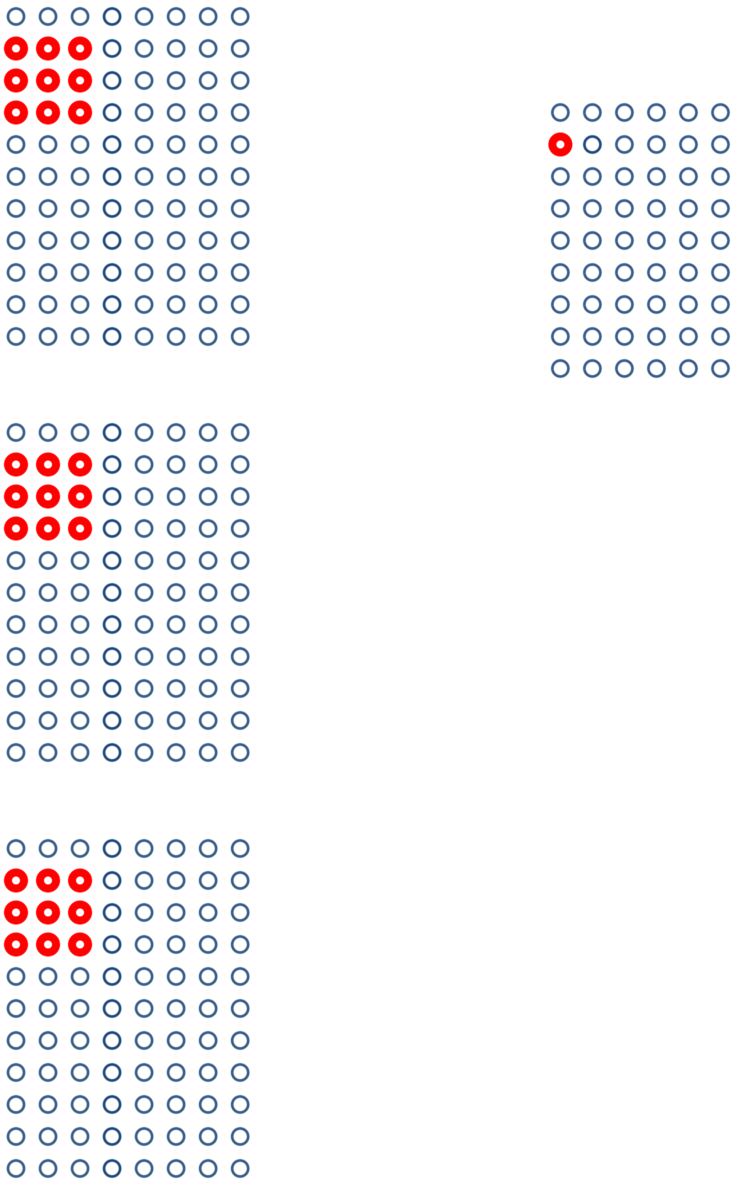
- The convolution result at location (1,5) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of Stride 1



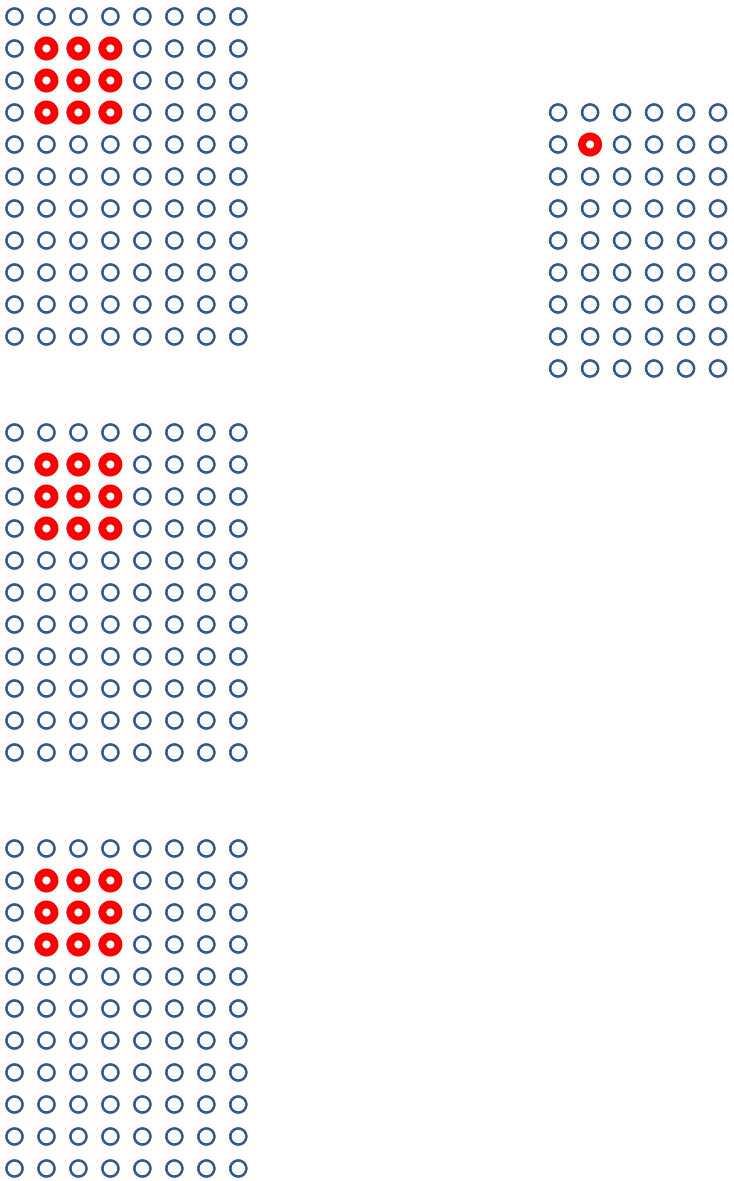
- The convolution result at location (1,6) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of Stride 1



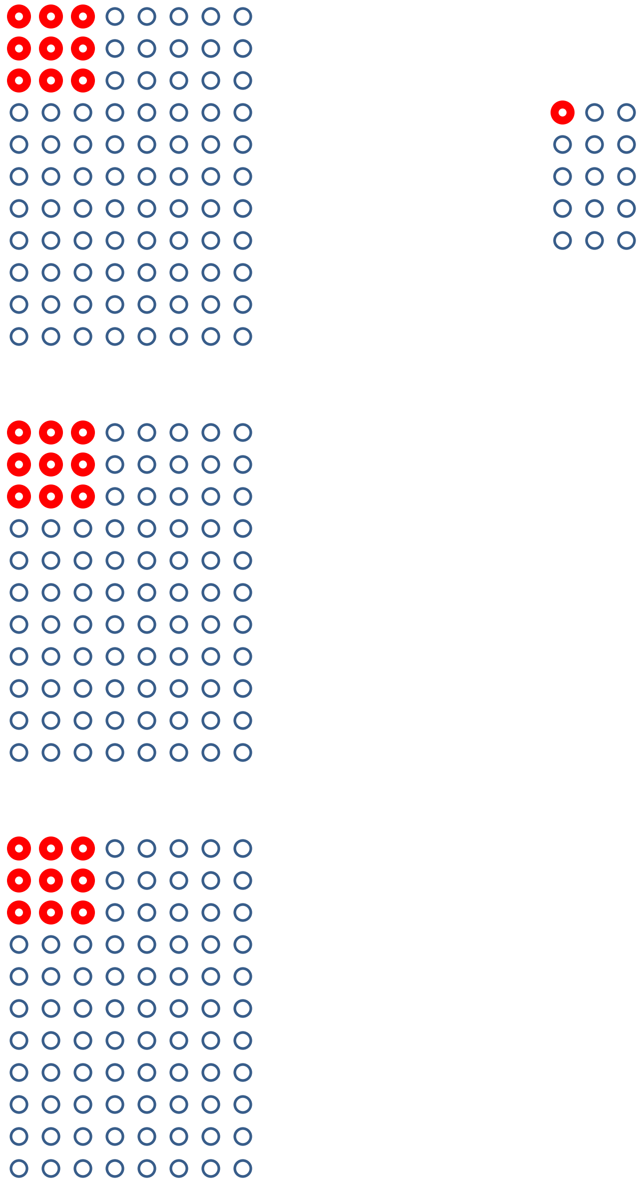
- The convolution result at location (2,1) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of Stride 1



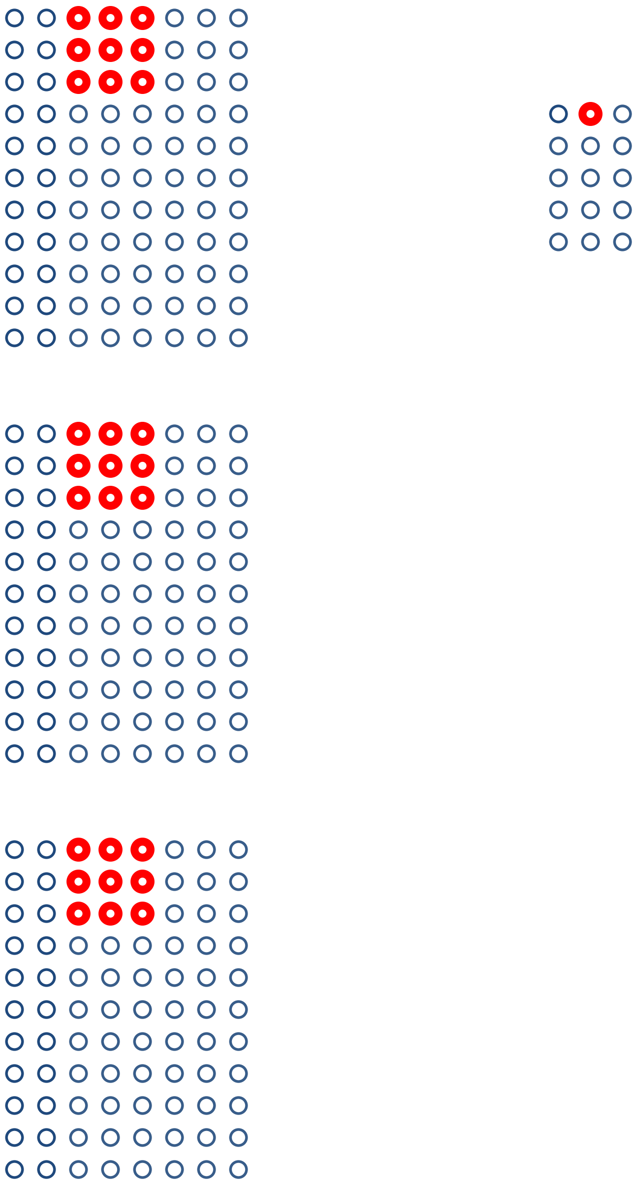
- The convolution result at location (2,2) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.
- And so on…

Example of Stride 2



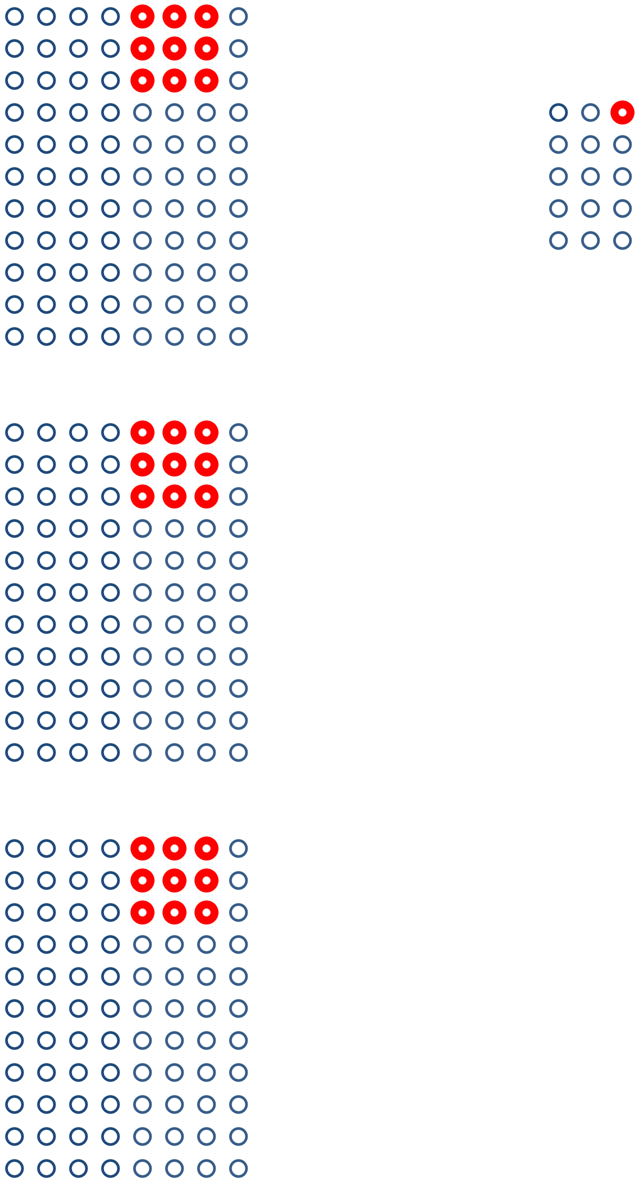
- The convolution result at location (1,1) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of Stride 2



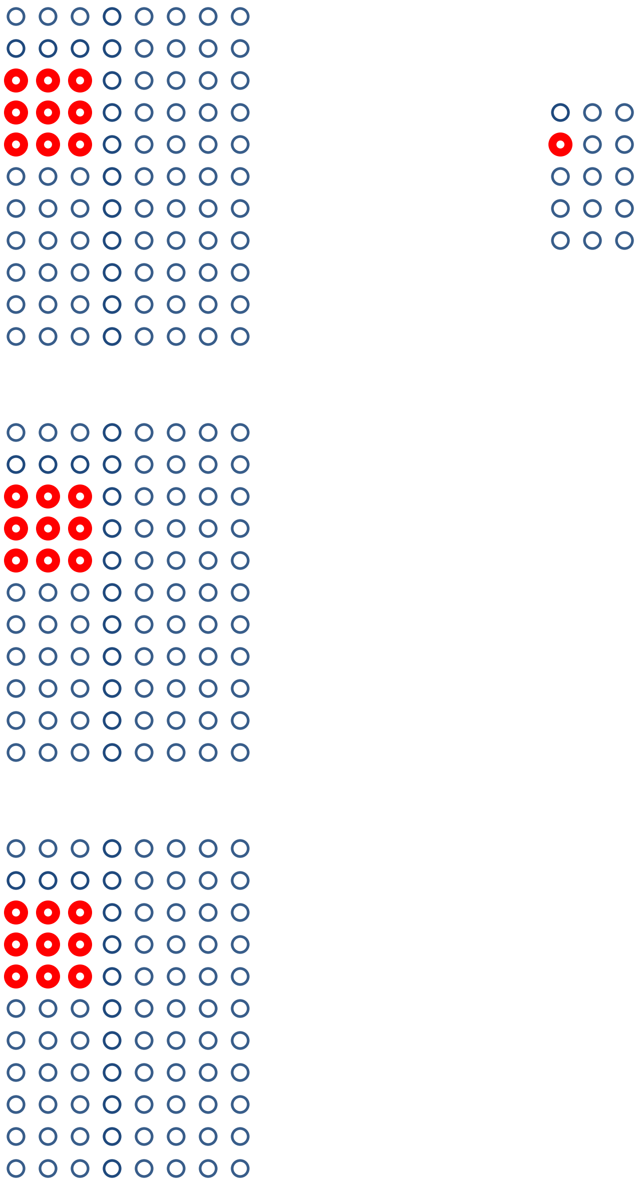
- The convolution result at location (1,2) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of Stride 2



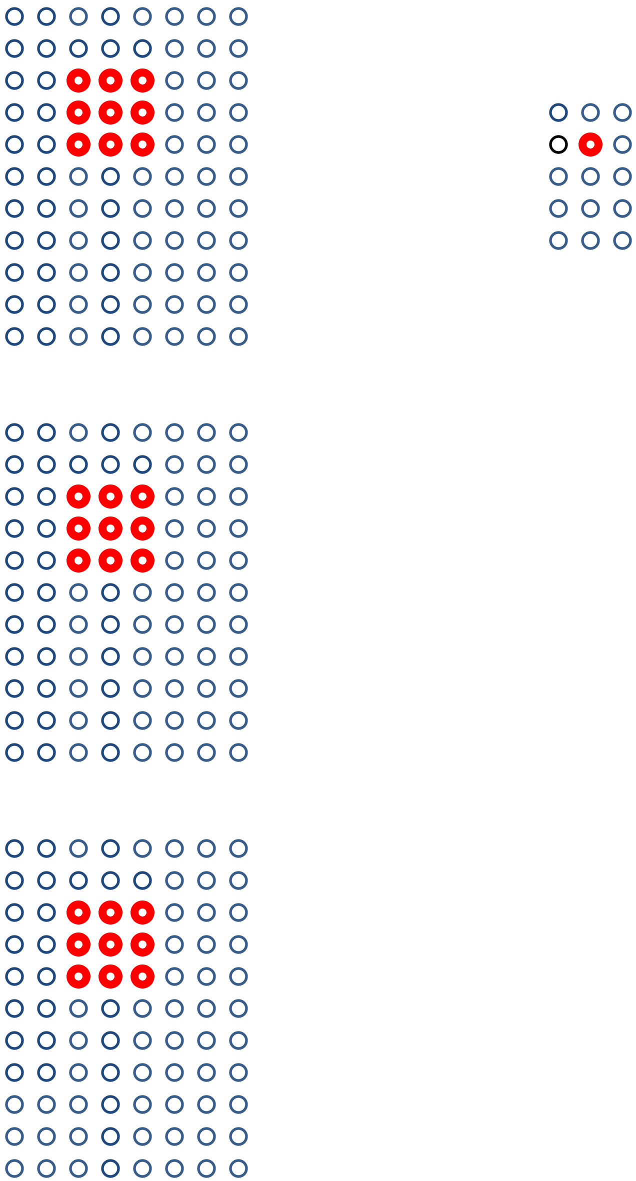
- The convolution result at location (1,3) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of Stride 2



- The convolution result at location (2,1) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

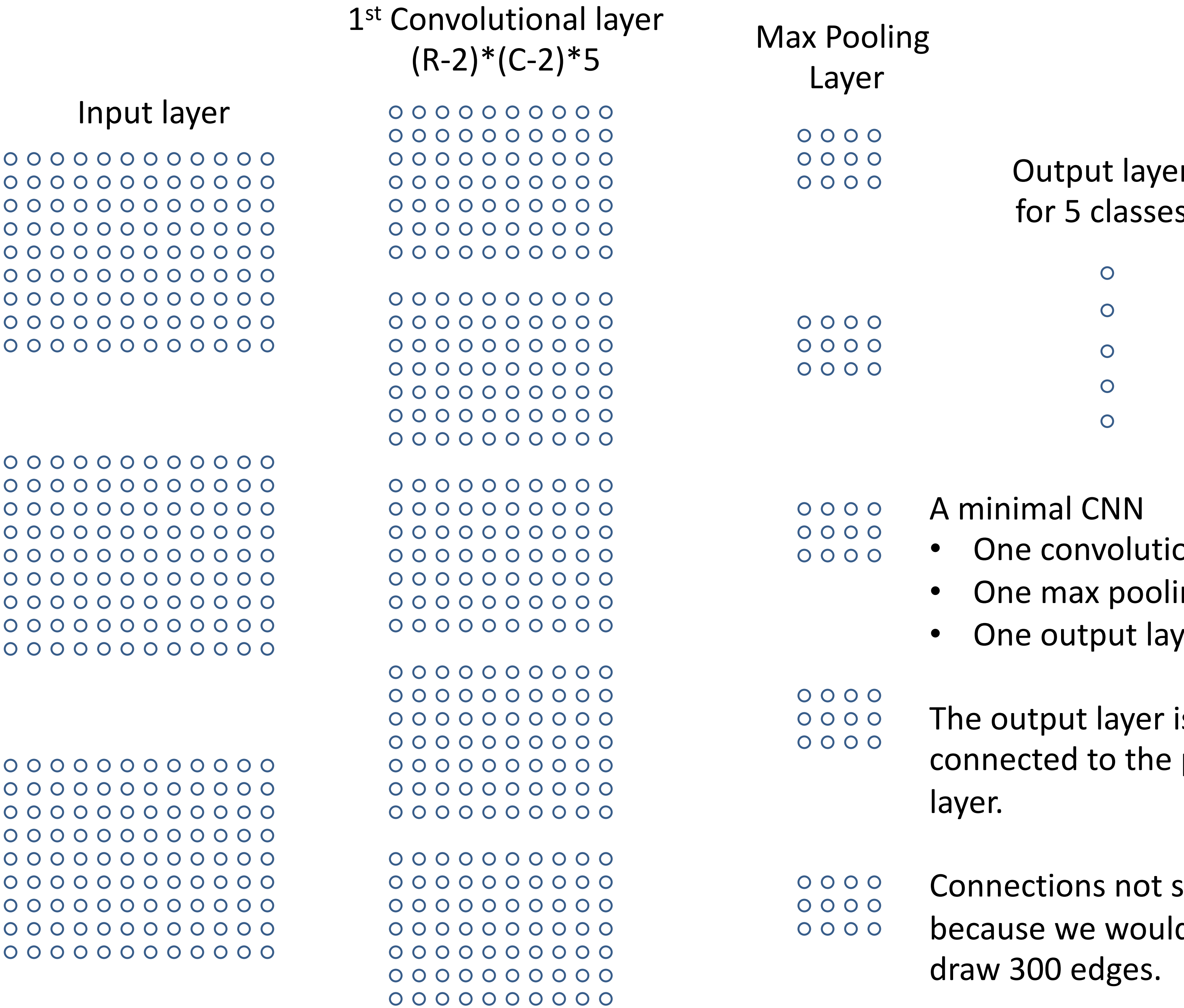
Example of Stride 2



- The convolution result at location (2,2) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Fully Connected Output Layer

- The output layer has as many units as the number of classes.
- The layer preceding the output layer is called the **last hidden layer**.
- Usually the output layer is fully connected to its previous layer.
  - “Fully connected” means that all units in the output layer take as input all outputs of the last hidden layer.
- The last hidden layer can be a convolutional layer or a max pooling layer.



A minimal CNN

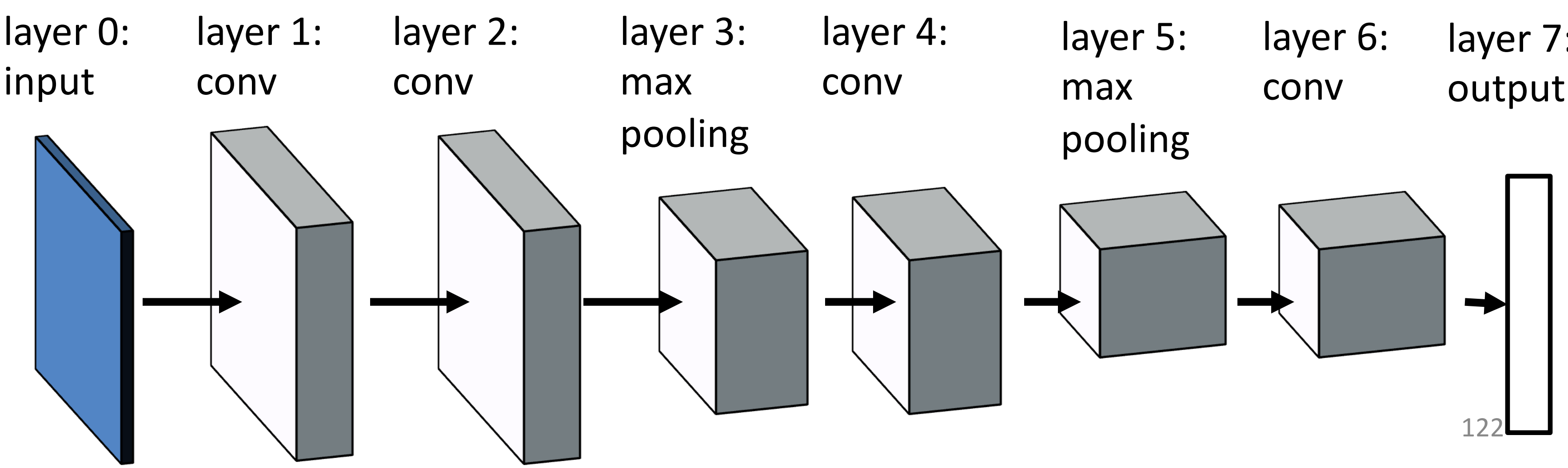
- One convolutional layer.
- One max pooling layer
- One output layer.

The output layer is fully connected to the previous layer.

Connections not shown because we would need to draw 300 edges.

Example of a CNN Layout

- This is an example of a CNN with six hidden layers.
  - Four convolutional layers with stride 1.
  - Two max pooling layers.



Transfer Learning

- ImageNet is a public dataset has over 14 million images, over 20,000 classes (such as "balloon" or "strawberry").
  - http://www.image-net.org/
  - https://en.wikipedia.org/wiki/ImageNet
- CNNs can be (and have been) trained on such large datasets.
- Suppose that you have some training images of three animals that were recently discovered.
  - Let's say, 100 images for each animal.
  - You want a classifier that recognizes each of the two animals.
- Is the ImageNet data useful?
  - ImageNet does not contain images of those two animals.

Transfer Learning

- Simple approach: train a classifier on your 300 images.
- 300 images is a rather small dataset, so you would use a relatively simple model.
  - Complicated models would probably overfit.
  - At the same time, a simple model would probably have limited accuracy.

Transfer Learning

- Transfer learning approach:
  - 1st step: train a CNN on ImageNet.
    - Actually, these days you can just download a pre-trained model, so you do not have to do the training yourself.
  - 2nd step: throw away the output layer of the CNN.
    - The last layer recognizes ImageNet classes, which you don’t care about.
  - 3rd step: create a new output layer with three units, and connect it to the last hidden layer of the pre-trained CNN.
  - 4th step: train **just the weights of the new output layer** with your new dataset.

Transfer Learning

- Why is it called “transfer learning”?
  – Because, in order to learn a classifier for our three animals, we are transferring information learned from training examples (ImageNet) that did not include those three animals.
  – So, to recognize some classes, we use (partly) training examples from other classes.
- What information are we transferring?
  – All the layers and weights of the ImageNet-trained model, except for the output layer.

Transfer Learning

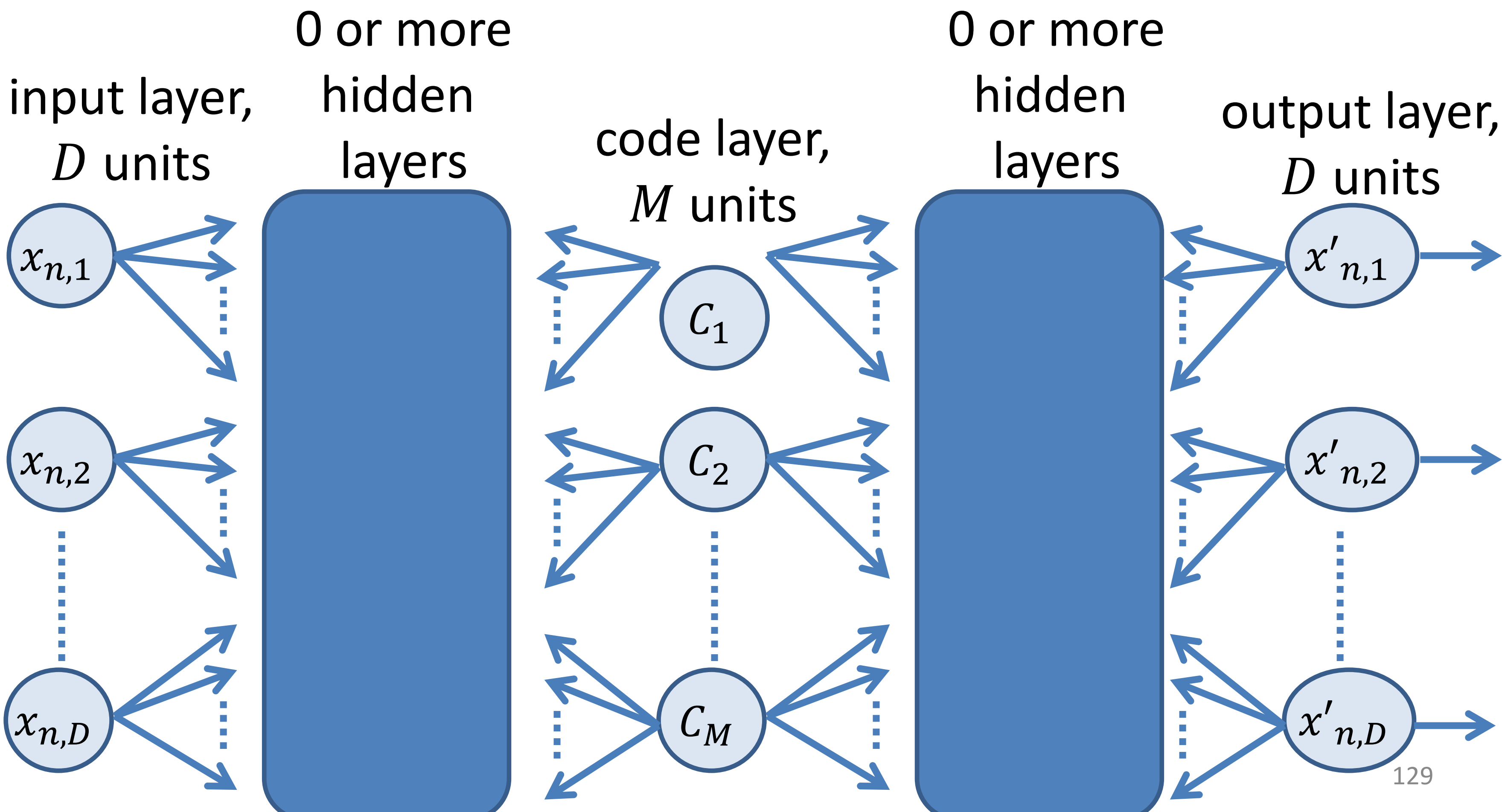
- Why would transfer learning be useful?
- The last hidden layer of the pre-trained model is used to recognize over 20,000 classes.
- Presumably, that last hidden layer computes features that are useful for recognizing a very wide variety of visual objects and shapes.
- Those features have been optimized using millions of training examples.
- Transfer learning assumes that those features would be useful for our three animals as well.

Transfer Learning

- Overall, we have a trade-off:
- 1st choice: Learn a model from scratch, using our three hundred images.
  - The model will be optimized exclusively for the three animals we want to recognize.
  - However, it will be a simple model, trained on a small training set.
- 2nd choice: Transfer learning.
  - Most of the model is optimized using examples that do not contain our three animals.
  - However, the features extracted by that model have been heavily optimized, and may be more useful than what we can learn from just 300 examples.

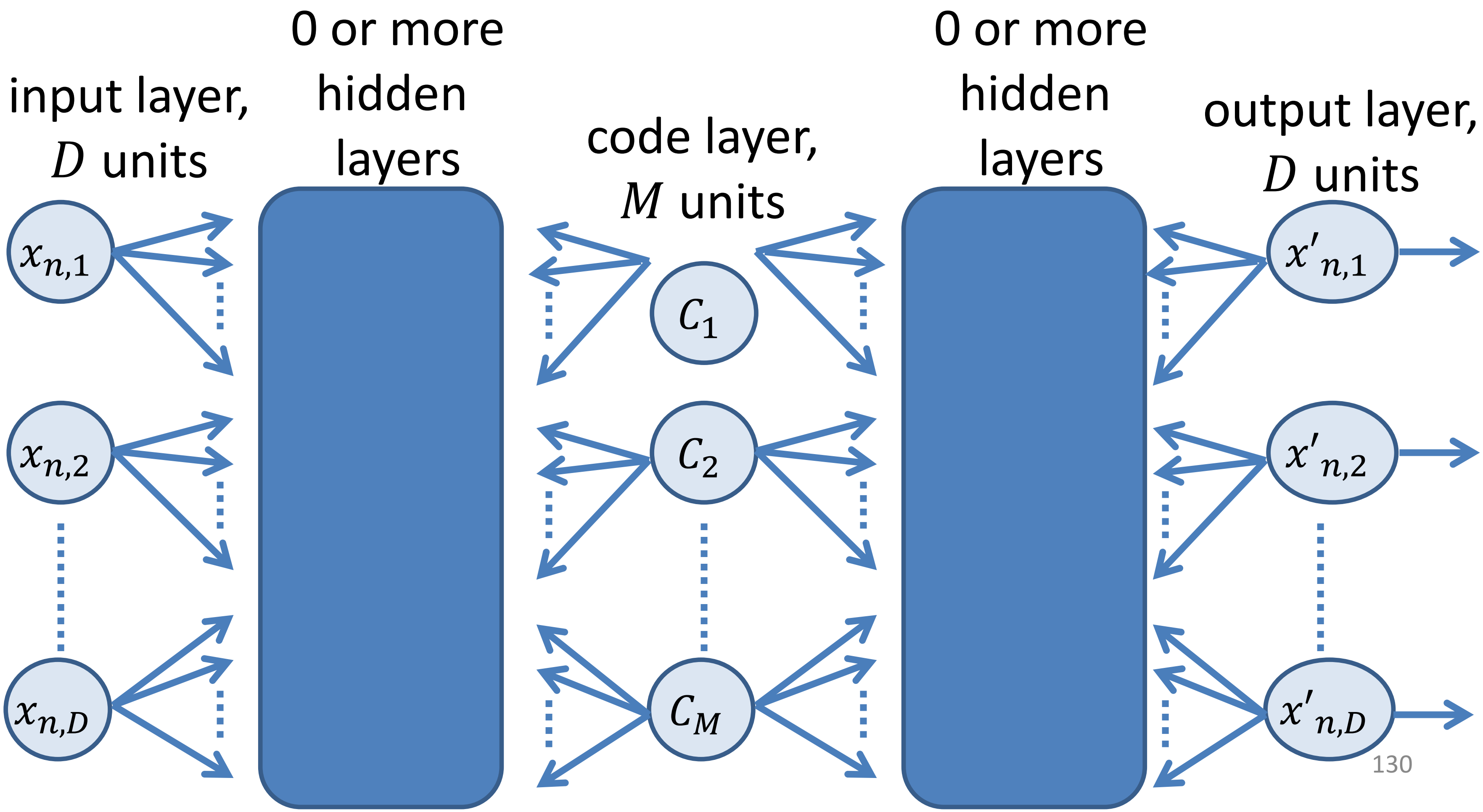
Autoencoders

- An autoencoder is a neural network solving the same problem as PCA.
- Goal: achieve a low-dimensional representation of the input.
- Key difference from PCA: PCA computes a linear projection, autoencoders are non-linear.



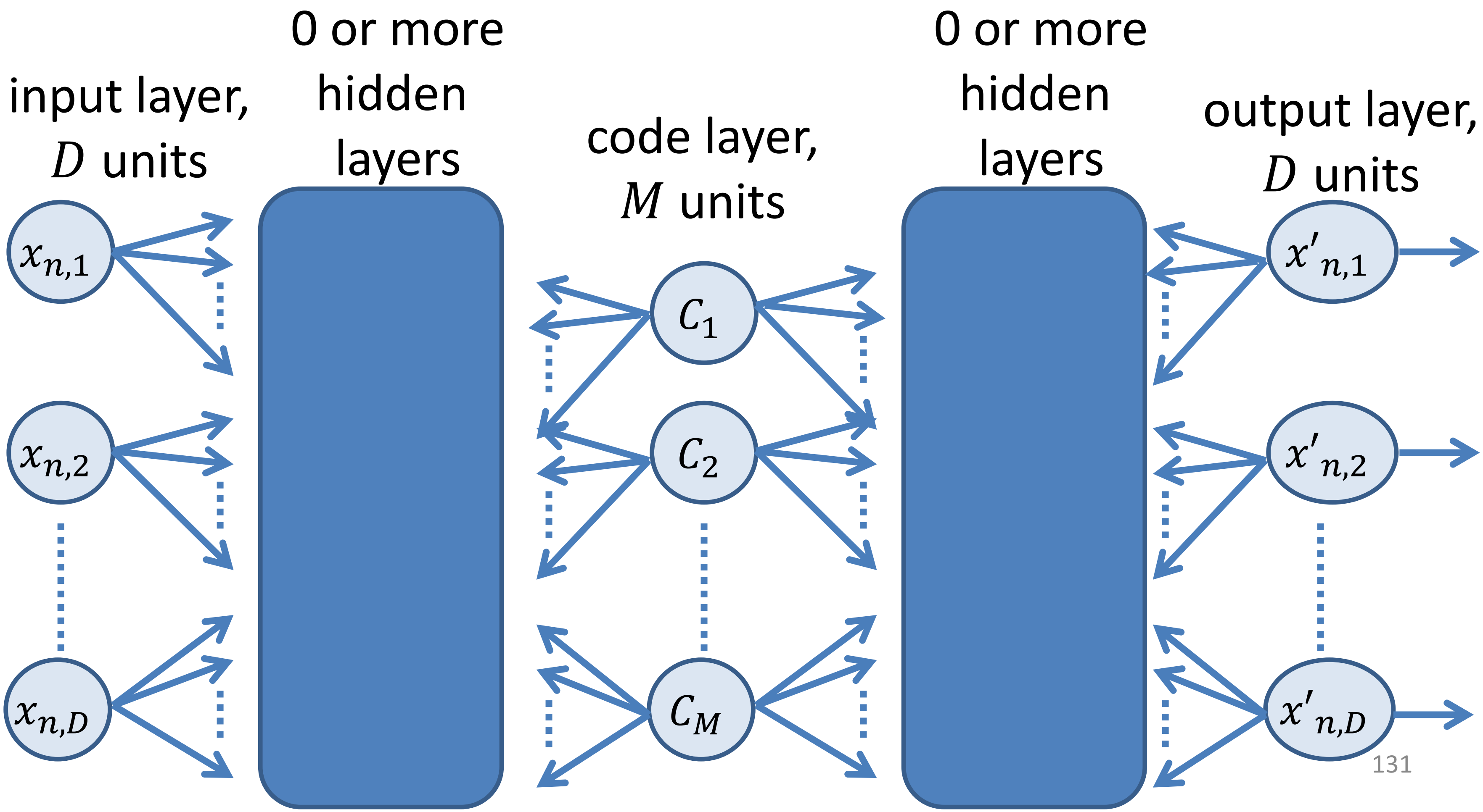
Autoencoders

- An autoencoder is a neural network, can be trained with backpropagation or other methods.
- The target output for input $\boldsymbol{x}_n$ is $\boldsymbol{x}_n$.
- One of the layers is the **code layer**, with M units, where typically $M \ll \mathrm{D}$.



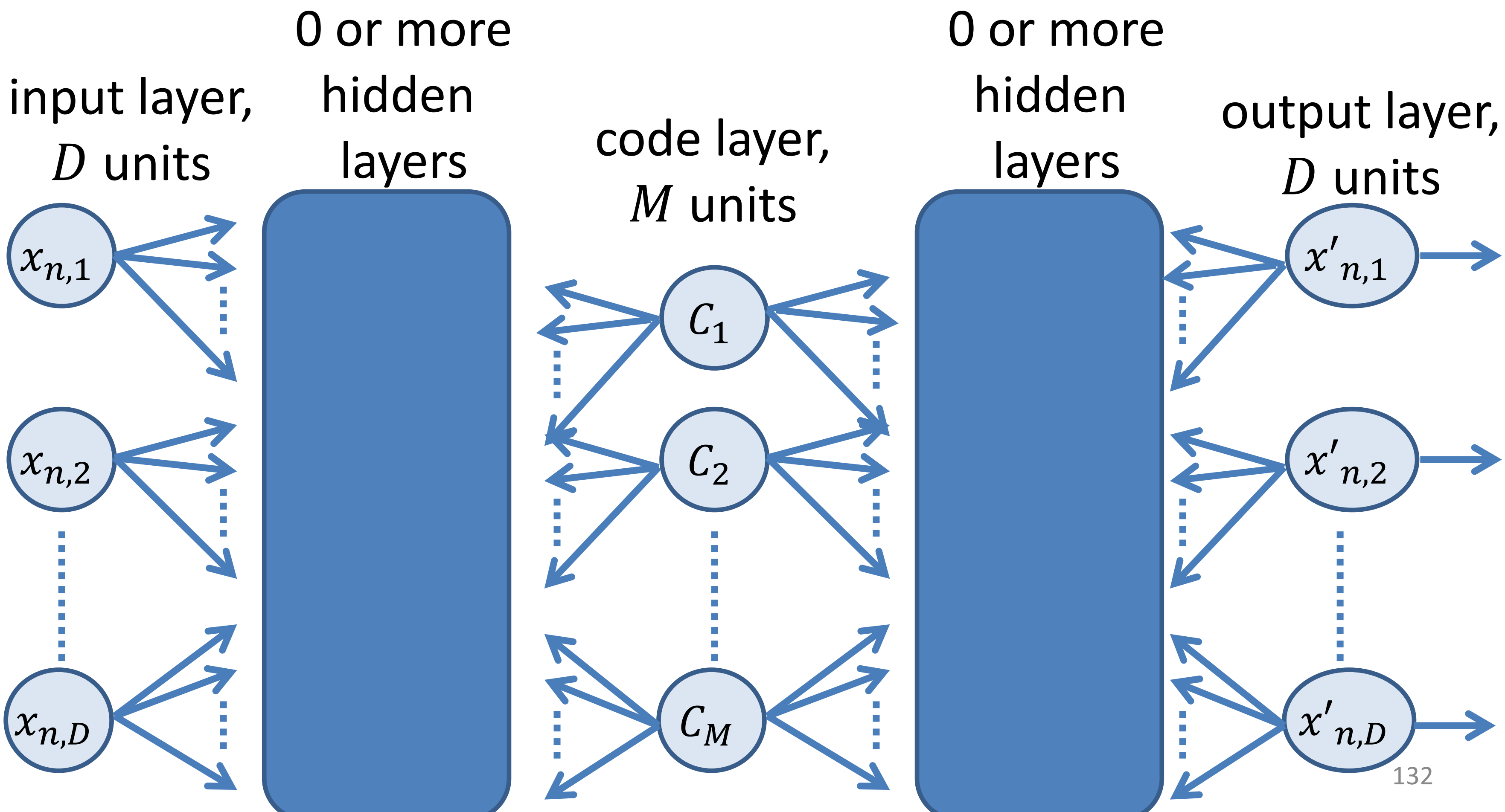
Autoencoders

- A trained autoencoder is used to define a projection $F(\boldsymbol{x})$, mapping $\boldsymbol{x}$ to the activations (sums of weighted inputs) of the code layer units.
- F maps D-dimensional vectors to M-dimensional vectors.



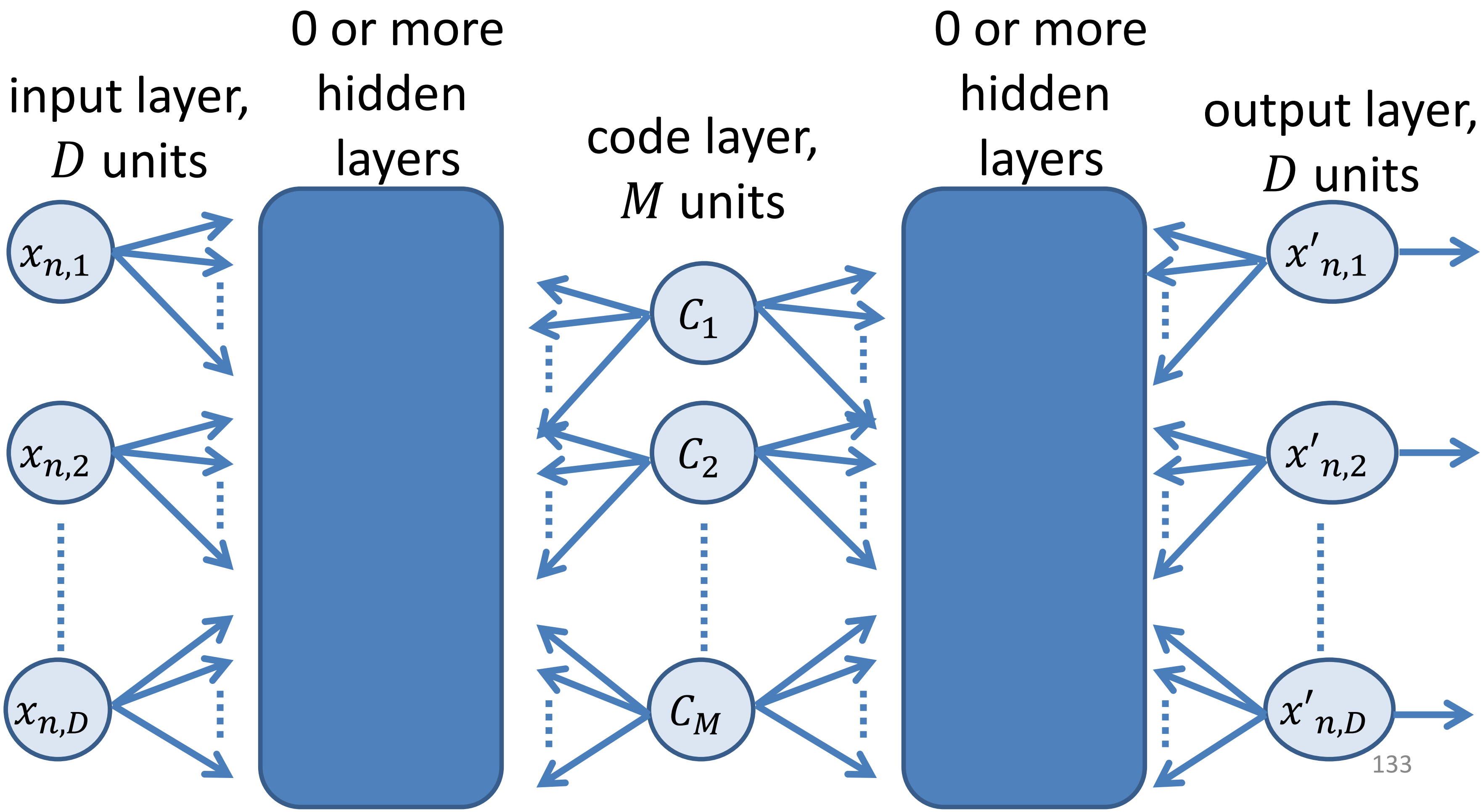
Autoencoders

- A trained autoencoder also defines a backprojection $B(\mathbf{z})$, mapping the activations of the code layers to the output of the output units.
- B maps M-dimensional vectors to D-dimensional vectors.



Autoencoders

- The whole network computes the composition $F(B(\boldsymbol{x}))$.
- F is typically a nonlinear projection.



On Training Neural Networks

- Before the 2010s, most people were skeptical about the practicality of training large neural networks.
  - Even shallow neural networks had a large number of parameters.
  - It took lots of data and lots of time to estimate good values for such networks.
  - Deeper networks made that problem even harder, because they had even more parameters.
- Now we know that deep neural networks are indeed trainable, actually better than other models.
  - Prerequisite: a sufficient (i.e., very large) amount of training data.

Neural Networks, pre-2010

- Before 2010, neural networks were usually not the first choice for people doing research in AI (computer vision, speech recognition, …).
- Other methods were more popular, gave usually better results:
  - Boosting methods.
  - Decision trees and random forests.
  - Support vector machines.
  - Probabilistic graphical models (e.g., Hidden Markov Models for speech recognition).

Post 2010 – What Changed

- Some deep learning methods that are now popular were known in the 1990s.
  - Convolutional Neural Networks proposed by Yann LeCun in the 1990s [LeCun95].
- Limitations in hardware did not allow using and evaluating such methods with sufficient training data.
- Improved hardware (CPUs, GPUs, RAM) eventually allowed experiments with sufficiently large datasets.
- In computer vision, deep neural networks started outperforming other methods by large margins around 2012.

Post 2010 – What Changed

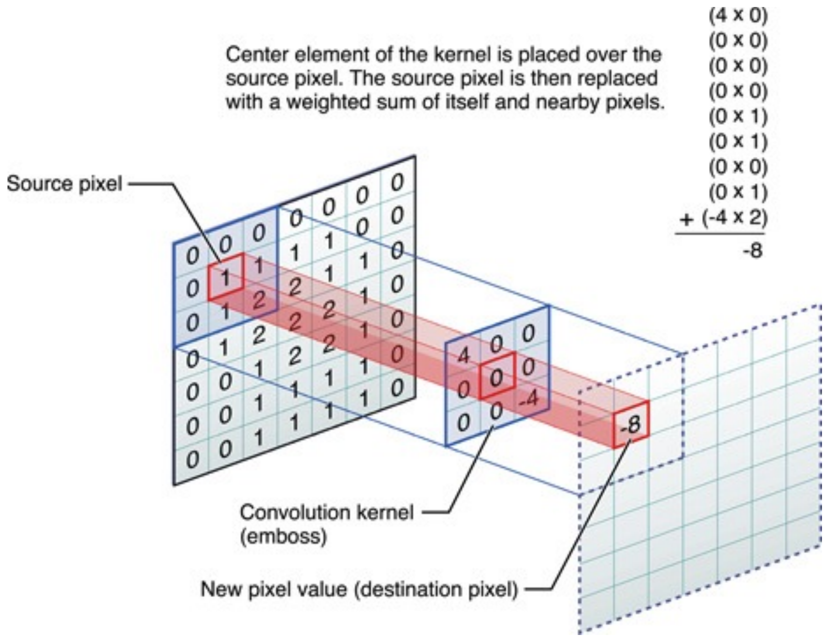
- Initial successes started a positive feedback loop:
  - More researchers started working on improving deep neural networks.
  - New improvements led to even better results.
  - Even better results encouraged even more people to start using deep neural networks…
- Continuing improvements in hardware, and continuing algorithmic improvements allow the use of ever deeper models, with even better results:
  - 7 hidden layers in 2012.
  - 152 hidden layers in 2016.

Performance vs. Size of Data

- Pre-2010, relatively smaller training sets:
  - Other machine learning methods tended to outperform neural network methods.
- Post-2010, larger training sets:
  - Deep neural networks get a big bump in accuracy, and outperform other methods.
  - Compared to other methods, deep neural networks are more capable of taking advantage of large amounts of data.

Convolutional Neural Networks

- Convolutional Neural Networks (CNNs) are the dominant paradigm in deep learning models for computer vision.
  - Originally proposed by Yann LeCun in the 1990s [LeCun95].
- Hidden layers perform convolutions on their input.
  - Convolutional layers have a far smaller number of parameters, compared to fully-connected layers.
  - Fewer parameters can be trained with smaller amounts of data.



Credit: figure from apple.com developer website.

Success Stories – ImageNet

- Task: classify each image into over 1000 categories.
- Before deep learning: 25% error rate.
- 2012: deep learning model, 16% error rate.
  - Had a huge impact in the computer vision community.
  - Reference: [Kriz12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, NIPS 2012.
- Now: under 5% error rate.

Images from the ImageNet dataset.

Success Stories – Face Recognition

- 99.6% accuracy in recognizing a dataset of 5,000 individuals.
  - Reference: [Kem16] Ira Kemelmacher-Shlizerman, Steve Seitz, Daniel Miller, Evan Brossard, CVPR 2016.
- Face verification used for unlocking smartphones and laptops.
- Social media websites identify people in photos.
  - Google Photos, Facebook…

Images from the MegaFace dataset.

Success Stories - Translation

- Google Translate switched in 2015 to a deep learning model.
- Accuracies improved significantly, according to ratings provided by humans (a score of 6 means "perfect").

From	To	Previous model	Deep learning	Human translation
English	Spanish	4.885	5.428	5.504
English	French	4.932	5.295	5.496
English	Chinese	4.035	4.594	4.987
Spanish	English	4.872	5.187	5.372
French	English	5.046	5.343	5.404
Chinese	English	3.694	4.263	4.636

Results from: [Wu16] Yonghui Wu et al., "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", at arxiv.org, 2016.

Limitations – Amounts of Data

- Deep learning methods do great when lots of training data is available.
- However, it is often hard to find sufficient amounts of data.
  - Example: for face recognition, Google uses a proprietary dataset with tens of millions of individuals. As a university, we cannot compete with that.
  - Example: for American Sign Language recognition, NO current dataset is sufficiently comprehensive to train an acceptably accurate general-purpose translation system.
- Humans need less data to learn, so there is room for improvement.

Limitations - Accuracy

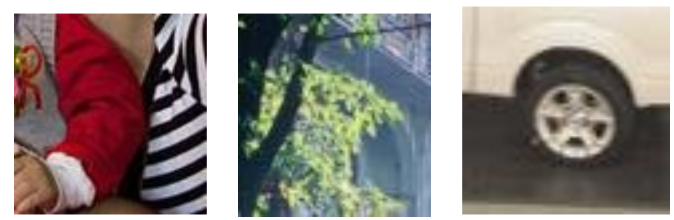
- In most tasks, accuracy is still far from being on par with human accuracy. Just a few examples:
- Mistakes in face detection:

Examples from MegaFace website

- Mistakes in machine translation:
  - Google Translate, input in Greek:
    - Δε φτάνει να θέλεις να κάνεις κάτι, πρέπει και να το μπορείς.
  - Google Translate output:
    - You do not want to do something, you have to.

Limitations - Accuracy

- In most tasks, accuracy is still far from being on par with human accuracy. Just a few examples:
- Mistakes in face detection:

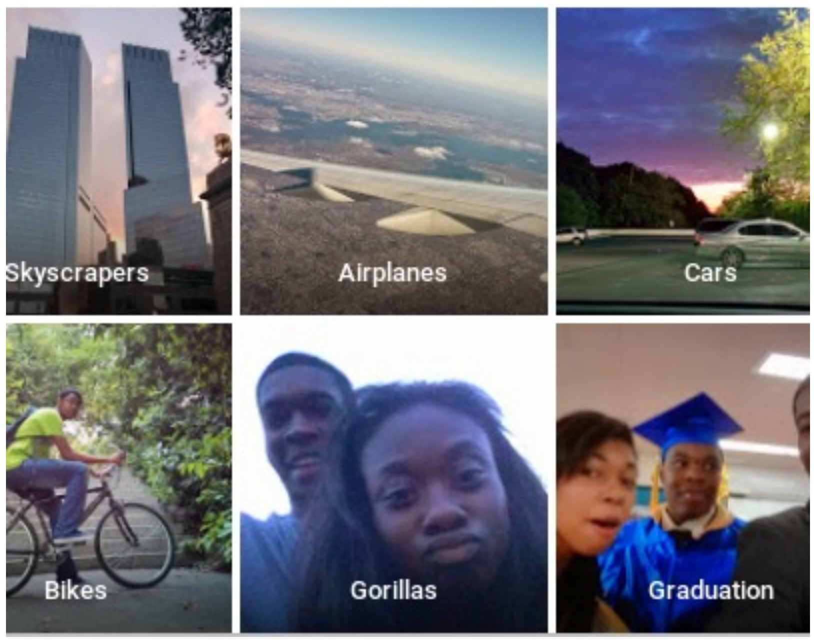


Examples from [MegaFace website]

- Mistakes in machine translation:
  - Google Translate, input in Greek:
    - Δε φτάνει να θέλεις να κάνεις κάτι, πρέπει και να το μπορείς.
  - Google Translate output:
    - You do not want to do something, you have to.
  - My translation:
    - It is not enough to want to do something, you also have to be able to do it.

Limitations - Bias

- Example: black people labeled as gorillas by Google photos.



Picture retrieved from https://twitter.com/jackyalcine/status/615329515909156865
See articles at Mashable, New York Magazine, The Guardian.

Limitations - Bias

- Example: a language model was trained to complete analogies, such as:
  - Input: Paris is to France as Rome is to ???
  - Output: Italy.
- That model provided some heavily biased answers:
  - Input: Man is to computer programmer as woman is to ???
  - Output:

Limitations - Bias

- Example: a language model was trained to complete analogies, such as:
  - Input: Paris is to France as Rome is to ???
  - Output: Italy.
- That model provided some heavily biased answers:
  - Input: Man is to computer programmer as woman is to ???
  - Output: homemaker.

Limitations - Bias

- Example: a language model was trained to complete analogies, such as:
  - Input: Paris is to France as Rome is to ???
  - Output: Italy.
- That model provided some heavily biased answers:
  - Input: Man is to computer programmer as woman is to ???
  - Output: homemaker.
  - Input: Man is to doctor as woman is to ???
  - Output:

Limitations - Bias

- Example: a language model was trained to complete analogies, such as:
  - Input: Paris is to France as Rome is to ???
  - Output: Italy.
- That model provided some heavily biased answers:
  - Input: Man is to computer programmer as woman is to ???
  - Output: homemaker.
  - Input: Man is to doctor as woman is to ???
  - Output: Nurse.

Reference: [Bol16] Tolga Bolukbasi, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, Adam Tauman Kalai: “Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings.” NIPS 2016: 4349-4357

Limitations - Bias

- Bias is a big problem, that needs to be overcome as deep learning models are used to make decisions that affect humans.
  - Systems that are trained with black dogs and white cats think that a white dog is a cat.
  - Similarly, systems trained based on human decisions, learn human biases.
    - Can lead to unfair decisions for loans, release from prison, college admissions, employment…
- An exacerbating factor is the lack of explainability of the output of deep learning models.
  - Decisions are influenced by millions of perceptrons, it is hard to tell if an output was influenced by bias.

Conclusions

- Deep learning models are “deep” in two aspects:
  - In terms of functionality: they are end-to-end models, directly mapping input to output.
  - In terms of design: deep neural networks with many hidden layers.
- For many years, people were skeptical about the practicality of training large neural networks.
- Better hardware, more memory, and more data have demonstrated that deep neural networks significantly outperform other machine learning methods in many tasks.
- Many success stories, many commercial applications:
  - Object recognition, face recognition, machine translation, speech recognition…
- Still, lots of room for improvement:
  - Improving accuracy, training with fewer examples, avoiding bias…

Conclusions

- 20 years ago, computer vision and machine learning were not that relevant to society.
- Now, these areas have huge financial impact, enabling exciting technologies.
- No one can predict the future, but there is potential for much larger impact than what we got so far:
  – Self-driving cars (banning human drivers as too careless and error-prone).
  – Robots doing house chores.
  – More accurate diagnosis from medical data, tests, images.
  – Computer tutors offering personalized help to students…
- We live in exciting times!!!

References

[Bol16] Tolga Bolukbasi, Kai-Wei Chang, James Y. Zou, Venkatesh Saligrama, Adam Tauman Kalai: “Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings.” NIPS 2016.

[Kem16] Ira Kemelmacher-Shlizerman, Steve Seitz, Daniel Miller, Evan Brossard: “The MegaFace Benchmark: 1 Million Faces for Recognition at Scale.” CVPR 2016.

[Kriz12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012.

[LeCun95] Y. LeCun and Y. Bengio: “Convolutional Networks for Images, Speech, and Time-Series.” in Arbib, M. A. (Eds), The Handbook of Brain Theory and Neural Networks, MIT Press, 1995.

[Lee09] Honglak Lee, Roger Grosse, Rajesh Ranganath and Andrew Y. Ng: “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations.” ICML 2009.

[Wu16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, Jeffrey Dean: “Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.” arxiv.org, 2016.