An introduction to Deep Learning

More at neuralnetworksanddeeplearning.com

Winter School on Quantitative Biology Learning and Artificial Intelligence

Alberto Pezzotta

ICTP – November 2018

What is this about?

Classification tasks

What is this about?

Classification tasks

“is this a cat or a dog?”

What is this about?

Classification tasks

“is this a cat or a dog?”

“what digit is this?”

What is this about?

Classification tasks

"is this a cat or a dog?"
"what digit is this?"

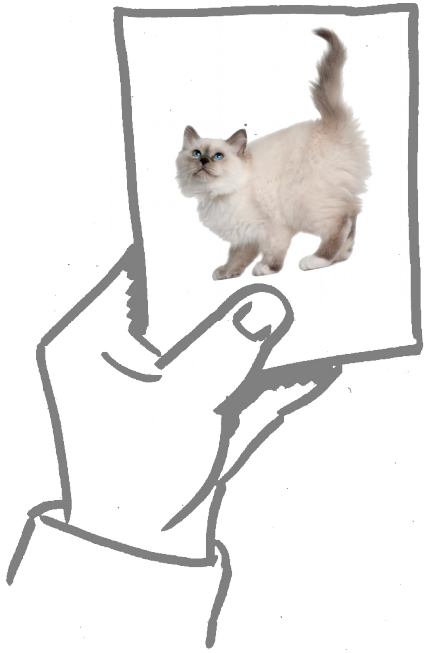
Supervised Learning: by examples, with a teacher

What is this about?

Classification tasks

"is this a cat or a dog?"
"what digit is this?"

Supervised Learning: by examples, with a teacher



`cat`

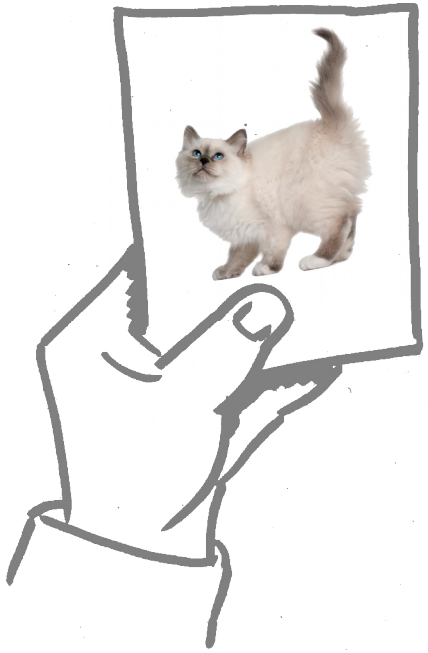
`dog`

What is this about?

Classification tasks

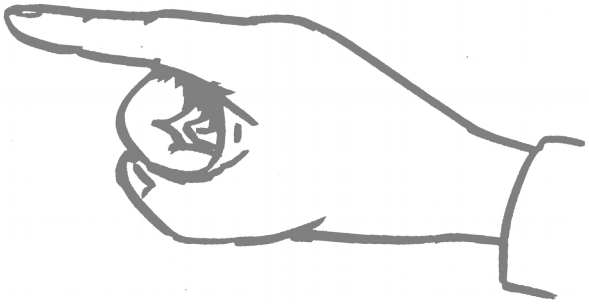
"is this a cat or a dog?"
"what digit is this?"

Supervised Learning: by examples, with a teacher



`cat`

`dog`



What is this about?

Classification tasks

"is this a cat or a dog?"
"what digit is this?"

Supervised Learning: by examples, with a teacher

`cat`

`dog`

What is this about?

Classification tasks

"is this a cat or a dog?"
"what digit is this?"

Supervised Learning: by examples, with a teacher

`cat`

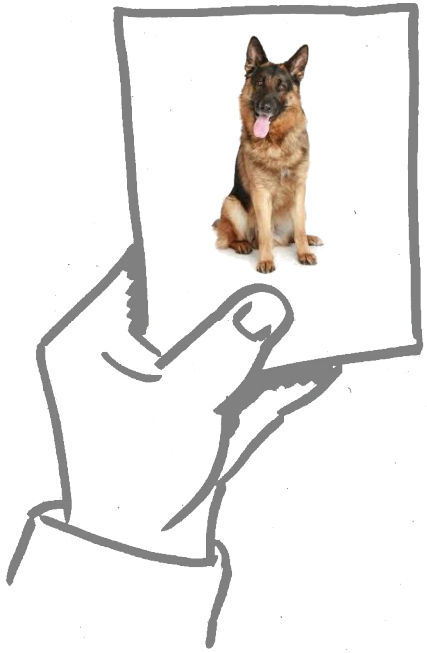
`dog`

What is this about?

Classification tasks

"is this a cat or a dog?"
"what digit is this?"

Supervised Learning: by examples, with a teacher



`cat`

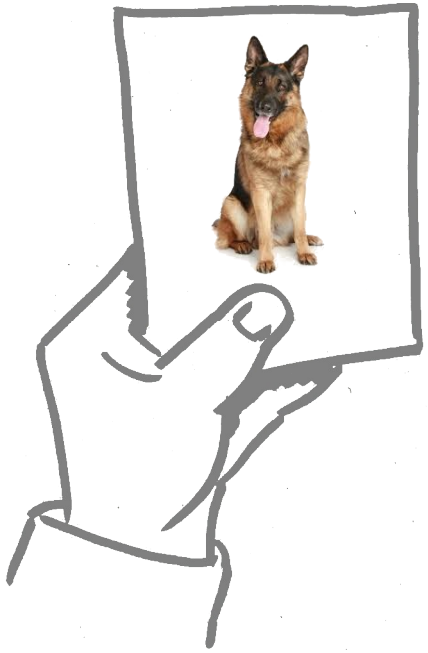
`dog`

What is this about?

Classification tasks

"is this a cat or a dog?"
"what digit is this?"

Supervised Learning: by examples, with a teacher



`cat`

??

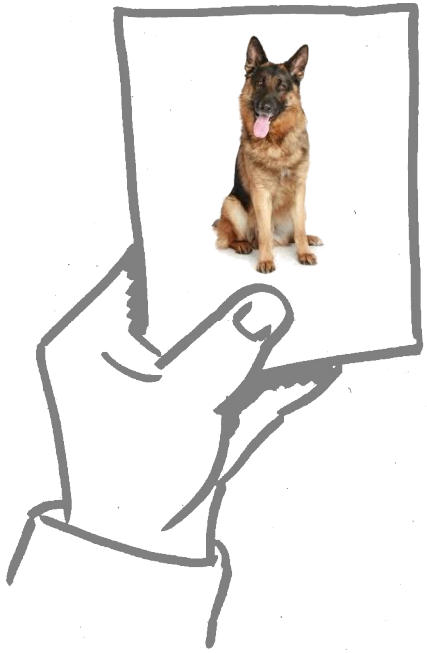
`dog`

What is this about?

Classification tasks

"is this a cat or a dog?"
"what digit is this?"

Supervised Learning: by examples, with a teacher



`cat`

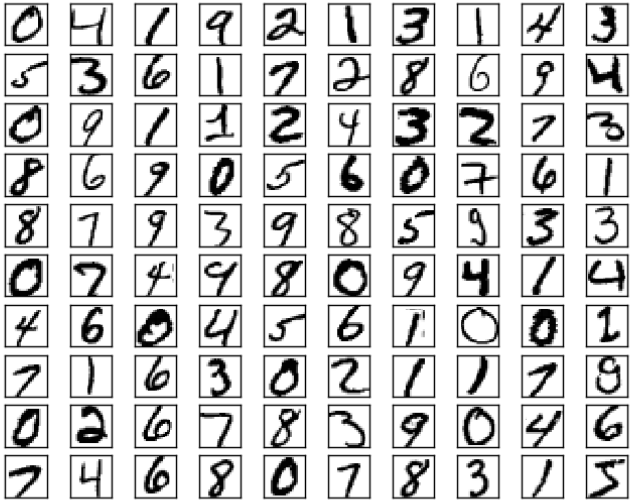
??

`dog`

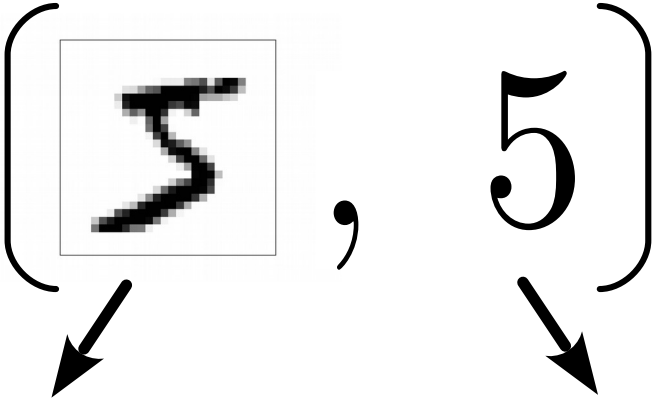
Generalization to unseen picture

Let us play with digits...
... the MNIST database

Classification of handwritten digits



70'000 input/output pairs



How to use them?

Input x

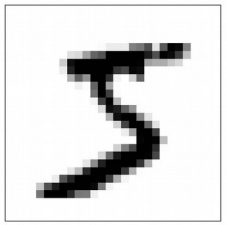
$28 \times 28,\ [0,\ 1]^{784}$

Correct output $y(x)$

$\{0, 1 \ldots 9\}$

Let us play with digits...

... the MNIST database

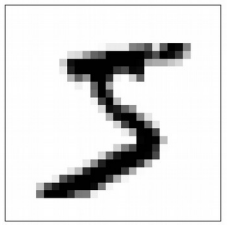
$$\left\{ \left(\text{  }, 5 \right) \quad \ldots \right\}$$

Input x

Correct output $y(x)$

Let us play with digits...

... the MNIST database

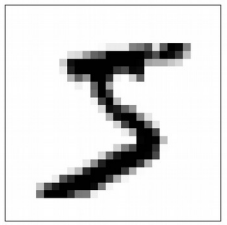
$$\left\{ \left(\text{  }, 5 \right) \quad \ldots \right\}$$

Input x Correct output $y(x)$

$$f_{\bar{w},\bar{b}} : [0,1]^{784} \rightarrow \{0, 1 \ldots 9\}$$

Let us play with digits...

... the MNIST database

$$\left\{ \left(\text{  }, 5 \right) \quad \dots \right\}$$

Input x Correct output $y(x)$

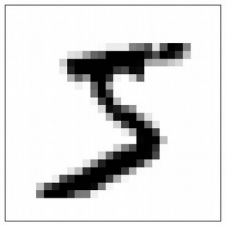
$$f_{\bar{w},\bar{b}} : [0,1]^{784} \rightarrow \{0, 1 \dots 9\}$$

A parametrized function...

How to **construct** it?

Let us play with digits...

... the MNIST database

$$\left\{ \left(\text{  }, 5 \right) \quad \dots \right\}$$

Input x Correct output $y(x)$

$$f_{\bar{w},\bar{b}} : [0,1]^{784} \to \{0, 1 \dots 9\}$$

A parametrized function...

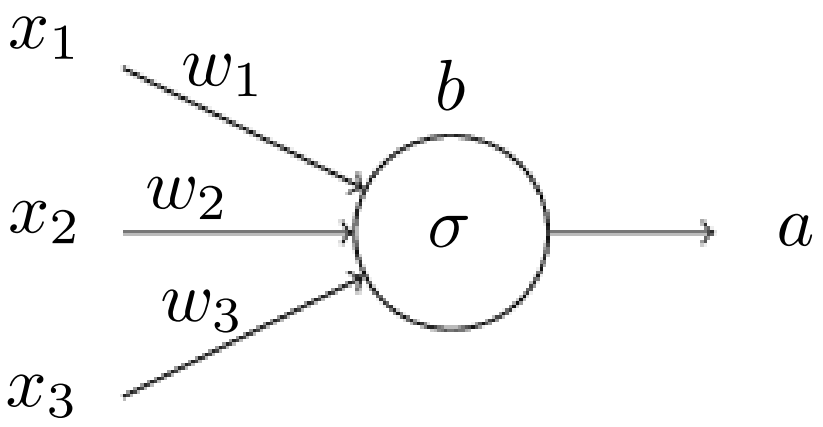
How to **construct** it?

Feed-Forward Neural Networks

Feed-Forward Neural Networks...

...the ingredients

from the input
or
from other neurons

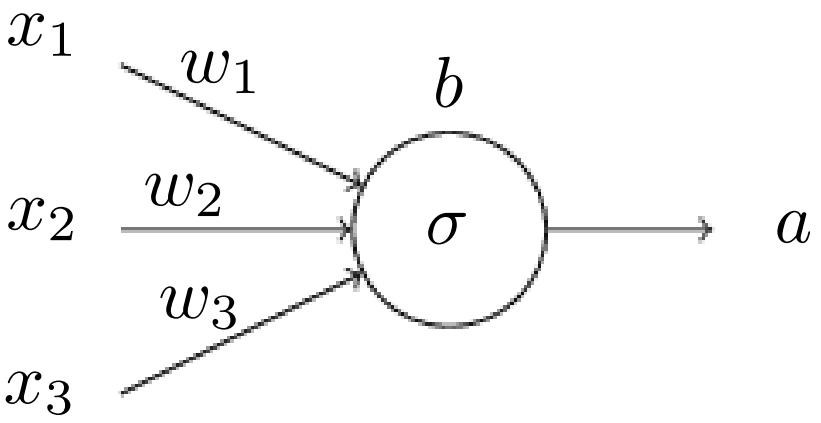


to output
or
to other neurons

Feed-Forward Neural Networks...

...the ingredients

from the input
or
from other neurons



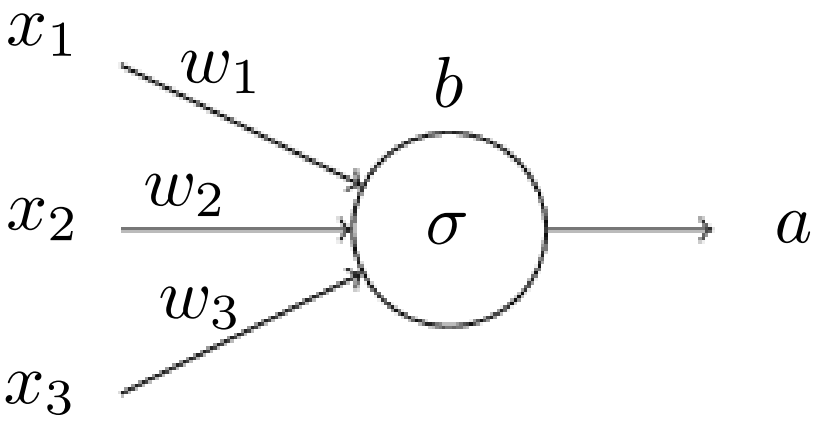
to output
or
to other neurons

weights w **bias** b **activity** a

Feed-Forward Neural Networks...

...the ingredients

from the input
or
from other neurons



to output
or
to other neurons

weights w **bias** b **activity** a

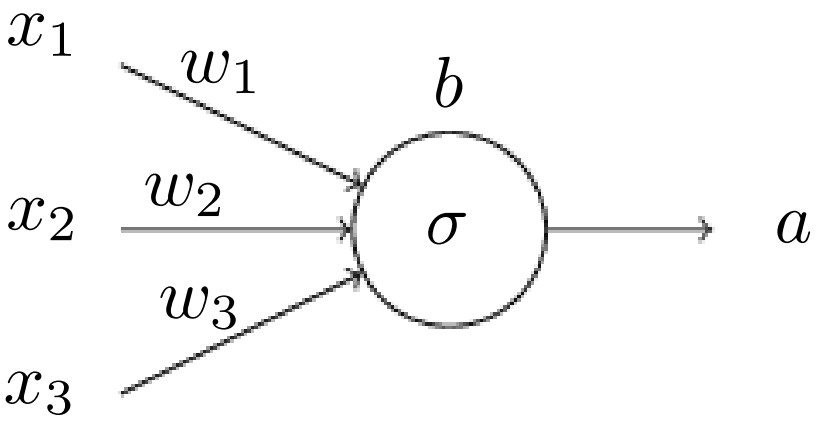
weighed input $z = b + \sum_i w_i \, x_i$

activation function σ, $a = \sigma(z)$

Feed-Forward Neural Networks...

...the ingredients

from the input
or
from other neurons

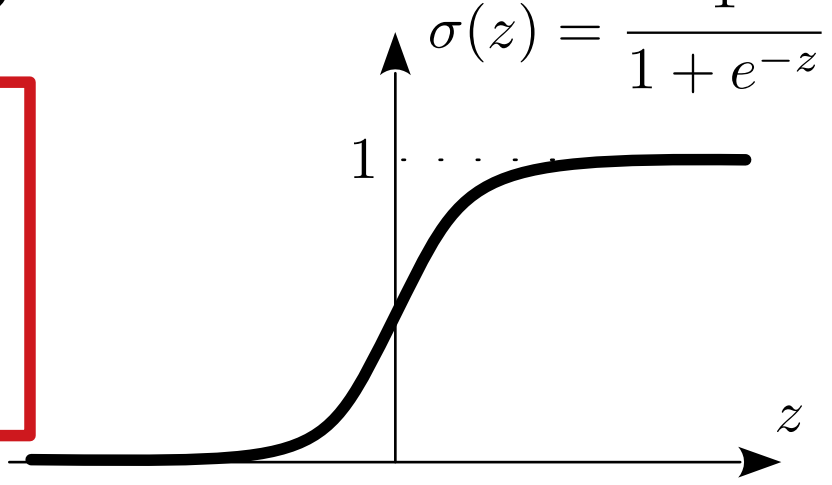


to output
or
to other neurons

weights w **bias** b **activity** a

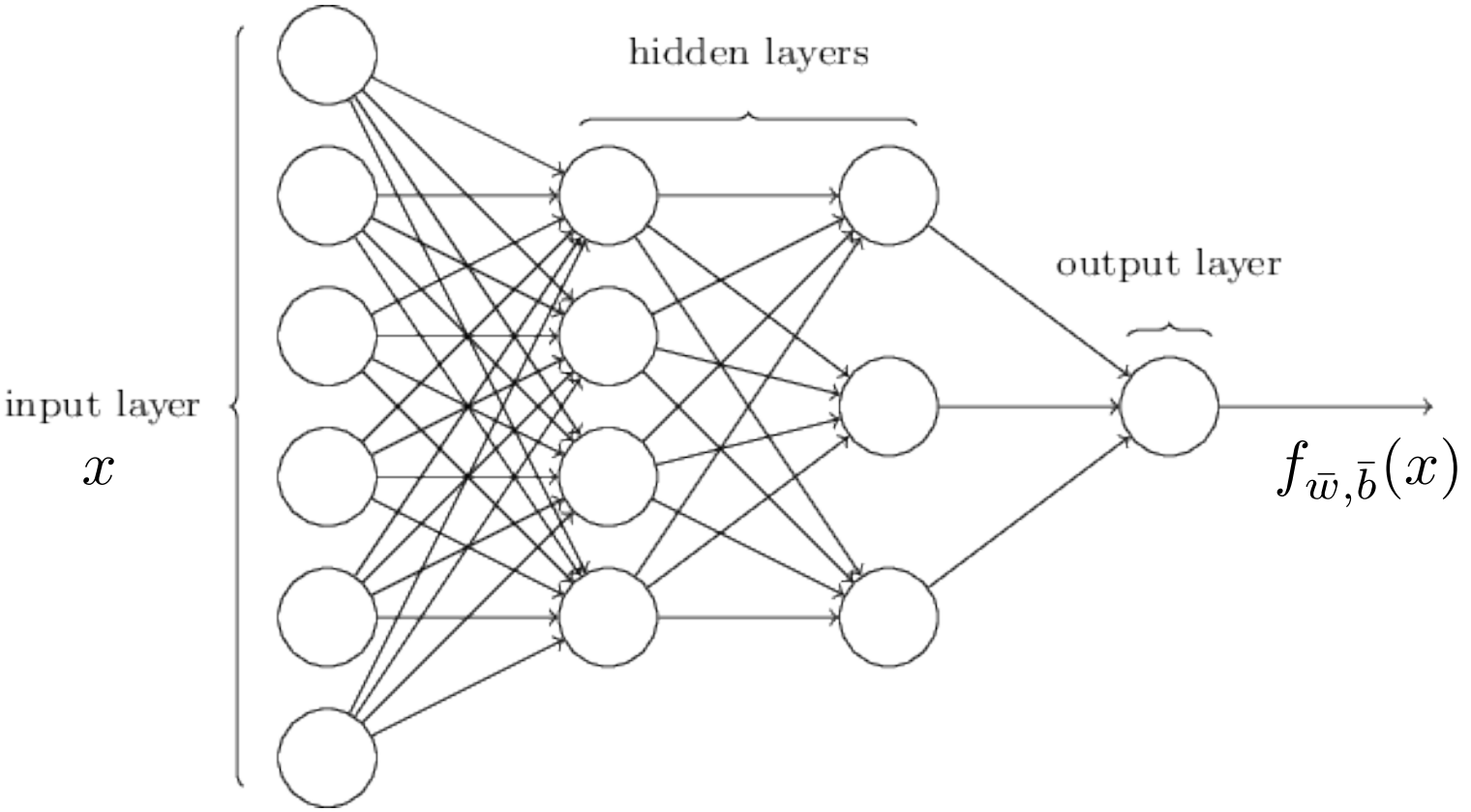
weighed input $z = b + \sum_i w_i \, x_i$

activation function σ, $a = \sigma(z)$



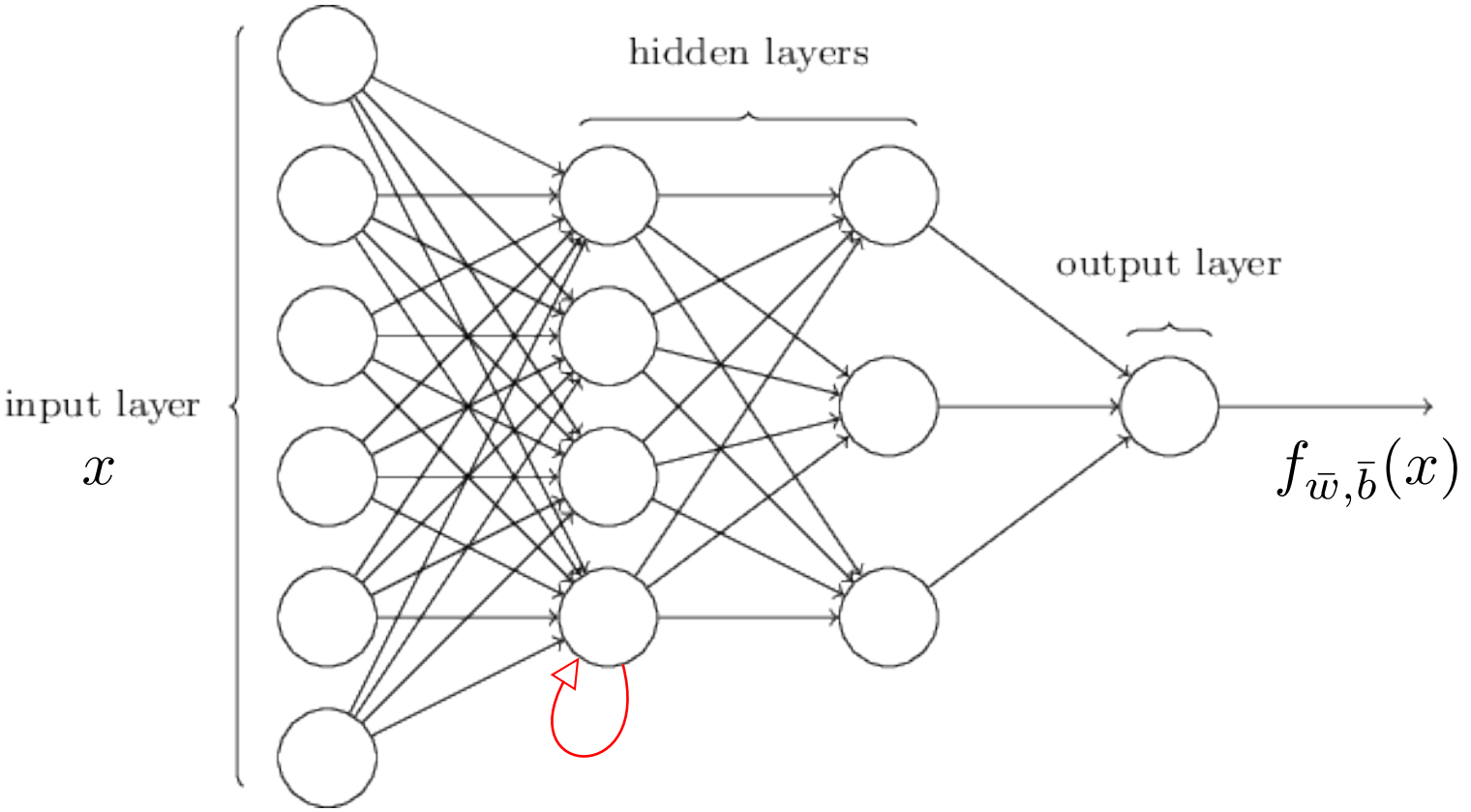
Feed-Forward Neural Networks...

...the architecture



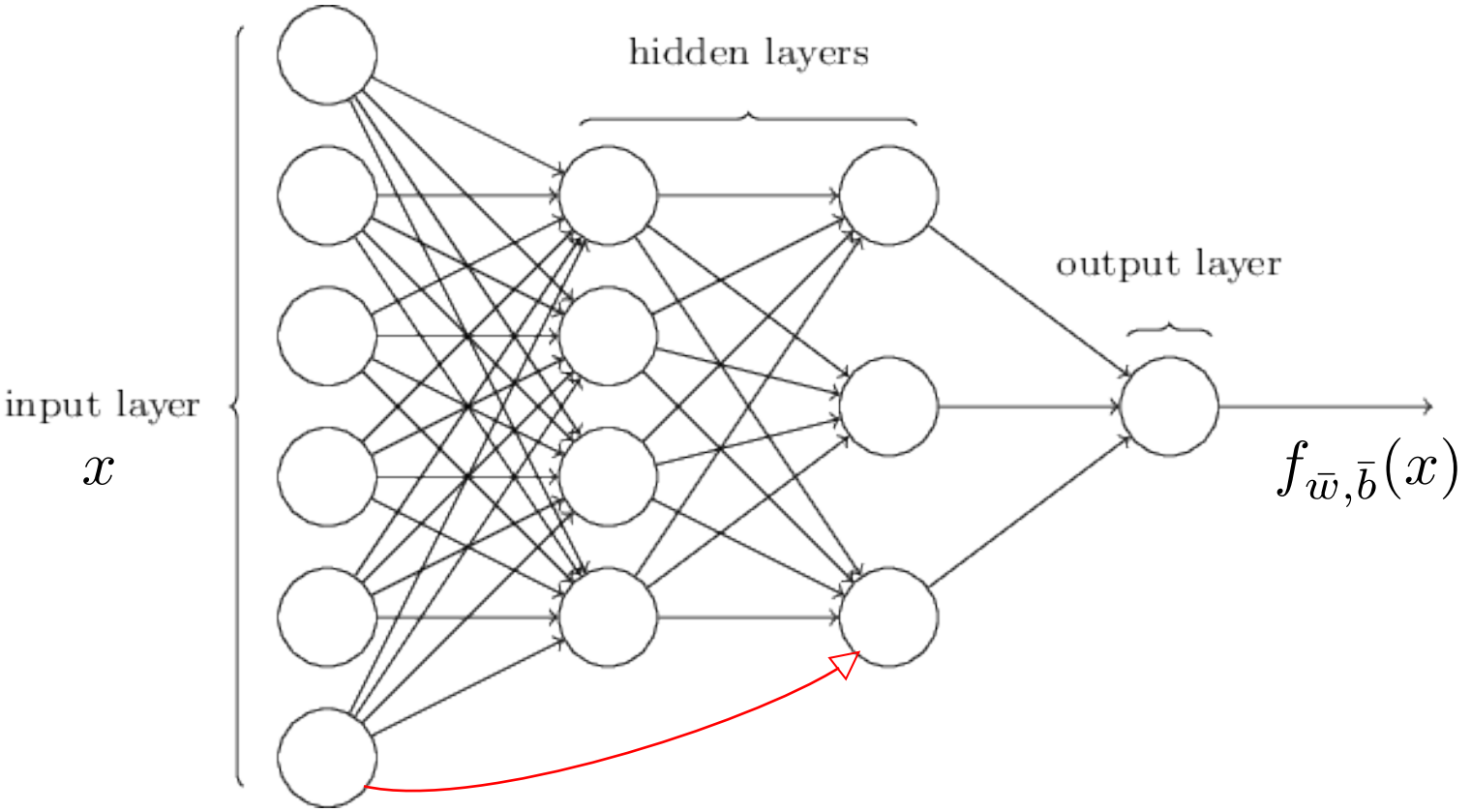
Feed-Forward Neural Networks...

...the architecture



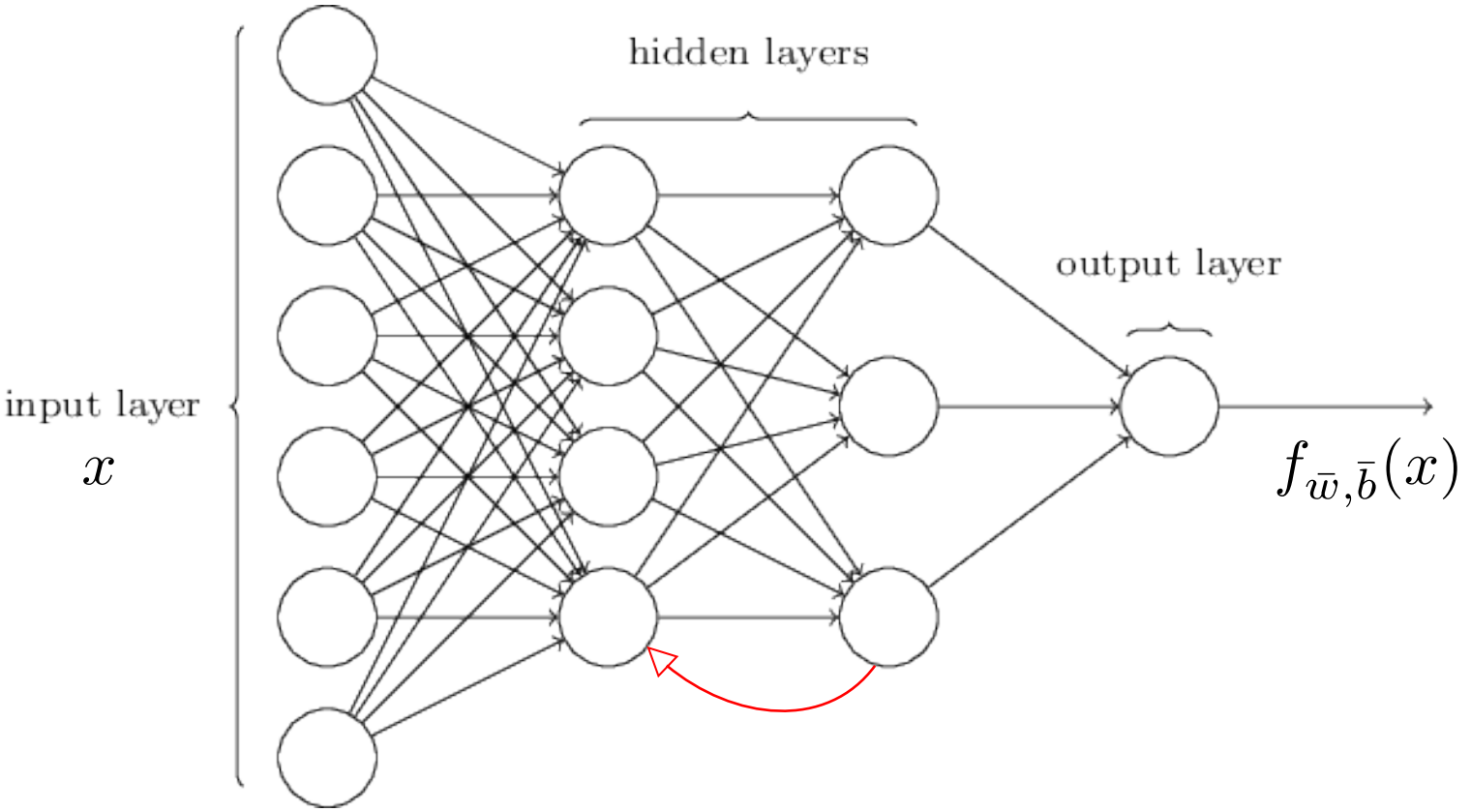
Feed-Forward Neural Networks...

...the architecture



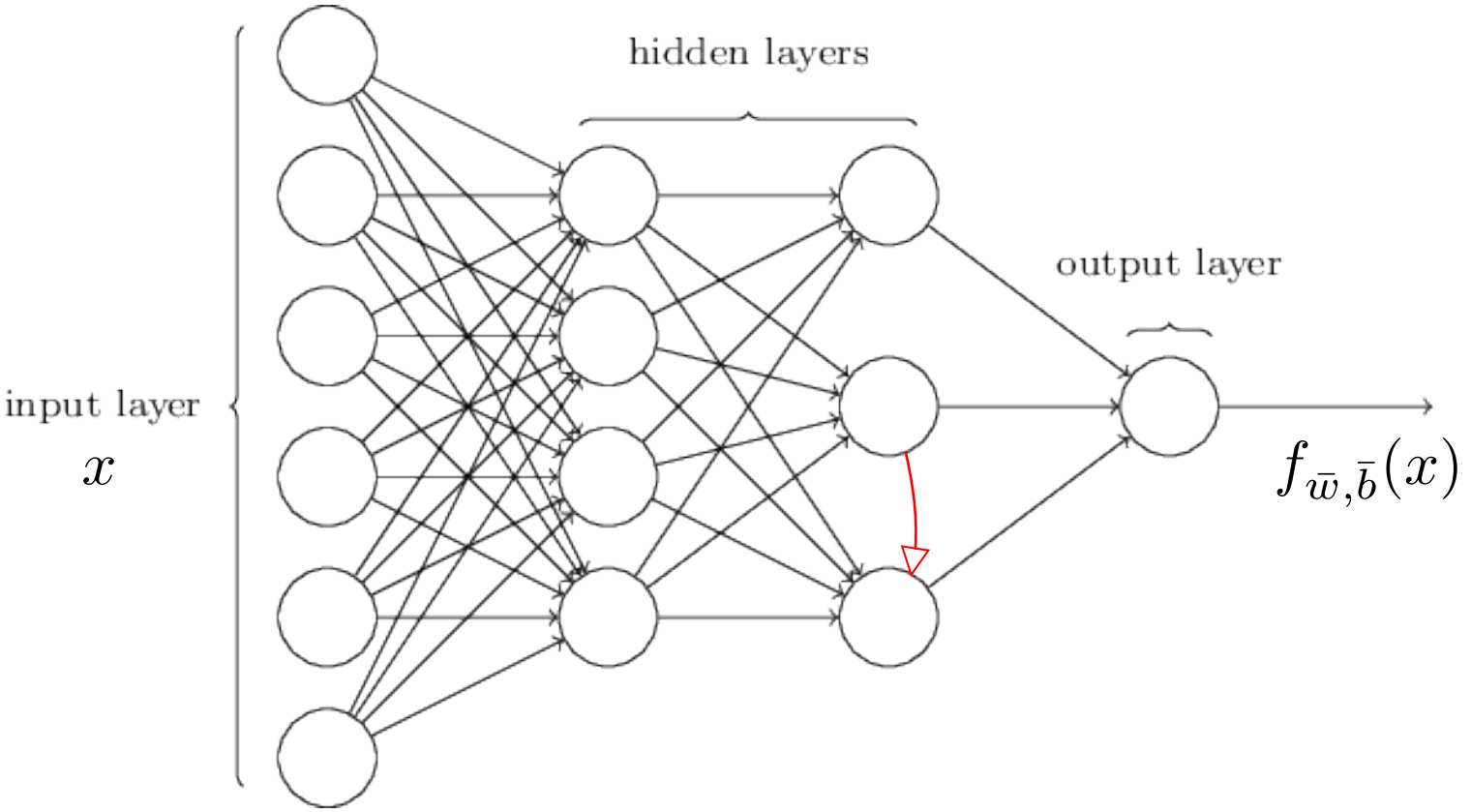
Feed-Forward Neural Networks...

...the architecture



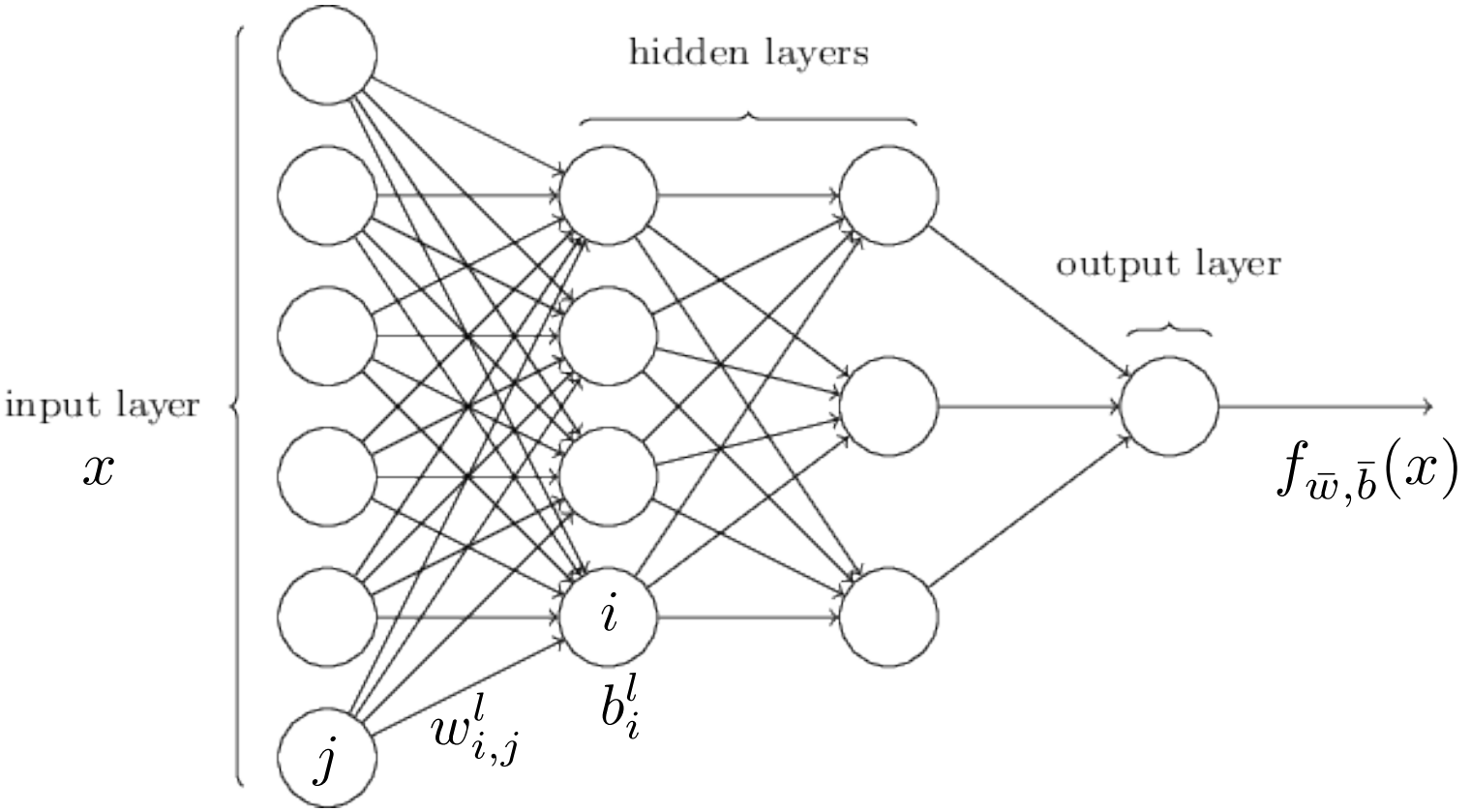
Feed-Forward Neural Networks...

...the architecture



Feed-Forward Neural Networks...

...the architecture

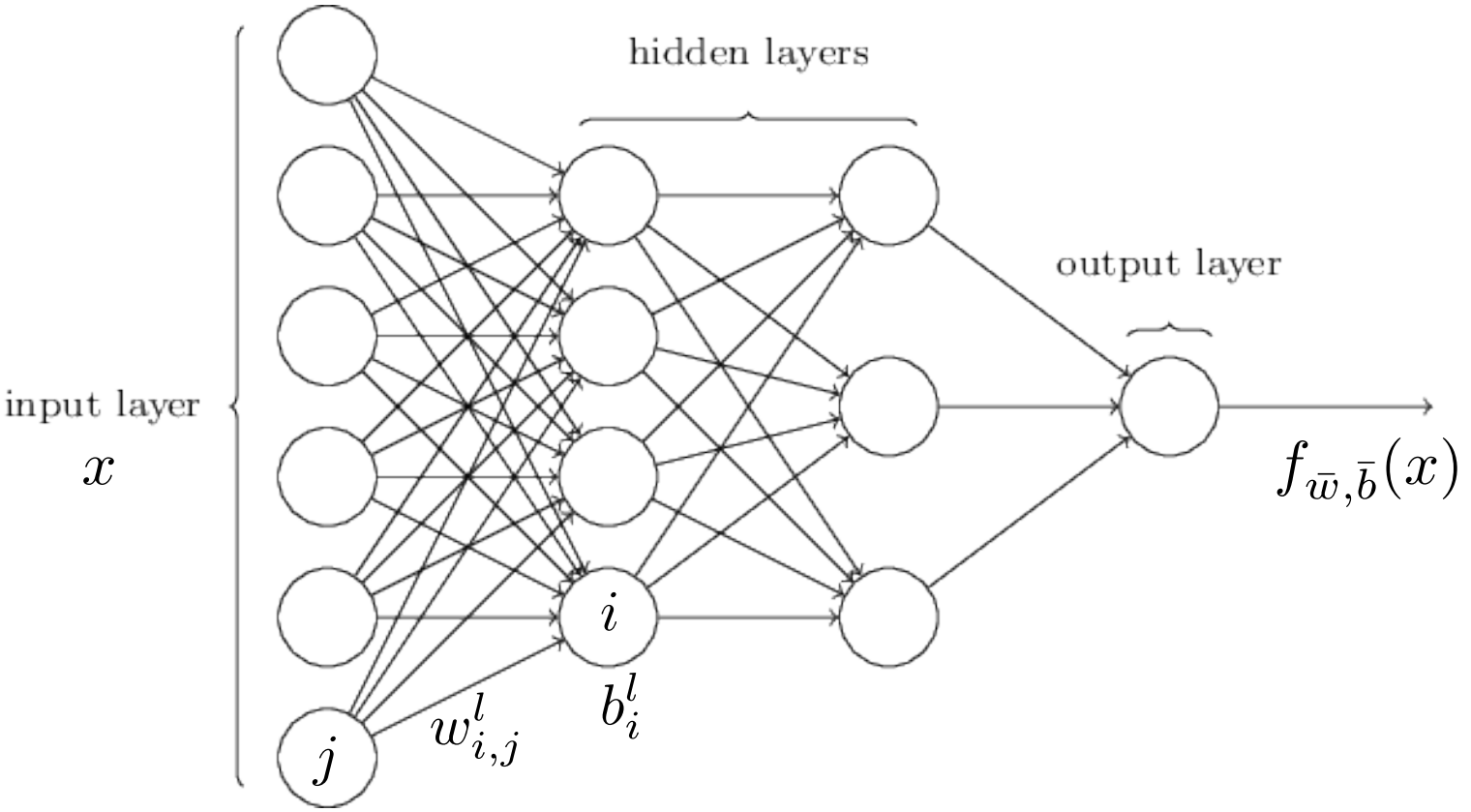


one **weight** per arrow
one **bias** per neuron

$$w^l_{i,j} \qquad b^l_i$$

Feed-Forward Neural Networks...

...the architecture



one **weight** per arrow
one **bias** per neuron

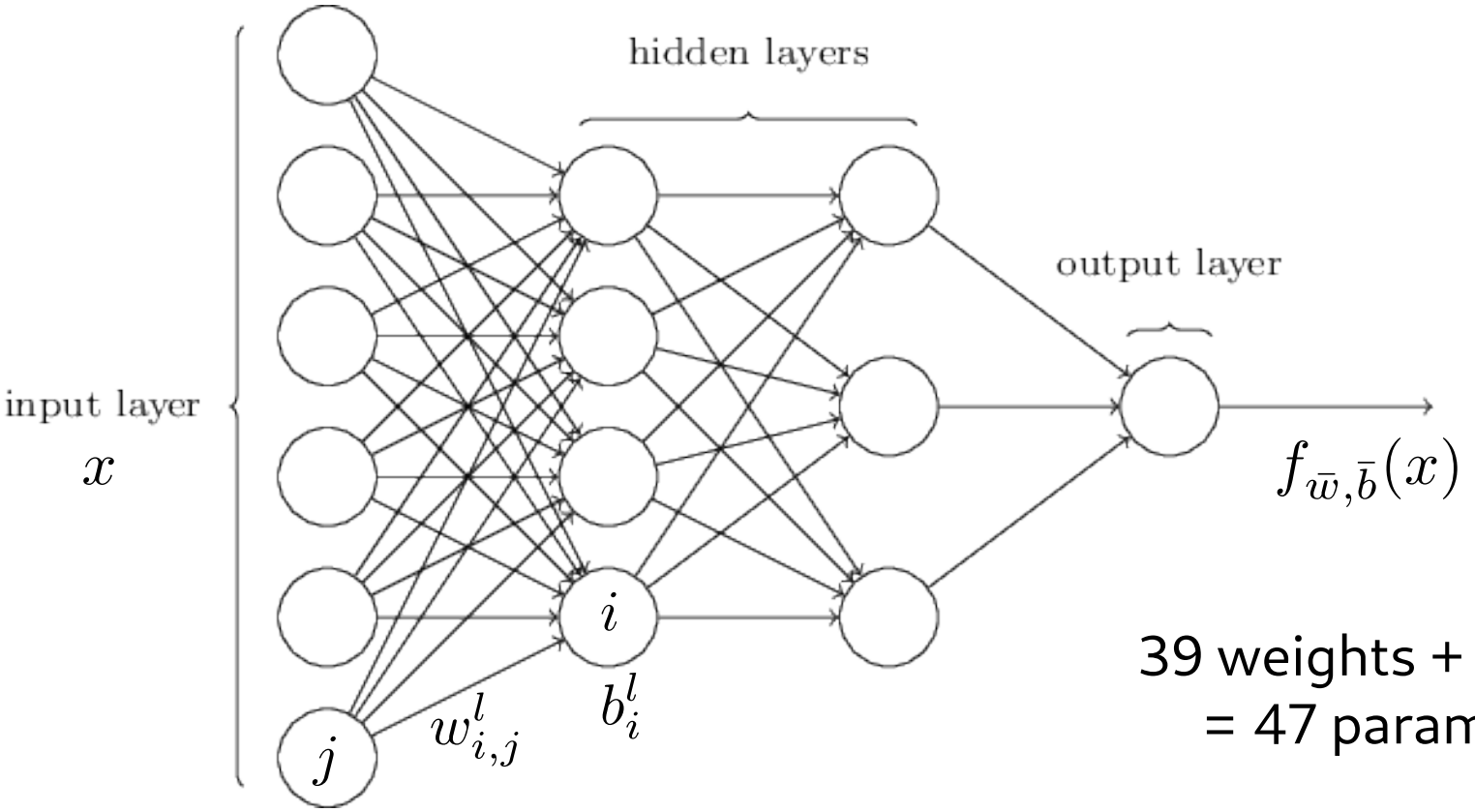
$w^l_{i,j}$ b^l_i

$$a^l_i = \sigma\left(z^l_i\right)$$

$$= \sigma\left(b^l_i + \sum_j w^l_{i,j} a^{l-1}_j\right)$$

Feed-Forward Neural Networks...

...the architecture



39 weights + 8 biases
= 47 parameters

one **weight** per arrow
one **bias** per neuron

$w^l_{i,j}$ b^l_i

$$a^l_i = \sigma\left(z^l_i\right)$$

$$= \sigma\left(b^l_i + \sum_j w^l_{i,j} a^{l-1}_j\right)$$

Feed-Forward Neural Networks...

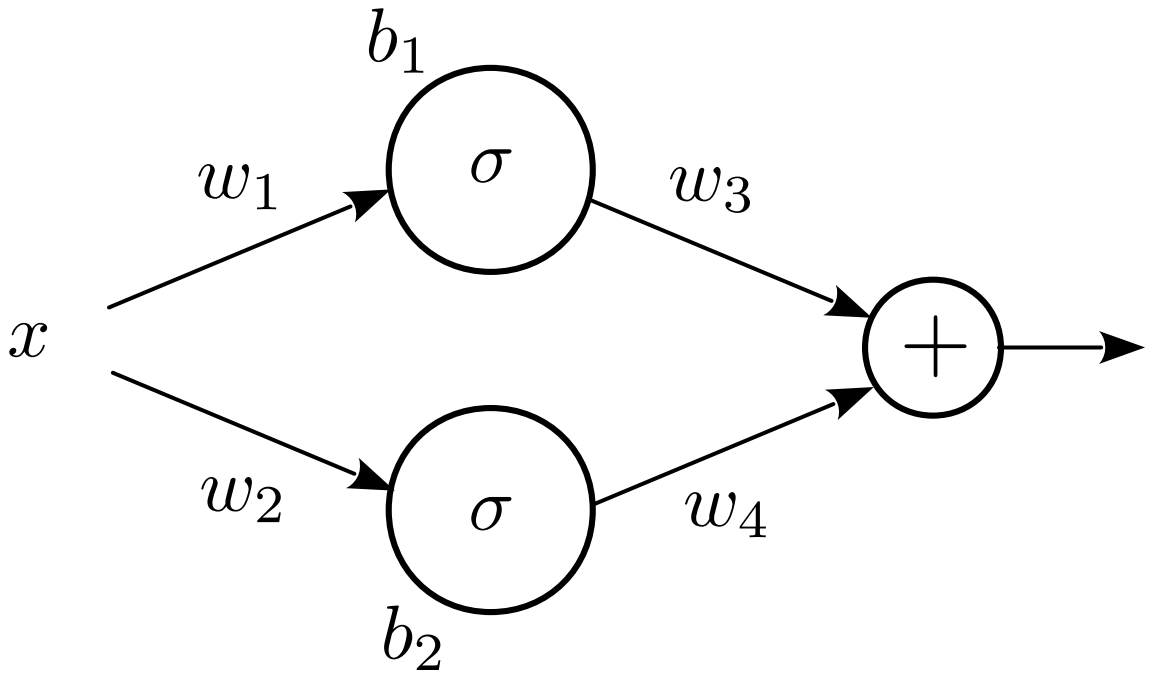
...universal function approximators

[Cybenko, G. (1989) *Approximations by superpositions of sigmoidal functions*]

Feed-Forward Neural Networks...

...universal function approximators

[Cybenko, G. (1989) *Approximations by superpositions of sigmoidal functions*]

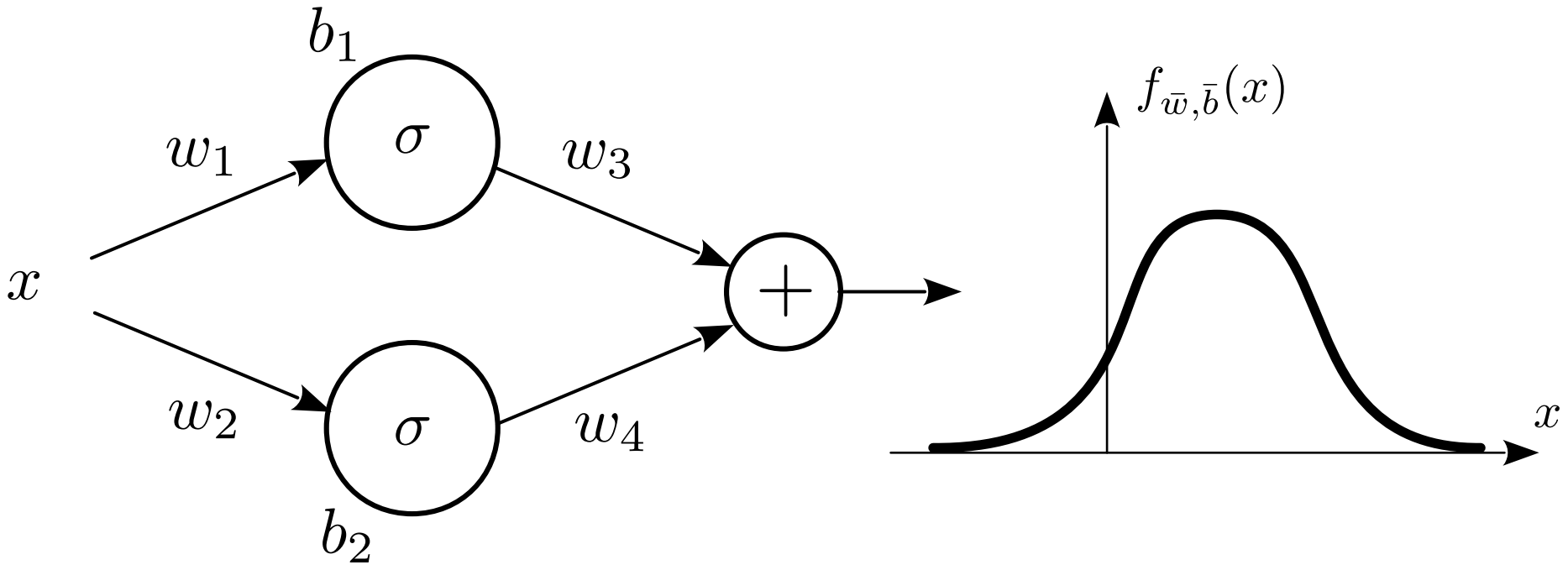


$$f_{\bar{w},\bar{b}}(x) = w_3\,\sigma\big(b_1 + w_1 x\big) + w_4\,\sigma\big(b_2 + w_2 x\big)$$

Feed-Forward Neural Networks...

...universal function approximators

[Cybenko, G. (1989) *Approximations by superpositions of sigmoidal functions*]

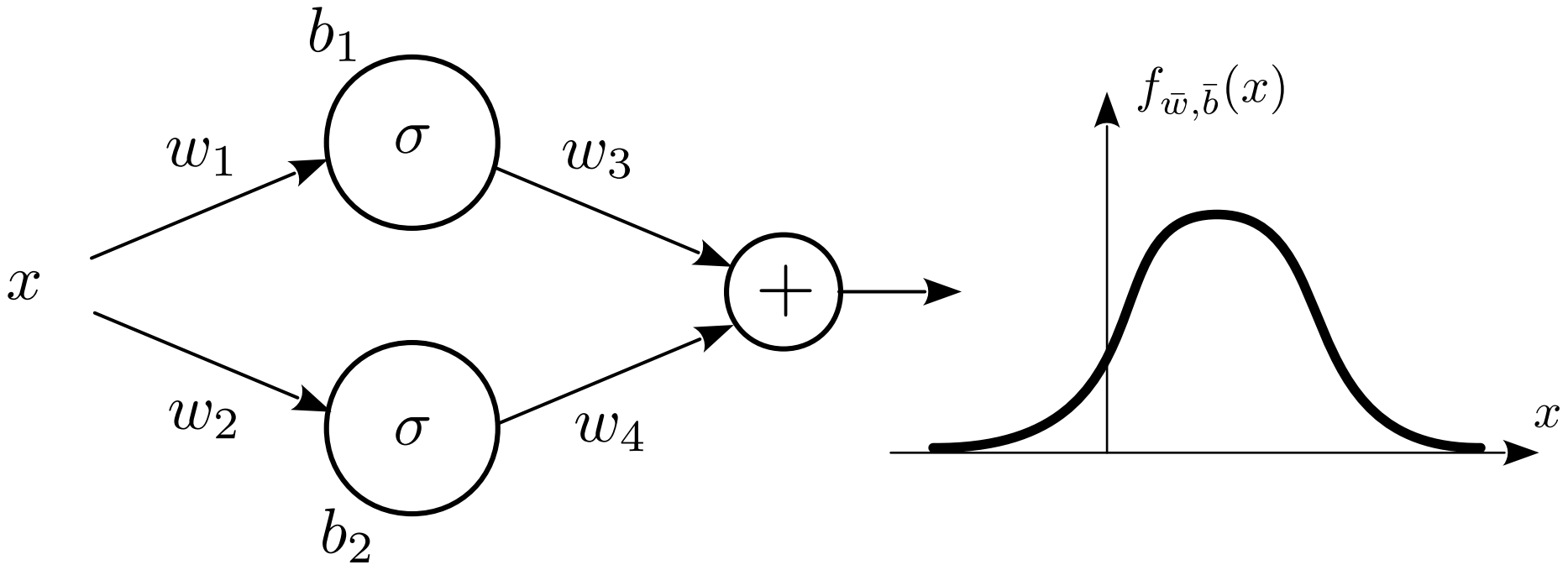


$$f_{\bar{w},\bar{b}}(x) = w_3\,\sigma\big(b_1 + w_1 x\big) + w_4\,\sigma\big(b_2 + w_2 x\big)$$

Feed-Forward Neural Networks...

...universal function approximators

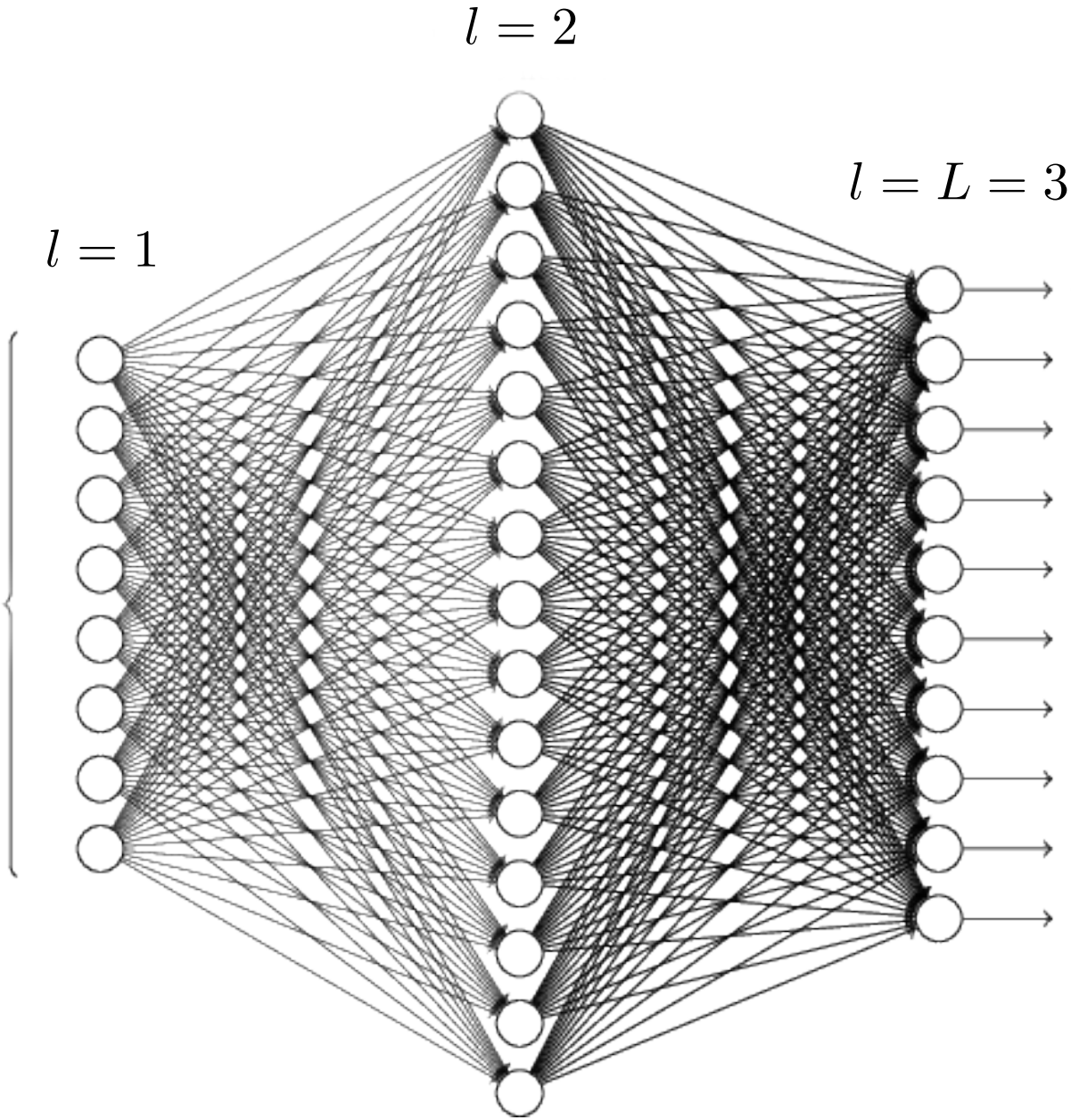
[Cybenko, G. (1989) *Approximations by superpositions of sigmoidal functions*]



$$f_{\bar{w},\bar{b}}(x) = w_3\,\sigma\big(b_1 + w_1 x\big) + w_4\,\sigma\big(b_2 + w_2 x\big)$$

Open Firefox: **www.geogebra.org**

Network for MNIST handwritten digit classification



Input layer:
784 neurons

$a^1(x) = x$

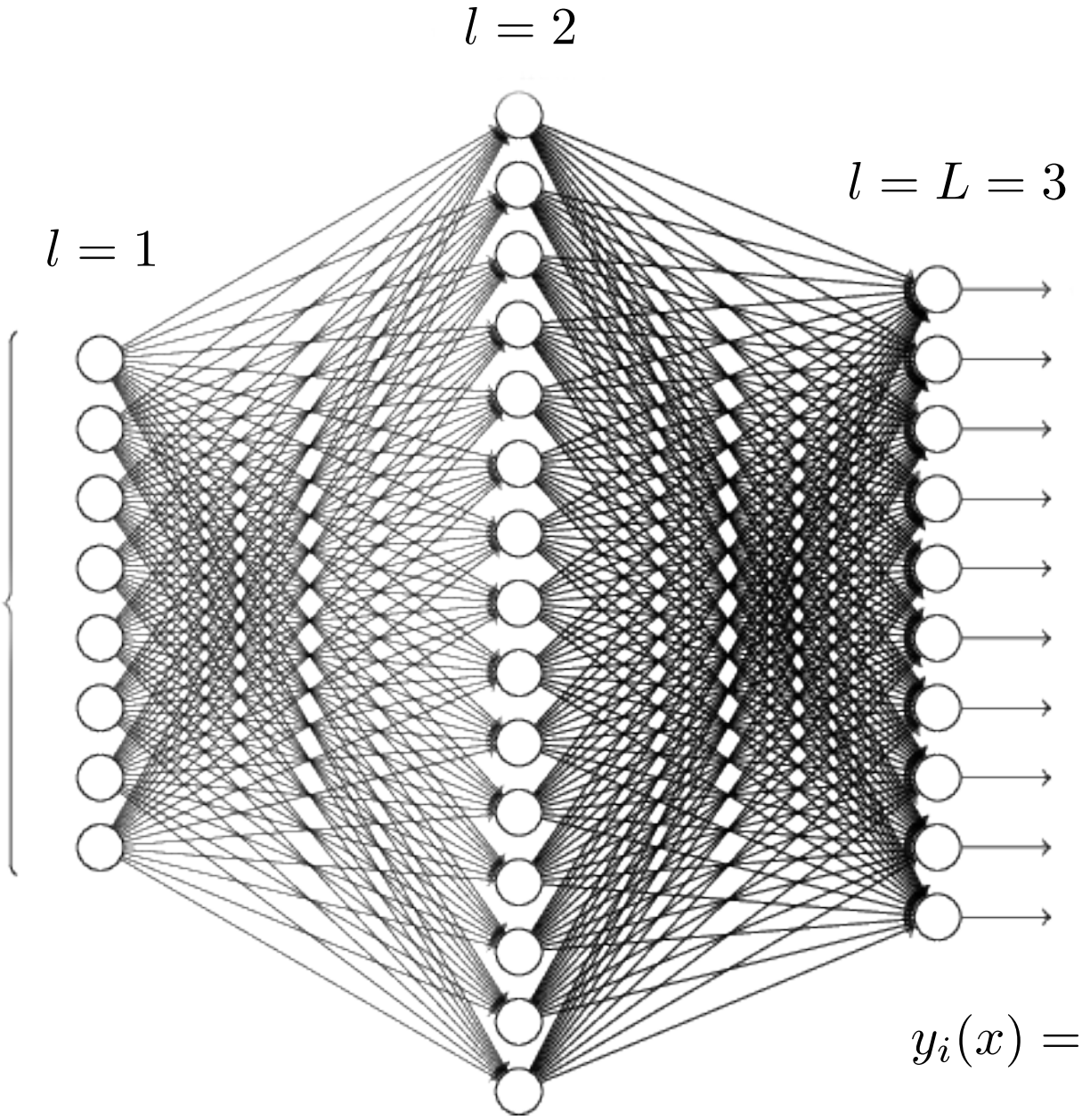
(grey-scale value of each pixel)

Output layer:
10 neurons

$a^L(x)$

index with maximum activity = predicted class

Network for MNIST handwritten digit classification



Input layer:
784 neurons

$a^1(x) = x$

(grey-scale value of each pixel)

Output layer:
10 neurons

$a^L(x)$

index with maximum activity = predicted class

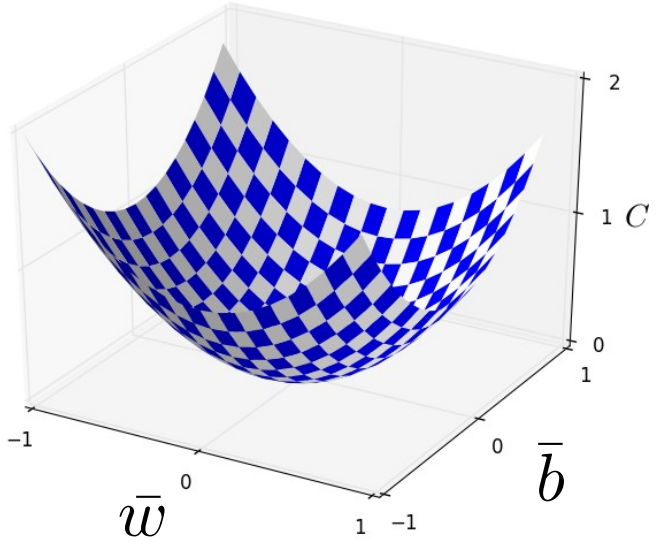
$$y_i(x) = \begin{cases} 1 & i = \text{digit} \\ 0 & \text{otherwise} \end{cases}$$

Learning through gradient descent

Minimize a **cost** (or **loss**) **function**:

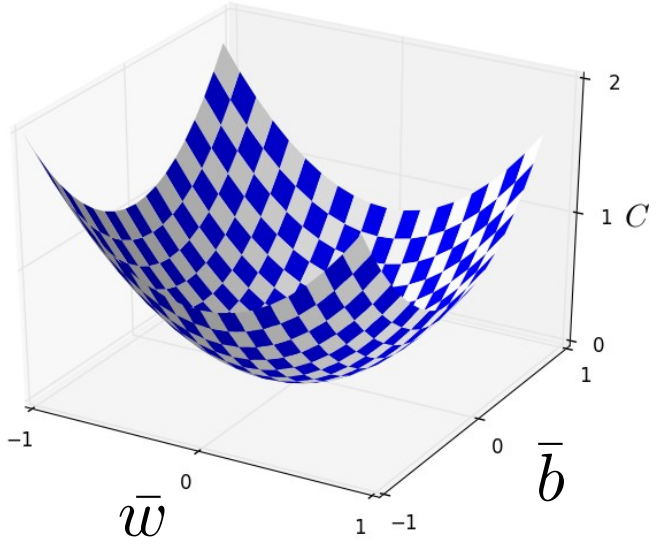
$$C(\bar{w}, \bar{b}) = \Big\langle C_x(\bar{w}, \bar{b}) \Big\rangle_{\substack{\text{training} \\ \text{data } x}} = \frac{1}{N} \sum_x C_x(\bar{w}, \bar{b})$$

training data



Learning through gradient descent

Minimize a **cost** (or **loss**) **function**:



$$C(\bar{w},\bar{b}) = \left\langle C_x(\bar{w},\bar{b}) \right\rangle_{\substack{\text{training} \\ \text{data } x}} = \frac{1}{N}\sum_x C_x(\bar{w},\,\bar{b})$$

training data

A possible choice: **quadratic** cost

$$C_x = \frac{1}{2}\sum_i \left(a_i^L(x) - y_i(x)\right)^2$$

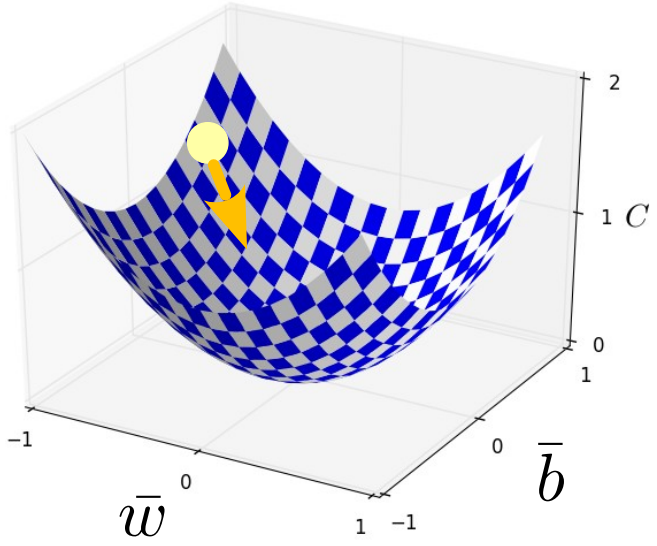
neurons in layer L (output)

Learning through gradient descent

Minimize a **cost** (or **loss**) **function**:

$$C(\bar{w}, \bar{b}) = \Big\langle C_x(\bar{w}, \bar{b}) \Big\rangle_{\substack{\text{training} \\ \text{data } x}} = \frac{1}{N} \sum_x C_x(\bar{w}, \bar{b})$$

training data



A possible choice: **quadratic** cost

$$C_x = \frac{1}{2} \sum_i \left(a_i^L(x) - y_i(x) \right)^2$$

neurons in layer L (output)

Gradient descent update rules:

$$w_{i,j}^l \leftarrow w_{i,j}^l - \eta \frac{\partial C}{\partial w_{i,j}^l}$$

$$b_i^l \leftarrow b_i^l - \eta \frac{\partial C}{\partial b_i^l}$$

η

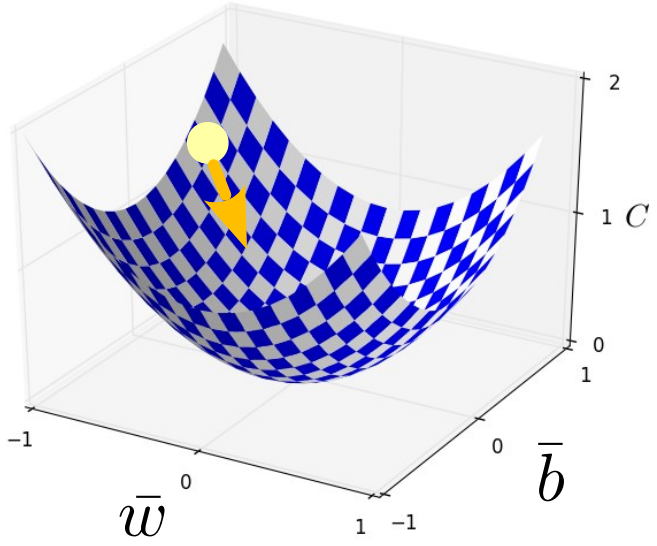
learning rate

Learning through gradient descent

Minimize a **cost** (or **loss**) **function**:

$$C(\bar{w}, \bar{b}) = \Big\langle C_x(\bar{w}, \bar{b}) \Big\rangle_{\substack{\text{training} \\ \text{data } x}} = \frac{1}{N} \sum_x C_x(\bar{w}, \bar{b})$$

training data



A possible choice: **quadratic** cost

$$C_x = \frac{1}{2} \sum_i \left(a_i^L(x) - y_i(x) \right)^2$$

neurons in layer L (output)

Gradient descent update rules:

$$w_{i,j}^l \leftarrow w_{i,j}^l - \eta \frac{\partial C}{\partial w_{i,j}^l}$$

$$b_i^l \leftarrow b_i^l - \eta \frac{\partial C}{\partial b_i^l}$$

η

learning rate

$$\frac{\partial C}{\partial w_{i,j}^l}, \frac{\partial C}{\partial b_i^l} = \; ?$$

The backpropagation algorithm

The **error**:

How much does the cost change as an effect of a change in the input of a given neuron in a given layer?

The backpropagation algorithm

The **error**:

How much does the cost change as an effect of a change in the input of a given neuron in a given layer?

$$\delta_i^l \equiv \frac{\Delta C_x}{\Delta z_i^l} \rightarrow \frac{\partial C_x}{\partial z_i^l}$$

The backpropagation algorithm

The **error**:

How much does the cost change as an effect of a change in the input of a given neuron in a given layer?

$$\delta_i^l \equiv \frac{\Delta C_x}{\Delta z_i^l} \rightarrow \frac{\partial C_x}{\partial z_i^l}$$

Small if the parameters are close to **optimal**.

The backpropagation algorithm

The **error**:

How much does the cost change as an effect of a change in the input of a given neuron in a given layer?

$$\delta_i^l \equiv \frac{\Delta C_x}{\Delta z_i^l} \rightarrow \frac{\partial C_x}{\partial z_i^l}$$

Small if the parameters are close to **optimal**.

$$\frac{\partial C_x}{\partial w_{ij}^l}$$

The backpropagation algorithm

The **error:**

How much does the cost change as an effect of a change in the input of a given neuron in a given layer?

$$\delta_i^l \equiv \frac{\Delta C_x}{\Delta z_i^l} \rightarrow \frac{\partial C_x}{\partial z_i^l}$$

Small if the parameters are close to **optimal.**

$$\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

chain rule $z_i^l = b_i^l + \sum_j w_{i,j}^l \, a_j^{l-1}$

The backpropagation algorithm

The **error:**

How much does the cost change as an effect of a change in the input of a given neuron in a given layer?

$$\delta_i^l \equiv \frac{\Delta C_x}{\Delta z_i^l} \rightarrow \frac{\partial C_x}{\partial z_i^l}$$

Small if the parameters are close to **optimal.**

$$\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l} = \delta_i^l \, a_j^{l-1}$$

chain rule $z_i^l = b_i^l + \sum_j w_{i,j}^l \, a_j^{l-1}$

The backpropagation algorithm

The **error:**

How much does the cost change as an effect of a change in the input of a given neuron in a given layer?

$$\delta_i^l \equiv \frac{\Delta C_x}{\Delta z_i^l} \rightarrow \frac{\partial C_x}{\partial z_i^l}$$

Small if the parameters are close to **optimal.**

$$\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l} = \delta_i^l \, a_j^{l-1}$$

chain rule $z_i^l = b_i^l + \sum_j w_{i,j}^l \, a_j^{l-1}$

$$\frac{\partial C_x}{\partial b_i^l}$$

The backpropagation algorithm

The **error:**

How much does the cost change as an effect of a change in the input of a given neuron in a given layer?

$$\delta_i^l \equiv \frac{\Delta C_x}{\Delta z_i^l} \rightarrow \frac{\partial C_x}{\partial z_i^l}$$

Small if the parameters are close to **optimal.**

$$\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l} = \delta_i^l \, a_j^{l-1}$$

chain rule $z_i^l = b_i^l + \sum_j w_{i,j}^l \, a_j^{l-1}$

$$\frac{\partial C_x}{\partial b_i^l} = \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial b_i^l}$$

The backpropagation algorithm

The **error:**

How much does the cost change as an effect of a change in the input of a given neuron in a given layer?

$$\delta_i^l \equiv \frac{\Delta C_x}{\Delta z_i^l} \rightarrow \frac{\partial C_x}{\partial z_i^l}$$

Small if the parameters are close to **optimal.**

$$\frac{\partial C_x}{\partial w_{ij}^l} = \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l} = \delta_i^l \, a_j^{l-1}$$

chain rule $z_i^l = b_i^l + \sum_j w_{i,j}^l \, a_j^{l-1}$

$$\frac{\partial C_x}{\partial b_i^l} = \frac{\partial C}{\partial z_i^l} \frac{\partial z_i^l}{\partial b_i^l} = \delta_i^l$$

The backpropagation algorithm

From the activity of the **output** layer...

$$\delta_i^L = \frac{\partial C_x}{\partial z_i^L}$$

The backpropagation algorithm

From the activity of the **output** layer…

predicted – desired

$$\delta_i^L = \frac{\partial C_x}{\partial z_i^L} = \overbrace{\left(a_i^L(x) - y_i(x)\right)}\, \sigma'\left(z_i^L\right)$$

$$C_x = \sum_{\substack{j \\ \text{layer } L}} \frac{1}{2}\left(a_j^L(x) - y_j(x)\right)^2 \qquad C = \frac{1}{N}\sum_x C_x$$

The backpropagation algorithm

From the activity of the **output** layer…

$$\delta_i^L = \frac{\partial C_x}{\partial z_i^L} = \overbrace{\left(a_i^L(x) - y_i(x)\right)}^{\text{predicted – desired}} \sigma'\left(z_i^L\right)$$

$$C_x = \sum_{\substack{j \\ \text{layer } L}} \frac{1}{2}\left(a_j^L(x) - y_j(x)\right)^2 \qquad C = \frac{1}{N}\sum_x C_x$$

… and **back-propagate** to the previous layers

$$\delta_j^{l-1} = \frac{\partial C_x}{\partial z_j^{l-1}}$$

The backpropagation algorithm

From the activity of the **output** layer…

$$\delta_i^L = \frac{\partial C_x}{\partial z_i^L} = \overbrace{\left(a_i^L(x) - y_i(x)\right)}^{\text{predicted – desired}} \sigma'\left(z_i^L\right)$$

$$C_x = \sum_{\substack{j \\ \text{layer } L}} \frac{1}{2}\left(a_j^L(x) - y_j(x)\right)^2 \qquad C = \frac{1}{N}\sum_x C_x$$

… and **back-propagate** to the previous layers

$$\delta_j^{l-1} = \frac{\partial C_x}{\partial z_j^{l-1}} = \sum_i \frac{\partial C_x}{\partial z_i^l}\,\frac{\partial z_i^l}{\partial z_j^{l-1}}$$

chain rule $\quad z_i^l = b_i^l + \sum_j w_{i,j}^l\, a_j^{l-1}$

The backpropagation algorithm

From the activity of the **output** layer…

$$\delta_i^L = \frac{\partial C_x}{\partial z_i^L} = \overbrace{\left(a_i^L(x) - y_i(x)\right)}^{\text{predicted – desired}} \sigma'\left(z_i^L\right)$$

$$C_x = \sum_{\substack{j \\ \text{layer } L}} \frac{1}{2}\left(a_j^L(x) - y_j(x)\right)^2 \qquad C = \frac{1}{N}\sum_x C_x$$

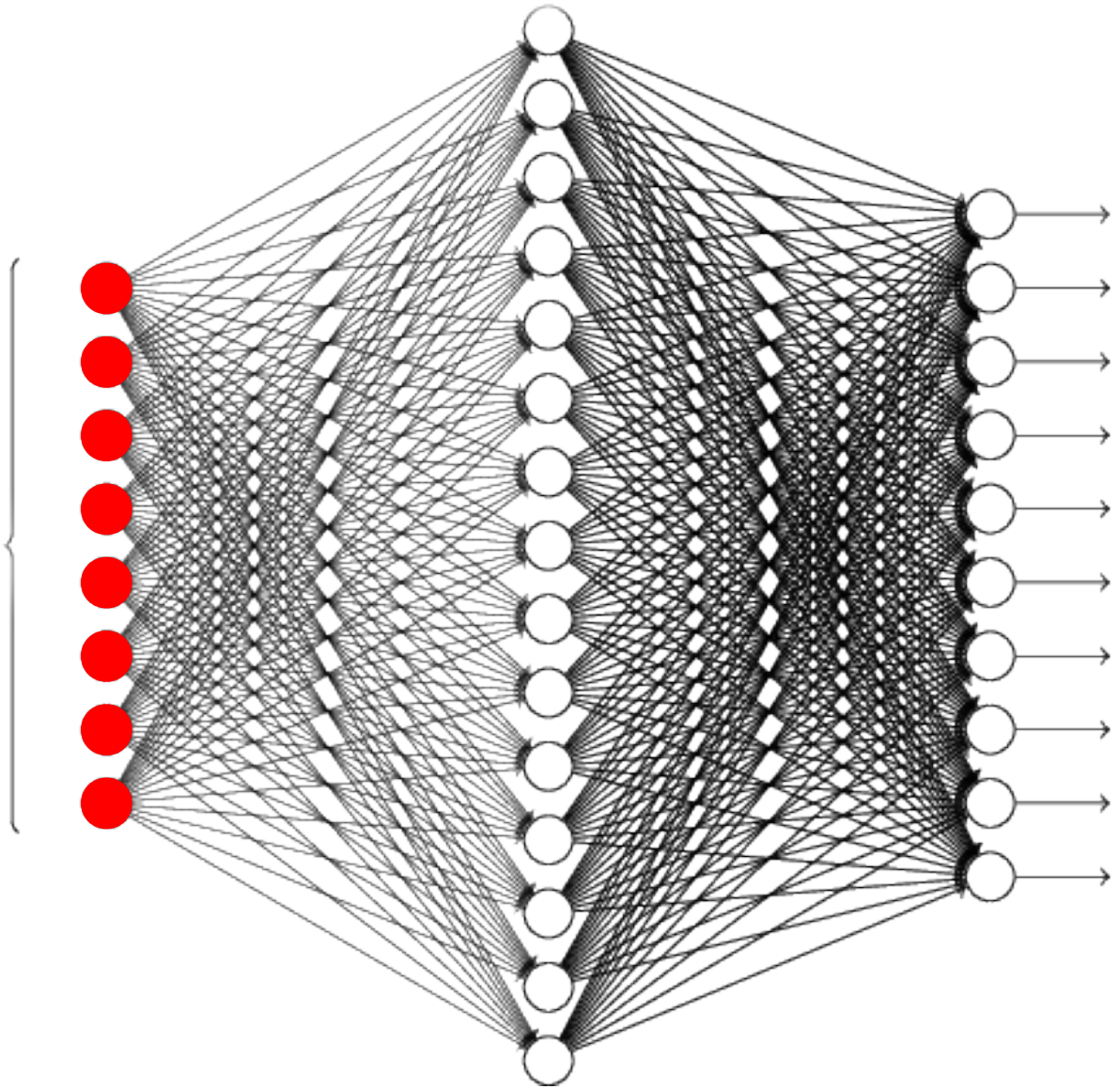
… and **back-propagate** to the previous layers

$$\delta_j^{l-1} = \frac{\partial C_x}{\partial z_j^{l-1}} = \sum_i \frac{\partial C_x}{\partial z_i^l}\,\frac{\partial z_i^l}{\partial z_j^{l-1}} = \sum_i \delta_i^l\, w_{ij}^l\, \sigma'\left(z_j^{l-1}\right)$$

chain rule $\quad z_i^l = b_i^l + \sum_j w_{i,j}^l\, a_j^{l-1}$

The backpropagation algorithm

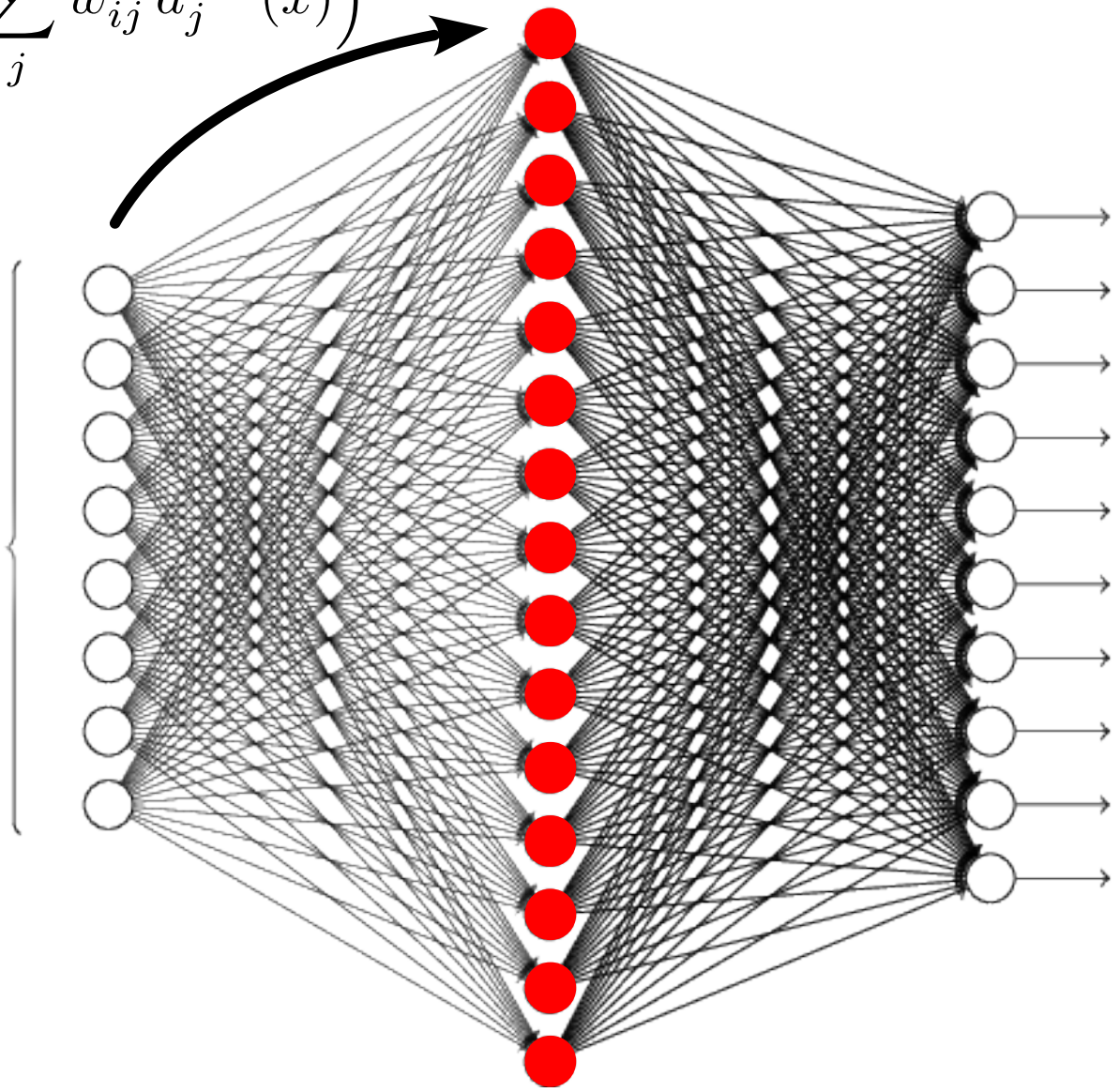
For every input x **1. Feed-forward**



The backpropagation algorithm

For every input x **1. Feed-forward**

$$a_i^l(x) = \sigma\left(b_i^l + \sum_j w_{ij}^l \, a_j^{l-1}(x)\right)$$



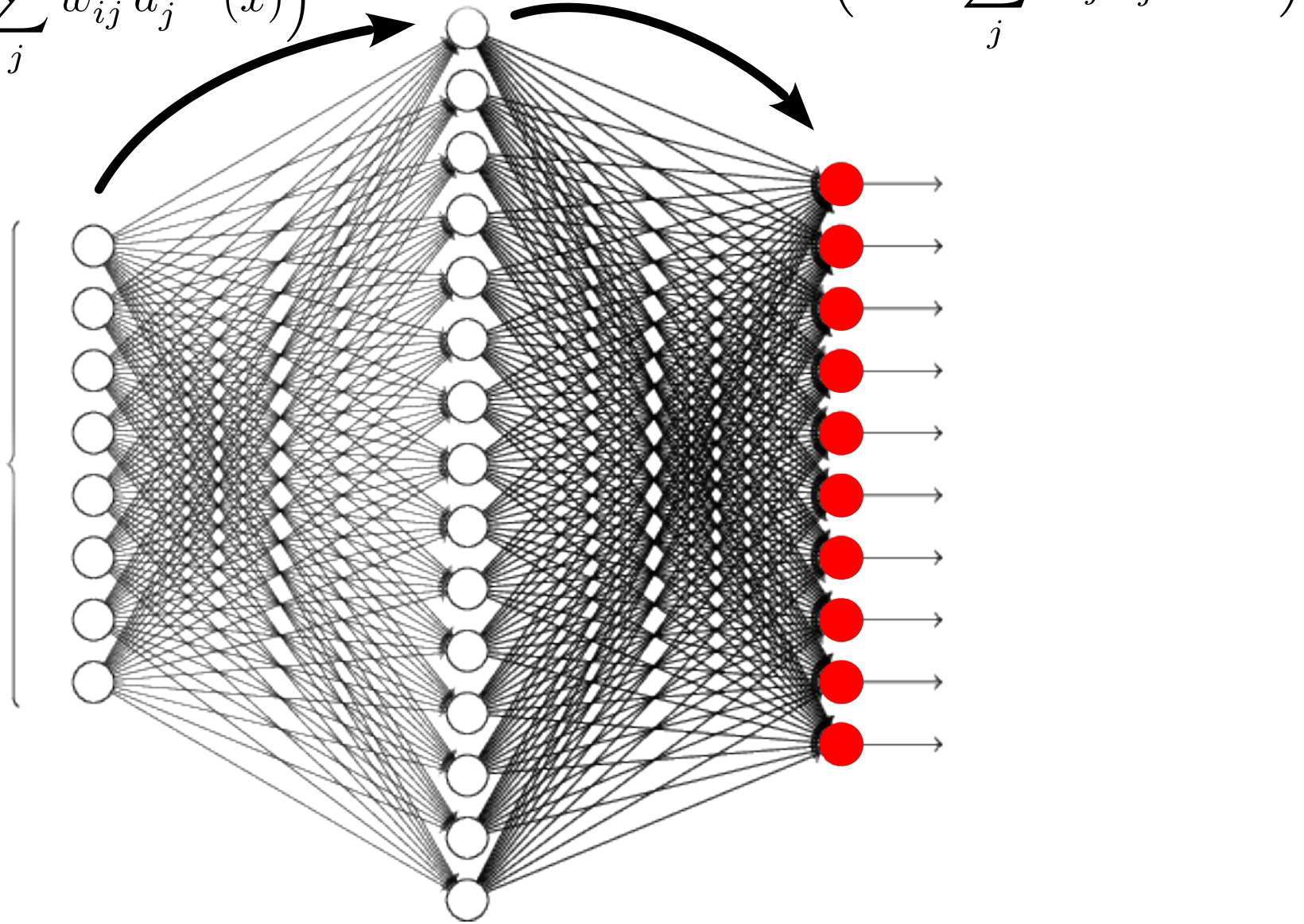
The backpropagation algorithm

For every input x

1. Feed-forward

$$a_i^l(x) = \sigma\left(b_i^l + \sum_j w_{ij}^l \, a_j^{l-1}(x)\right)$$

$$a_i^L(x) = \sigma\left(b_i^L + \sum_j w_{ij}^L \, a_j^{L-1}(x)\right)$$

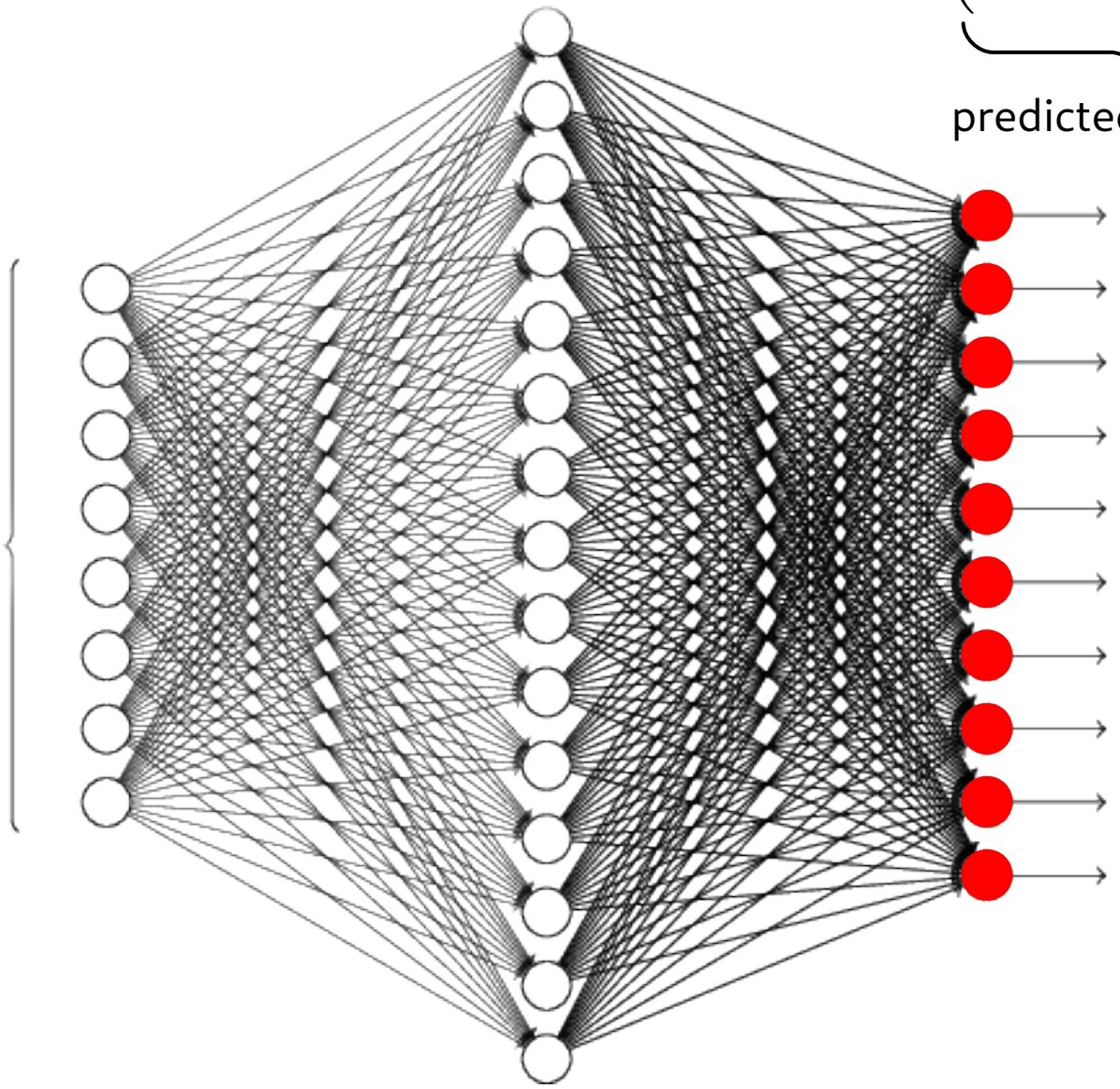


The backpropagation algorithm

For every input x

2. Calculate the **error** at **output**

$$\delta_i^L(x) = \underbrace{\left(a_i^L(x) - y_i(x)\right)}_{\text{predicted – desired}} \sigma'\left(z_i^L\right)$$

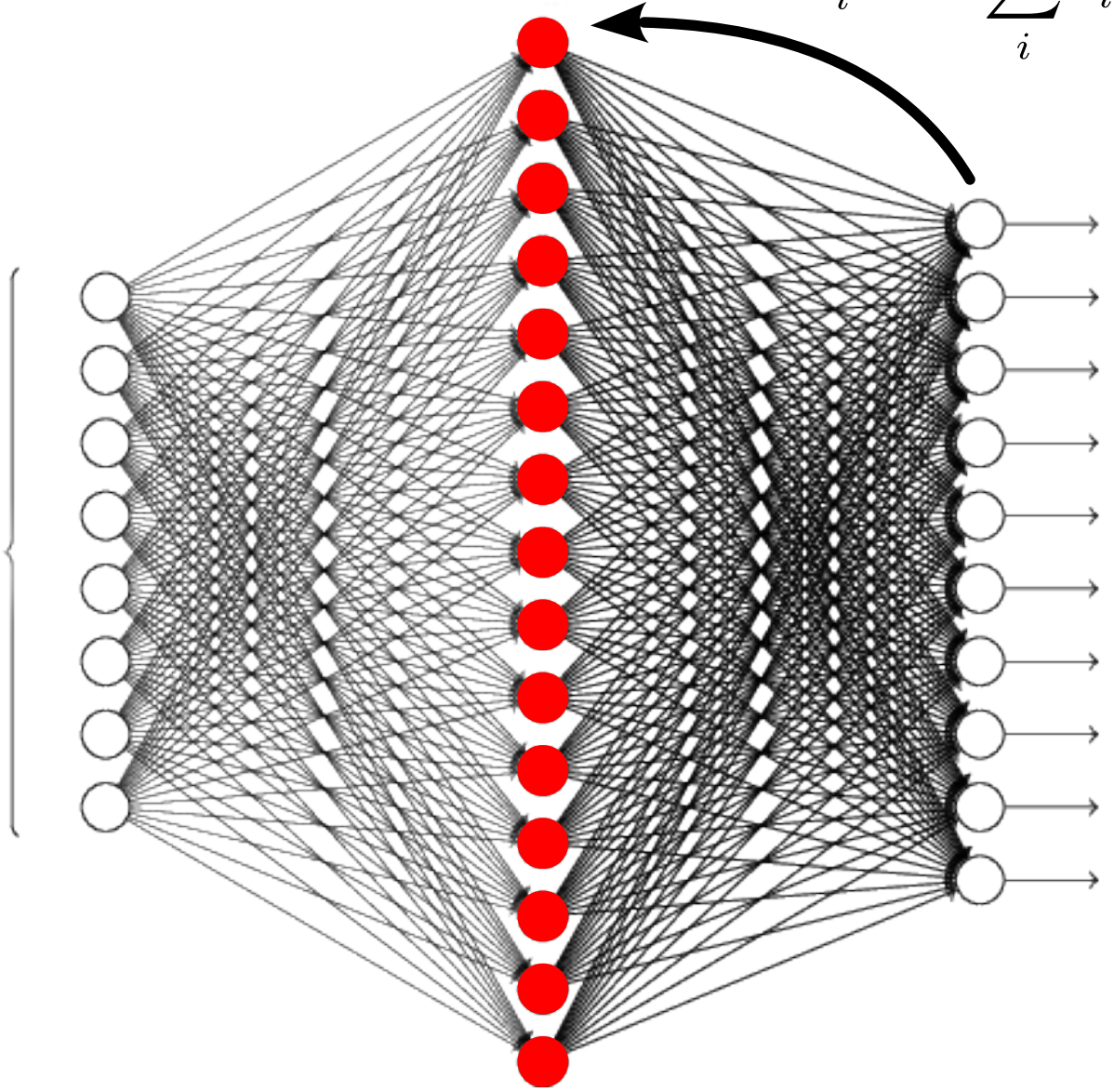


The backpropagation algorithm

For every input x

3. **Back-propagate** the error

$$\delta_i^{l-1} = \sum_i \delta_i^l \, w_{ij}^l \, \sigma'\left(z_j^{l-1}\right)$$



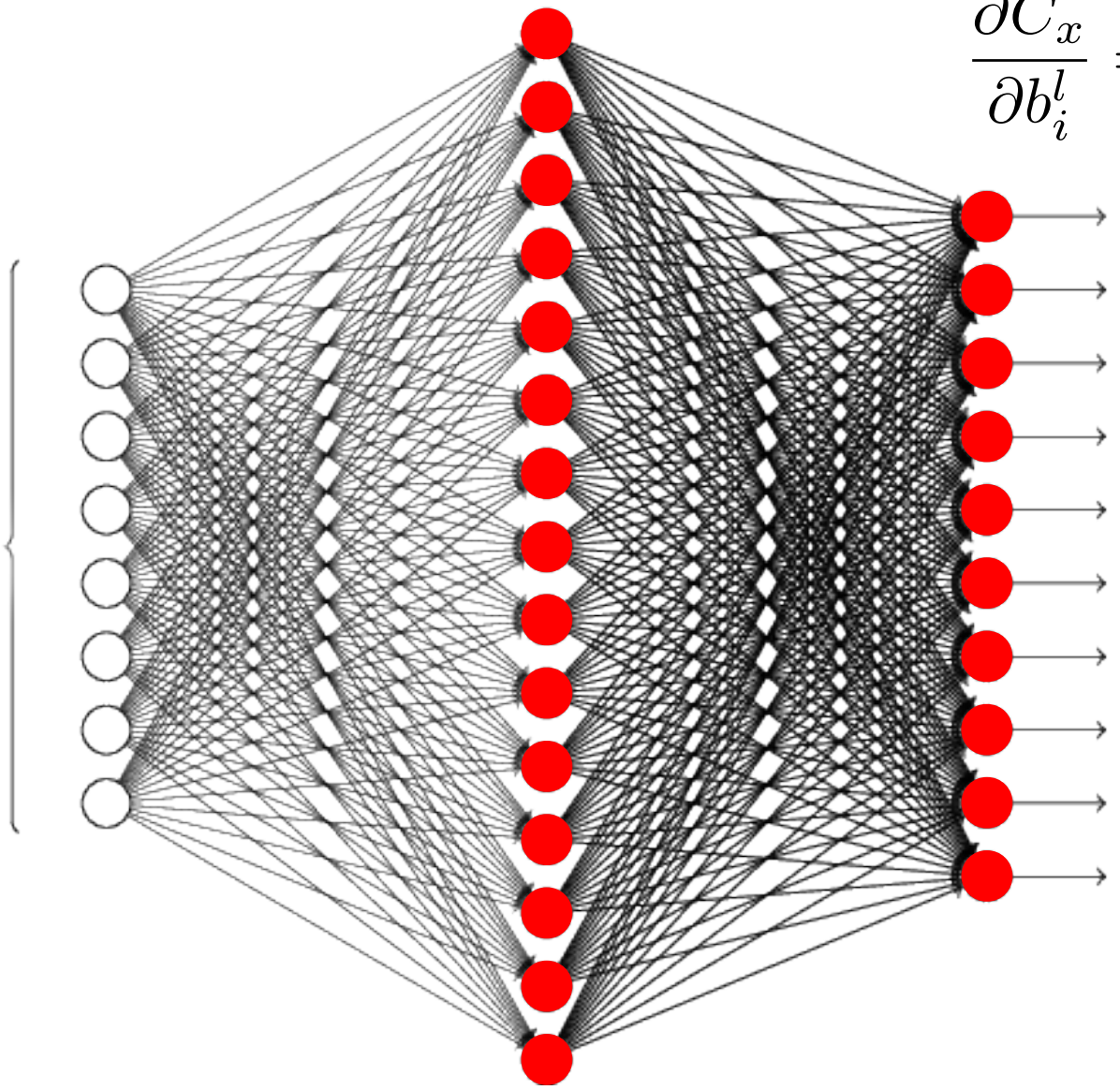
The backpropagation algorithm

For every input x

4. Calculate the **derivatives**

$$\frac{\partial C_x}{\partial w_{ij}^l} = \delta_i^l \, a_j^{l-1}$$

$$\frac{\partial C_x}{\partial b_i^l} = \delta_i^l$$



The mini-batch update

$$C = \frac{1}{N} \sum_x C_x \quad \Rightarrow \quad \nabla C = \frac{1}{N} \sum_x \nabla C_x$$

The mini-batch update

$$C = \frac{1}{N} \sum_x C_x \quad \Rightarrow \quad \nabla C = \frac{1}{N} \sum_x \nabla C_x$$

Divide the training dataset into M **mini-batches** of size S: $N = M \times S$

{training data} = { {mini-batch 1}, {mini-batch 2}, ... {mini-batch M} }

The mini-batch update

$$C = \frac{1}{N} \sum_x C_x \quad \Rightarrow \quad \nabla C = \frac{1}{N} \sum_x \nabla C_x$$

Divide the training dataset into M **mini-batches** of size S: $N = M \times S$

{training data} = { {mini-batch 1}, {mini-batch 2}, ... {mini-batch M} }

Average **within** mini-batches first...

$$\overline{C}_m = \frac{1}{S} \sum_{\substack{\text{mini-batch} \\ m}} C_x \quad \Rightarrow \quad \overline{\nabla C}_m = \frac{1}{S} \sum_{\substack{\text{mini-batch} \\ m}} \nabla C_x$$

The mini-batch update

$$C = \frac{1}{N}\sum_x C_x \quad \Rightarrow \quad \nabla C = \frac{1}{N}\sum_x \nabla C_x$$

Divide the training dataset into M **mini-batches** of size S: $N = M \times S$

{training data} = { {mini-batch 1}, {mini-batch 2}, ... {mini-batch M} }

Average **within** mini-batches first...

$$\overline{C}_m = \frac{1}{S}\sum_{\substack{\text{mini-batch}\\ m}} C_x \quad \Rightarrow \quad \overline{\nabla C}_m = \frac{1}{S}\sum_{\substack{\text{mini-batch}\\ m}} \nabla C_x$$

... and use the **mini-batch estimate** of the gradient for gradient descent

$$w \leftarrow w - \eta\,\overline{\nabla_w C}_m \qquad b \leftarrow b - \eta\,\overline{\nabla_b C}_m$$

(then **iterate** over all mini-batches)

The mini-batch update

$$C = \frac{1}{N}\sum_x C_x \quad \Rightarrow \quad \nabla C = \frac{1}{N}\sum_x \nabla C_x$$

Divide the training dataset into M **mini-batches** of size S: $N = M \times S$

{training data} = { {mini-batch 1}, {mini-batch 2}, ... {mini-batch M} }

Average **within** mini-batches first...

$$\overline{C}_m = \frac{1}{S}\sum_{\substack{\text{mini-batch}\\ m}} C_x \quad \Rightarrow \quad \overline{\nabla C}_m = \frac{1}{S}\sum_{\substack{\text{mini-batch}\\ m}} \nabla C_x$$

... and use the **mini-batch estimate** of the gradient for gradient descent

$$w \leftarrow w - \eta\,\overline{\nabla_w C}_m \qquad b \leftarrow b - \eta\,\overline{\nabla_b C}_m$$

(then **iterate** over all mini-batches)

Stochastic Gradient Descent (SGD)

Put your hands on the code!

Open the **terminal**

```
$   cd <main directory>/Lecture1

$   jupyter-notebook MNIST_notebook.ipynb
```