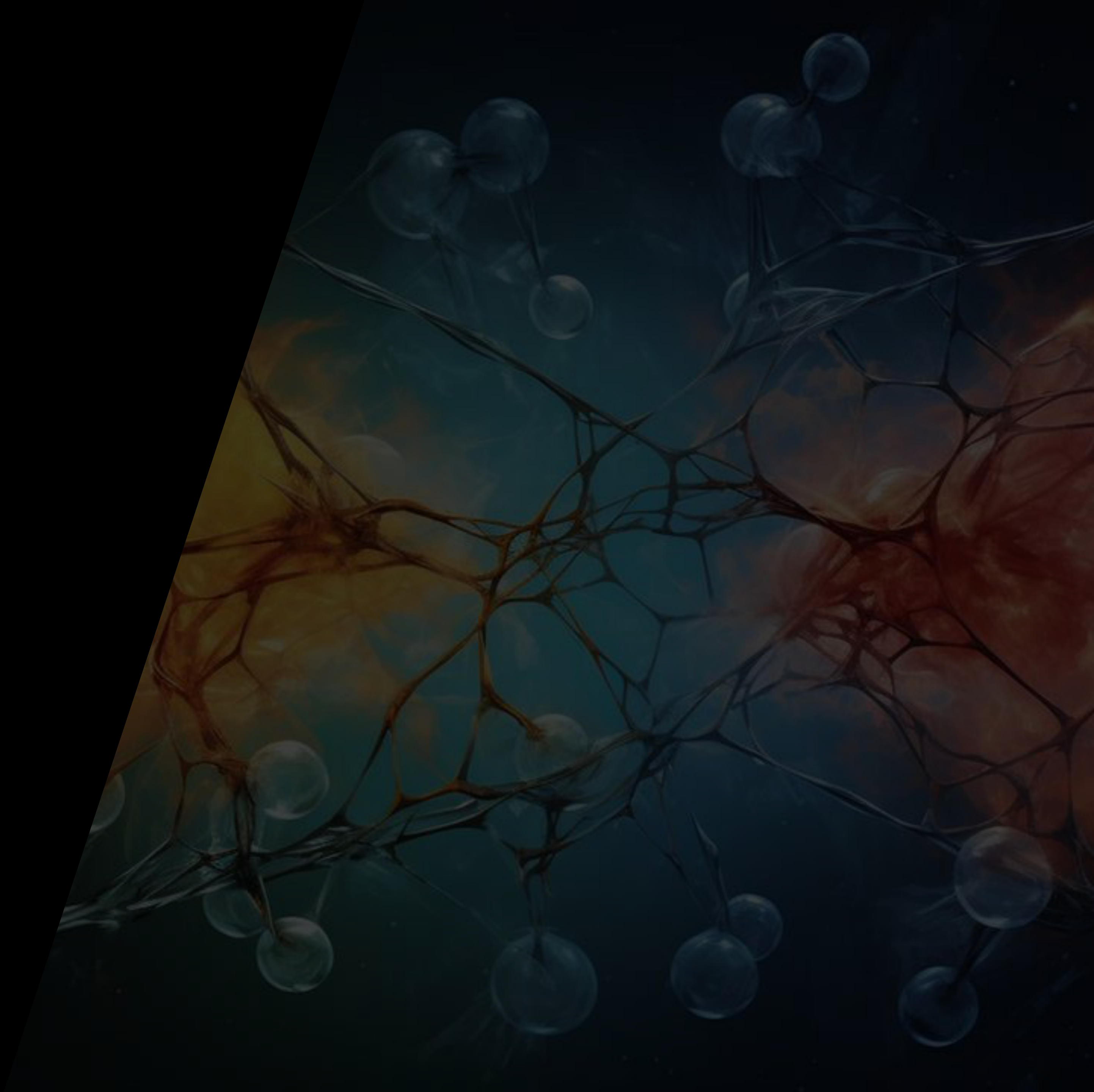
IV – Neural networks

Contents:

- *Neurons, activation functions*
- *Learning features with feed-forward neural networks*
- *Convolutional neural networks: invariances, convolutions, pooling*

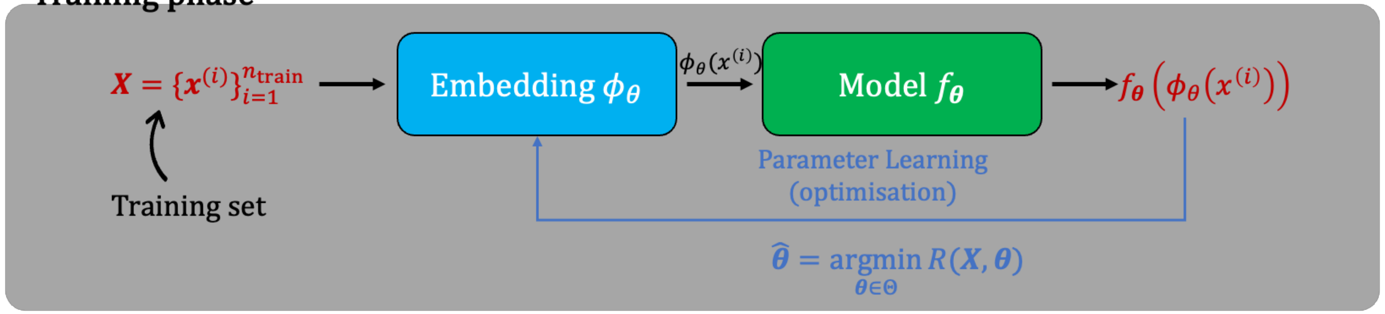


Artificial neural networks: neurons

Artificial neural networks

- How can we handcraft features $\phi(\boldsymbol{x})$ for complex problems like real-life image classification allowing the use of simple linear decision models?
- **The idea of neural network is to parameterise the embedding $\boldsymbol{\phi_\theta}(\mathrm{x})$ and find the basis linearising the problem.**

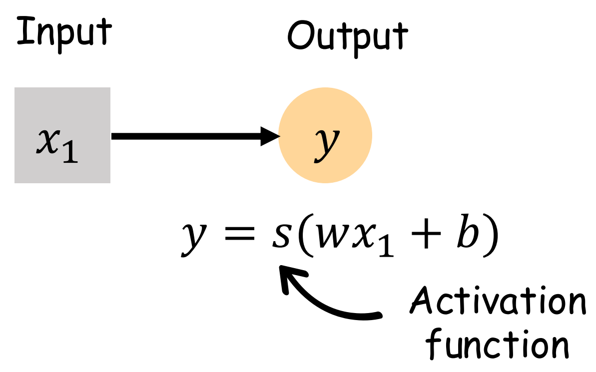
Training phase

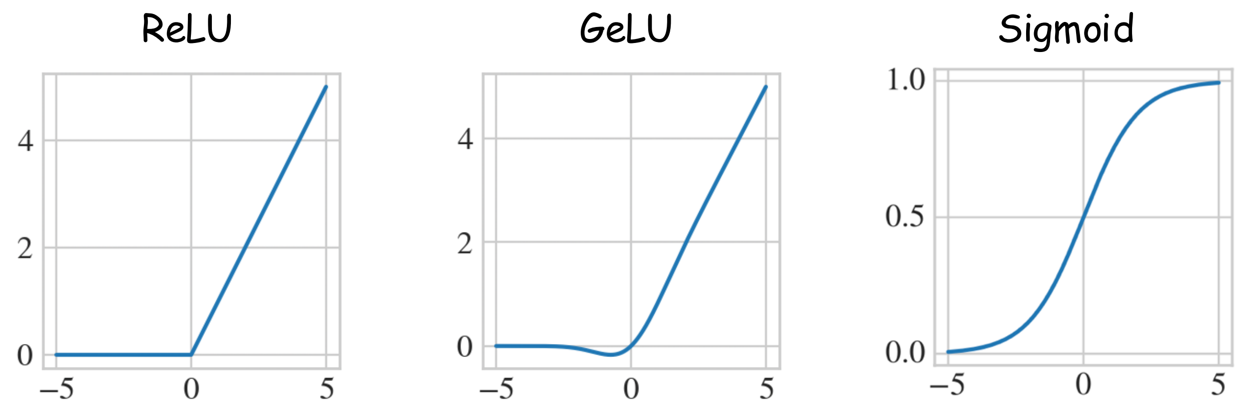


Images from the ImageNet dataset

Artificial neural networks

- Building block of neural networks: the **neuron**
- Made of three operations: it first multiplies the input by a **weight**, then adds a **bias**, and finally applies an **activation function** s to the result
- If s is the identity, you recognize the linear regression model with $\theta_0 = b$ and $\theta_1 = w$. To represent non-linear functions, s must be non-linear



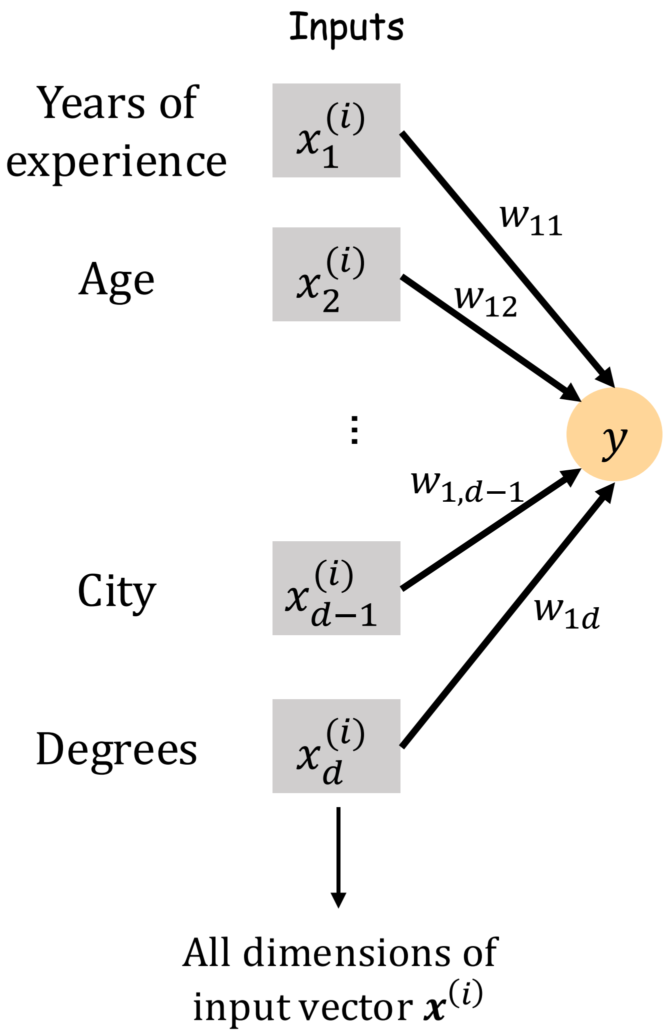


$s(x) = \max(0, x)$ $\quad$ $s(x) = \frac{x}{2}\left(1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $\quad$ $s(x) = \frac{1}{1 + \exp(-x)}$

- This model is motivated by biological neurons and the activation function mimics the activation or inhibition through **non-linear (and differentiable) functions**
- Can take different forms but some examples include the **ReLU or sigmoid functions**

Artificial neural networks: more features

Artificial neural networks



Notation: w_{ij} is the weight linking the unit j of the first layer to the unit i of the next layer

Output

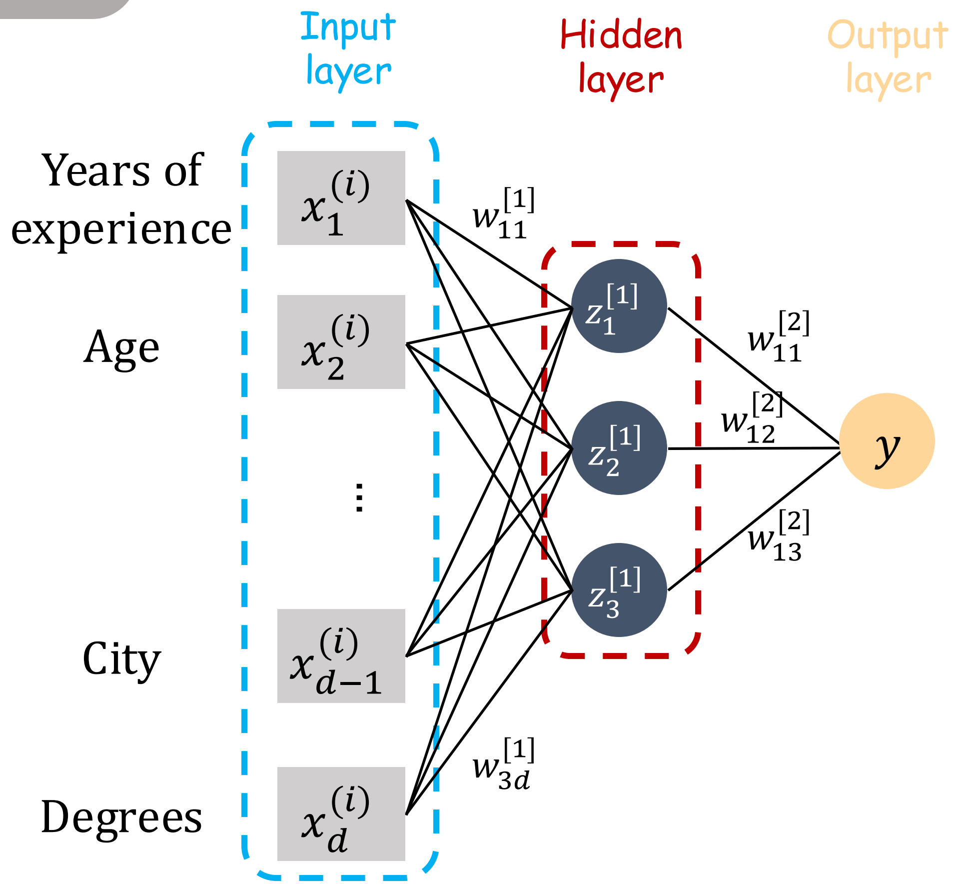
$$y = s\left(\sum_{j=1}^{d} w_{1j} x_j^{(i)} + b\right)$$

Salary

$$y = s\left(\boldsymbol{w}_1 \cdot \boldsymbol{x}^{(i)} + b\right)$$

Such a network has $p = d + 1$ parameters (as in multi-feature linear regression)

Artificial neural networks



- Units in a same layer do not interact
- Data go from input to output in a **feed-forward way**
- The width of the output layer corresponds to the number of classes/values that you want to predict
- The **activation** of the unit j in layer one is given by

Layer 1

Weights of layer 1 for unit j (vectorized)

$$z_j^{[1]} = s\left(\boldsymbol{w}_j^{[1]} \cdot \boldsymbol{x}^{(i)} + b_j^{[1]}\right)$$

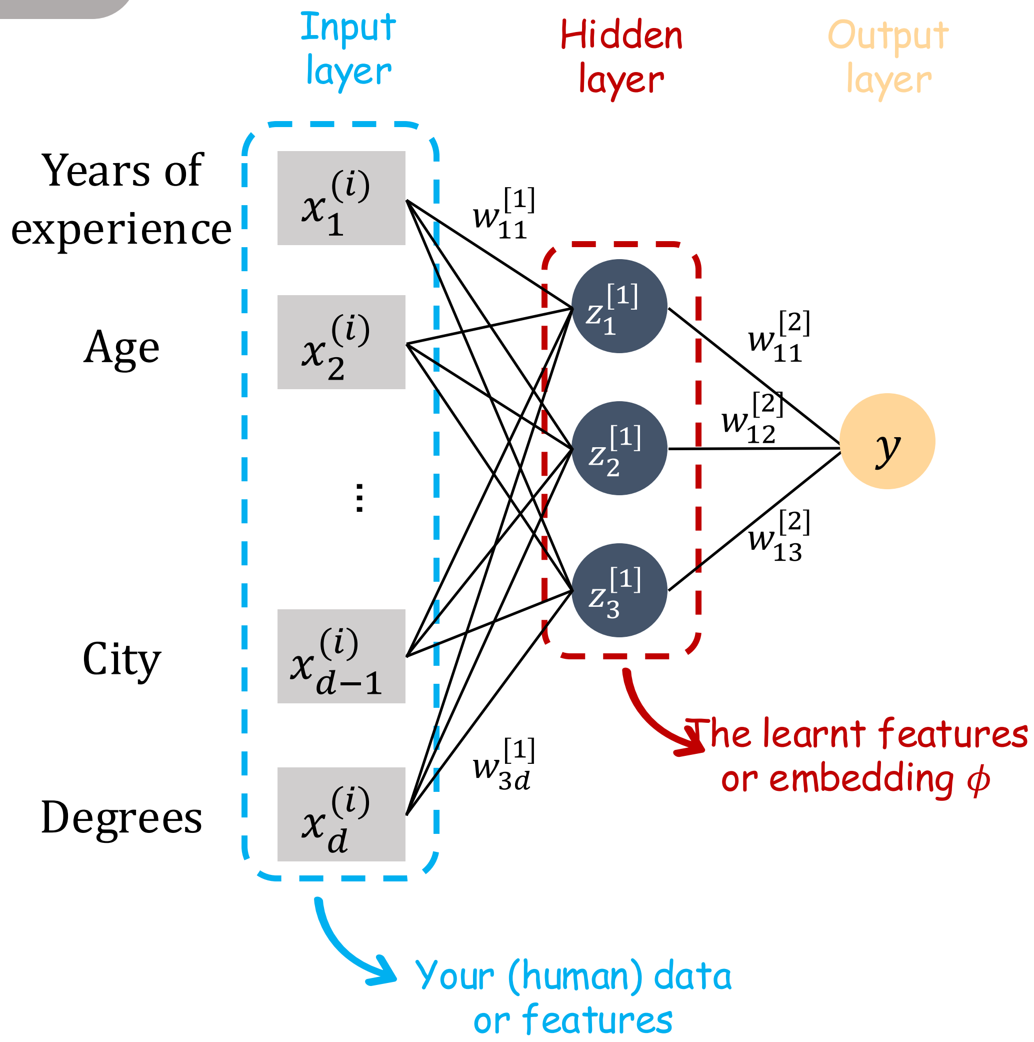
Unit j

Bias of the unit j in layer 1

- We also define the **pre-activation**

$$u_j^{[1]} = \boldsymbol{w}_j^{[1]} \cdot \boldsymbol{x} + b_j^{[1]}$$

Artificial neural networks



Let's have a look at the output of this network

$$y = s\left(\sum_{j=1}^{d} w_{1j}^{[2]} z_j^{[1]} + b^{[2]}\right)$$

$\{z_j\}_{j=1,\dots,3}$ act as **new features** to predict y

Artificial neural networks

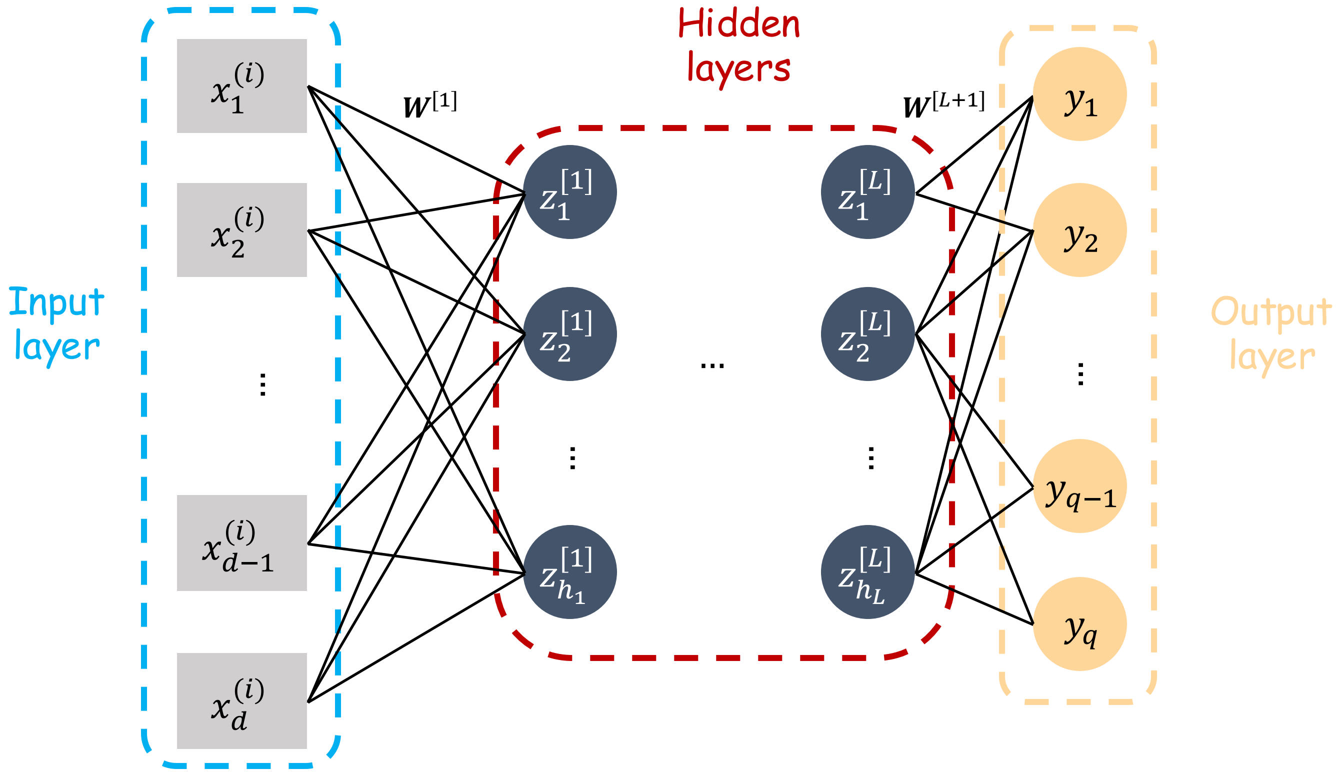
- A fully-connected neural network with d inputs, L hidden layers of width $h_1, h_2, \dots, h_L$ and an output layer of size q
- The j^{th} output is computed as

$$\boldsymbol{y} = s^{[L+1]}\left(\boldsymbol{W}^{[L+1]}\boldsymbol{z}^{[L]}\right) \qquad \text{with } \boldsymbol{z}^{[L]} = \left[1, z_1^{[L]}, z_2^{[L]}, \dots, z_{h_L}^{[L]}\right]^{\mathrm{T}}$$

$$\boldsymbol{y} = s^{[L+1]}\left(\boldsymbol{W}^{[L+1]} s^{[L]}\left(\boldsymbol{W}^{[L]} \dots s^{[1]}\left(\boldsymbol{W}^{[1]}\boldsymbol{x}\right)\right)\right)$$

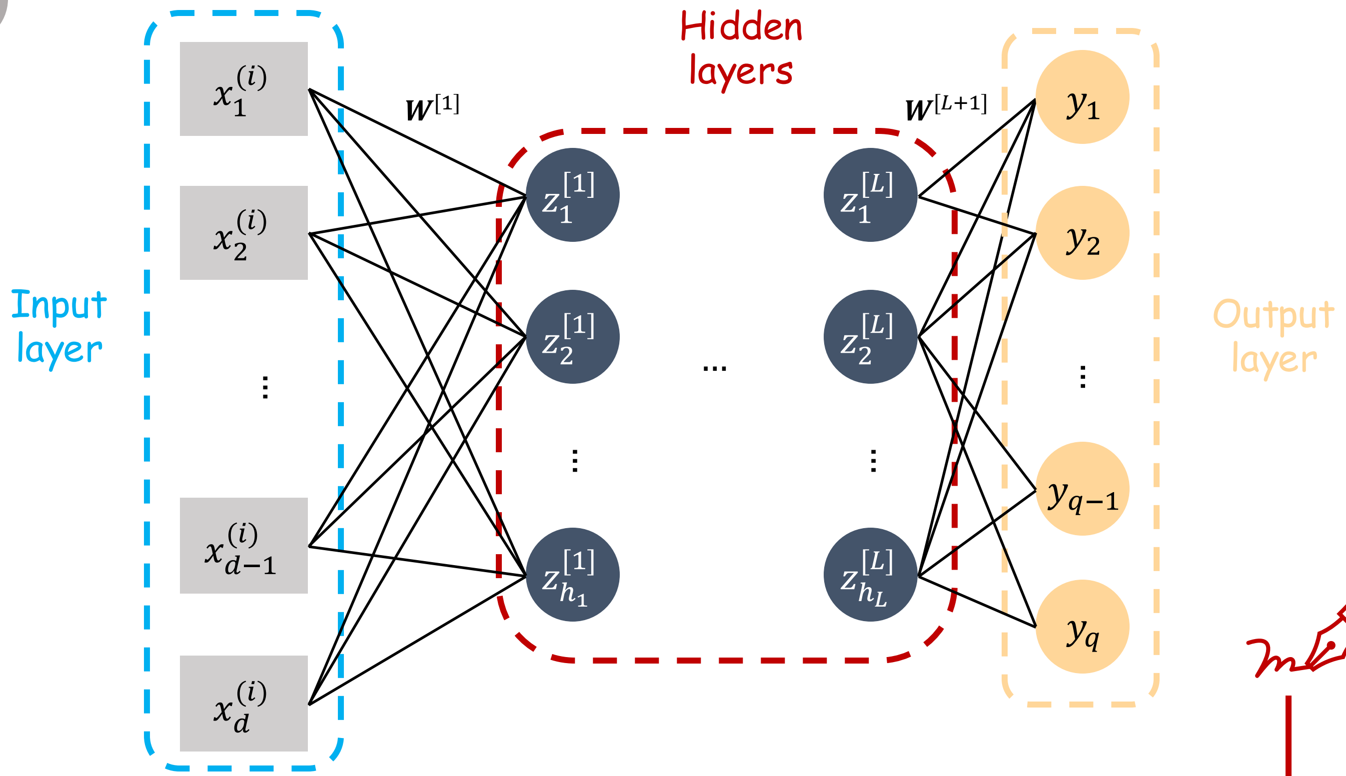
$$\boldsymbol{w}_j^{[l]} = \left[b_j^{[l]}, w_{1j}^{[l]}, w_{2j}^{[l]}, \dots, w_{h_l j}^{[l]}\right]$$

$$\boldsymbol{W}^{[l]} = \left[\boldsymbol{w}_1^{[l]}, \boldsymbol{w}_2^{[l]}, \dots, \boldsymbol{w}_{h_{l-1}}^{[l]}\right]^{\mathrm{T}} \in \mathbb{R}^{h_l \times (h_{l-1}+1)}$$



- In the end, a **neural network** is a, as other models, a function $f_{\boldsymbol{\theta}}: X \to Y$ of some parameters ($\boldsymbol{\theta}$ = **weights and biases**)
- $f_{\boldsymbol{\theta}}$ is a composition of non-linear function when s is non-linear, allowing to build non-linear estimators
- The parameters of the model are obtained by minimising the empirical risk (**cross-entropy for classification**, **MSE for regression**)

Artificial neural networks



How many parameters does this network have? (don't forget biases!)

+ Both regression and classification, learns features, good performances when enough data

− Need a lot of data to train, subject to overfitting, uninterpretable, targeted-purpose architectures usually work better

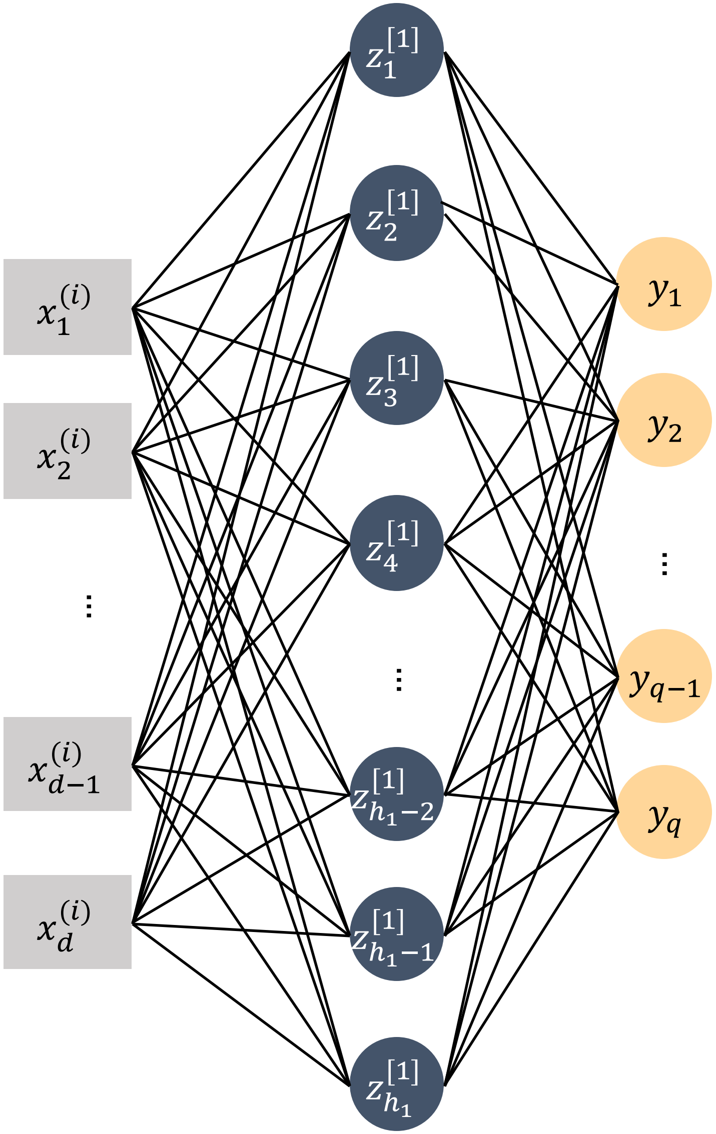
Artificial neural networks

- Single-layer neural networks are universal approximation (see [Cybenko 1989](#))

Theorem (informal)

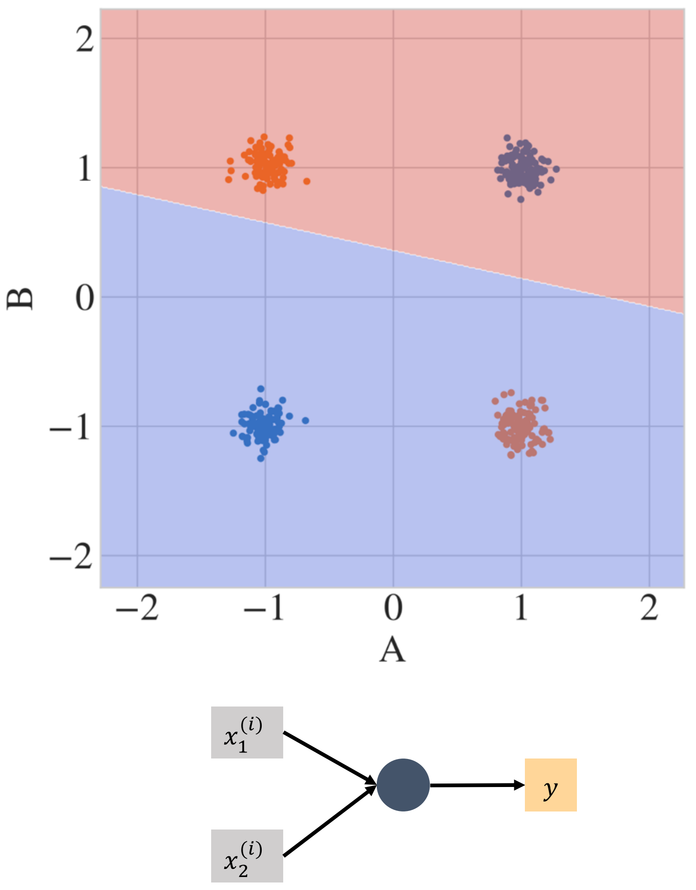
A fully-connected neural network with a single hidden layer $(L = 1)$ with enough neurons (h_1 large) can fit any arbitrary smooth function.

- In practice, this does ***not*** help: the width may need to scale exponentially with the function complexity.
- Empirically, **deep networks are more efficient than wide ones**.

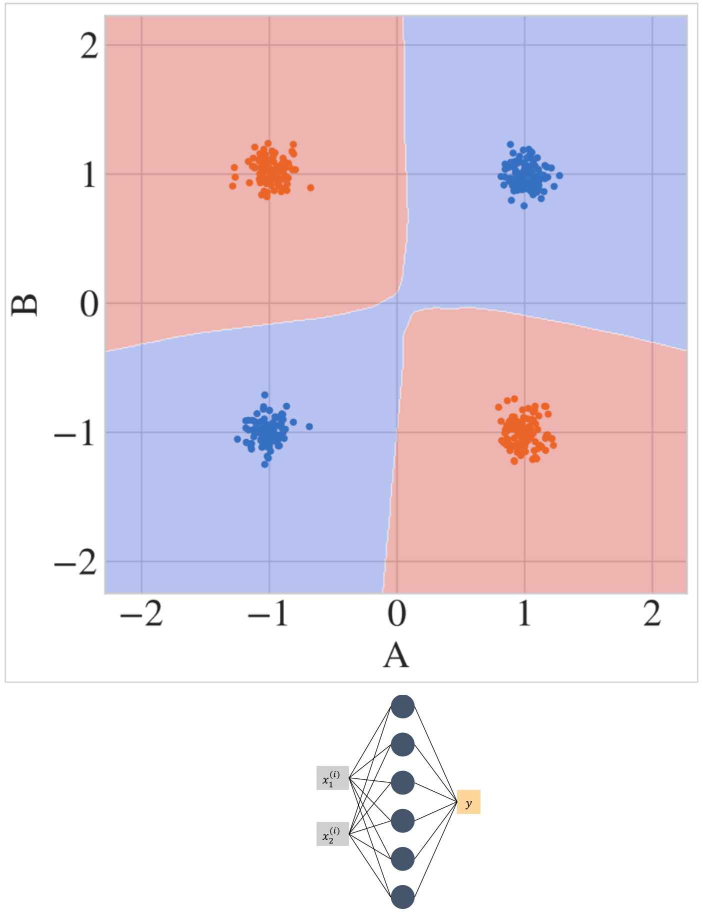


Artificial neural networks

1 hidden layer of 1 unit



1 hidden layer of 100 units



2 hidden layers of 10 units

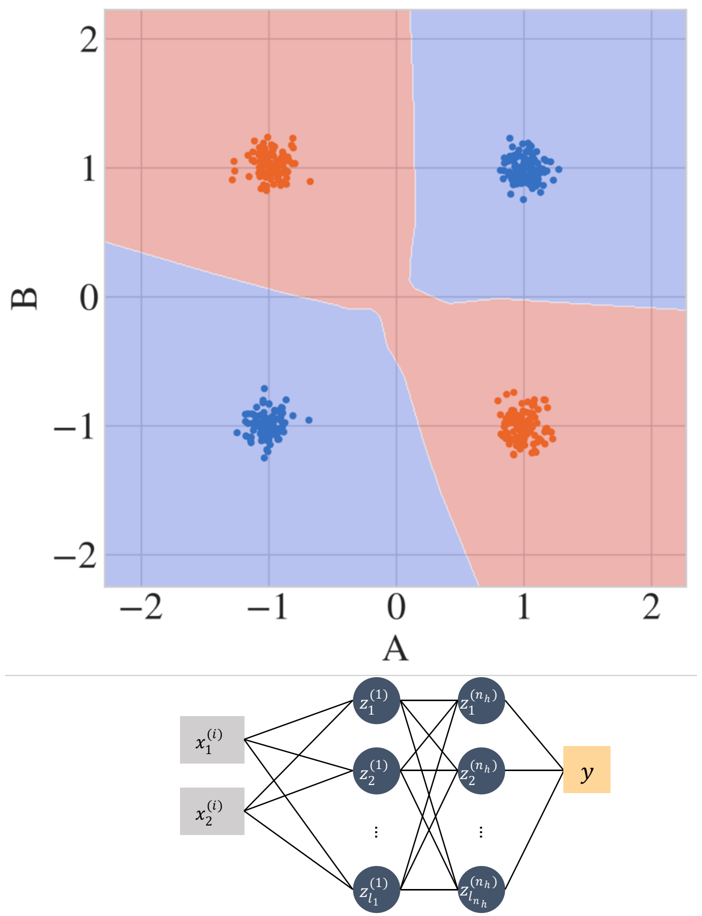
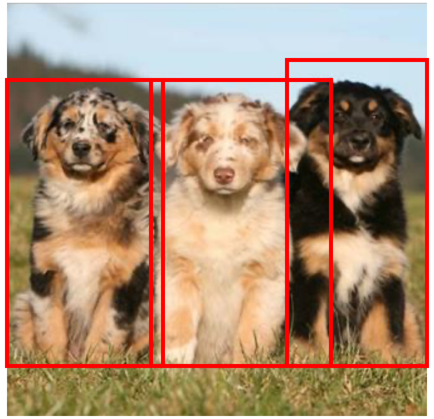


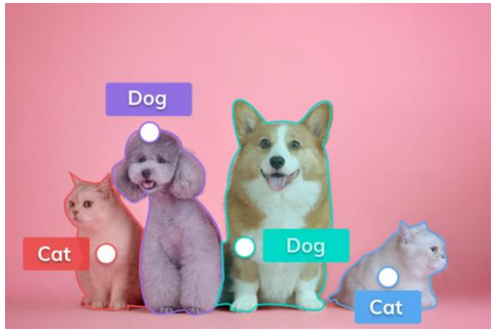
Image classification

Dog?

Object detection



Instance segmentation



Caption generation

"A dog and a little boy playing with a basketball in the grass"

Image classification

Dog?

12M pixels

$$x^{(i)} = \left[x_1^{(i)}, x_2^{(i)}, \dots, x_{12192768}^{(i)}\right]$$

$$d = 12192768$$

Why going beyond fully-connected neural network?

- Consider the simplest task with **classification**
- Standard images taken by a current smartphone are of size $4032 \times 3024 = 12\text{M}$ pixels
- A 2 layer-FCNN with just 100 (!) units has > 1B parameters

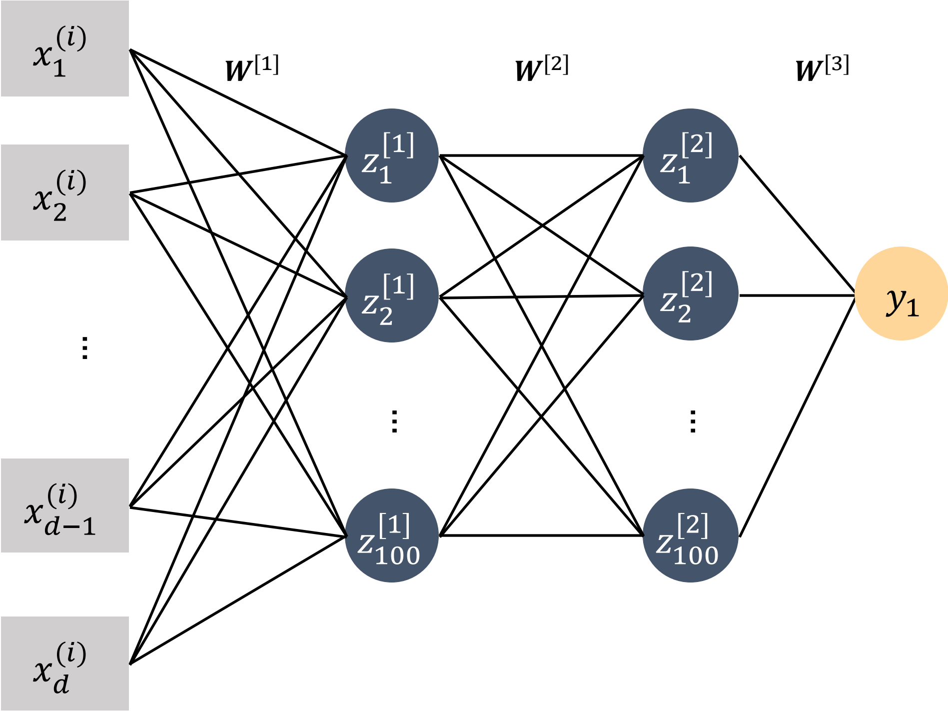
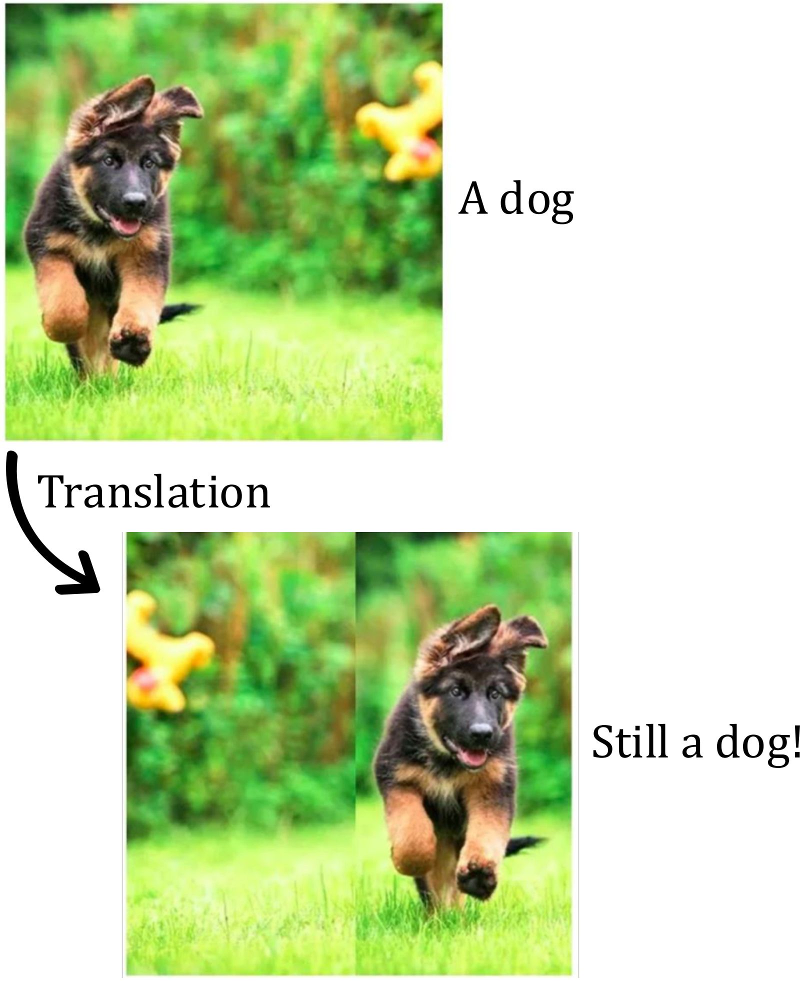


Image classification



Why going beyond fully-connected neural network?

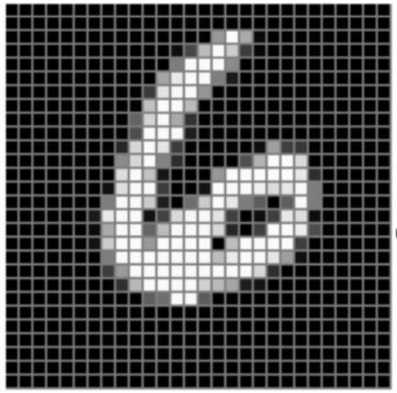
- Consider the simplest task with **classification**
- Standard images taken by a current smartphone are of size $4032 \times 3024 = 12M$ pixels
- A 2 layer-FCNN with just 100 (!) units has > 1B parameters
- FCNN are **unstructured** with no invariance with respect to **translations** or **local distortions**

They are all zeros!

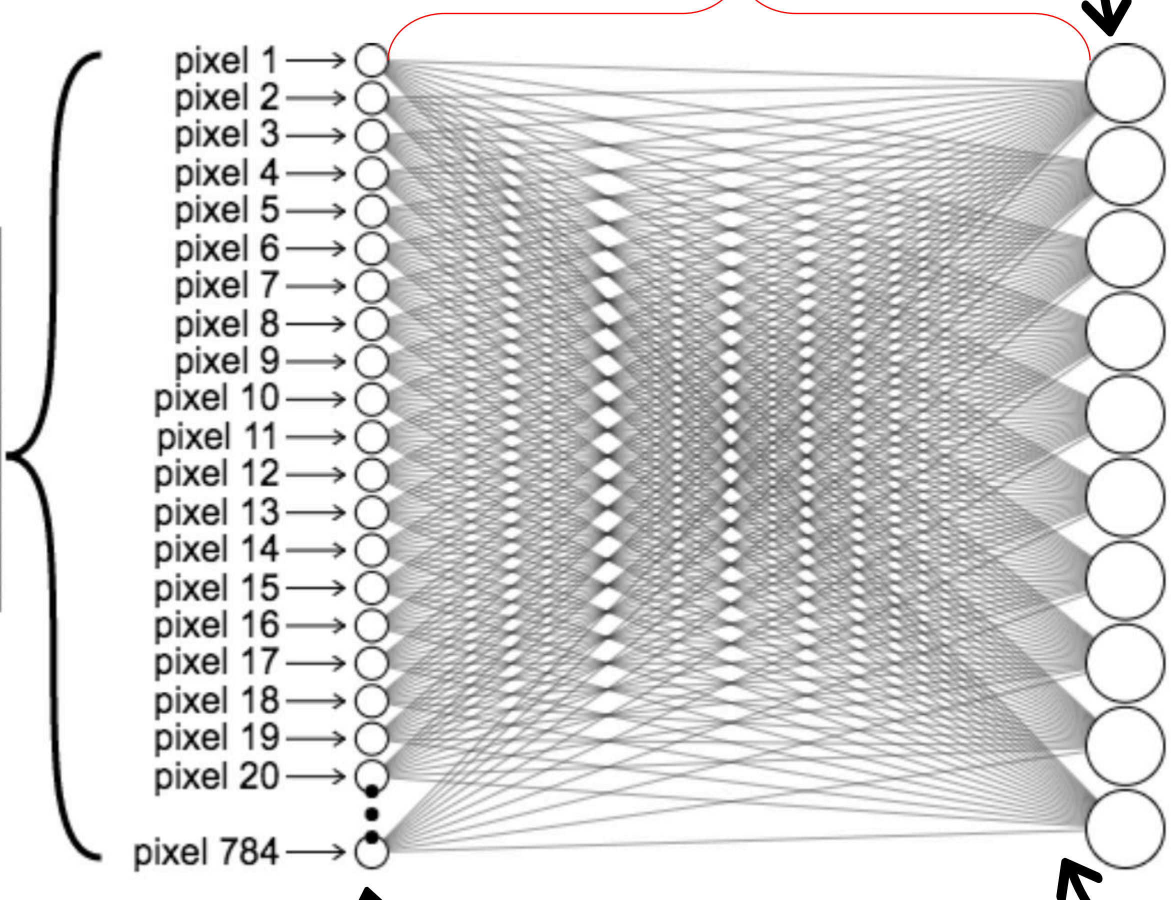
Image classification

One output neuron for each digit (0-9)

Learns an embedding



28x28 pixels



One input for each pixel

Softmax activation

In principle, an FCNN *could* learn that but at the cost of:

1) A huge training set,

2) A lot of redundant weights to encode the same features at different locations.

→ **Can we use this prior knowledge about image data to do better?**

Convolution

1D

$$(x * w)(t) = \sum_{k=-K}^{K} x(k)w(t-k)$$

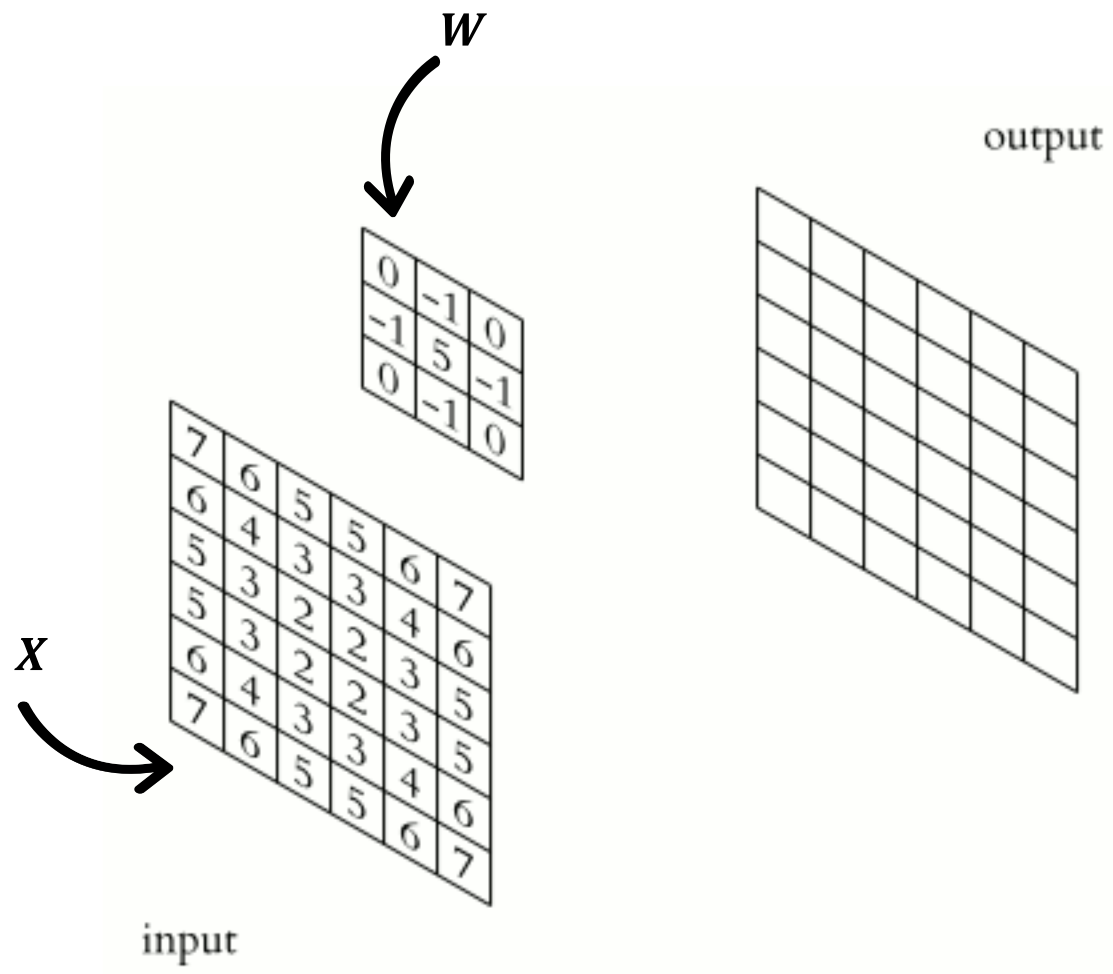
2D

$$(\boldsymbol{X} * \boldsymbol{W})_{ij} = \sum_{l=1}^{L}\sum_{k=1}^{K} x_{lk} w_{i-l,j-k}$$

Convolution operation is a **linear operation equivariant** to translation

$$\sum_{l=1}^{L}\sum_{k=1}^{K} x_{l+c,k+c} w_{i-l,j-k} = \sum_{l=1}^{L}\sum_{k=1}^{K} x_{lk} w_{i-l+c,j-k+c}$$

$$= (\boldsymbol{X} * \boldsymbol{W})_{i+c,j+c}$$



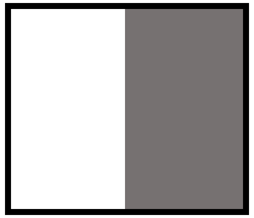
Animation from Wikipédia

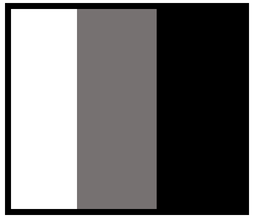
Example: edge detection

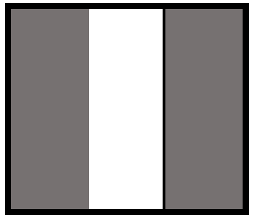
Convolution

Example of a kernel for vertical edge detection

$$
\underbrace{\begin{array}{|c|c|c|c|c|c|}\hline 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 \\ \hline \end{array}}_{\boldsymbol{X}} * \underbrace{\begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline 1 & 0 & -1 \\ \hline \end{array}}_{\boldsymbol{W}} = \underbrace{\begin{array}{|c|c|c|c|}\hline 0 & 3 & 3 & 0 \\ \hline 0 & 3 & 3 & 0 \\ \hline 0 & 3 & 3 & 0 \\ \hline 0 & 3 & 3 & 0 \\ \hline \end{array}}_{\boldsymbol{X} * \boldsymbol{W}}
$$







The border is enhanced!

Convolution

Example of a kernel for vertical edge detection

$$
\underbrace{\begin{array}{|c|c|c|c|c|c|c|c|}\hline 0&0&0&0&0&0&0&0\\\hline 0&1&1&1&0&0&0&0\\\hline 0&1&1&1&0&0&0&0\\\hline 0&1&1&1&0&0&0&0\\\hline 0&1&1&1&0&0&0&0\\\hline 0&1&1&1&0&0&0&0\\\hline 0&1&1&1&0&0&0&0\\\hline 0&0&0&0&0&0&0&0\\\hline\end{array}}_{\boldsymbol{X}}
*
\underbrace{\begin{array}{|c|c|c|}\hline 1&0&-1\\\hline 1&0&-1\\\hline 1&0&-1\\\hline\end{array}}_{\boldsymbol{W}}
=
\underbrace{\begin{array}{|c|c|c|c|c|c|}\hline -2&0&2&2&0&0\\\hline -2&0&3&3&0&0\\\hline -3&0&3&3&0&0\\\hline -3&0&3&3&0&0\\\hline -2&0&3&3&0&0\\\hline -2&0&2&2&0&0\\\hline\end{array}}_{\boldsymbol{X} * \boldsymbol{W}}
$$

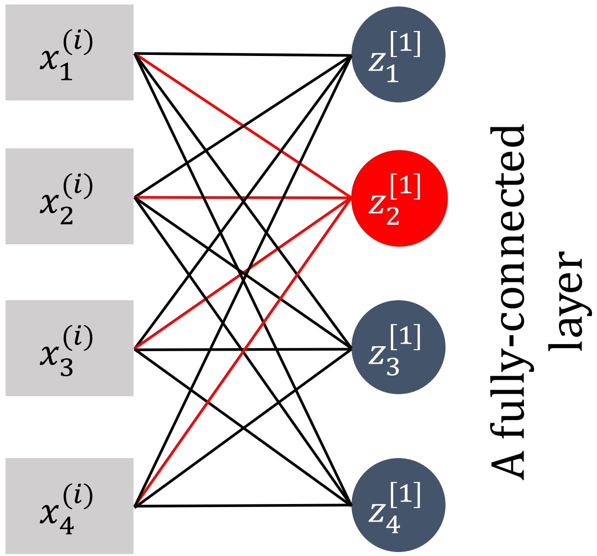
$\boldsymbol{X}$ and $\boldsymbol{X} * \boldsymbol{W}$ have the same size

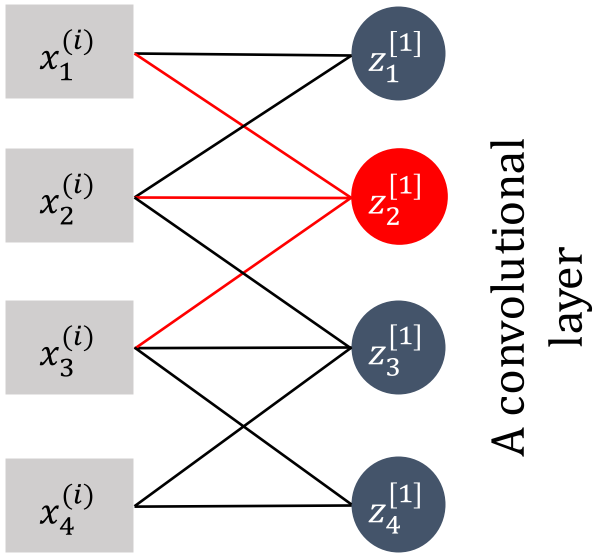
In practice, we use **padding** to add zeros around the image $\boldsymbol{X}$ so that the border of the image are seen as much as other pixels.

Convolutional layers

- **A convolutional layer** in a neural network is simply implementing the convolution operation with a filter (or kernel) $\boldsymbol{W}$ **shared across all the inputs**
- In 2 dimensions with a first layer of the same size as the input, a square weight matrix has $K \times K$ elements, while a fully connected layer has $d \times d$
- Typically, $K \ll d, K \sim O(1)$, usually 3 or 5.
- Units of a given layer ℓ only sees a subset of the activations of the previous layer $\ell - 1$: **local receptive field**
- **Deeper units** in the network **are influenced by more inputs**, hence learning higher-order features

The **convolutional layer** has three properties: **sparse interactions**, **parameters sharing** and **equivariant representation**.





Pooling and local invariance translation

Pooling layers

- To make the network **invariant to slight local translations**, we need an additional element
- Idea: local invariance to translations of features in a given window
- In 2D

$$z_{ij}^{[\ell+1]} = \max_{(r,s)\in K_1\times K_2} z_{rs}^{[\ell]} \qquad\qquad z_{ij}^{[\ell+1]} = \frac{1}{K_1 \times K_2}\sum_{r}^{K_1}\sum_{s}^{K_2} z_{rs}^{[\ell]}$$

1	0	1	3
4	2	1	5
0	3	4	1
2	8	7	6

$\longrightarrow$

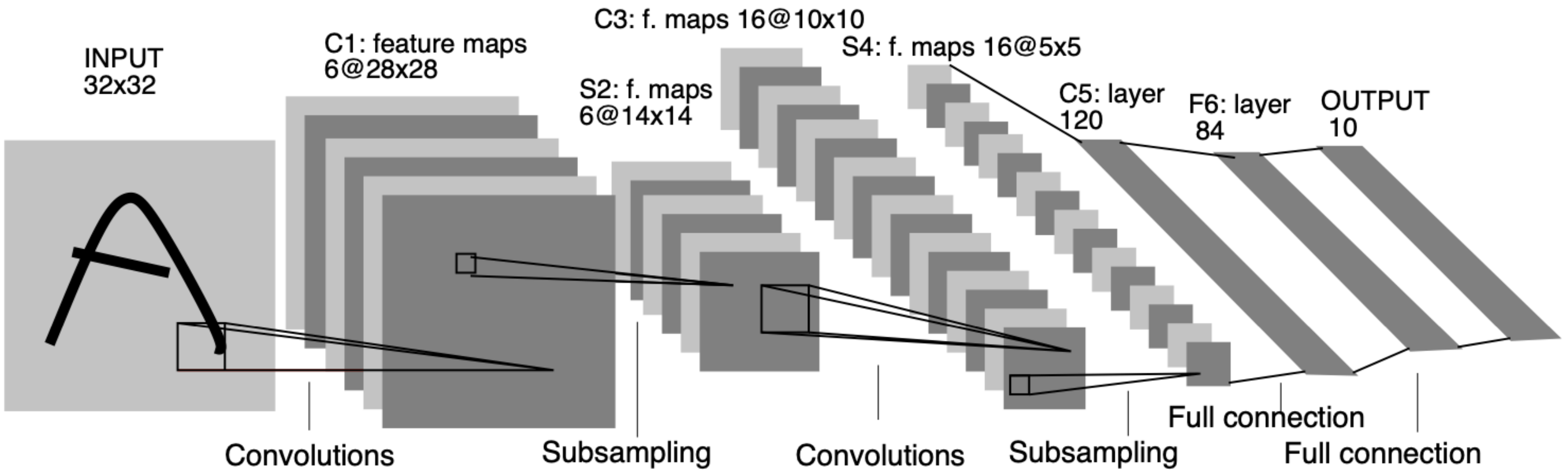
4	5
8	7

A max non-overlapping pooling layer with window of size 2x2

The **pooling layer** grants the network **some local invariance to translation** and **reduces the image size**

CNNs

A convolutional neural network (CNN) is a cascade of **convolutional layers** and **pooling layers** to ensure **invariance to small (local) shifting, scaling and distortions** through **local receptive field, shared weights and spatial sub-sampling**.



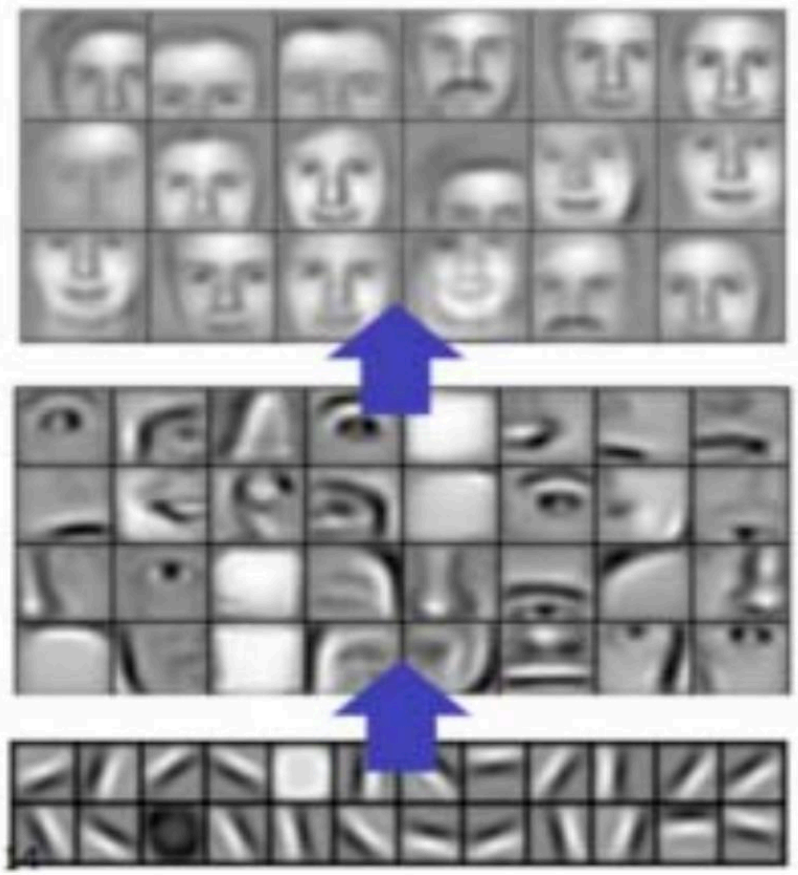
Architecture of LeNet-5 used for digit recognition in LeCun+98. It has 60,000 trainable parameters.

To know more about CNNs, see chapter 9 of Deep Learning from Goodfellow et al., 2013 or Dumoulin and Visin, 2018

Features learnt by a CNN

More specific, large-scale, high-order, features

Local, first-order, features (edges)



Layer 3

Layer 2

Layer 1

Depth

Image from Albawi et al., 2017