Deep Learning for Medical Imaging

Olivier Colliot, PhD
Research Director at CNRS
ARAMIS Lab – www.aramislab.fr
PRAIRIE – Paris Artificial Intelligence Research Institute

Maria Vakalopoulou, PhD
Assistant Professor at CentraleSupelec
Center for Visual Computing

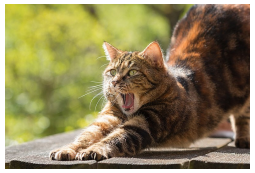
Part 2 – Classification (and regression)

Part 2 – Classification (and regression)
2.1 Machine learning refresher

Introduction

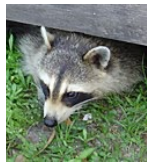
Training set: labeled examples

Cats



Racoons



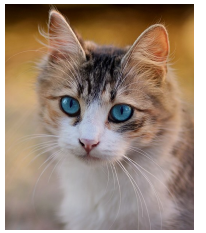


Trees



$$(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$$

Learning



Input x

$f \quad y = f(x)$

Cat **Output** y

Outputs y are classes: **classification** problem

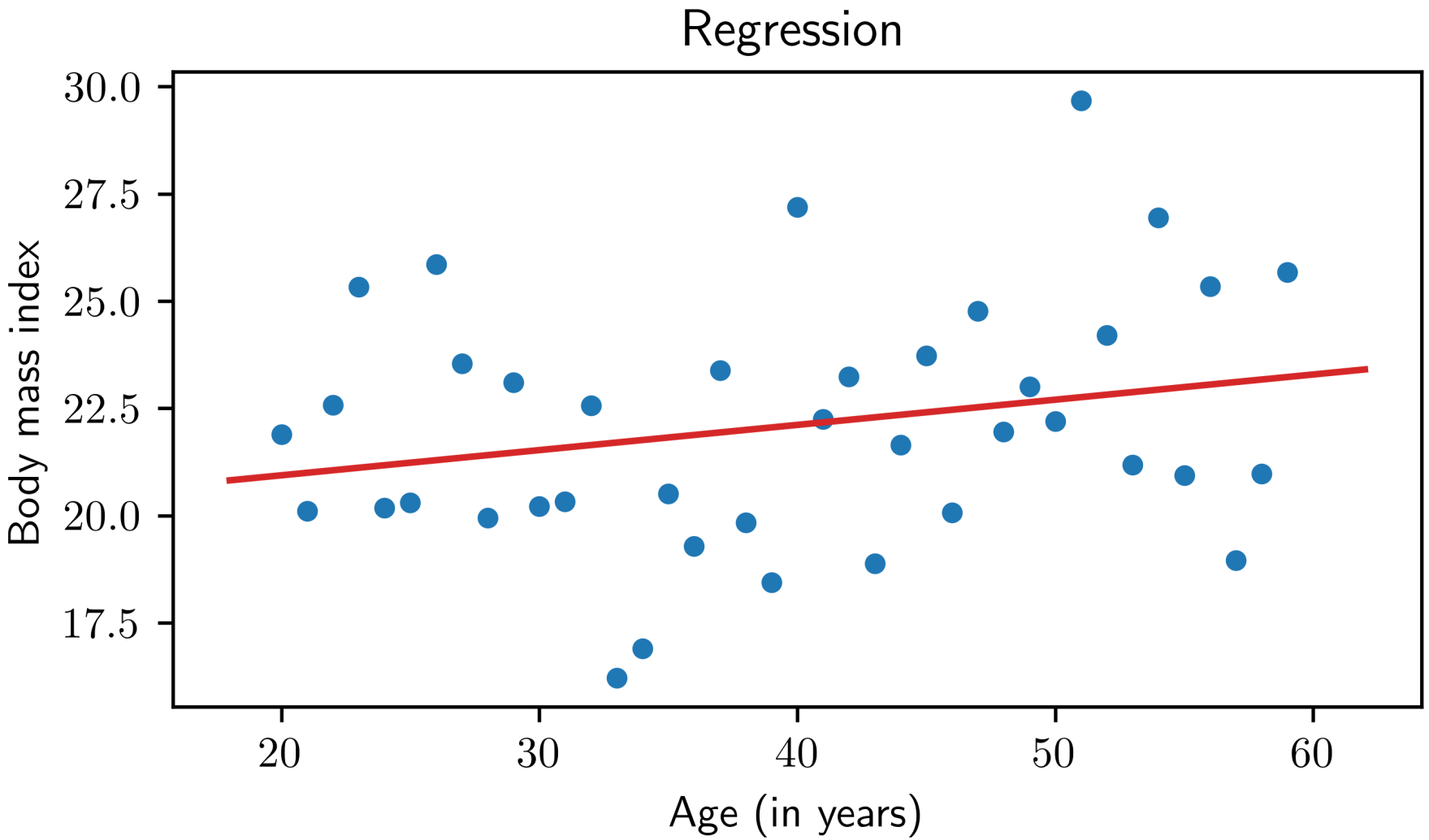
$$\hat{f} = \arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell(y^{(i)}, f(x^{(i)}))$$

Loss function: measures error in prediction

Images come from https://commons.wikimedia.org/ and have no usage restriction.

Refresher slides – will not be presented during the course

Introduction



Outputs y are numbers: **regression** problem

Notations

Input variable (multivariate): $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$

Output: y

Model: f, $y = f(x)$ The "artificial intelligence"

Model parameters: $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_p \end{bmatrix}$, indexed by j

Model, depending on parameters: $f(\boldsymbol{x}; \boldsymbol{w})$

Notations

Input variable (multivariate): $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$

Input features

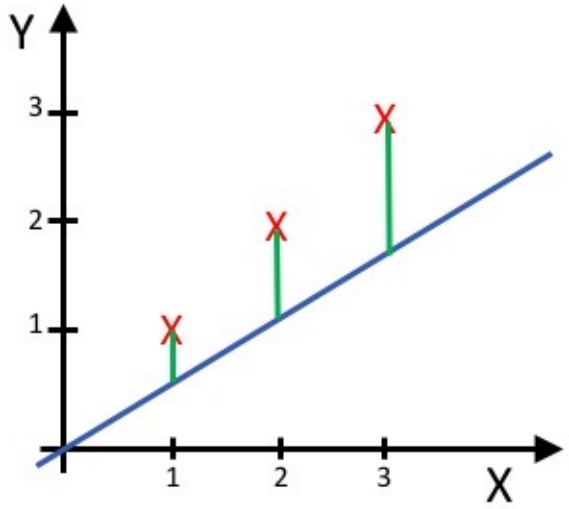
Output: y

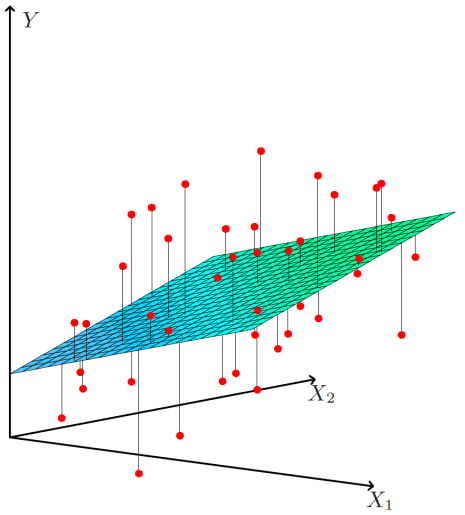
Model: f, $y = f(x)$

The "artificial intelligence"

Loss: $\ell(y, x)$

Quantifies how much the prediction is far from the true output





Notations

Input variable (multivariate): $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_p \end{bmatrix}$

Input features

Output: y

Model: $f,\ y = f(x)$

The "artificial intelligence"

Loss: $\ell(y, x)$

Quantifies how much the prediction is far from the true output

Cost function:

$$J(f) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, f(x^{(i)})\right)$$

How far are we from the true output across all training examples ?

Learning:

$$\hat{f} = arg \min_{f \in \mathcal{F}} J(f)$$

Learning: find the model with the minimal error

Optimization algorithm: method to find the minimum

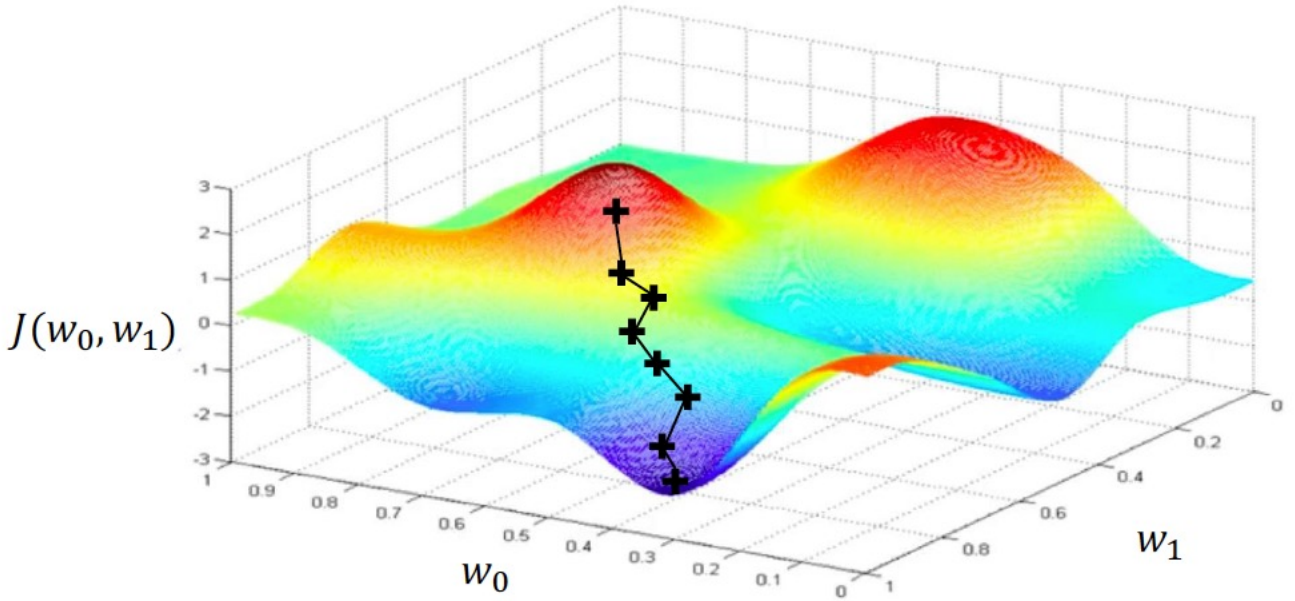
Notations

Learning:

$$\hat{f} = arg\min_{f \in \mathcal{F}} J(f)$$

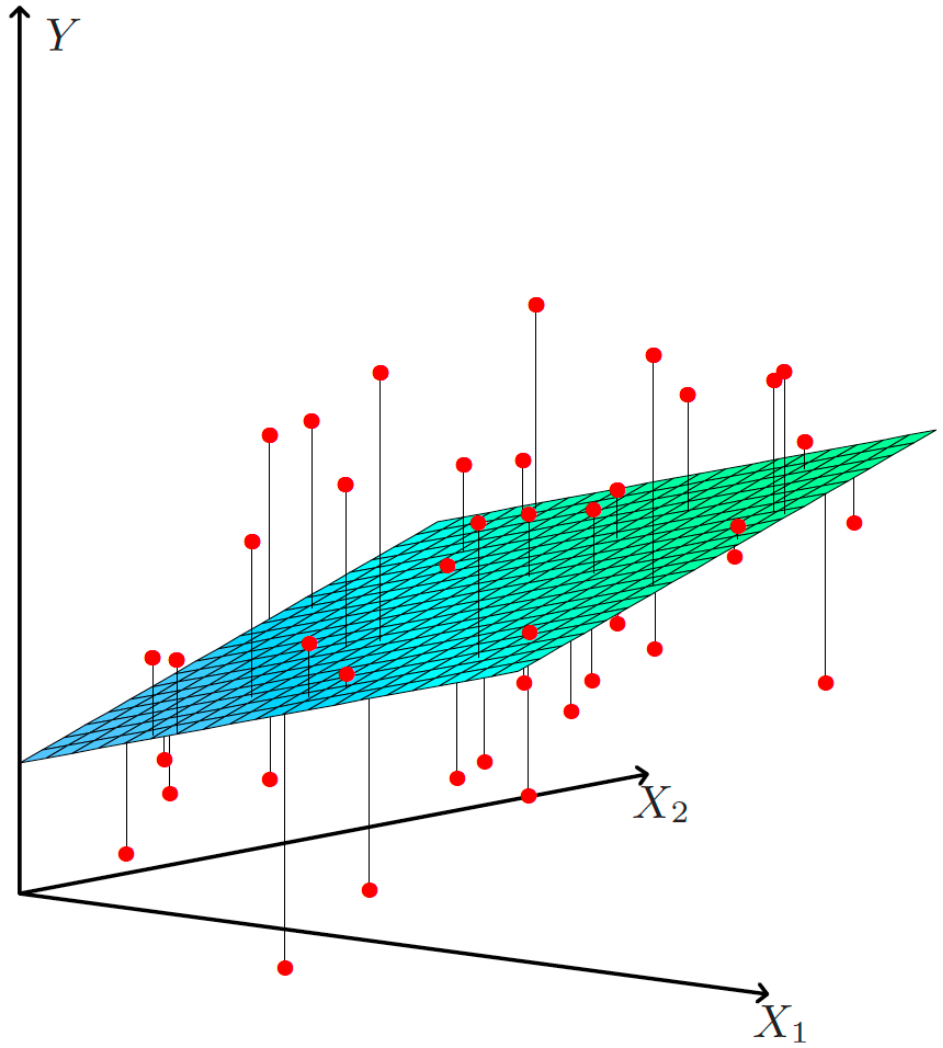
Learning: find the model with the minimal error

Optimization algorithm: method to find the minimum



Source: http://introtodeeplearning.com/

Linear regression



Model: $y = \sum_{j=0}^{p} w_j x_j$

Loss: $\ell(y, f(x)) = (y - f(x))^2$

Reduced notation: $y = \boldsymbol{w}^T \mathbf{x}$

Cost function:

$$J(f) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, f(x^{(i)})\right)$$

Image source: Hastie et al, Elements of statistical learning

Classification

The output y is a class.

Suppose we have K classes, then we could write $y \in \{C_1, \ldots, C_K\}$.

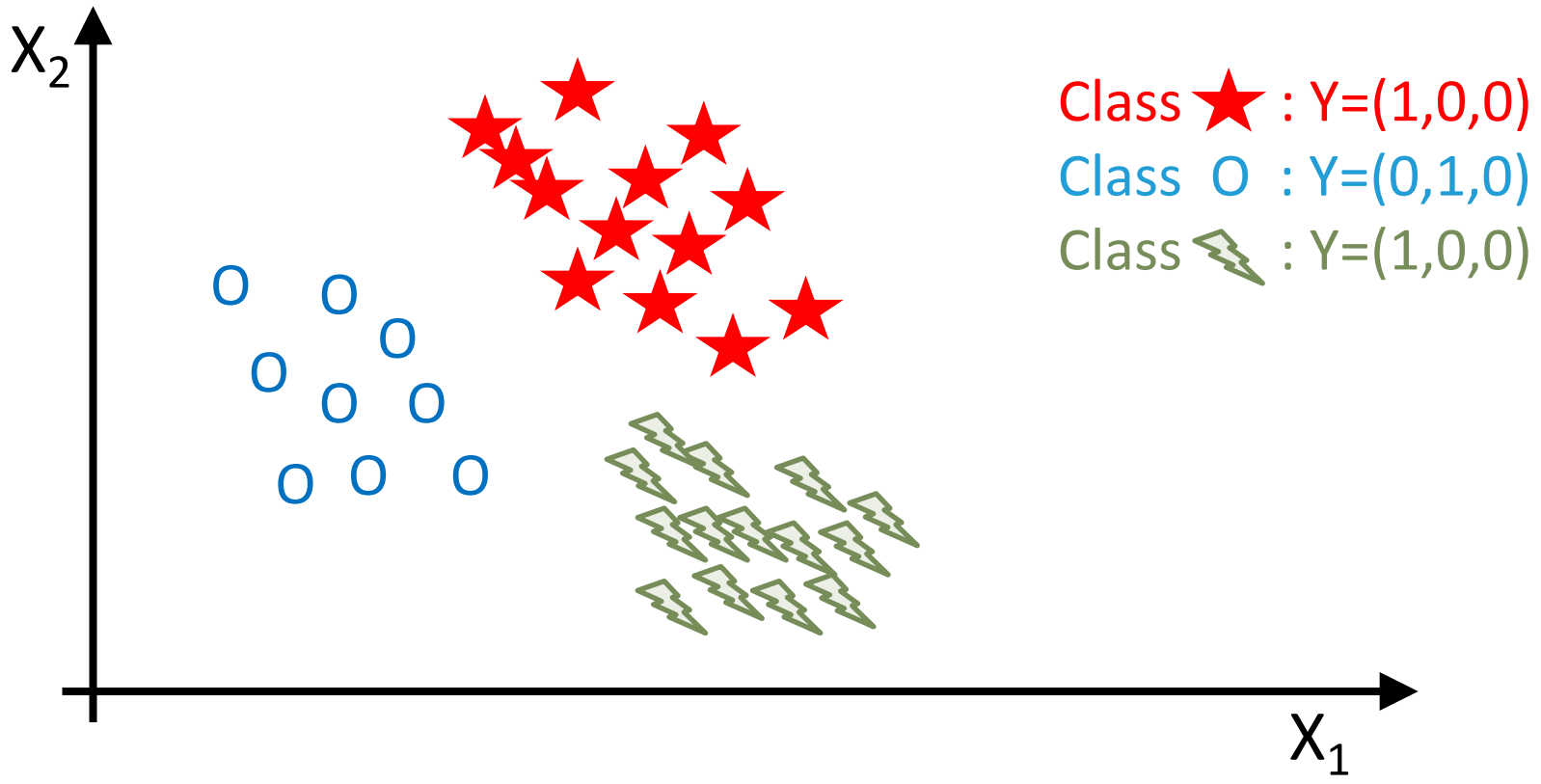
We need to encode this numerically.

Coding $y \in \{1, \ldots, K\}$ would not be meaningful because there is no order between the different classes.

Therefore, we will use indicator variables. The indicator y_k for class C_k is such that $y_k = 1$ if Y belongs to class C_k and 0 otherwise.

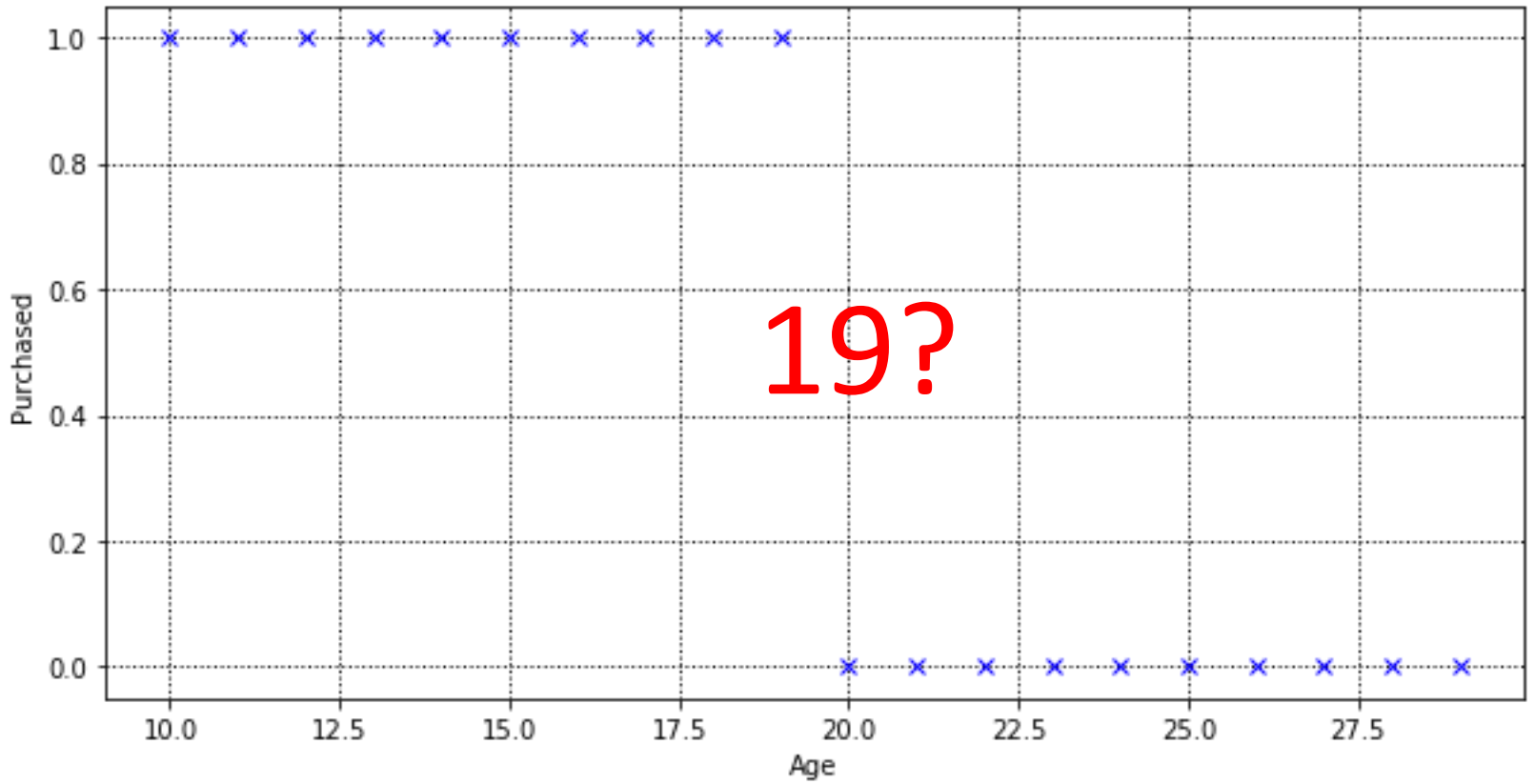
Thus, the ouput is the vector $\mathbf{y} = (y_1, \ldots, y_K)$.

Example: three classes



Classification

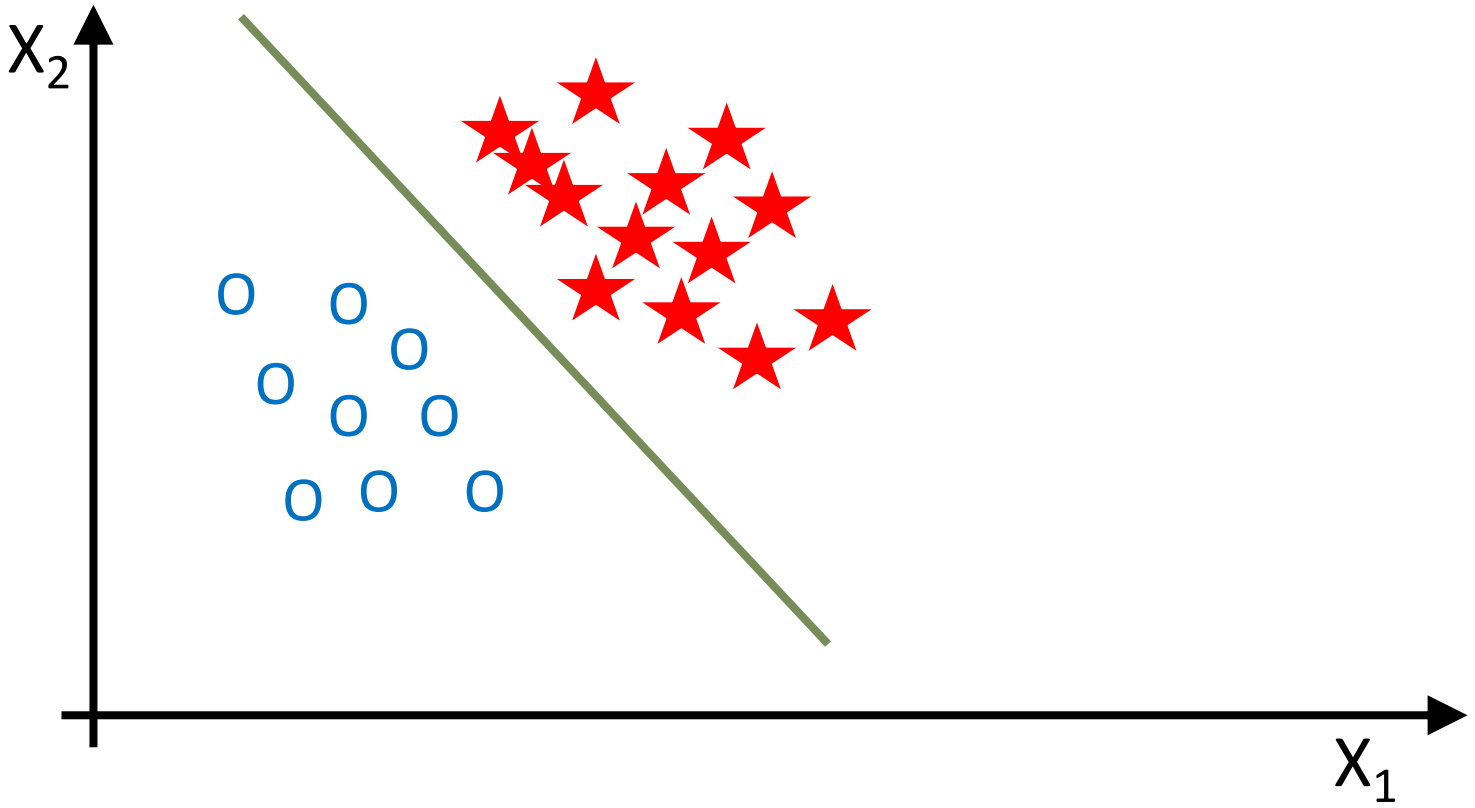
What we want is a decision boundary



With 1 variable, it is a number

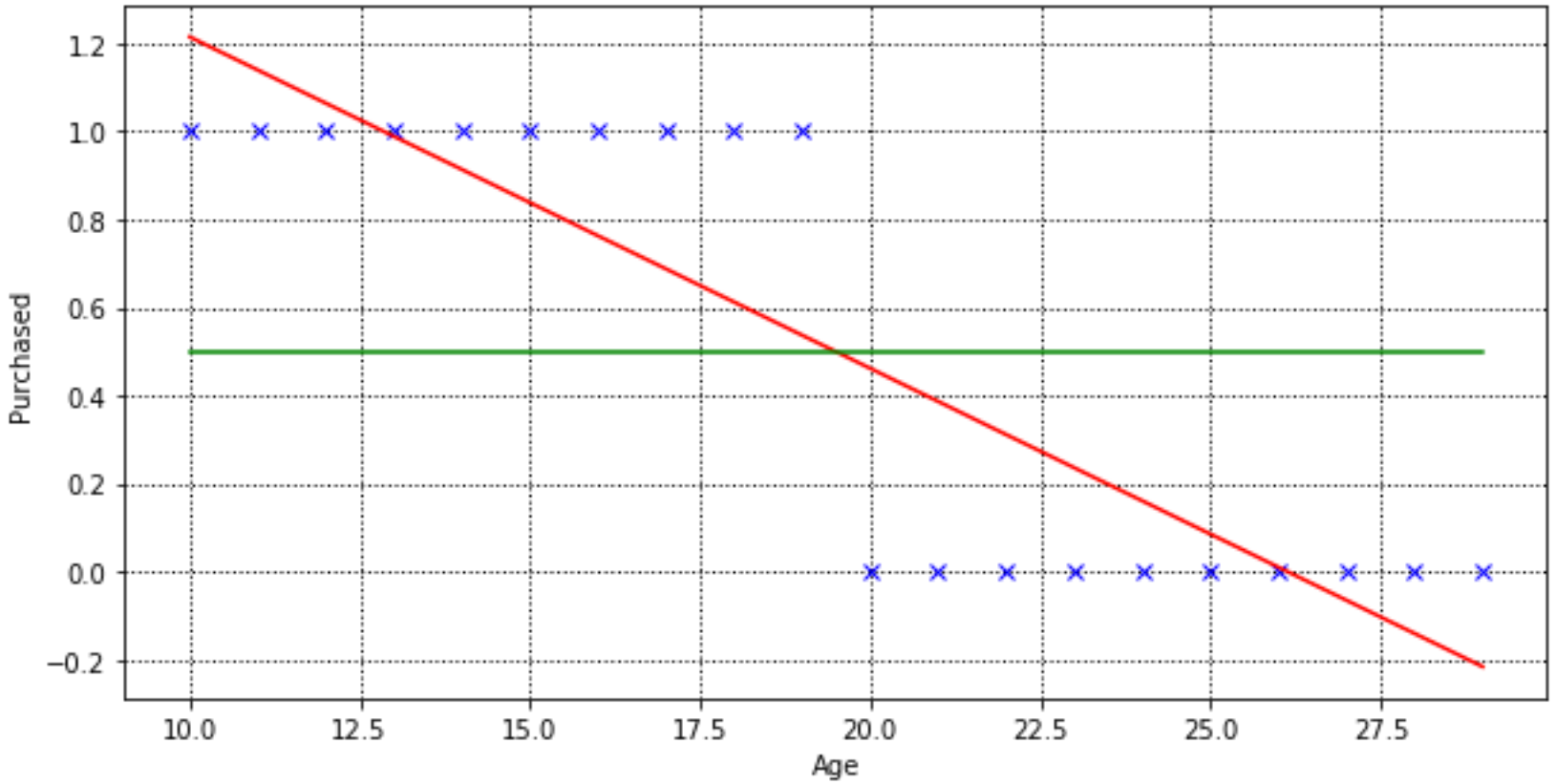
Decision boundary

What we want is a decision boundary



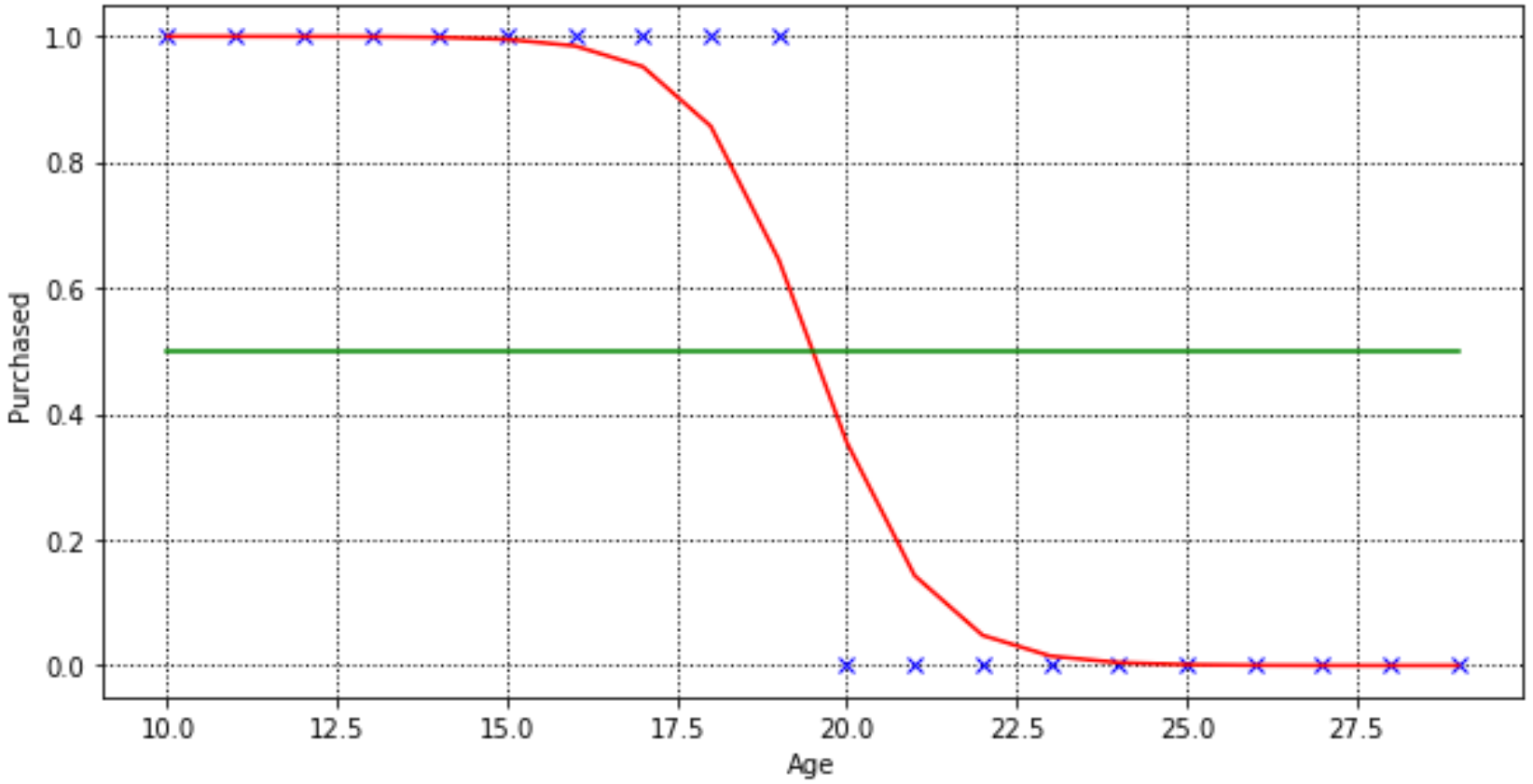
With 2 variables and 2 classes, it is a line if the separation is linear

Coming back to regression



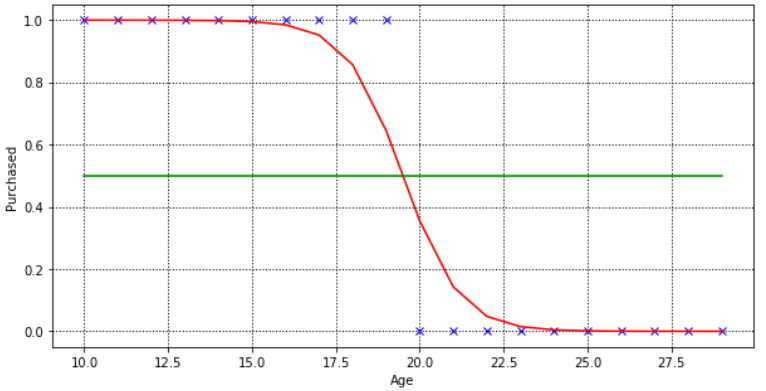
Problem: Y should be binary and not continuous

Coming back to regression



Logistic regression

Activation function



Logistic regression

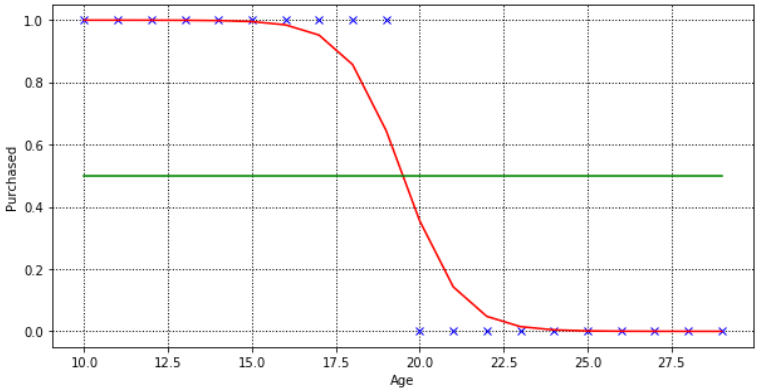
Activation function

Remember the model of linear regression:
$y = f(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$

Now we want $0 \leq f(\boldsymbol{x}) \leq 1$

Define $g(z) = \frac{1}{1+e^{-z}}$

This is called a logistic function.

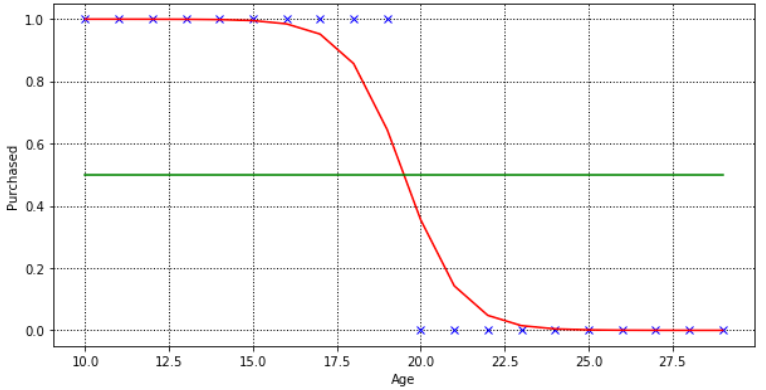


Note: il belongs to the more general class of sigmoid functions (functions which have an S shape).
For example: $\tanh(z)$, $\arctan(z)$
These will be called activation functions in neural networks.

$f(\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{w}^T \boldsymbol{x}}}$

Logistic regression

Probabilistic interpretation

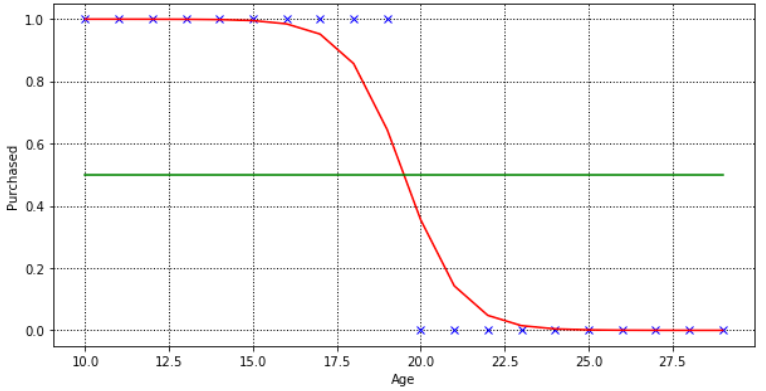


Logistic regression

Probabilistic interpretation

f(x) is the probability that Y=1 given the input x

In our example: probability that the client has purchased given his age



$$f(\boldsymbol{x}) = P(Y = 1 \mid \boldsymbol{x}, \boldsymbol{w})$$

For instance, $f(\boldsymbol{x}) = P(purchased \mid age, model)$

$P(Y = 0 \mid \boldsymbol{x}, \boldsymbol{w}) = 1 - f(X)$

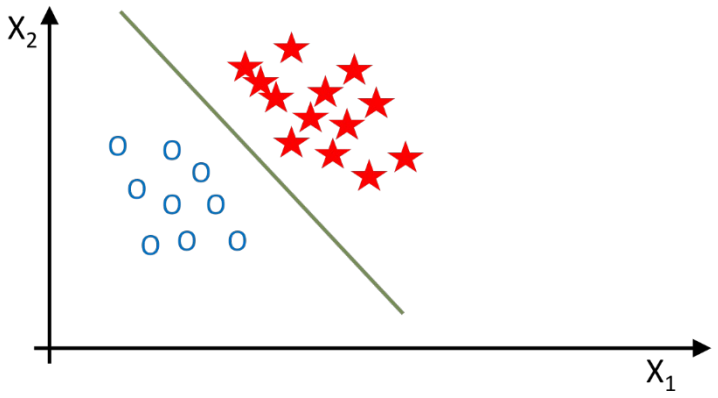
Logistic regression

Discriminant function

Predict Y=1 if $f(\boldsymbol{x}) \geq 0.5 \iff \boldsymbol{w}^T\boldsymbol{x} \geq 0$

Line

This is a **linear classification** technique even though g is a non-linear function



Logistic regression

Cost function

Cost function:

$$J(f) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, f(x^{(i)})\right)$$

Loss $\ell(y, f(x))$

Measures the difference between the predicted output and the true output

Learning:

$$\hat{f} = arg \min_{f \in \mathcal{F}} J(f)$$

Logistic regression

Loss

$$\ell(y, f(x))$$

Low if prediction is correct

High if prediction is wrong

Logistic regression

Loss

$$\begin{cases} \text{if } y = 1,\ \ell(y, f(x)) \text{ should be low when } f(x) \text{ is high} \\ \text{if } y = 0,\ \ell(y, f(x)) \text{ should be low when } f(x) \text{ is low} \end{cases}$$

$$\begin{cases} \text{if } y = 1,\ \ell(y, f(x)) = -\log(f(x)) \\ \text{if } y = 0,\ \ell(y, f(x)) = -\log(1 - f(x)) \end{cases}$$

This can be summarized as

$$\ell(y, f(x) = -y \log(f(x) - (1 - y) log(1 - f(x))$$

Logistic regression

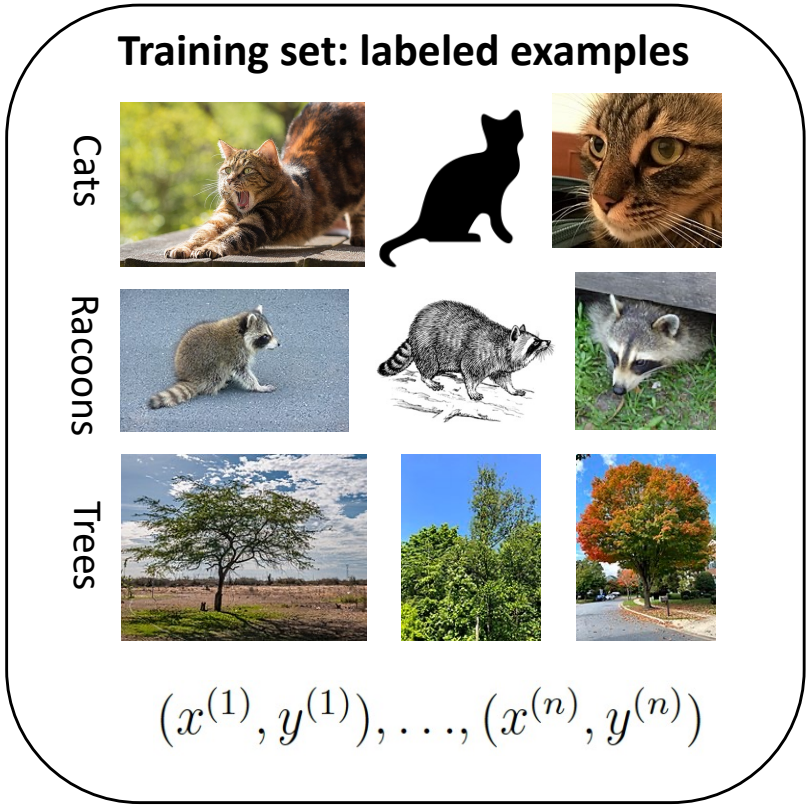
Cost function

$$\ell(y, f(x) = -y \log(f(x) - (1-y) log(1 - f(x))$$

$$J(f) = -\frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} \log(f(x^{(i)}) + (1 - y^{(i)}) log(1 - f(x^{(i)})) \right)$$

Part 2 – Classification (and regression)
2.2 Introduction to deep learning

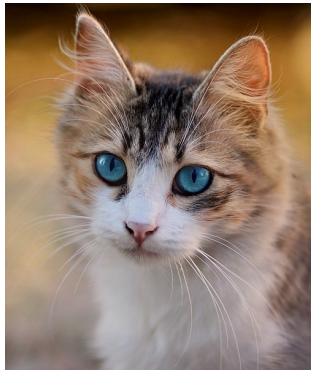
Introduction



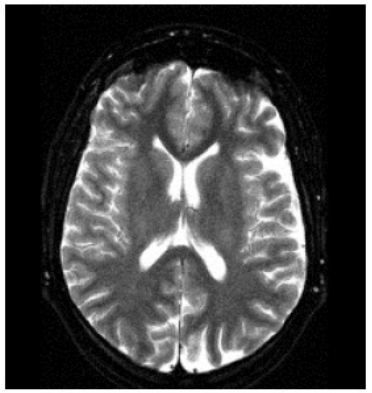
$$\hat{f} = arg\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(y^{(i)}, f(x^{(i)})\right)$$

Introduction

- There are many cases where the inputs are not numbers



Natural images



Magnetic resonance images

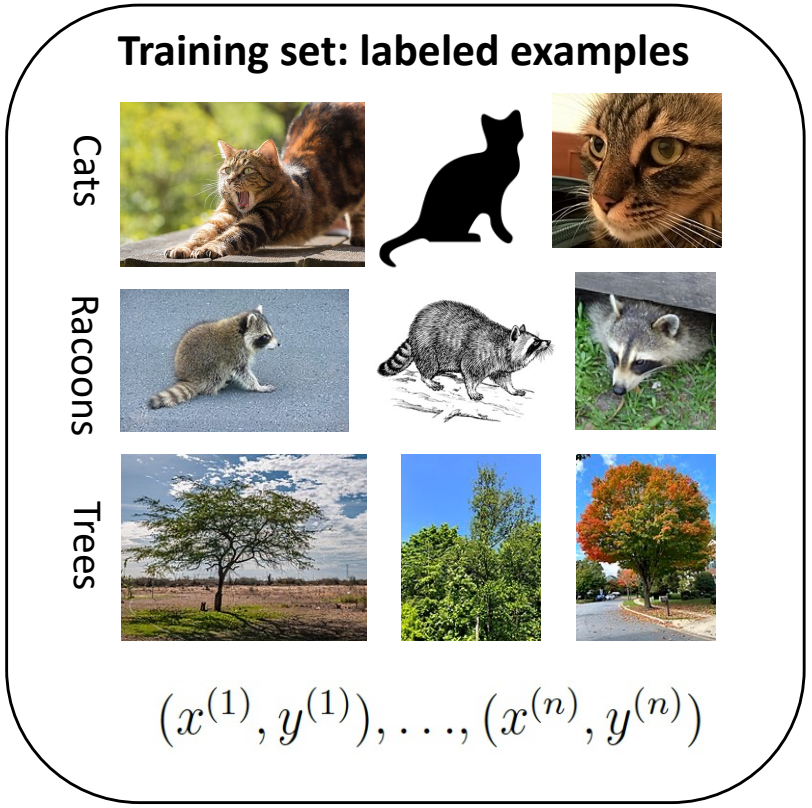
1. 'To be, or not to be: that is the question'

(Hamlet Act 3, Scene 1)

Text

DNA sequence

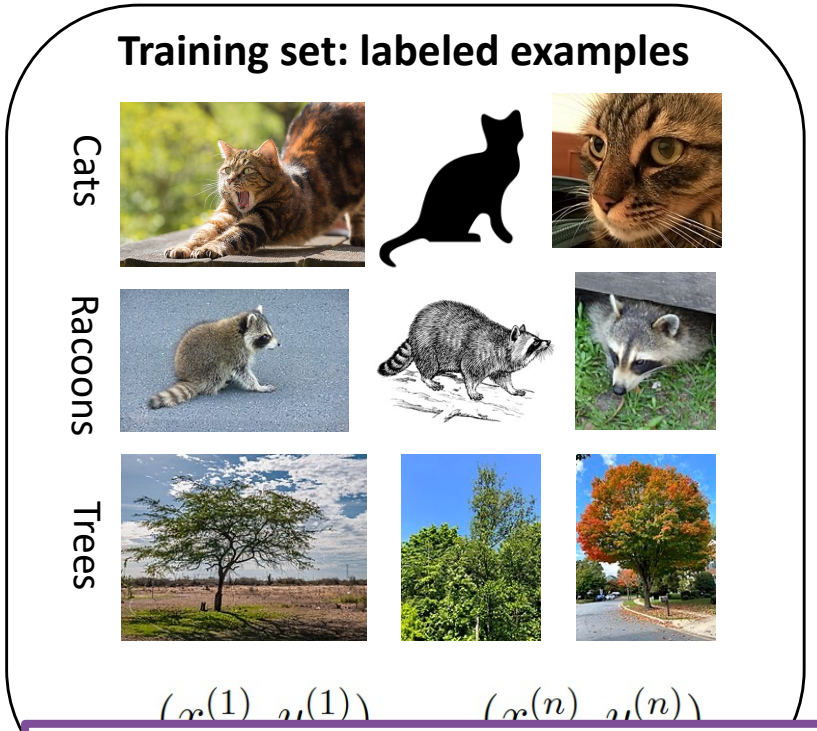
Learning features



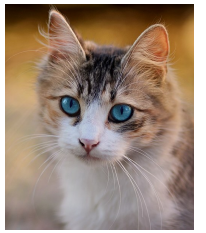
ϕ is the feature extractor

$$\hat{f} = arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \ell\left(y_i, f\left(\phi(x_i)\right)\right)$$

Learning features



Learning



Input x

$f \quad y = f(x)$

Cat

Output y

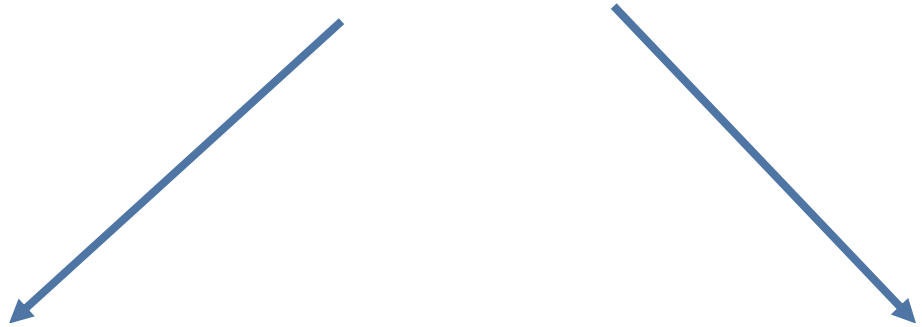
ϕ is the feature extractor

ϕ maps the set of inputs to the set of features $\mathbb{R}^p$

$$\hat{f} = arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} \ell\left(y_i, f\left(\phi(x_i)\right)\right)$$

Learning features

How to define ϕ?



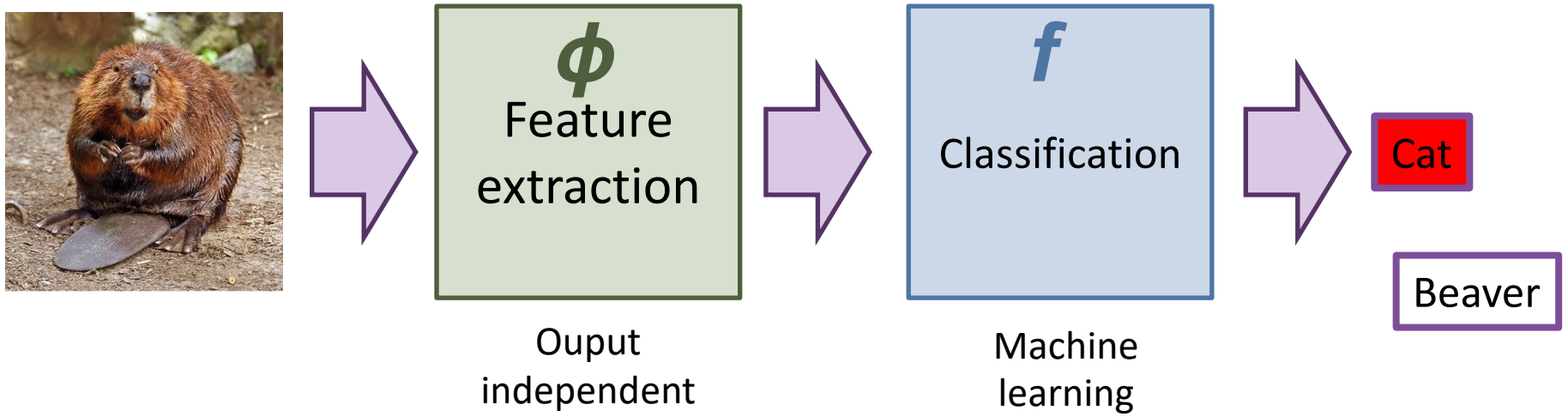
Define it "manually"

Feature engineering
"Hand-crafted" features

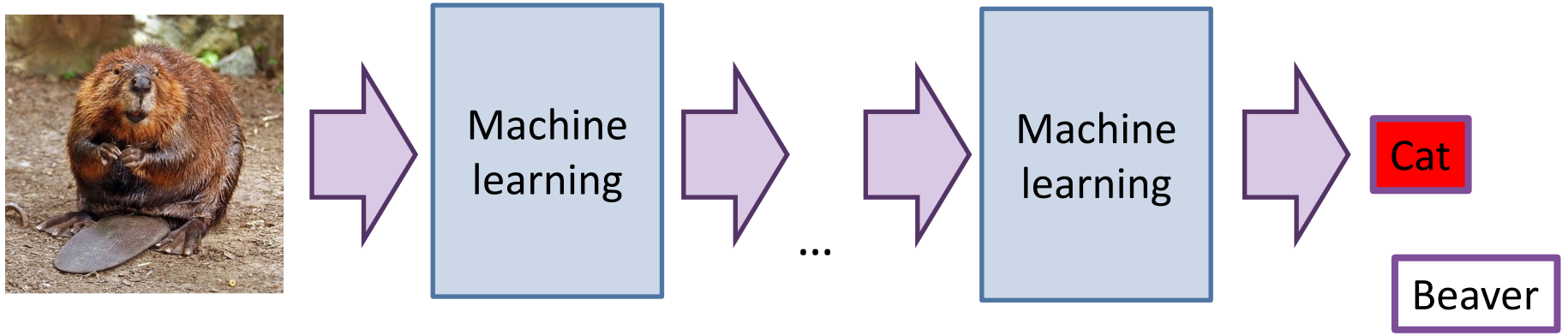
Learn it

This is the approach used in deep learning but not only

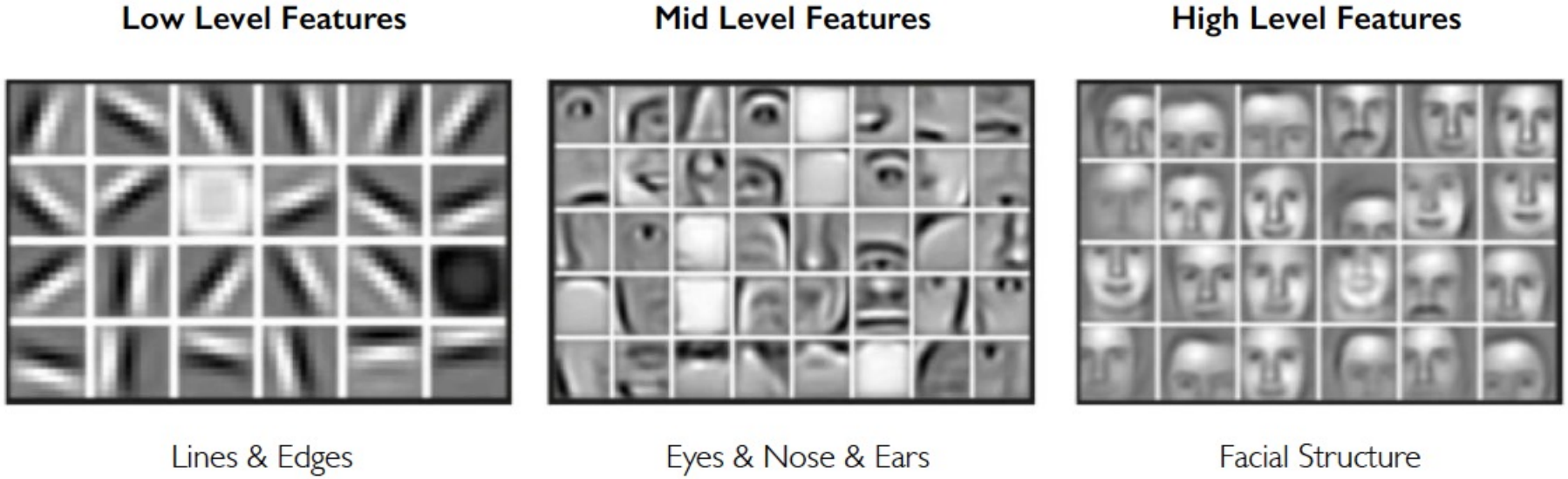
Learning features



Deep learning



Learning features



Source: http://introtodeeplearning.com/

What is deep learning?

- Good old Neural Networks, with more layer/modules
- Automatic extraction of features
- Non-linear, hierarchical, abstract representations of data
- Flexible models with any input/output type and size

Source: https://m2dsupsdlclass.github.io/lectures-labs/slides/01_intro_to_deep_learning/index.html#7

Why now?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

Neural Networks date back decades, so why the resurgence?

1. Big Data

- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware

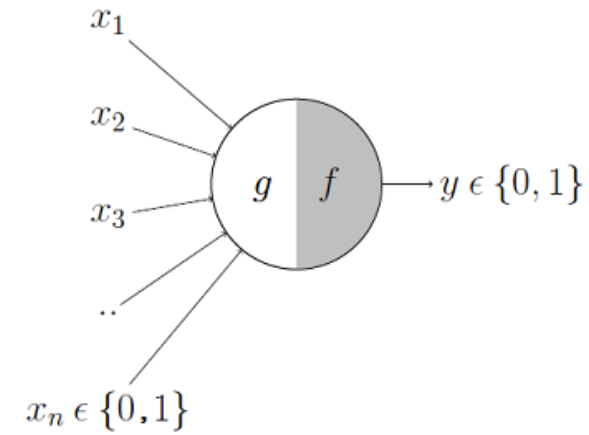
- Graphics Processing Units (GPUs)
- Massively Parallelizable

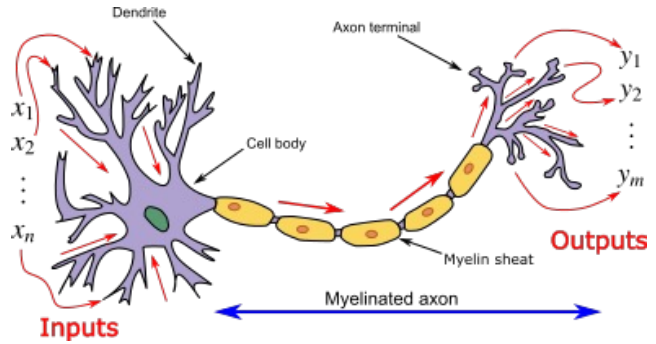
Source: http://introtodeeplearning.com/

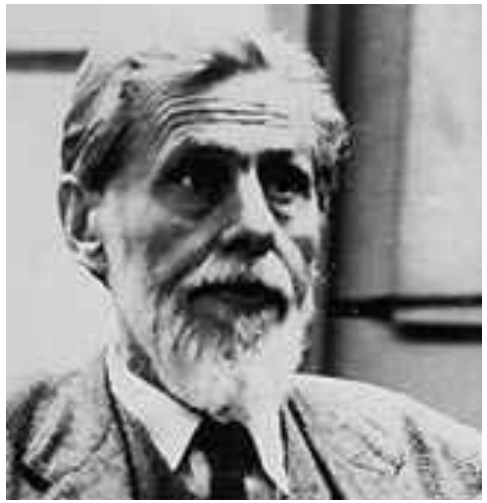
Part 2 – Classification (and regression)
2.2 Introduction to deep learning
2.2.1 Perceptron – a single layer neural network

Perceptron

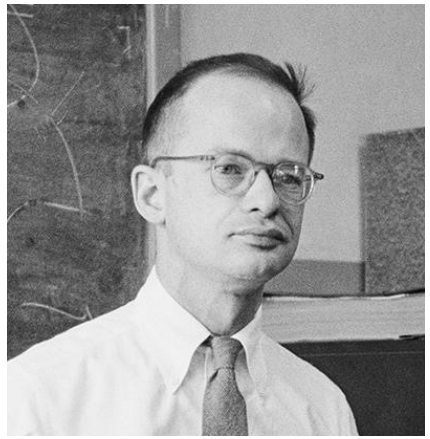
- **McCulloch-Pitt neuron**
  - Artificial neuron model







Warren McCulloch

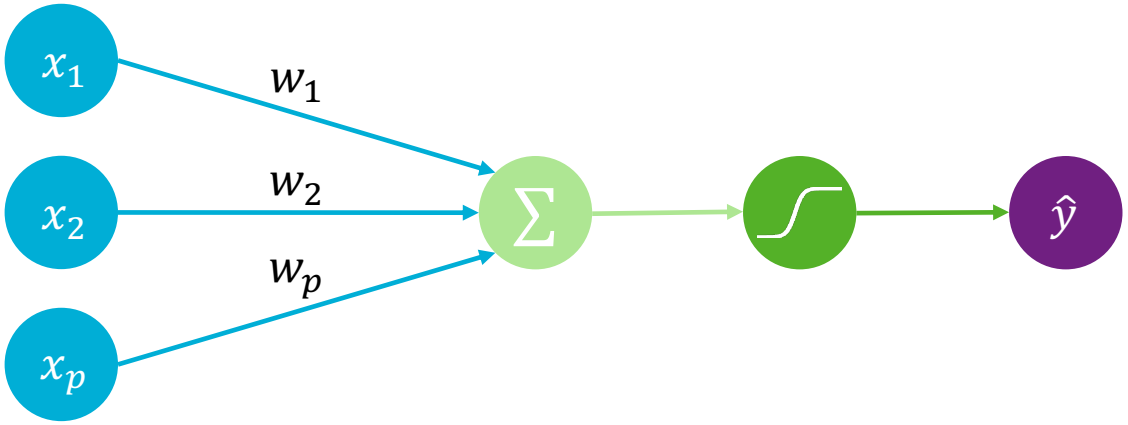


Walter Pitts

Source: https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1

ARAMIS LAB
BRAIN DATA SCIENCE

Perceptron (1958)



Inputs Weights Sum Non-Linearity Output

Frank Rosenblatt

Output

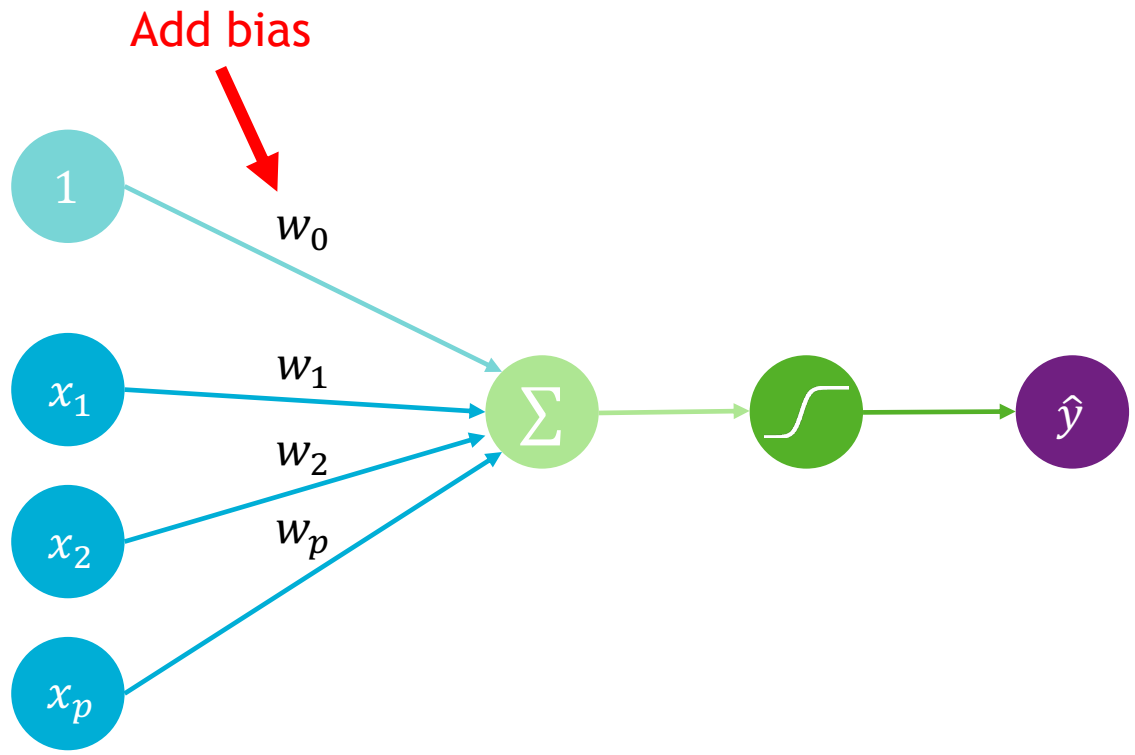
Linear combination of inputs

$$\hat{y} = g\left(\sum_{j=1}^{p} x_j w_j\right)$$

Non-linear activation function

Here this is a perceptron with a single output, so this is simply an artificial neuron

Perceptron (1958)



Output $\hat{y}$, Bias w_0, Linear combination of inputs $\sum_{j=1}^{p} x_j w_j$, Non-linear activation function g

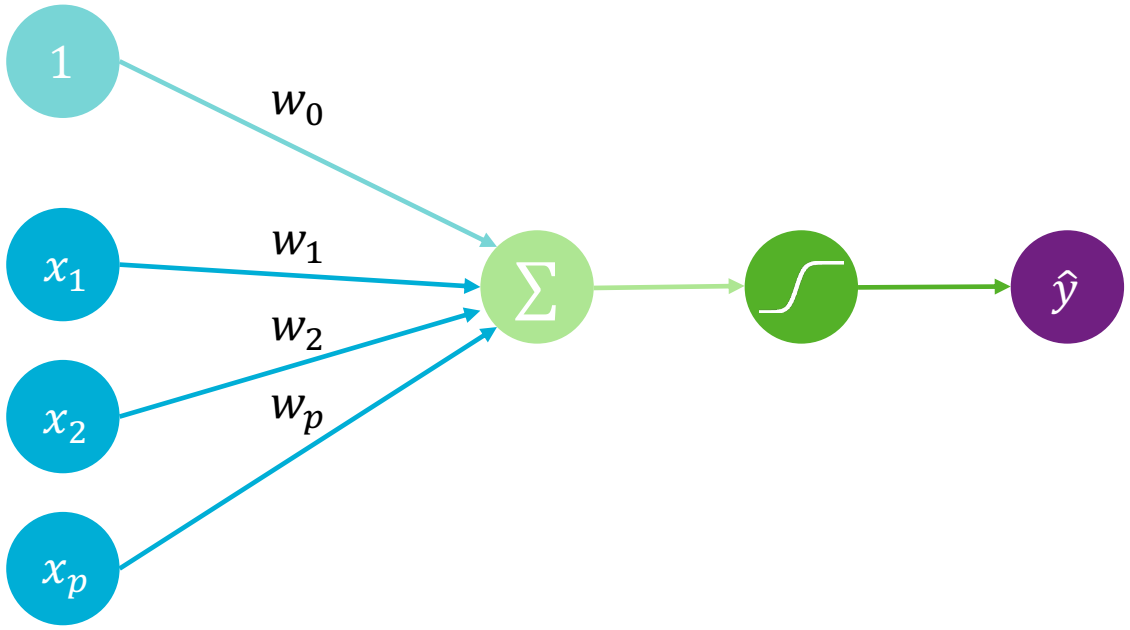
$$\hat{y} = g\left(w_0 + \sum_{j=1}^{p} x_j w_j\right)$$

Matrix notation

$$\hat{y} = g(\boldsymbol{X}^T \boldsymbol{W})$$

where $\boldsymbol{X} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_p \end{bmatrix}$

Perceptron (1958)



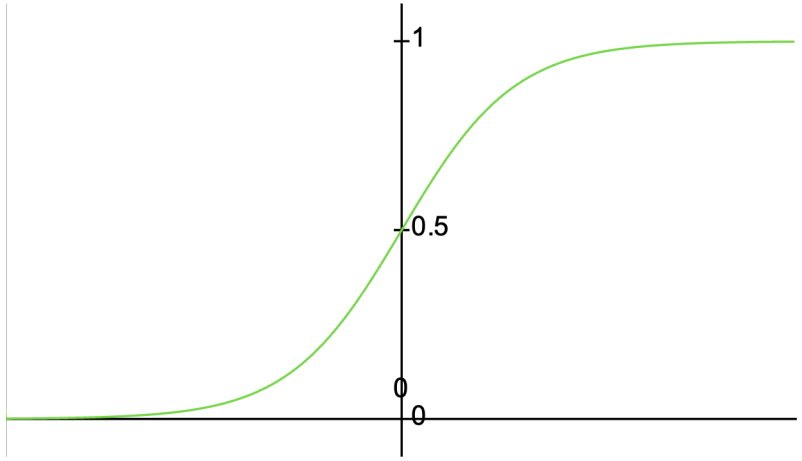
Inputs Weights Sum Non-Linearity Output

nn.Linear nn.Sigmoid

Activation function

$$\hat{y} = g(\boldsymbol{X}^T \boldsymbol{W})$$

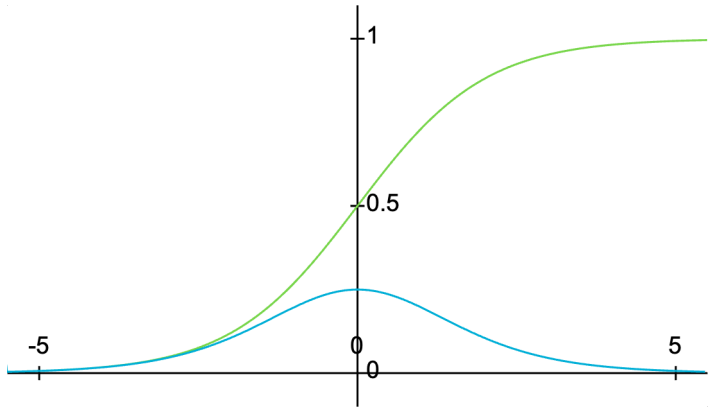
Sigmoid function: $$g(z) = \frac{1}{1 + e^{-z}}$$



→ Logistic regression

Activation functions

Sigmoid

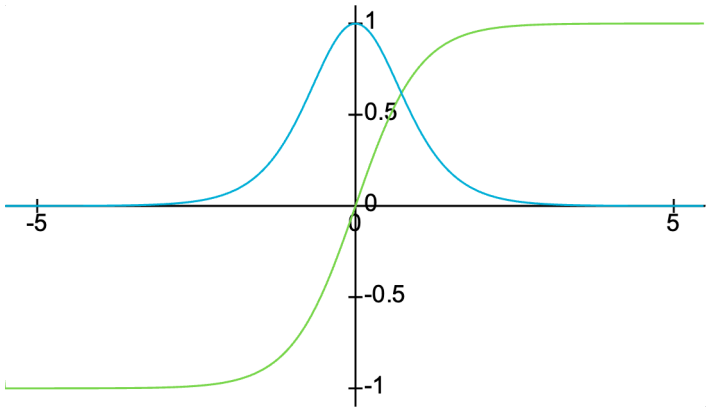


$$\mathrm{g}(z) = \frac{1}{1 + e^{-z}}$$

$$\mathrm{g}'(z) = g(z)(1 - g(z))$$

nn.Sigmoid

Hyperbolic tangent

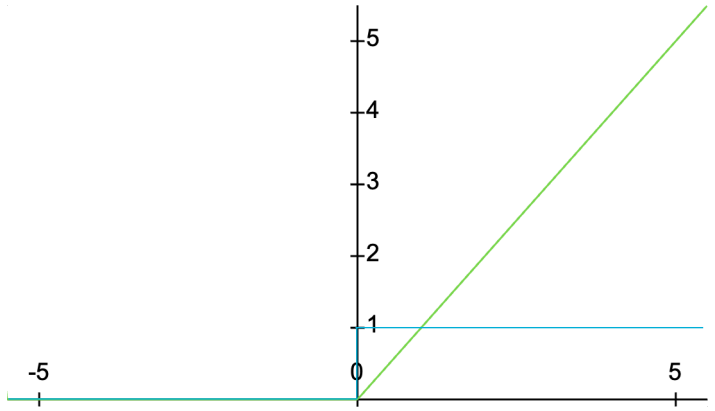


$$\mathrm{g}(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

nn.tanh

Rectified linear unit (ReLU)



$$\mathrm{g}(z) = \max(0, z)$$

$$\mathrm{g}'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

nn.ReLU

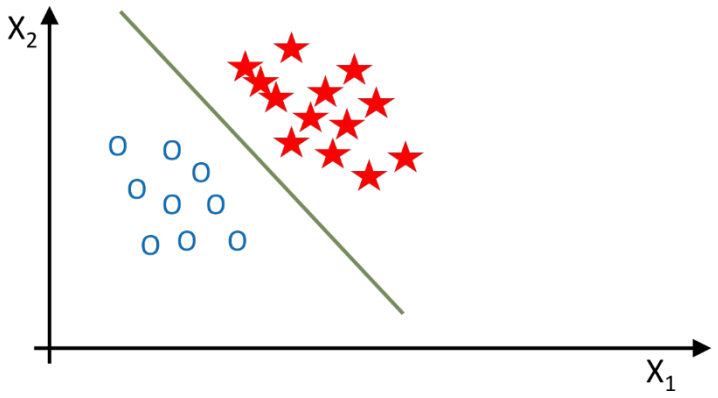
Perceptron (1958)

As logistic regression, the (single-layer) perceptron is a linear classifier

Predict Y=1 if $f(\boldsymbol{x}) \geq 0.5 \iff \boldsymbol{w}^T\boldsymbol{x} \geq 0$

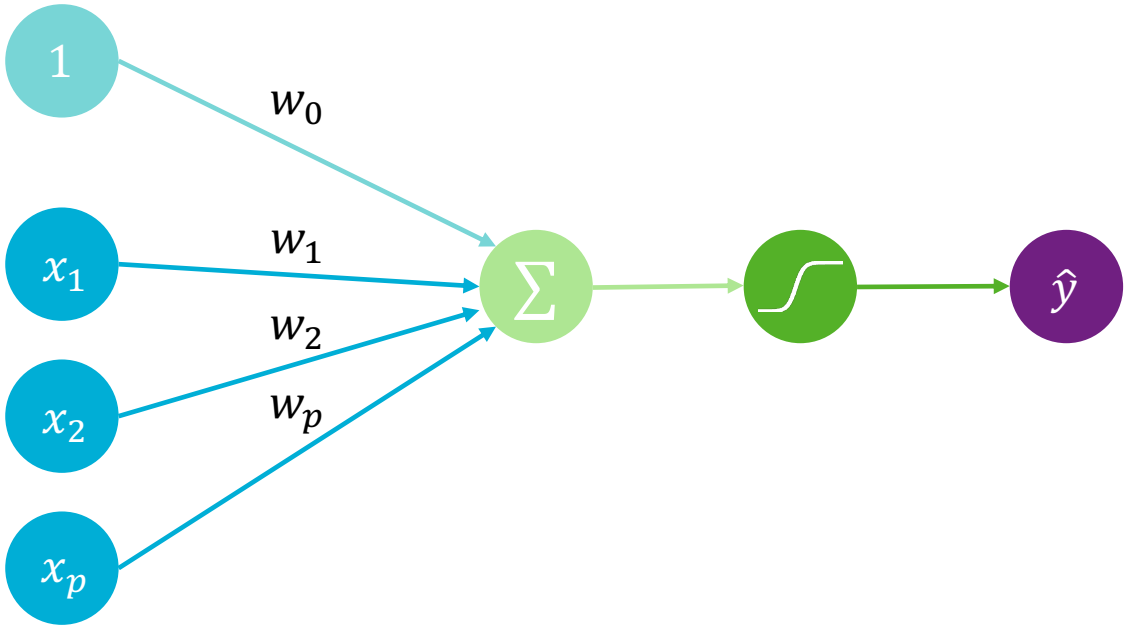
Line

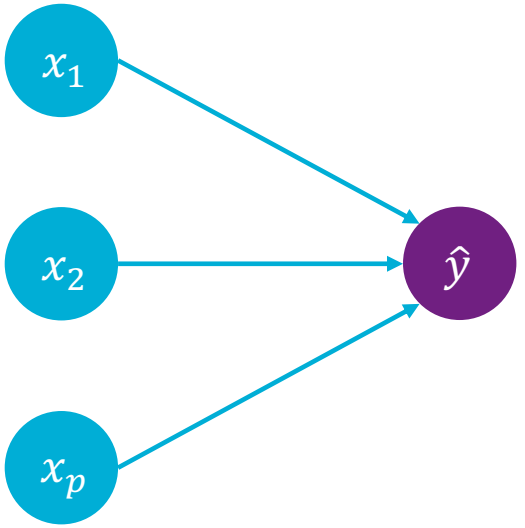
This is a **linear classification** technique even though g is a non-linear function



Perceptron (1958)

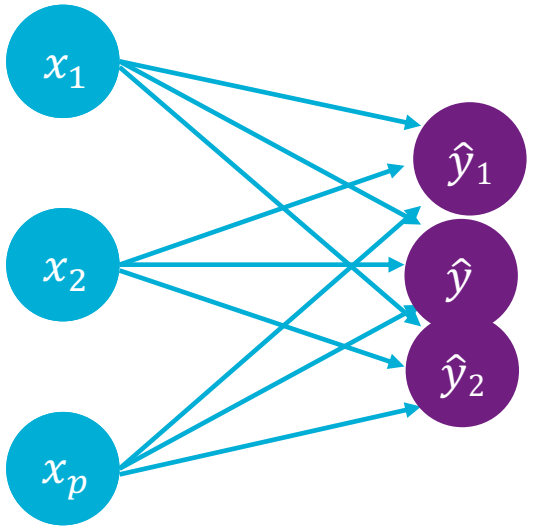
Simplified notation





Perceptron (1958)

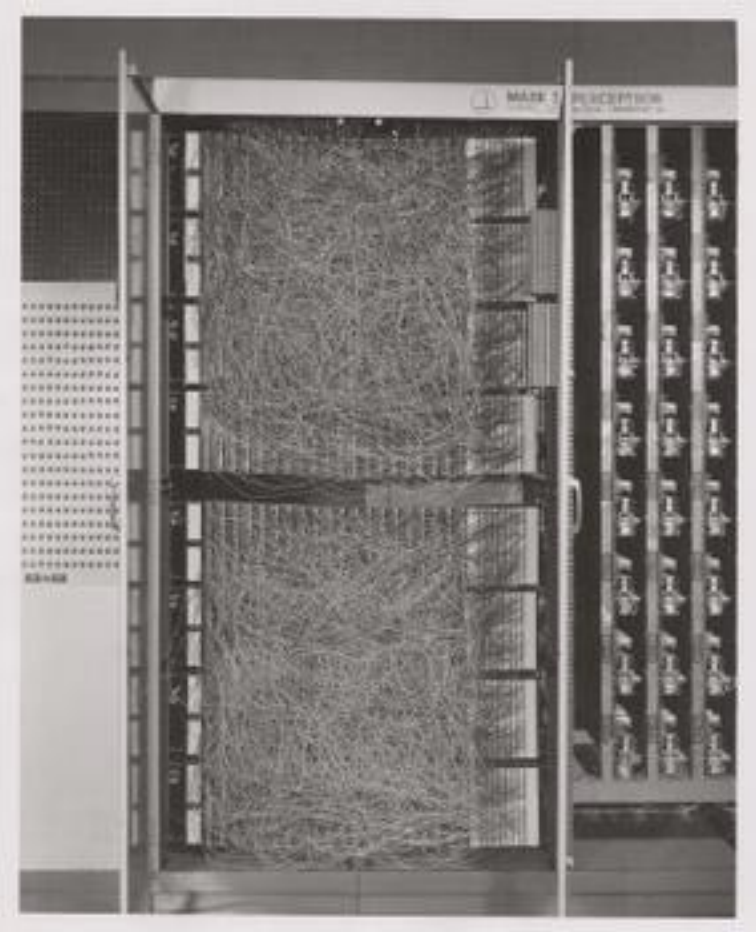
Perceptron with multiple outputs



Each output is associated with a set of weights

$$\hat{y}_i = g(\boldsymbol{X}^T \boldsymbol{W}_i)$$

1958: The perceptron



Mark I Perceptron machine, hardware implementation of the perceptron algorithm**. It was connected to a camera** with 20×20 cadmium sulfide photocells to make a 400-pixel image. To the right, arrays of **potentiometers** that implemented the adaptive weights.

Funded by US Navy

https://en.wikipedia.org/wiki/Perceptron

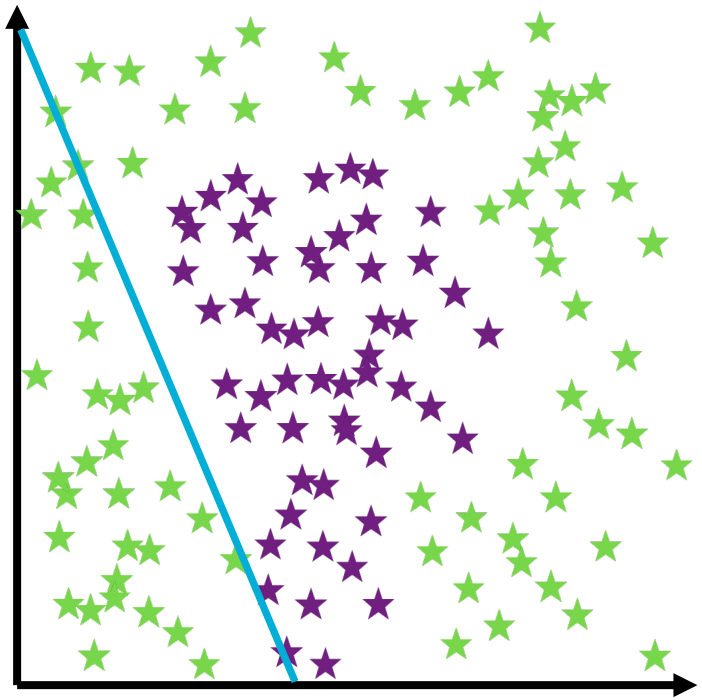
1958: The perceptron

The New York Times

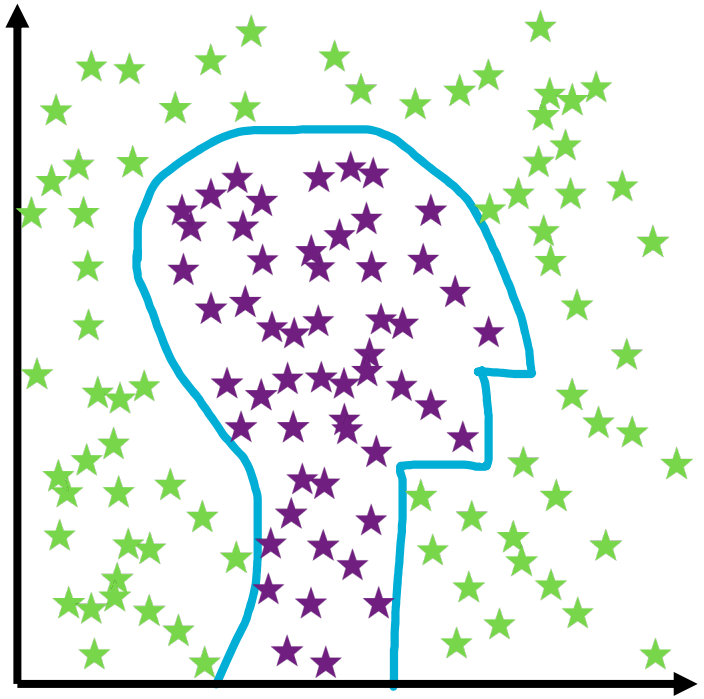
WASHINGTON, July 7 (UPI) -- The Navy revealed the embryo of an electronic computer today that it expects **will be able to walk, talk, see, write, reproduce itself and be conscious of its existence**.

Perceptron (1958)

Limitation of linear classifiers



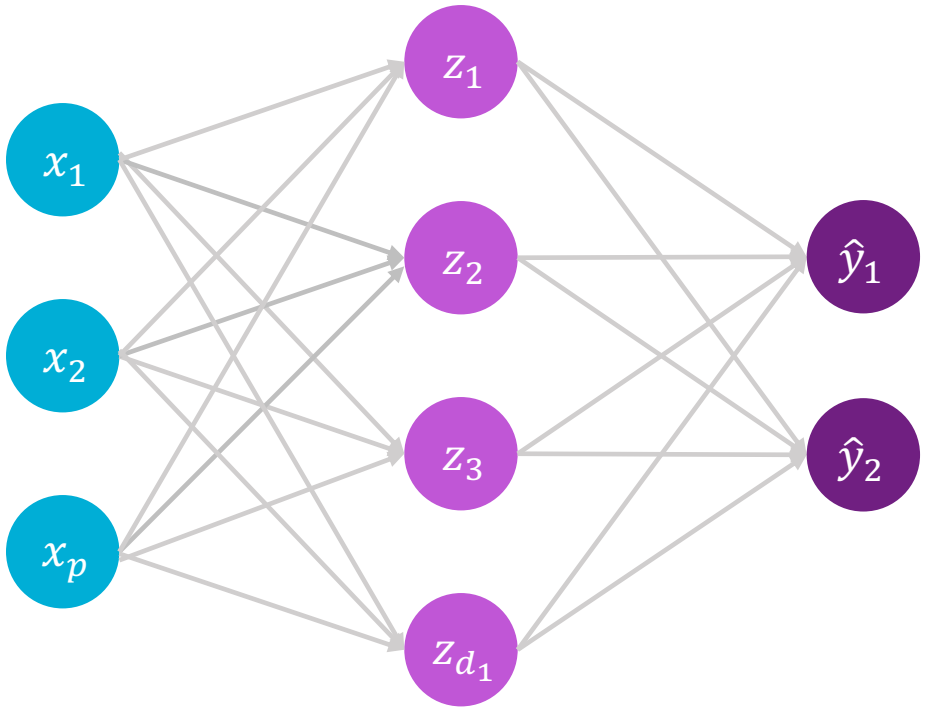
Linear decisions even though the activation is non-linear



Non-linear decisions
→ will require more layers (and still use non-linear activations, of course)

Part 2 – Classification (and regression)
2.2 Introduction to deep learning
2.2.2 Multi-layer perceptron

One hidden layer



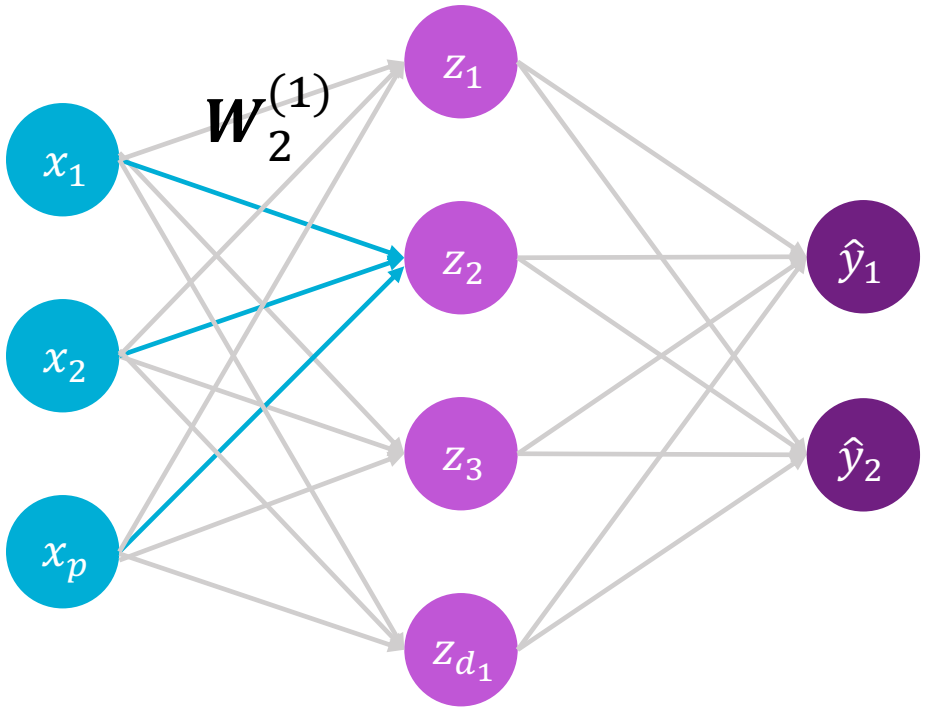
$$\hat{y}_i = g\left(\boldsymbol{Z}^T \boldsymbol{W}_i^{(2)}\right)$$

$$z_i = g\left(\boldsymbol{X}^T \boldsymbol{W}_i^{(1)}\right)$$

Index of the layer

Index of the neuron within the layer

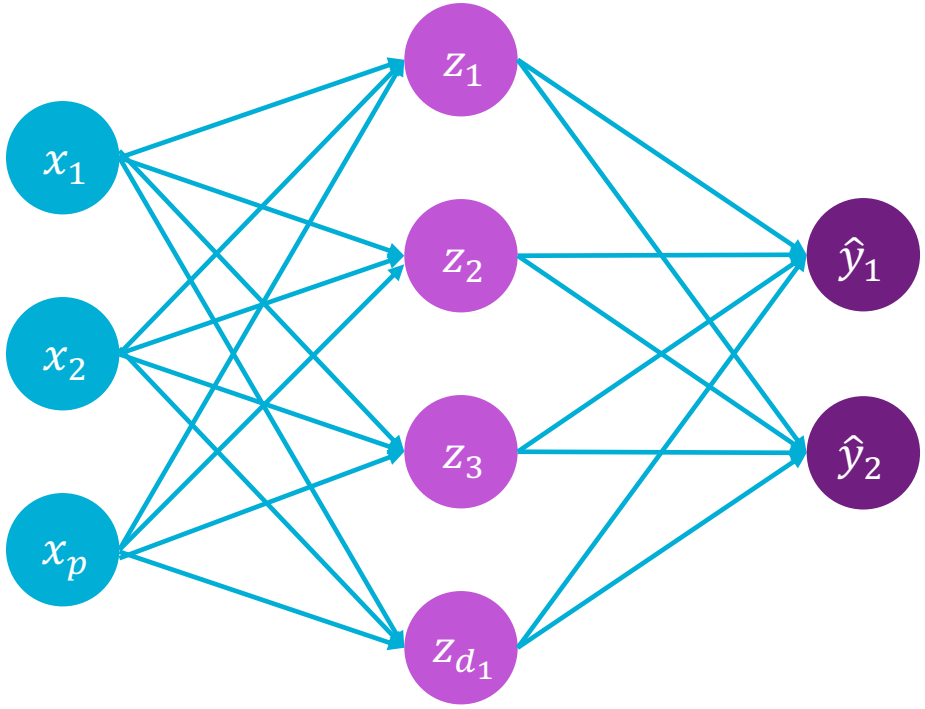
One hidden layer



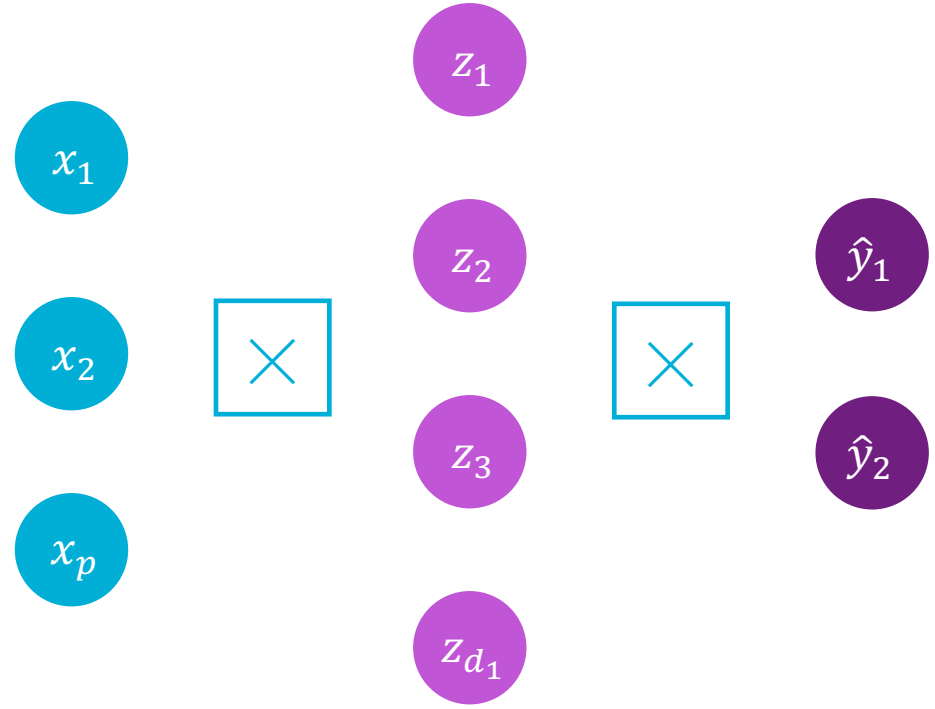
$$\hat{y}_i = g\left(\boldsymbol{Z}^T \boldsymbol{W}_i^{(2)}\right)$$

$$z_i = g\left(\boldsymbol{X}^T \boldsymbol{W}_i^{(1)}\right)$$

Simplified notation



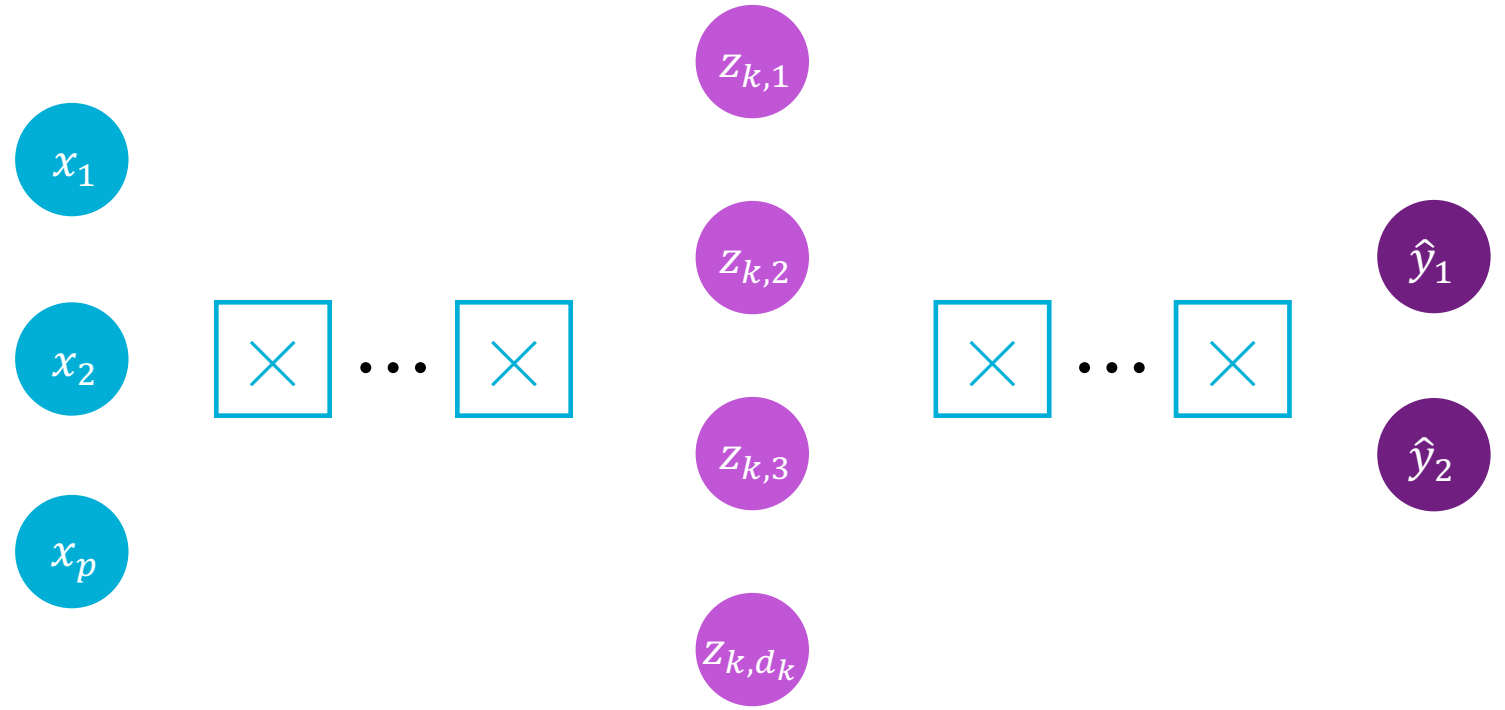
Inputs Hidden Output



Inputs Hidden Output

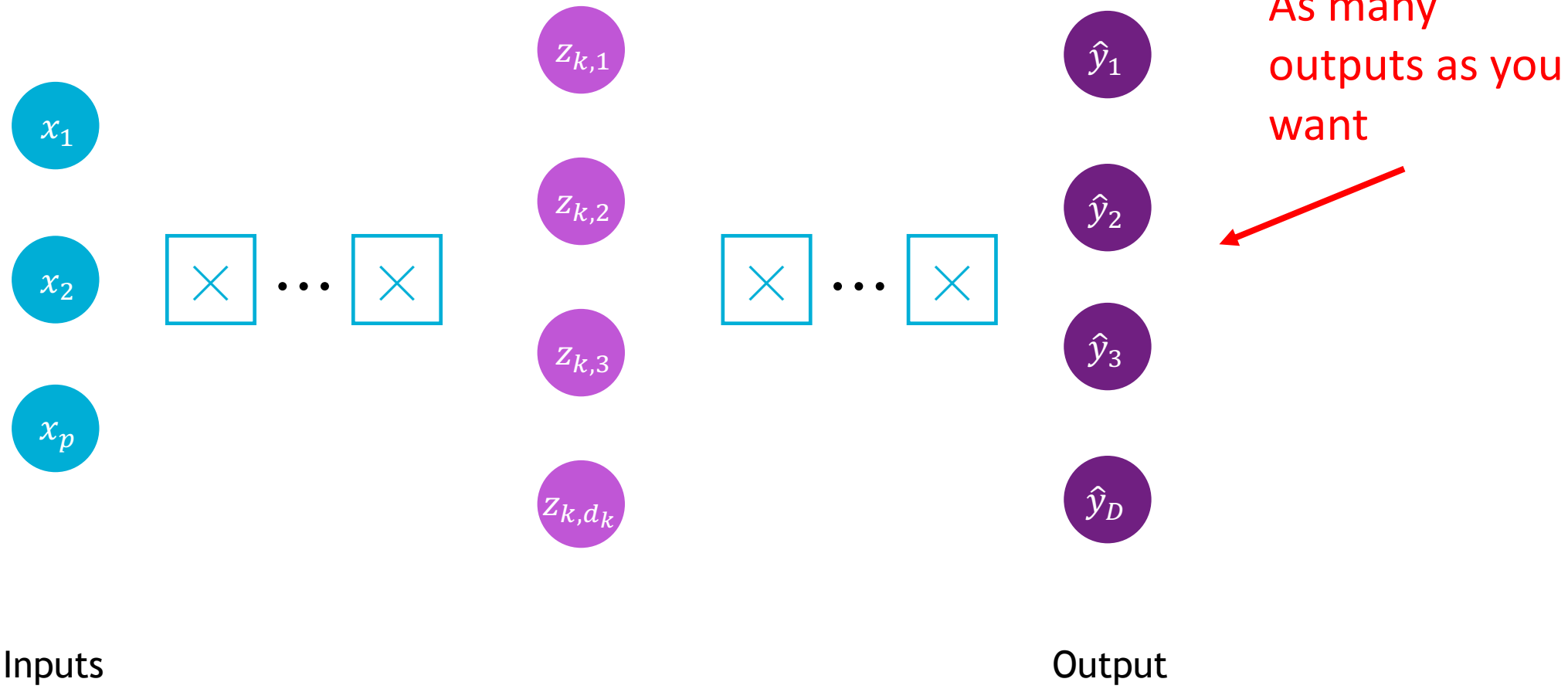
Fully connected layers or dense layers

Deep neural network



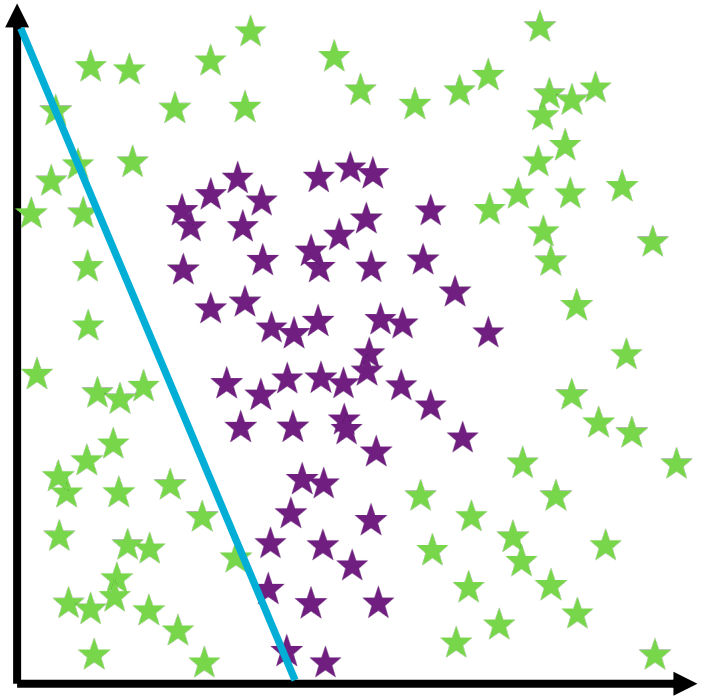
$$\boldsymbol{W} = \{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \ldots\}$$

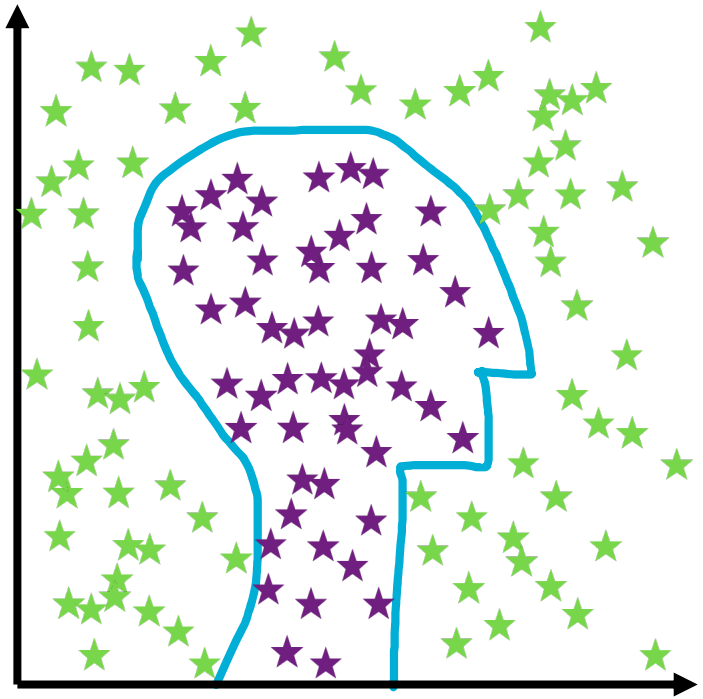
Deep neural network



Non-linearities

A neural network with multiple layers can learn non-linear boundaries because we use non-linear activation functions

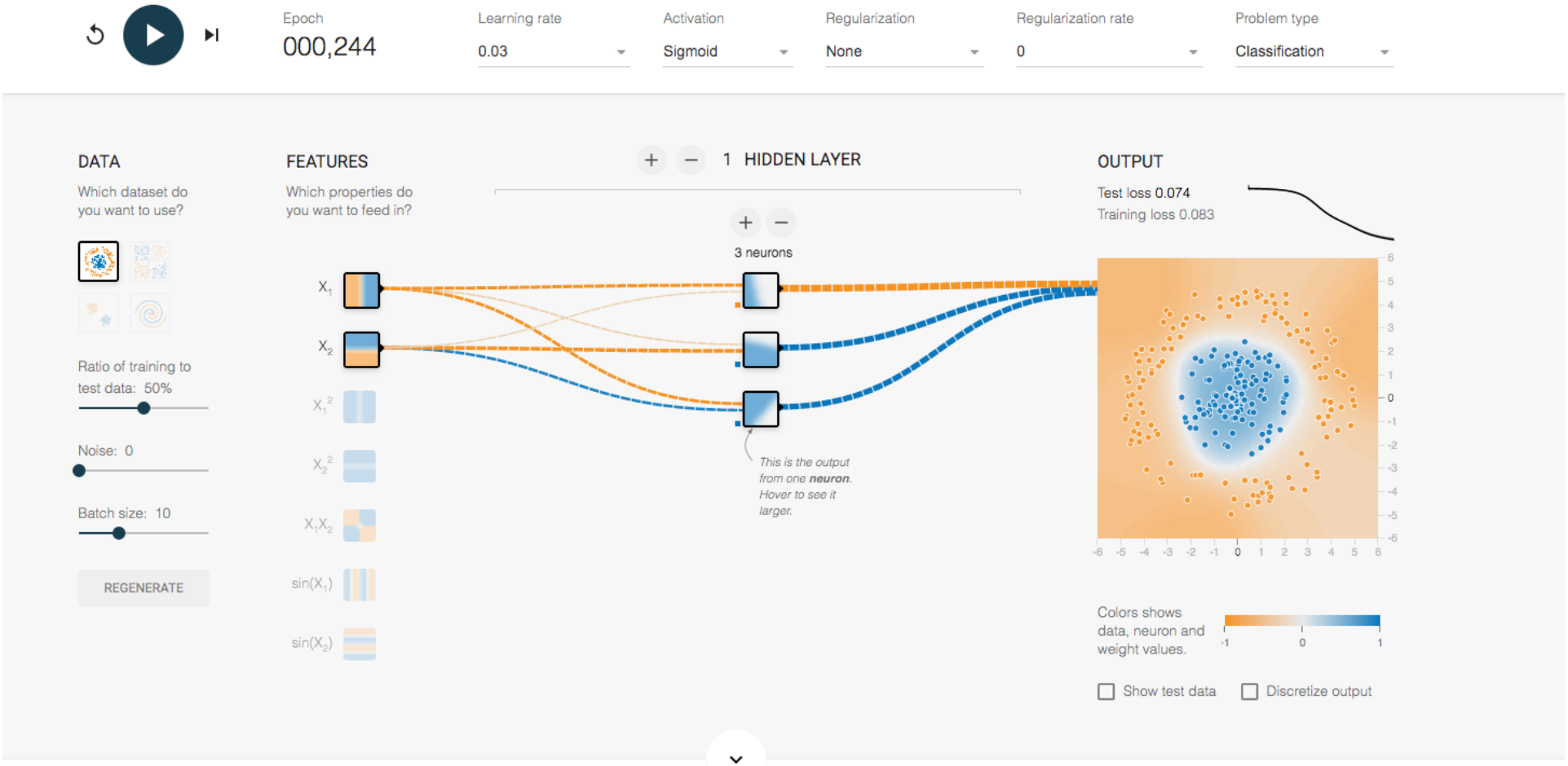




If the activations were linear the decision would be linear, no matter the number of layers

Intuition about neural networks

You can play with : http://playground.tensorflow.org



Part 2 – Classification (and regression)
2.2 Introduction to deep learning
2.2.3 Loss and cost function

Cost function

Cost function: $J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} l\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$

Predicted Actual

Nothing new here

Loss for classification

Cost function: $$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} l\big(f\big(x^{(i)};\boldsymbol{W}\big), y^{(i)}\big)$$

Predicted Actual

Loss for classification:

$$\ell(y, f(x) = -y\log(f(x) - (1-y)log(1-f(x))$$

Same as in logistic regression

Called cross-entropy loss

nn.CrossEntropyLoss

Loss for regression

Cost function: $$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} l\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Loss for regression:

$$\text{Loss: } \ell(y, f(x)) = (y - f(x))^2$$

Same as in linear regression

Least squares loss

nn.MSELoss

Part 2 – Classification (and regression)
2.2 Introduction to deep learning
2.2.4 Training

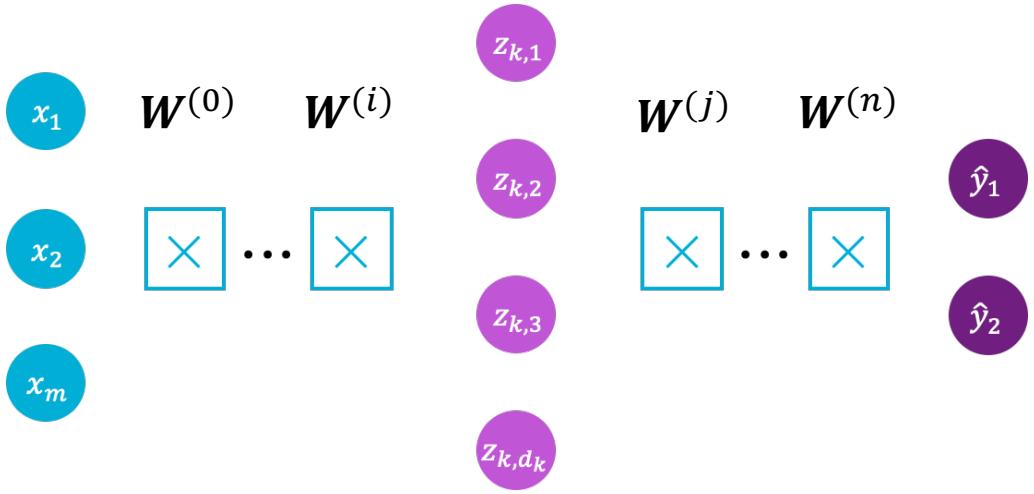
Training

Loss optimisation

- **Find the network weights that achieve the lowest loss**

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\operatorname{argmin}} J(\boldsymbol{W}) = \underset{\boldsymbol{W}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} l\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W} = \left\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \ldots\right\}$$

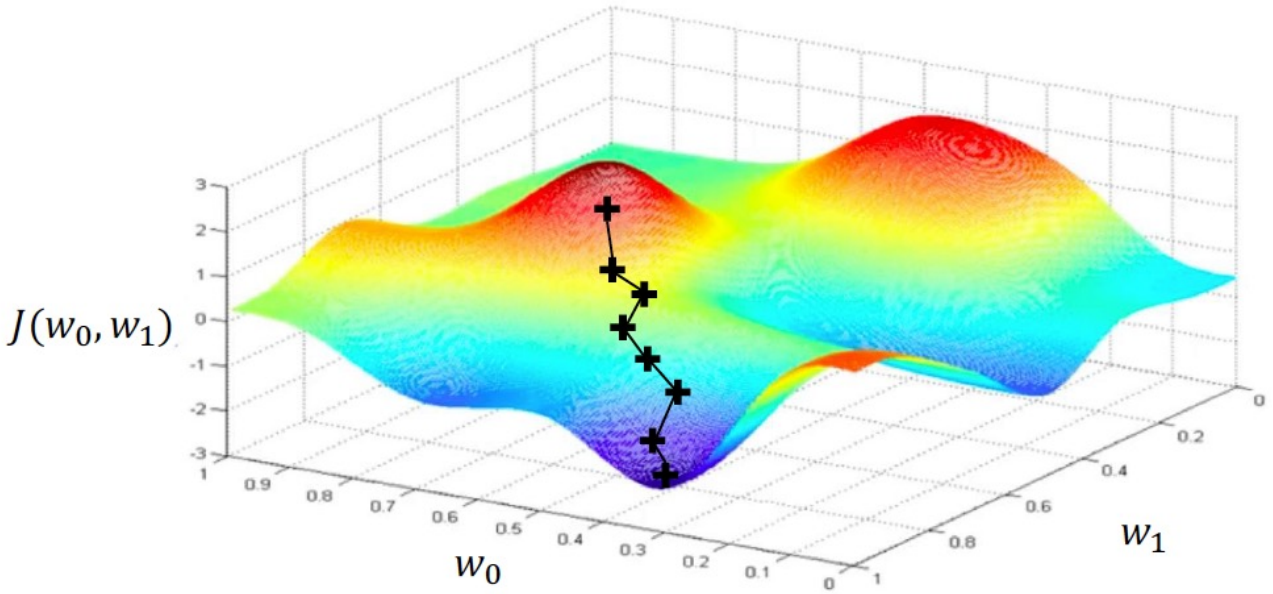


Optimization

Gradient descent

Algorithm

1. Initialise weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence
   a. Compute gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
   b. Update weights $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
3. Return weights



Source: http://introtodeeplearning.com/

Optimization

Computing gradients: backpropagation

$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_2}$$

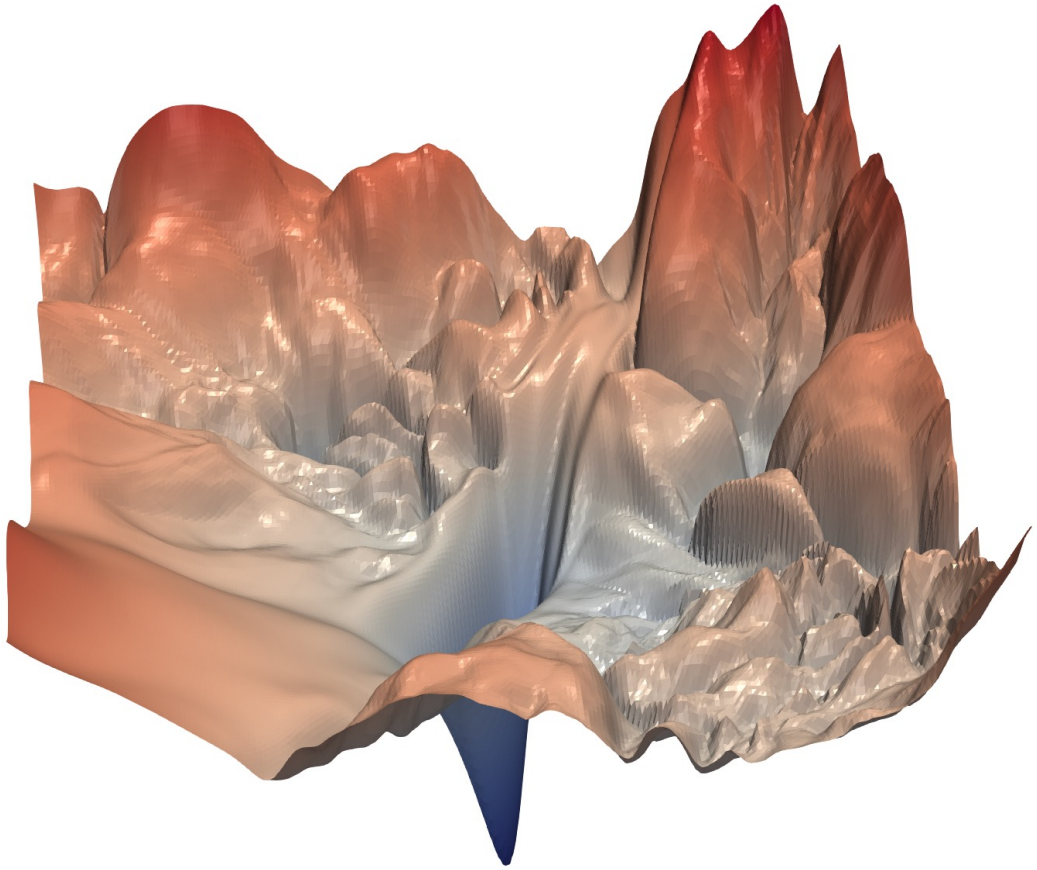
$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}$$

Optimization

Gradient descent in practice

Algorithm

1. Initialise weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence
   a. Compute gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
   b. Update weights $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
3. Return weights



Li et al., Visualizing the Loss Landscape of Neural Nets, NIPS, 2018

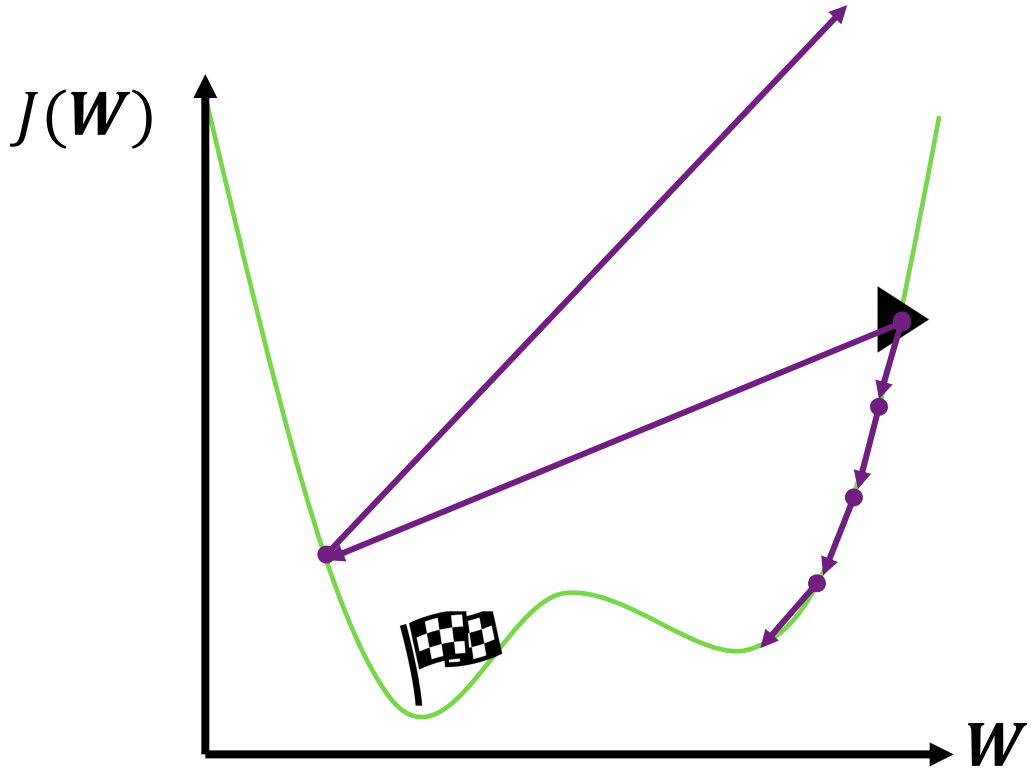
Optimization

Learning rate

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

- If η is too small: slow to converge, may be trapped in local minima
- If η is too large: may diverge

→ Adaptative learning rate



Optimization

Adaptative learning rate

- Learning rates are no longer fixed
- Can be made larger or smaller depending on:
  - how large the gradient is
  - how fast learning is happening
  - the size of particular weights
  - etc.
- Algorithms:
  - Adam [Kingma et al., Adam: A Method for Stochastic Optimization, 2014]
  - Adadelta [Zeiler et al., ADADELTA: An Adaptive Learning Rate Method, 2012]
  - Adagrad [Duchi et al., Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, 2011]
  - RMSProp [Hinton, Neural Networks for Machine Learning]

optim.Adam

Optimization

Standard gradient descent

Algorithm

1. Initialise weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence
   a. Compute gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$ — Can be expensive to compute if the number of samples is very large
   b. Update weights $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
3. Return weights

Optimization

Stochastic gradient descent with one sample

Algorithm

1. Initialise weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence
   a. Pick single sample i
   b. Compute gradient $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$ Cheap to compute but **very noisy**
   c. Update weights $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
3. Return weights

Optimization

Stochastic gradient descent with several samples (mini-batch)

Algorithm

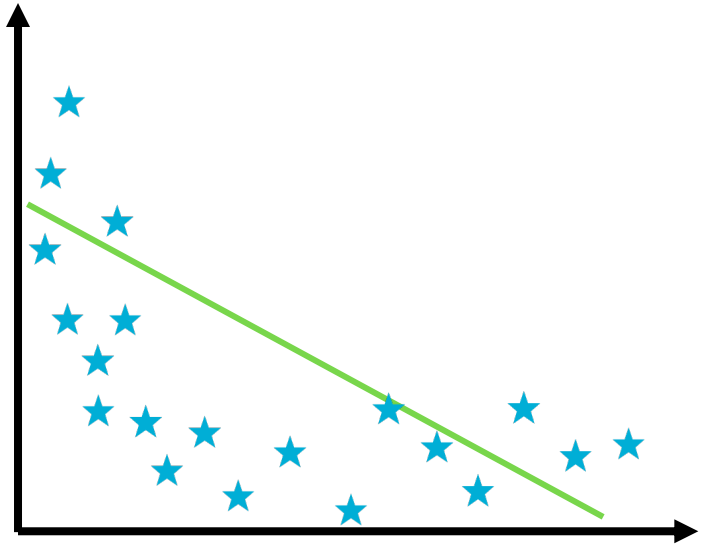
1. Initialise weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence
   a. Pick batch of B samples
   b. Compute gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
   c. Update weights $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
3. Return weights

Cheap to compute and good estimate of the true gradient

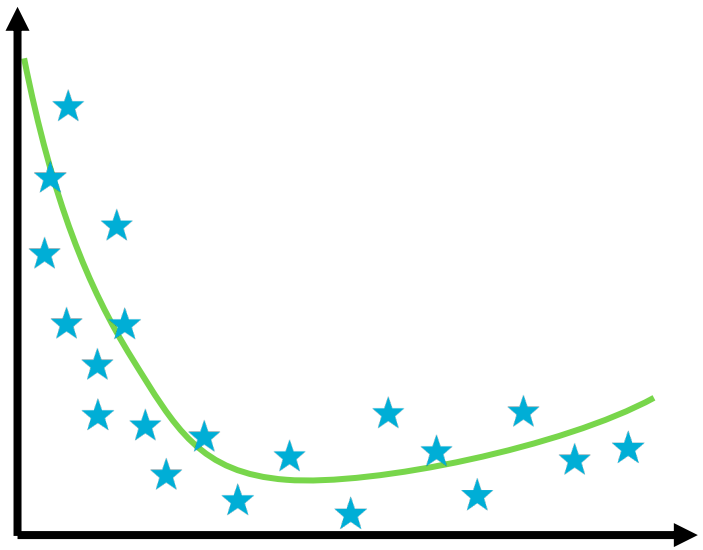
Part 2 – Classification (and regression)
2.2 Introduction to deep learning
2.2.5 Overfitting and regularization

Overfitting

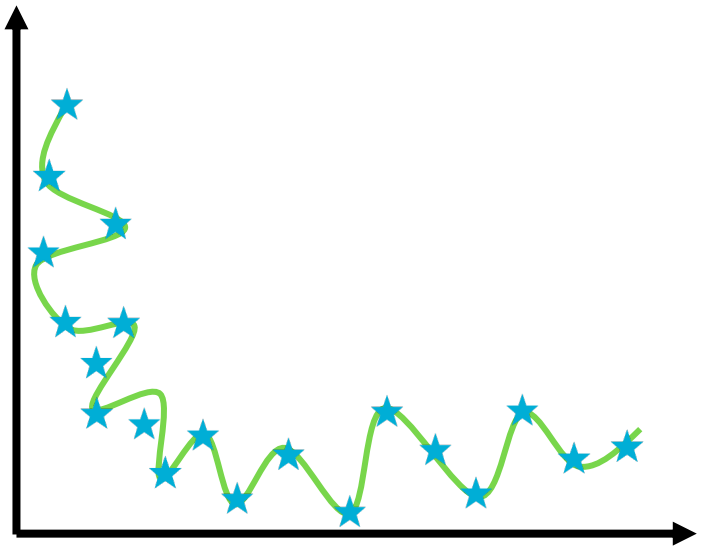
Overfitting



Underfitting



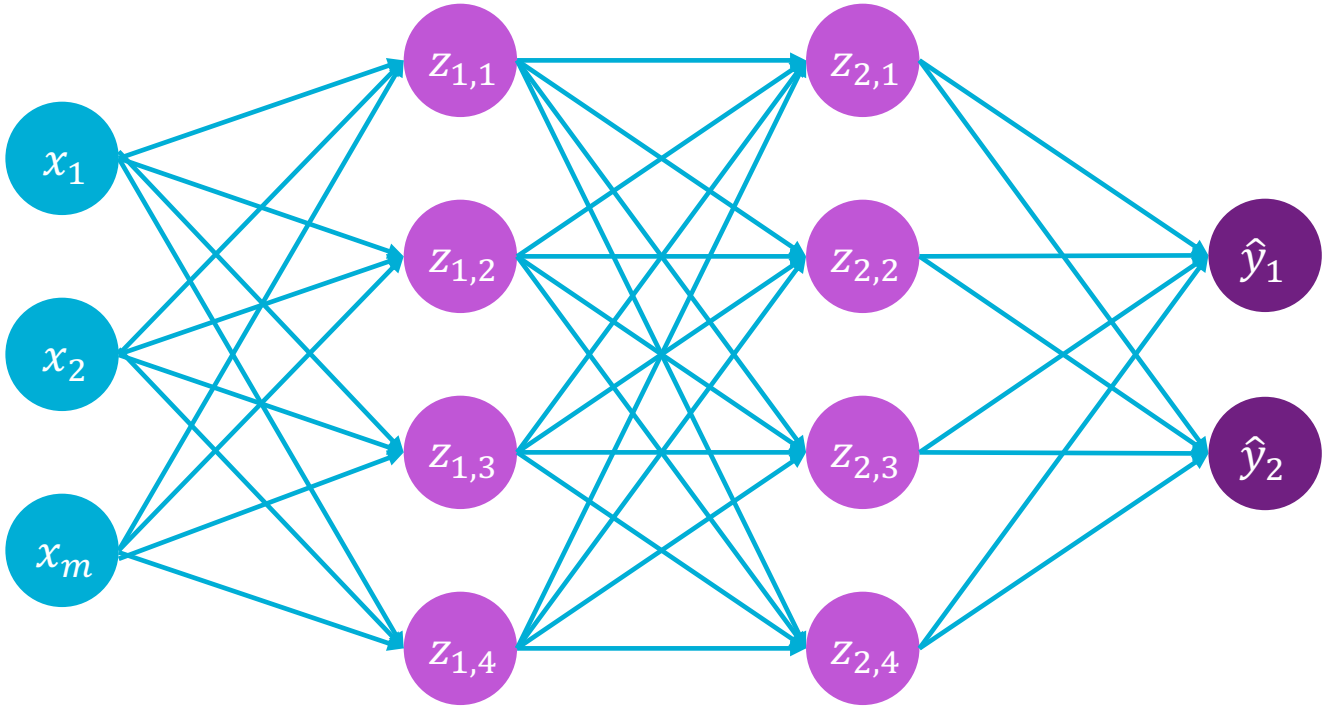
Ideal fit



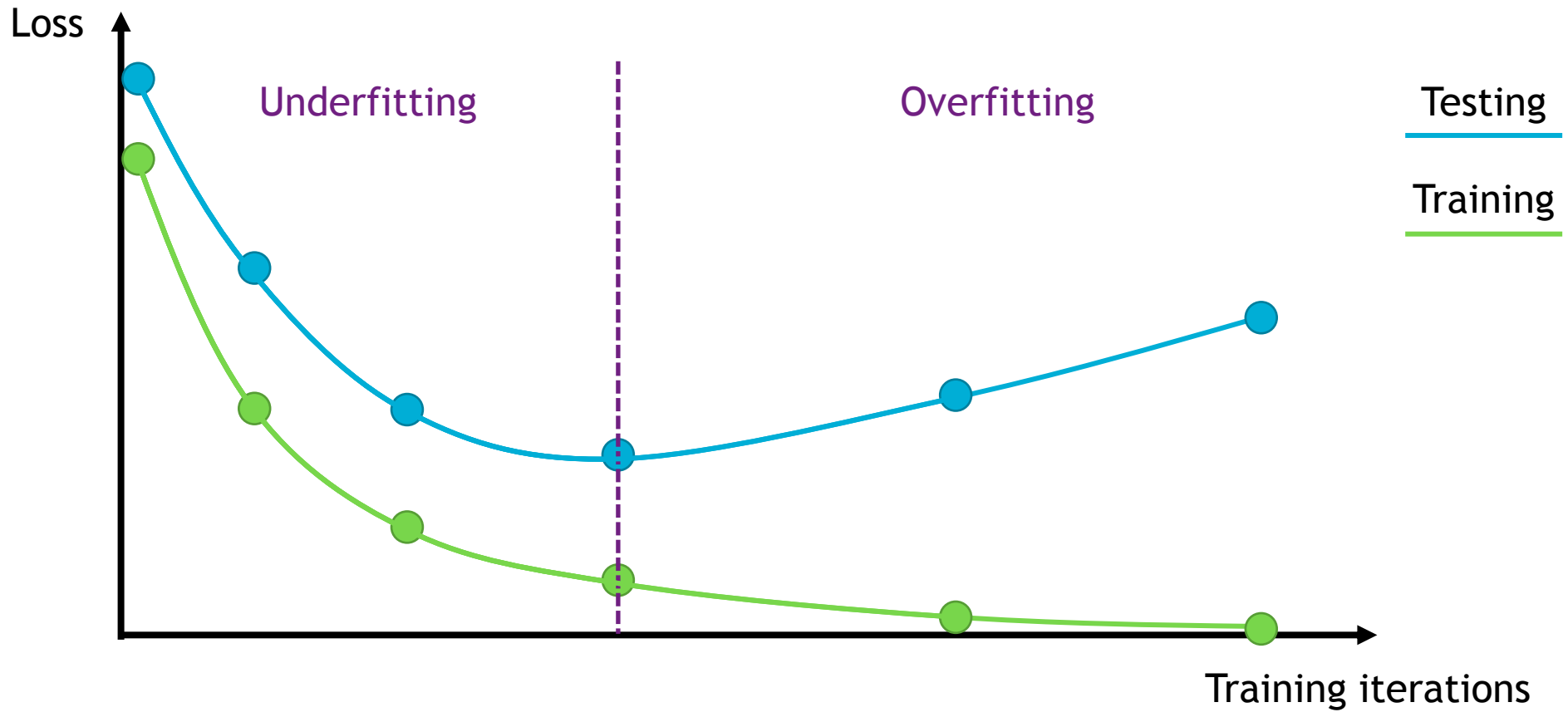
Overfitting

Regularization: drop-out

nn.Dropout



Early stopping



Course starts here

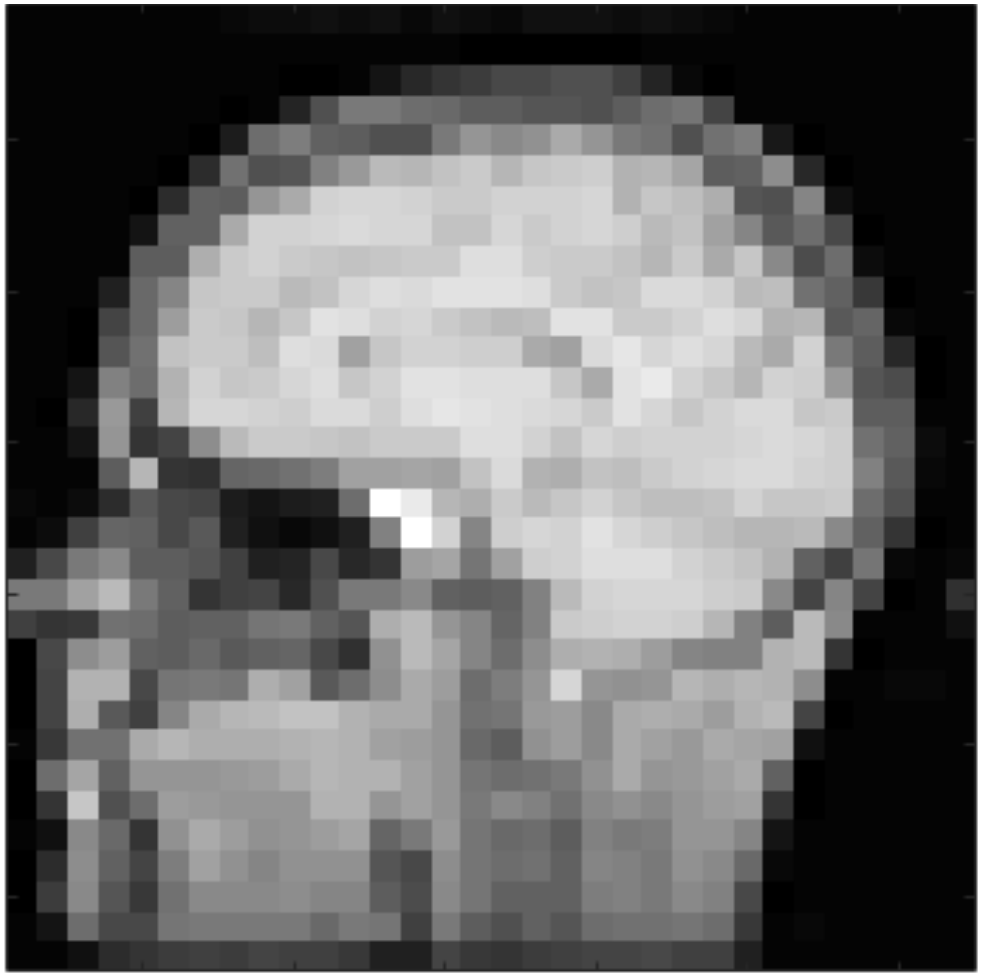
Part 2 – Classification (and regression)
2.3 Convolutional neural networks (CNNs)

CNN

- Specific type of neural network
  - Well adapted to image data
- State-of-the-art for many image analysis tasks
- Inspired from biological vision (visual cortex)

Using an image as input of a neural network

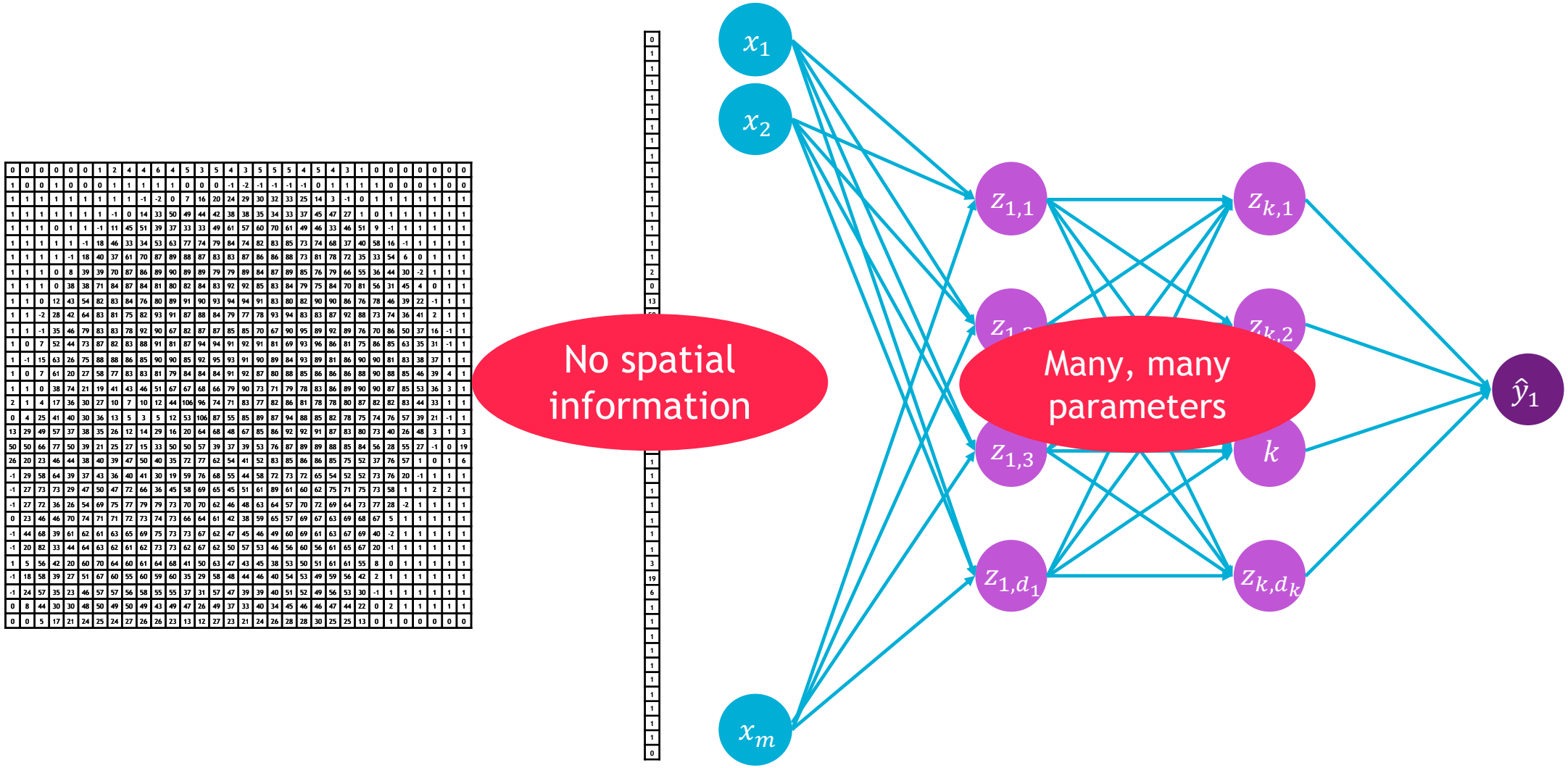
Image = matrix of numbers



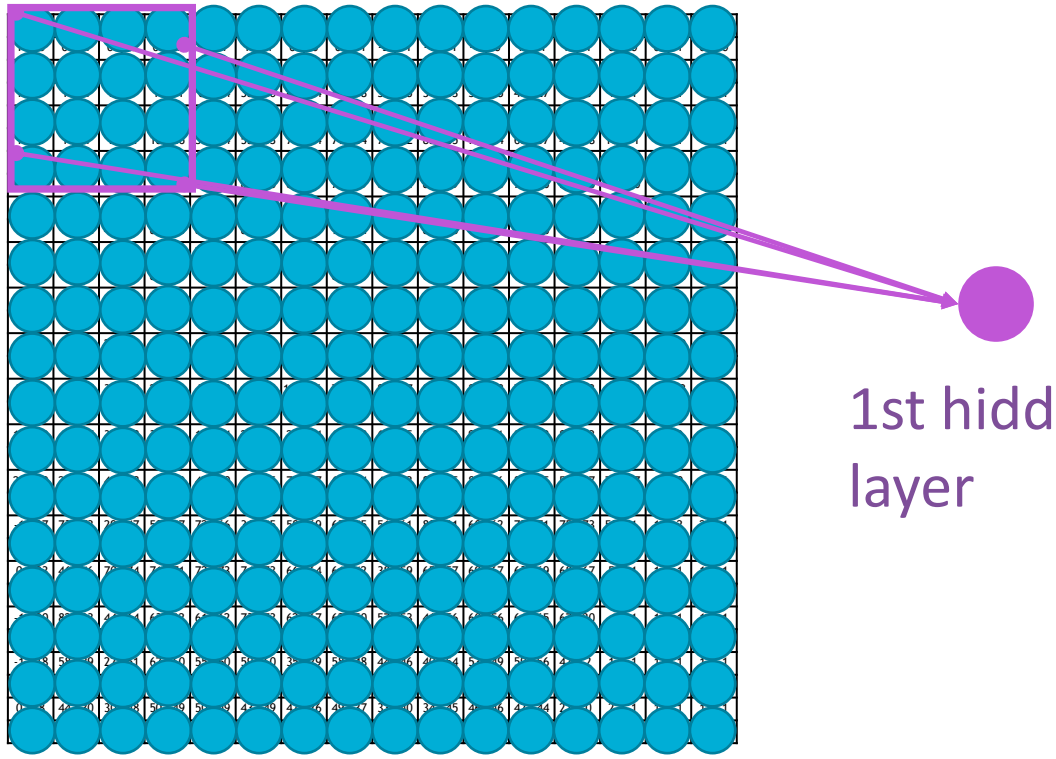
0	0	0	0	0	0	1	2	4	4	6	4	5	3	5	4	3	5	5	5	4	5	4	3	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	1	1	1	1	0	0	0	-1	-2	-1	-1	-1	-1	0	1	1	1	1	0	0	0	1	0	0
1	1	1	1	1	1	1	1	1	-1	-2	0	7	16	20	24	29	30	32	33	25	14	3	-1	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	-1	0	14	33	50	49	44	42	38	38	35	34	33	37	45	47	27	1	0	1	1	1	1	1	1
1	1	1	0	1	1	-1	11	45	51	39	37	33	33	49	61	57	60	70	61	49	46	33	46	51	9	-1	1	1	1	1	1
1	1	1	1	1	-1	18	46	33	34	53	63	77	74	79	84	74	82	83	85	73	74	68	37	40	58	16	-1	1	1	1	1
1	1	1	1	-1	18	40	37	61	70	87	89	88	87	83	83	87	86	86	88	73	81	78	72	35	33	54	6	0	1	1	1
1	1	1	0	8	39	39	70	87	86	89	90	89	89	79	79	89	84	87	89	85	76	79	66	55	36	44	30	-2	1	1	1
1	1	1	0	38	38	71	84	87	84	81	80	82	84	83	92	92	85	83	84	79	75	84	70	81	56	31	45	4	0	1	1
1	1	0	12	43	54	82	83	84	76	80	89	91	90	93	94	94	91	83	80	82	90	90	86	76	78	46	39	22	-1	1	1
1	1	-2	28	42	64	83	81	75	82	93	91	87	88	84	79	77	78	93	94	83	83	87	92	88	73	74	36	41	2	1	1
1	1	-1	35	46	79	83	83	78	92	90	67	82	87	87	85	85	70	67	90	95	89	92	89	76	70	86	50	37	16	-1	1
1	0	7	52	44	73	87	82	83	88	91	81	87	94	94	91	92	91	81	69	93	96	86	81	75	86	85	63	35	31	-1	1
1	-1	15	63	26	75	88	88	86	85	90	90	85	92	95	93	91	90	89	84	93	89	81	86	90	90	81	83	38	37	1	1
1	0	7	61	20	27	58	77	83	83	81	79	84	84	84	91	92	87	80	88	85	86	86	86	88	90	88	85	46	39	4	1
1	1	0	38	74	21	19	41	43	46	51	67	67	68	66	79	90	73	71	79	78	83	86	89	90	90	87	85	53	36	3	1
2	1	4	17	36	30	27	10	7	10	12	44	106	96	74	71	83	77	82	86	81	78	78	80	87	82	82	83	44	33	1	1
0	4	25	41	40	30	36	13	5	3	5	12	53	106	87	55	85	89	87	94	88	85	82	78	75	74	76	57	39	21	-1	1
13	29	49	57	37	38	35	26	12	14	29	16	20	64	68	48	67	85	86	92	92	91	87	83	80	73	40	26	48	3	1	3
50	50	66	77	50	39	21	25	27	15	33	50	50	57	39	37	39	53	76	87	89	89	88	85	84	56	28	55	27	-1	0	19
26	20	23	46	44	38	40	39	47	50	40	35	72	77	62	54	41	52	83	85	86	86	85	75	52	37	76	57	1	0	1	6
-1	29	58	64	39	37	43	36	40	41	30	19	59	76	68	55	44	58	72	73	72	65	54	52	52	73	76	20	-1	1	1	1
-1	27	73	73	29	47	50	47	72	66	36	45	58	69	65	45	51	61	89	61	60	62	75	71	75	73	58	1	1	2	2	1
-1	27	72	36	26	54	69	75	77	79	79	73	70	70	62	46	48	63	64	57	70	72	69	64	73	77	28	-2	1	1	1	1
0	23	46	46	70	74	71	71	72	73	74	73	66	64	61	42	38	59	65	57	69	67	63	69	68	67	5	1	1	1	1	1
-1	44	68	39	61	62	61	63	65	69	75	73	73	67	62	47	45	46	49	60	69	61	63	67	69	40	-2	1	1	1	1	1
-1	20	82	33	44	64	63	62	61	62	73	73	62	67	62	50	57	53	46	56	60	56	61	65	67	20	-1	1	1	1	1	1
1	5	56	42	20	60	70	64	60	61	64	68	41	50	63	47	43	45	38	53	50	51	61	61	55	8	0	1	1	1	1	1
-1	18	58	39	27	51	67	60	55	60	59	60	35	29	58	48	44	46	40	54	53	49	59	56	42	2	1	1	1	1	1	1
-1	24	57	35	23	46	57	57	56	58	55	55	37	31	57	47	39	39	40	51	52	49	56	53	30	-1	1	1	1	1	1	1
0	8	44	30	30	48	50	49	50	49	43	49	47	26	49	37	33	40	34	45	46	46	47	44	22	0	2	1	1	1	1	1
0	0	5	17	21	24	25	24	27	26	26	23	13	12	27	23	21	24	26	28	28	30	25	25	13	0	1	0	0	0	0	0

Using an image as input of a neural network

Fully connected neural network



Local connectivity

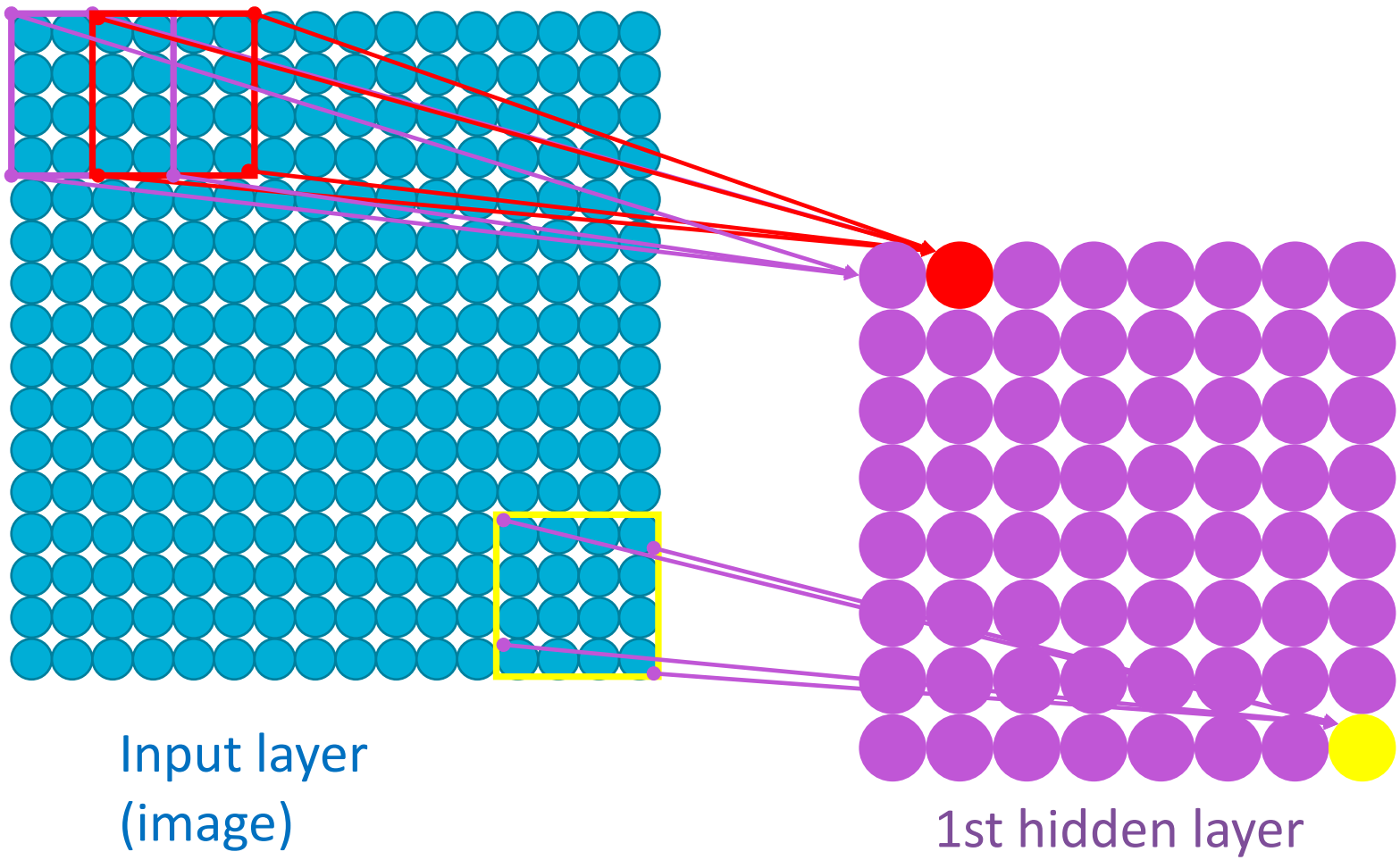


Idea: a given neuron only sees a part of the image

This part is called the **patch window** or **receptive field**

In practice, it will only be connected to the pixels of this patch window

Local connectivity

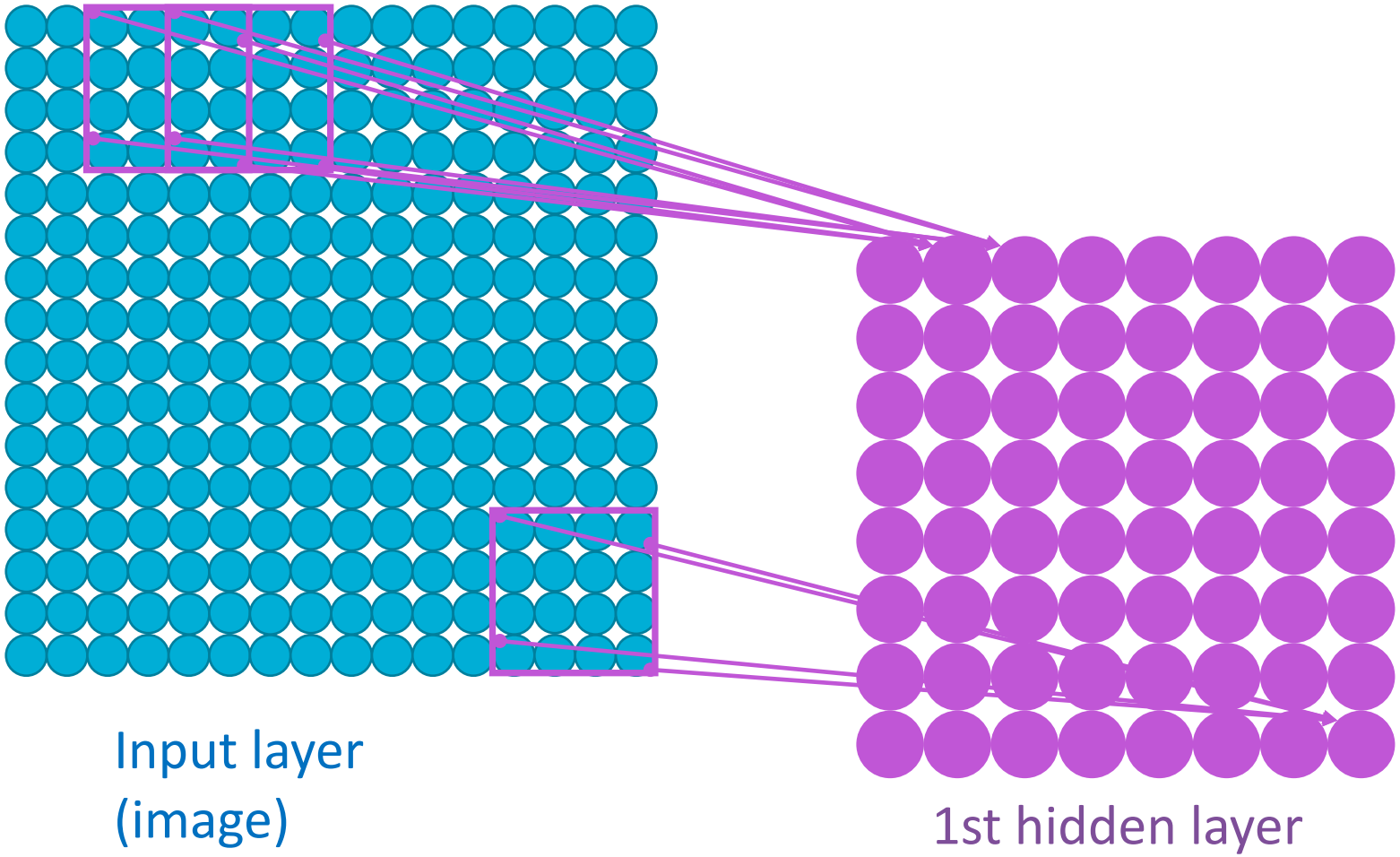


Same thing for all neurons

Slide patch window across input image

Neighbouring neurons in 1st hidden layer see neighbouring patches

Shared weights



Idea: all neurons share the same weights, they all learn the same parameters

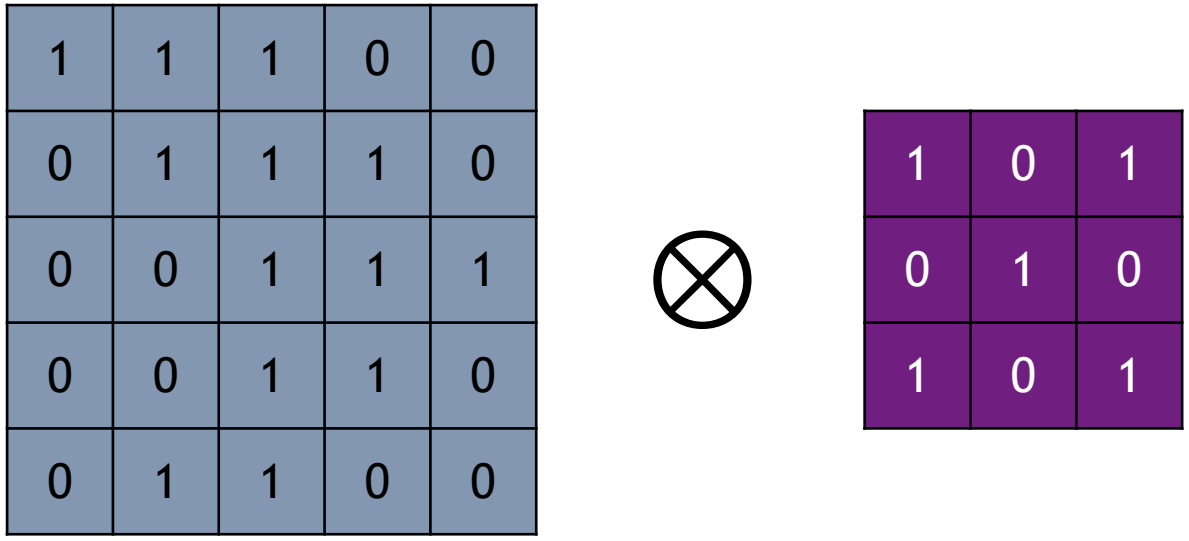
One set of weights will be called a filter

Neurons will be performing a discrete convolution operation

Discrete convolution

The convolution operation

- **Slide the filter over the input image**
- **Element-wise multiply**
- **Add the outputs**



Discrete convolution

The convolution operation

- **Slide the filter over the input image**
- **Element-wise multiply**
- **Add the outputs**

$$1\times1 + 1\times0 + 1\times1 + 0\times0 + 1\times1 + 1\times0 + 0\times1 + 0\times0 + 1\times1 = 4$$

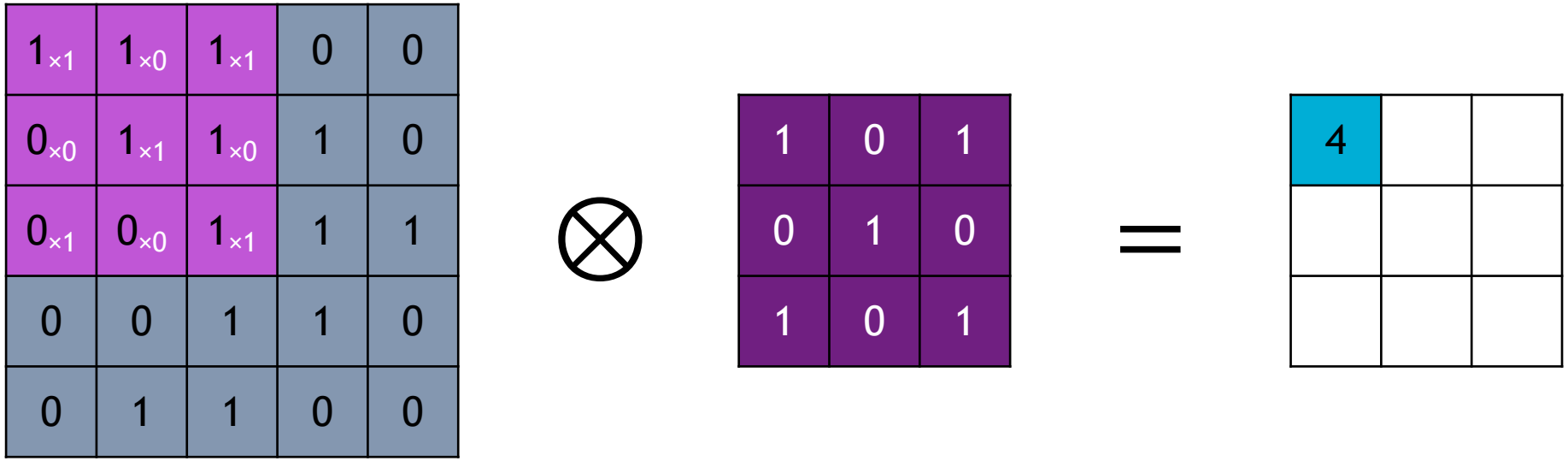


Image Filter Feature map

Discrete convolution

The convolution operation

- **Slide the filter over the input image**
- **Element-wise multiply**
- **Add the outputs**

$$1\times1 + 1\times0 + 0\times1 + 1\times0 + 1\times1 + 1\times0 + 0\times1 + 1\times0 + 1\times1 = 3$$

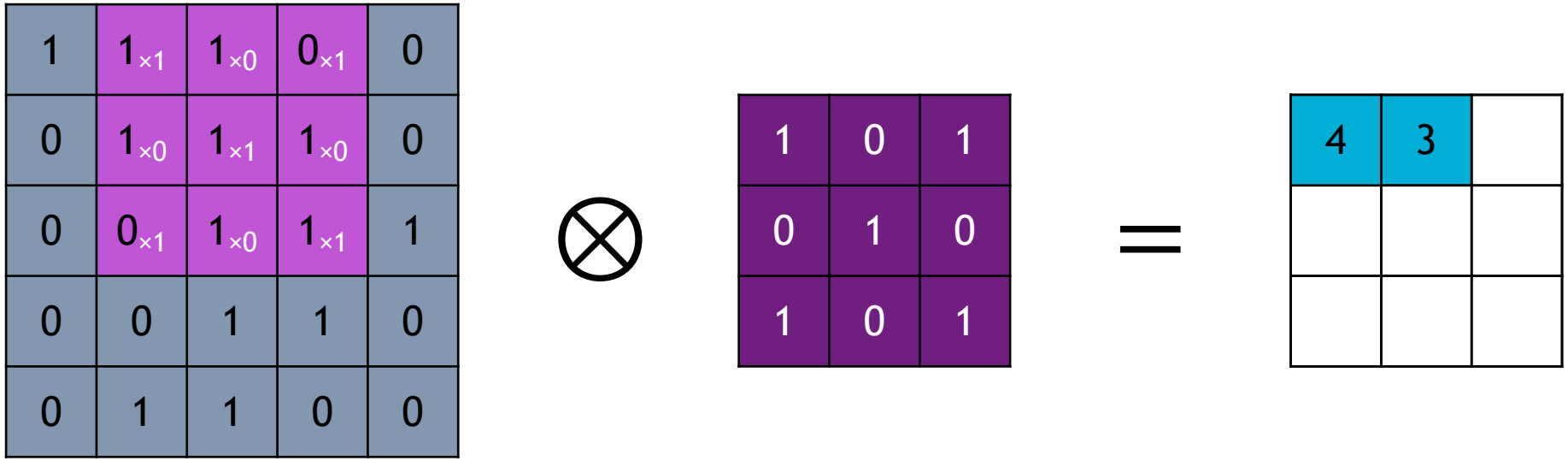


Image Filter Feature map

Discrete convolution

The convolution operation

- **Slide the filter over the input image**
- **Element-wise multiply**
- **Add the outputs**

$$1\times1 + 1\times0 + 1\times1 + 1\times0 + 1\times1 + 0\times0 + 1\times1 + 0\times0 + 0\times1 = 4$$

The filter will be learned

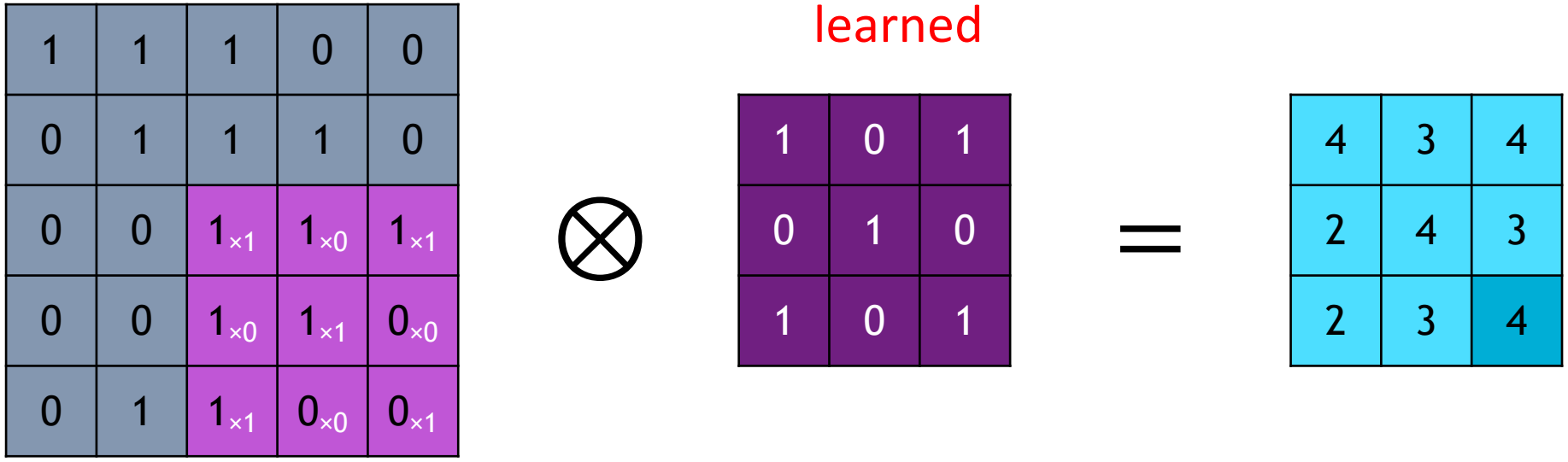
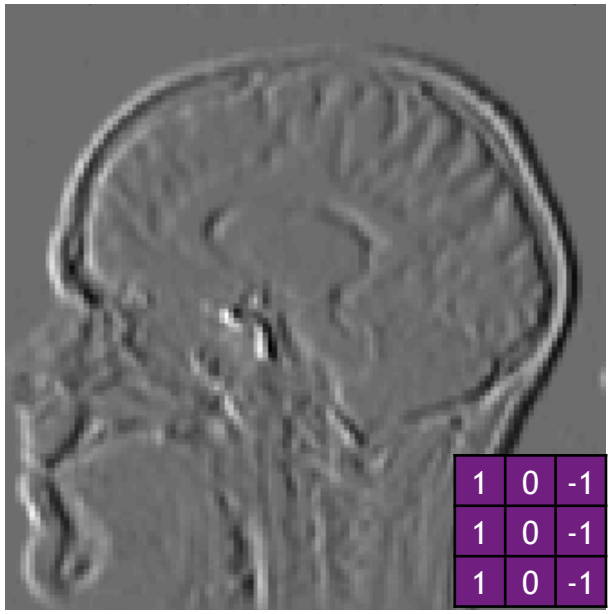


Image Filter=parameters=weights Feature map

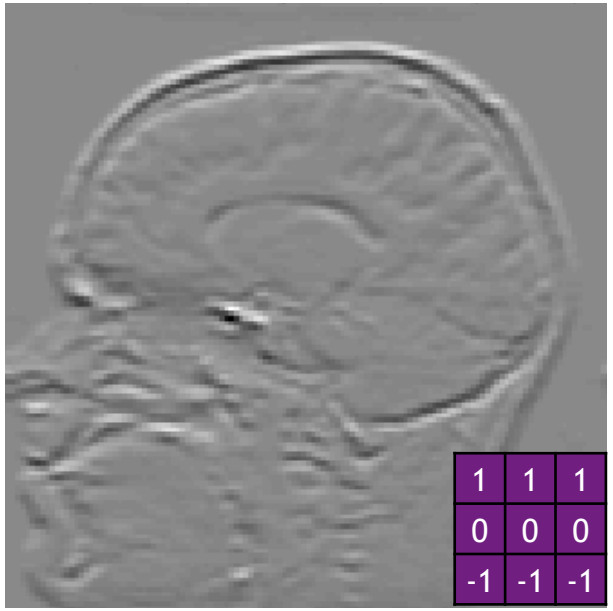
Discrete convolution

Different filters = different feature maps

Original image



Vertical edge detection



Horizontal edge detection

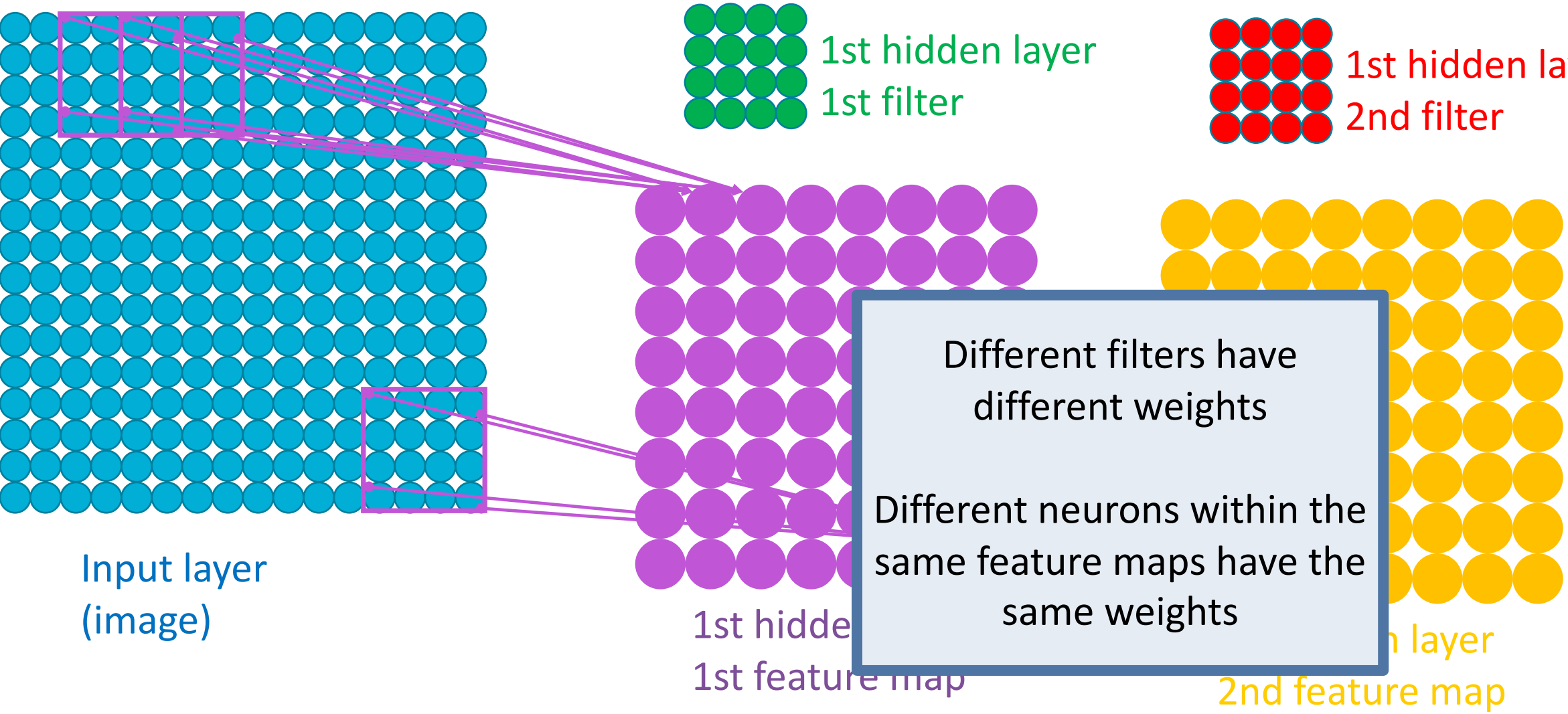
Remember : the filters are learned, not predefined
(e.g. there is no predefined vertical edge detector)

Discrete convolution

Visualizing the effect of different kernels:

https://setosa.io/ev/image-kernels/

Multiple filters (and multiple feature maps)



Define multiple filters to detect multiple characteristics (vertical lines, horiz.lines, corners…)

Padding

- **The size of the feature map is less than that of the image**

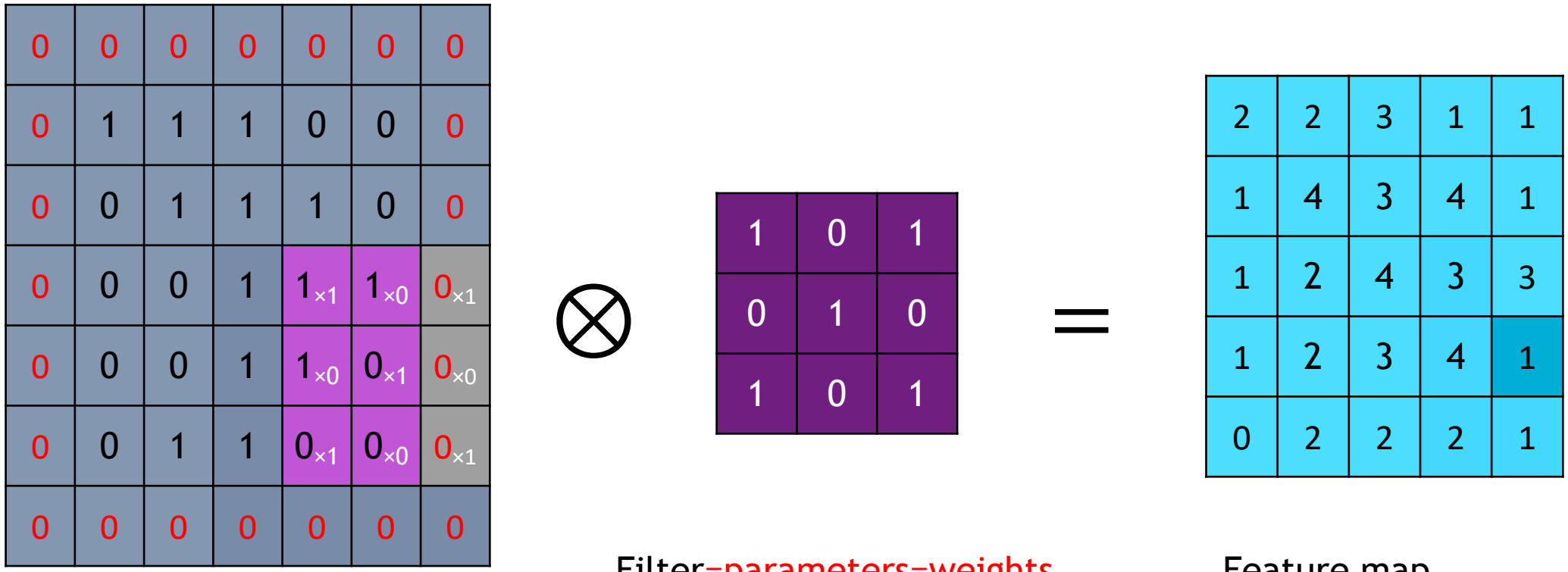
Formula (isotropic case)

$$O = I - k + 1$$

O: size of output (feature map)

I: size of input (image)

k: size of filter (along one dimension)



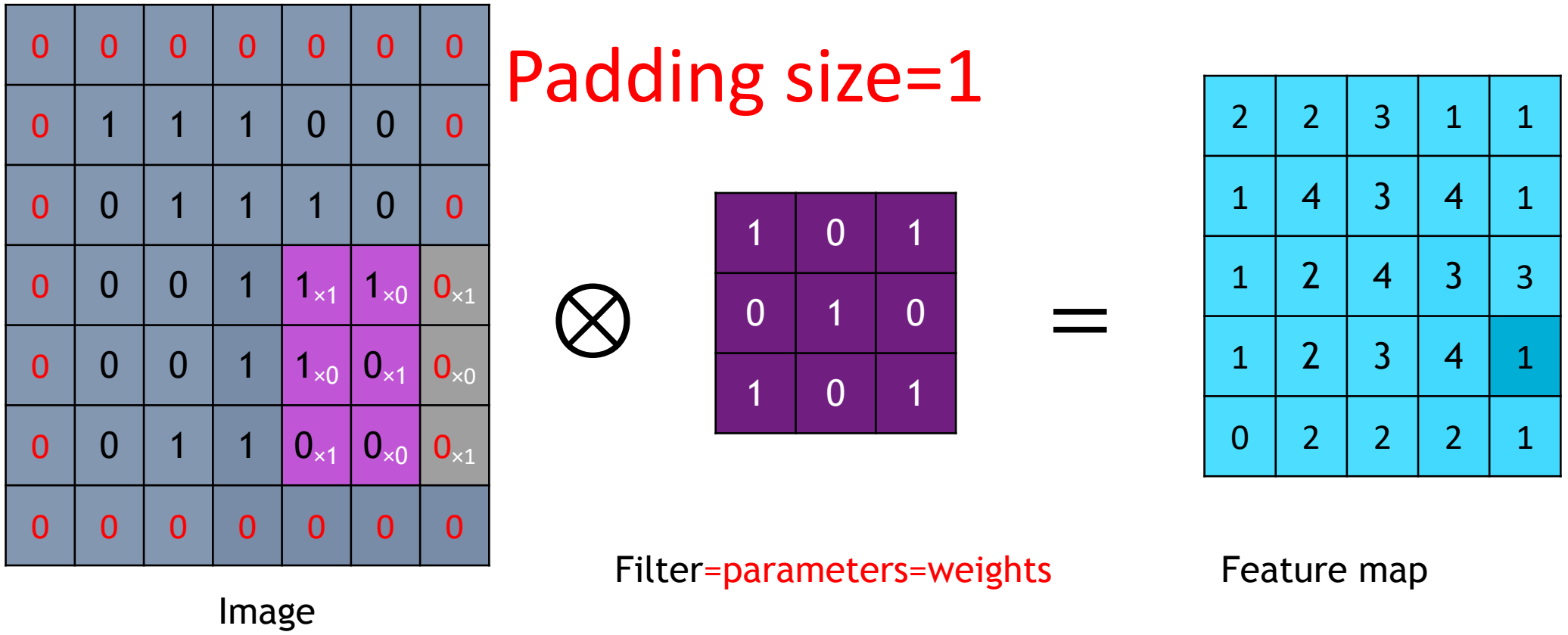
Image

Filter=parameters=weights

Feature map

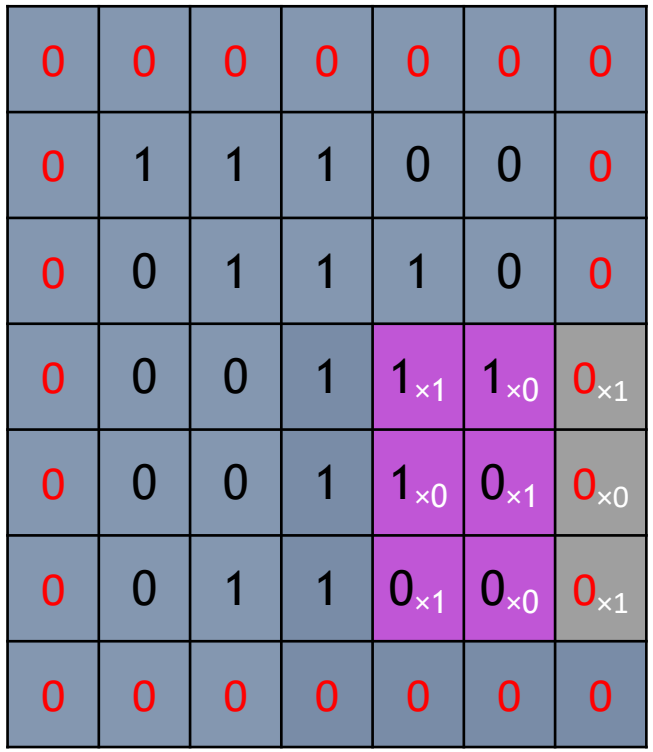
Padding

- **The size of the feature map is less than that of the image**
- **If you want to avoid that:**
  - **Use padding (add 0 on the borders)**

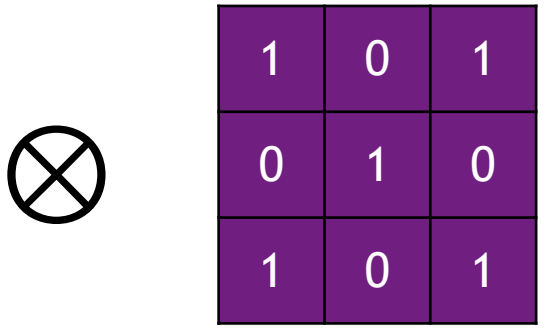


Padding

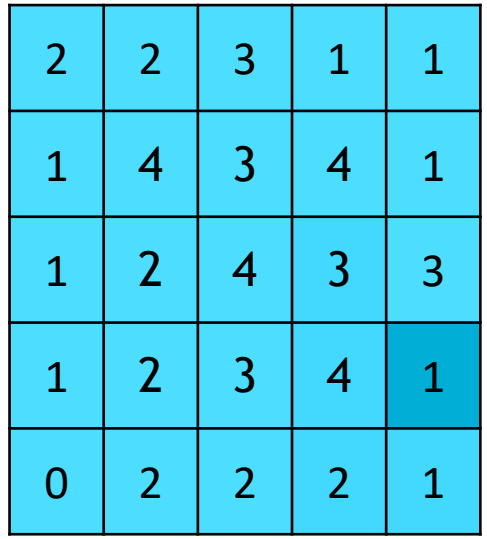
- **The size of the feature map is less than that of the image**
- **If you want to avoid that:**
  - **Use padding (add 0 on the borders)**



Image



Filter=parameters=weights



Feature map

Formula (isotropic case)

$$O = I - k + 2P + 1$$

O: size of output (feature map)

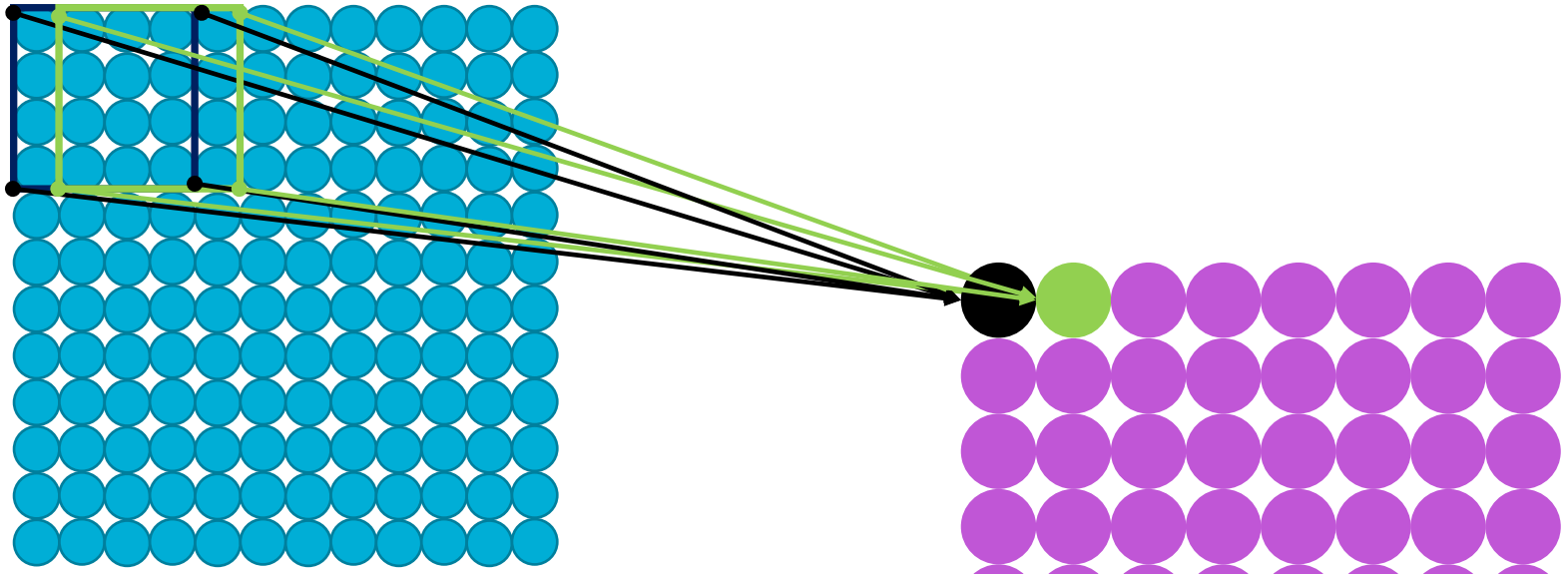
I: size of input (image)

k: size of filter (along one dimension)

P: padding size

Stride

Stride: how much the patch window is moved when moving from one neuron to its neighbour in the hidden layer



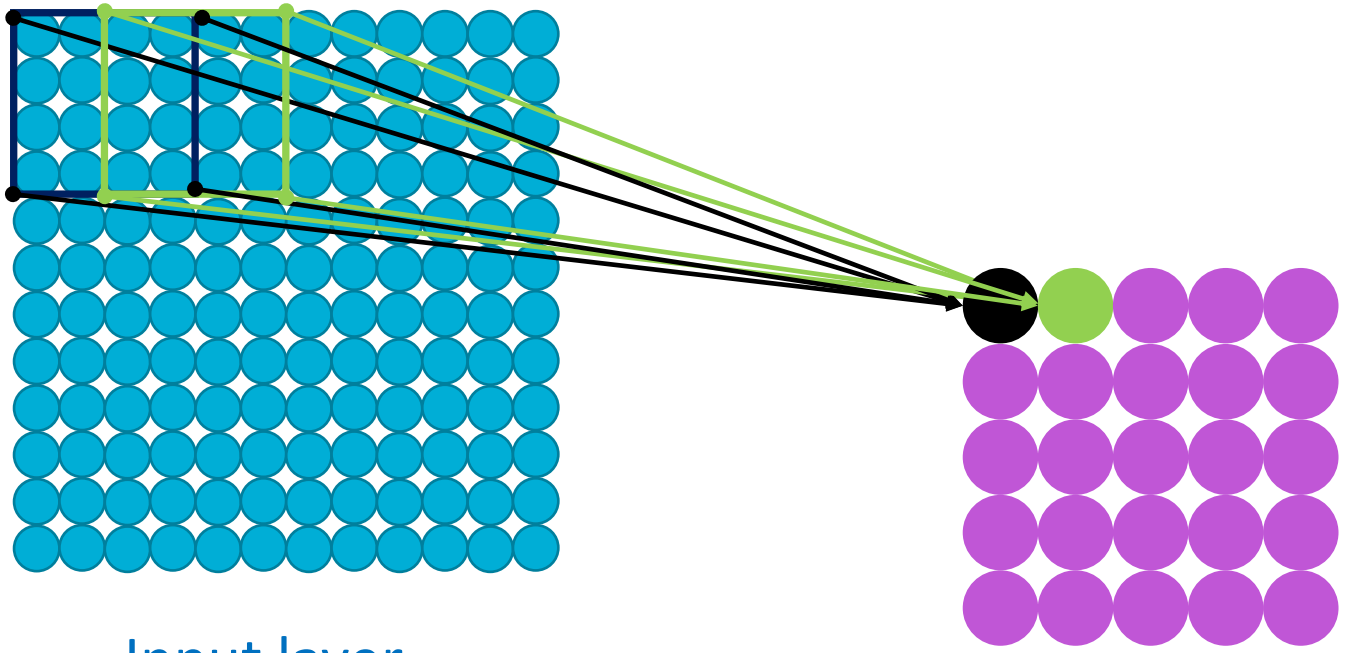
Stride=1

Input layer
(image)

1st hidden layer

Stride

Stride: how much the patch window is moved when moving from one neuron to its neighbour in the hidden layer



Input layer
(image)

1st hidden layer

Stride=2

Formula (isotropic case)

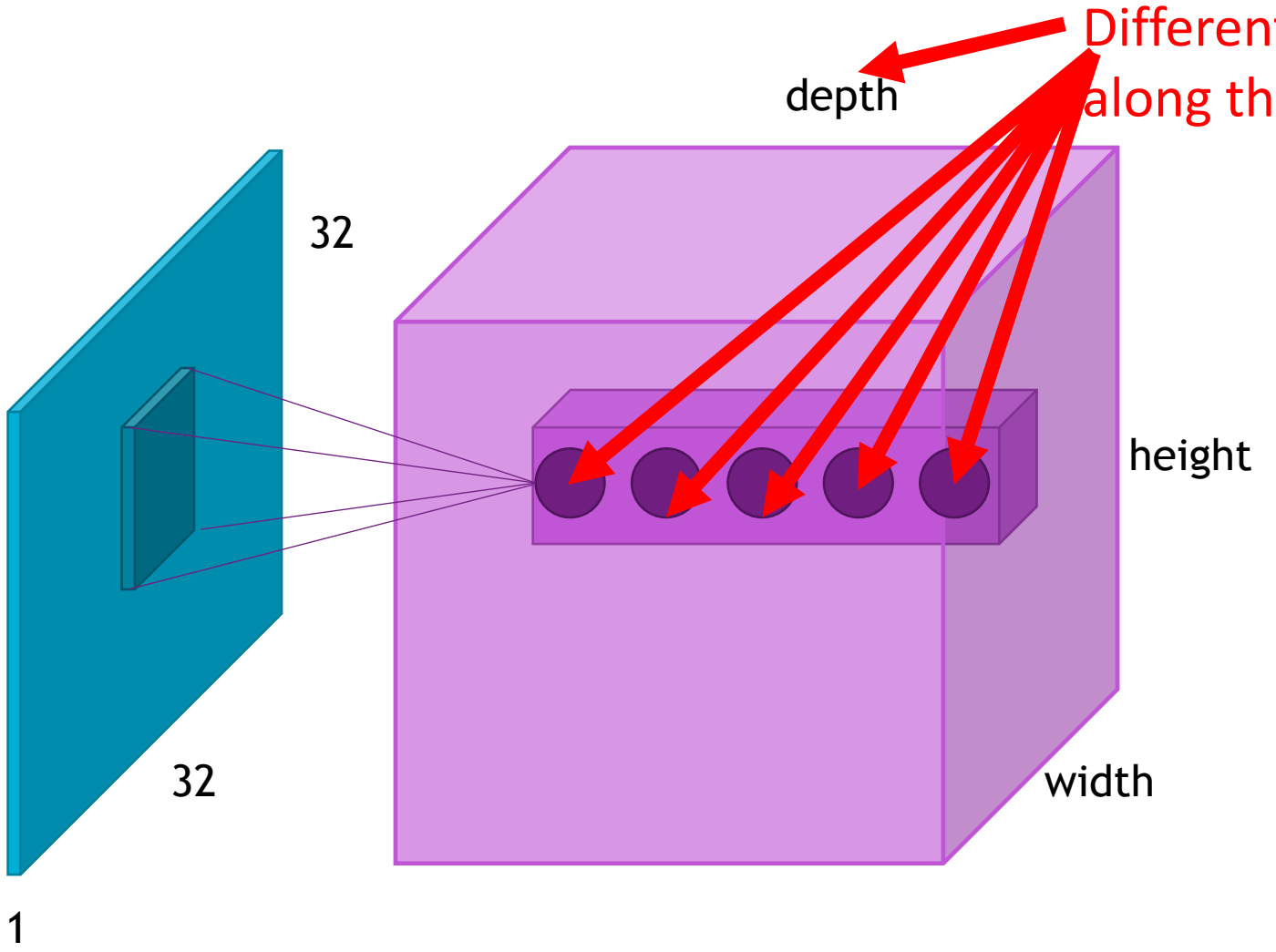
$$O = \frac{I - k + 2P}{S} + 1$$

O: size of output (feature map)
I: size of input (image)
k: size of filter (along one dimension)
P: padding size
S: stride

Multiple filters (and multiple feature maps)

Spatial arrangement of output volume

nn.Conv2d



Layer Dimensions:

$d \times w \times h$

d = n_channels = number of filters
(also called number of channels)

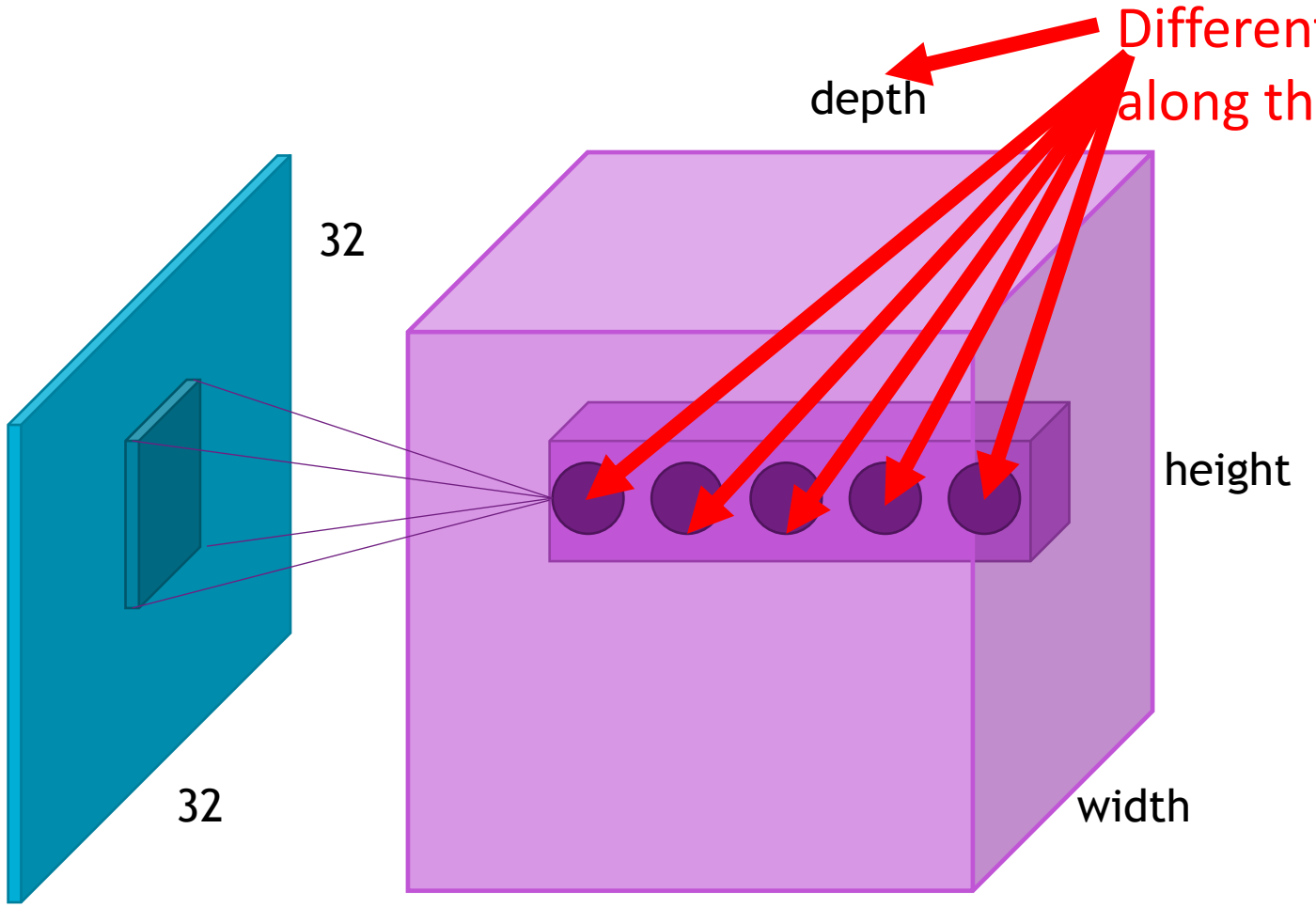
h & w = spatial dimensions defined
= O in isotropic case

$$O = \frac{I - k + 2P}{S} + 1$$

Multiple filters (and multiple feature maps)

Spatial arrangement of output volume

nn.Conv2d



Layer Dimensions:

$d \times w \times h$

d = n_channels = number of filters
(also called number of channels)

h & w = spatial dimensions defined
= O in isotropic case

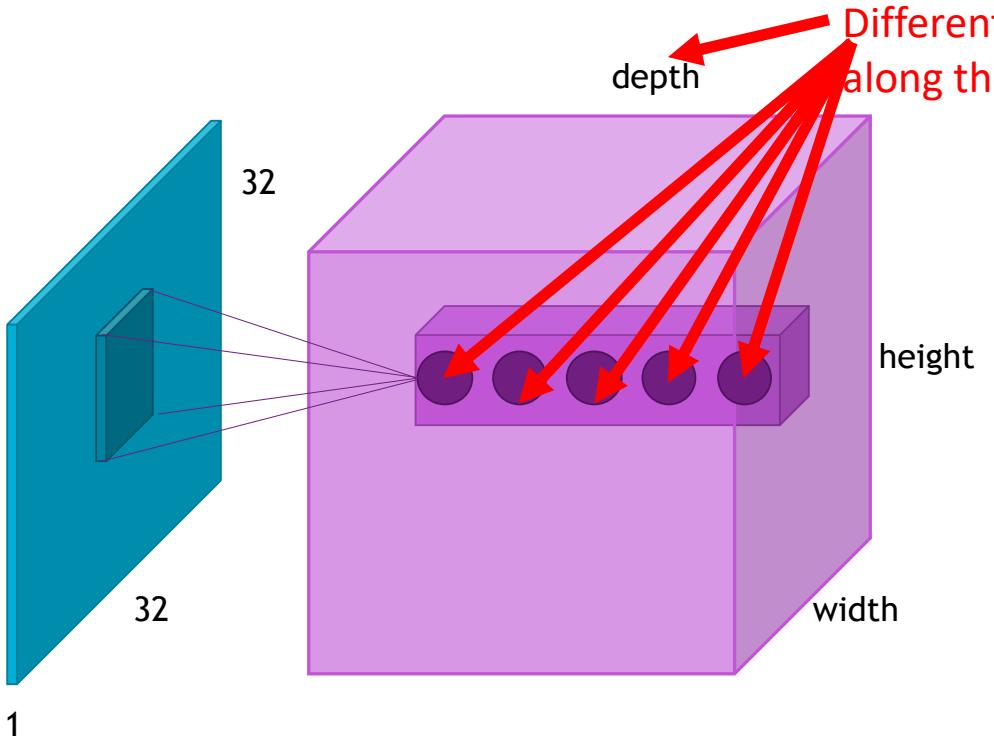
$$O = \frac{I - k + 2P}{S} + 1$$

Do I really need to compute all this? Not really, pyTorch does it for you except for the fully-connected layers (see after). Tip: execute, read the error message which gives you the results

Multiple filters (and multiple feature maps)

Spatial arrangement of output volume

nn.Conv2d



Layer Dimensions:

$d \times w \times h$

d = number of filters (also called number of channels)

h & w = spatial dimensions defined

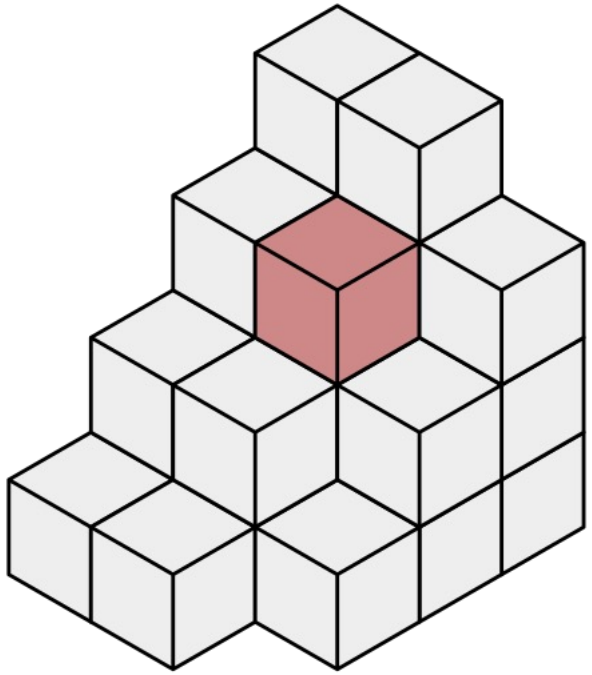
$$O = \frac{I - k + 2P}{S} + 1$$

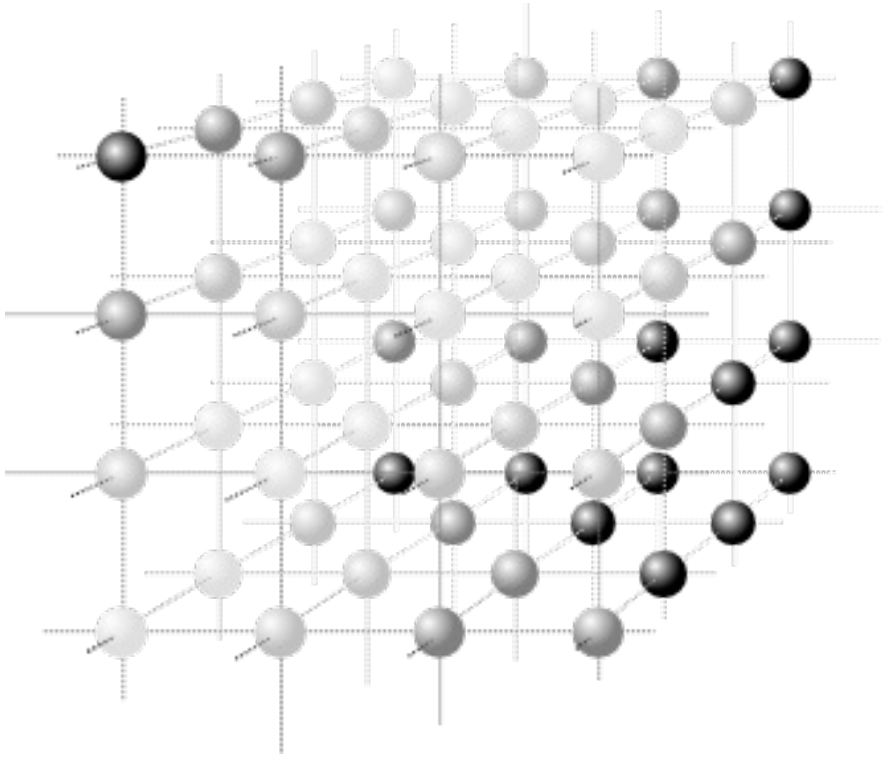
So each layer outputs a **3D array**

Actually **it is even a 4D array** because one additional dimension will be used to store the outputs computed for the different samples of the mini-batch

What about 3D images?

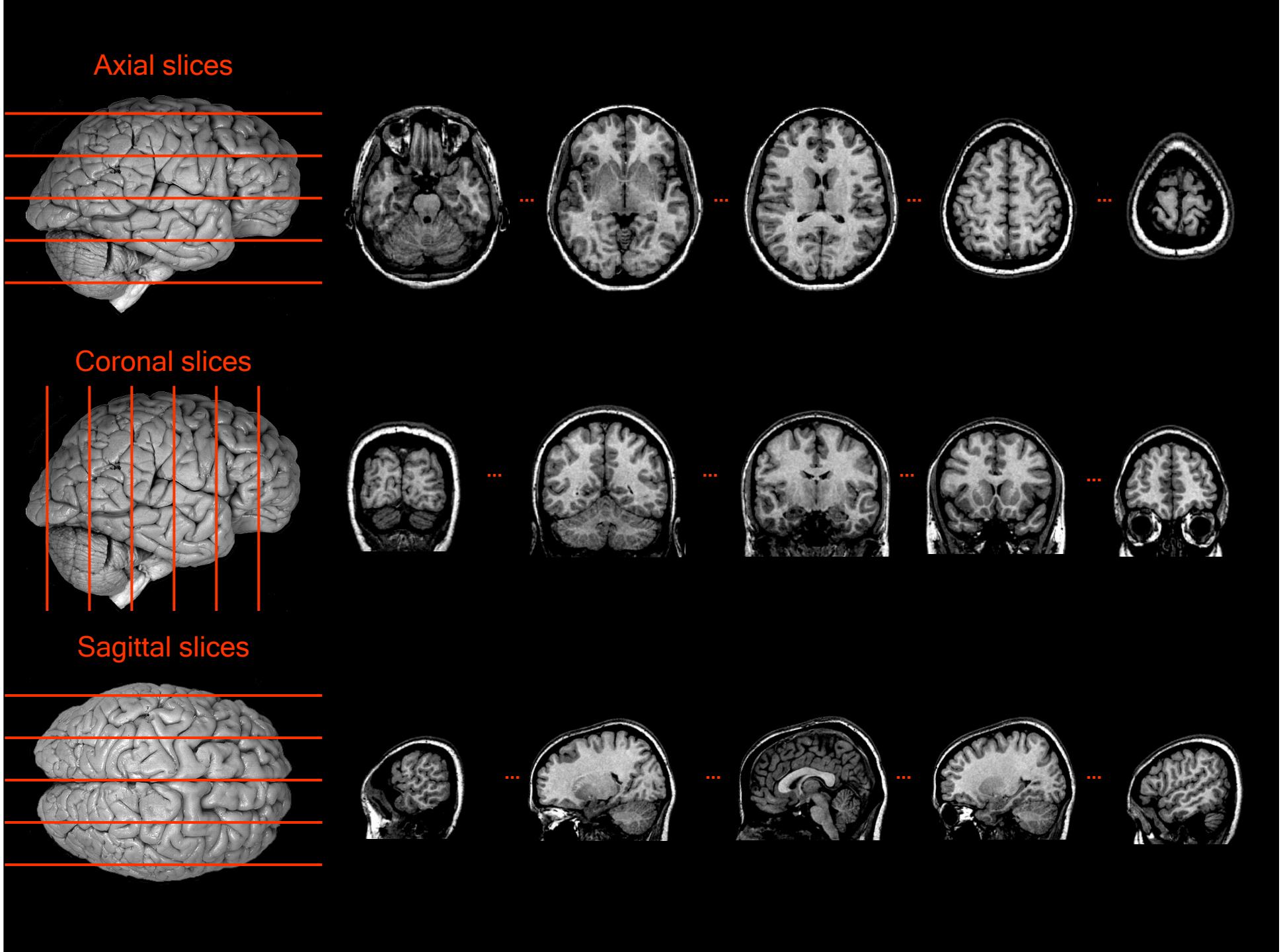
- Remember: many medical images are 3D



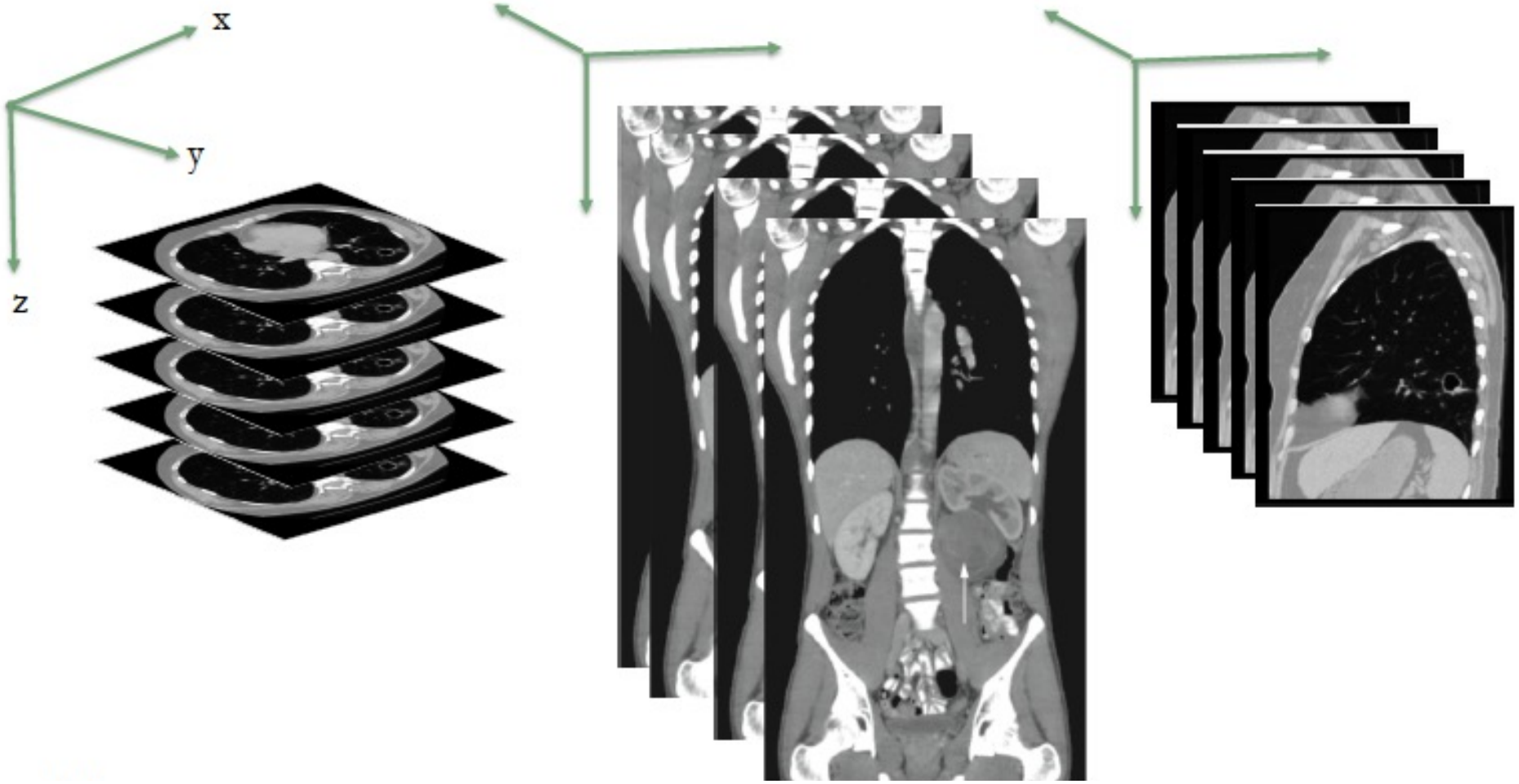


Voxel=volume element

Image source: https://en.wikipedia.org/wiki/Voxel


Axial slices
Coronal slices
Sagittal slices

What about 3D images?



I(x,y,z) denotes intensity value at voxel location (x,y,z)

Image source: http://www.cs.ucf.edu/~bagci/teaching/mic17.html

What about 3D images?

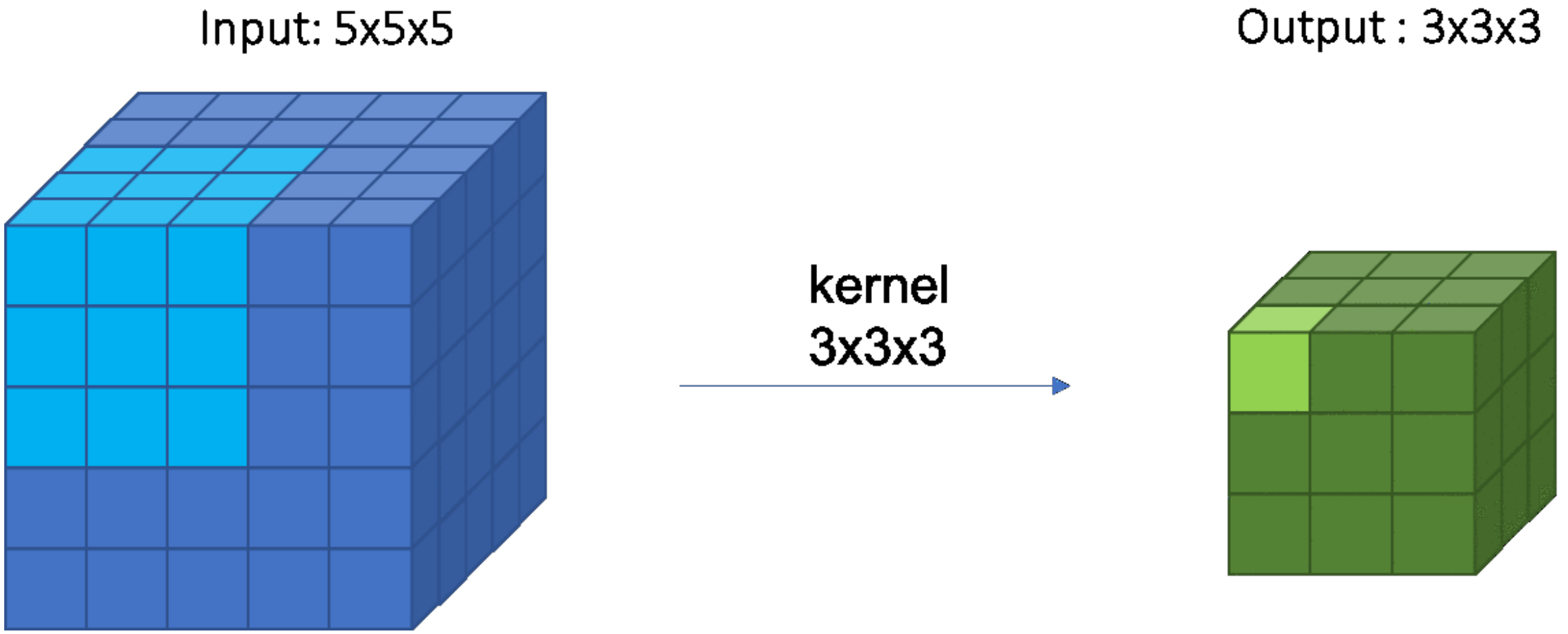
- Possible solution
  - Process 2D slices independently
  - Fuse the decisions for all the 2D slices to get a final decision for the patient

- Can be a solution in some case, but has drawbacks:
  - 3D spatial information is lost
  - Fusion is partly arbitrary

3D convolutions

nn.Conv3d

Nothing really different from the 2D case

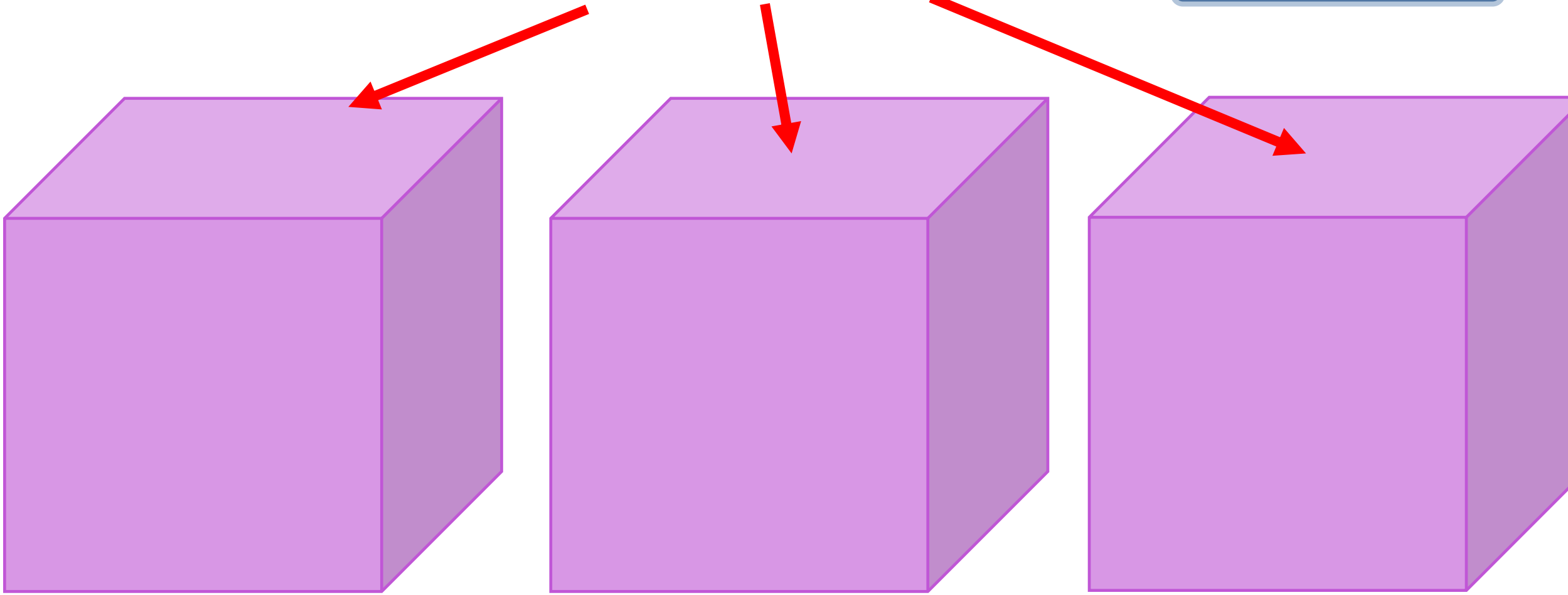


3D features maps

4D array

Each feature map is a 3D array

nn.Conv3d



So each layer outputs a **4D array**

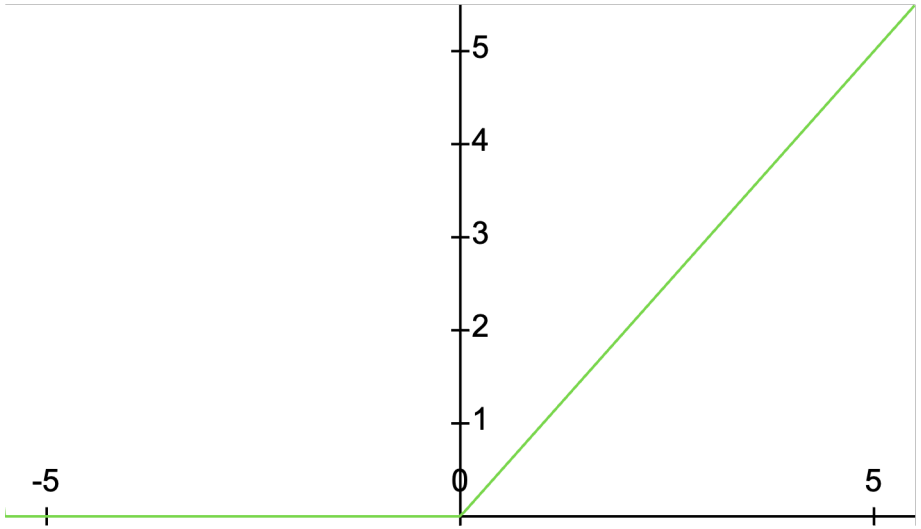
Actually **it is even a 5D array** (one additional dimension for the samples)

Non-linearity

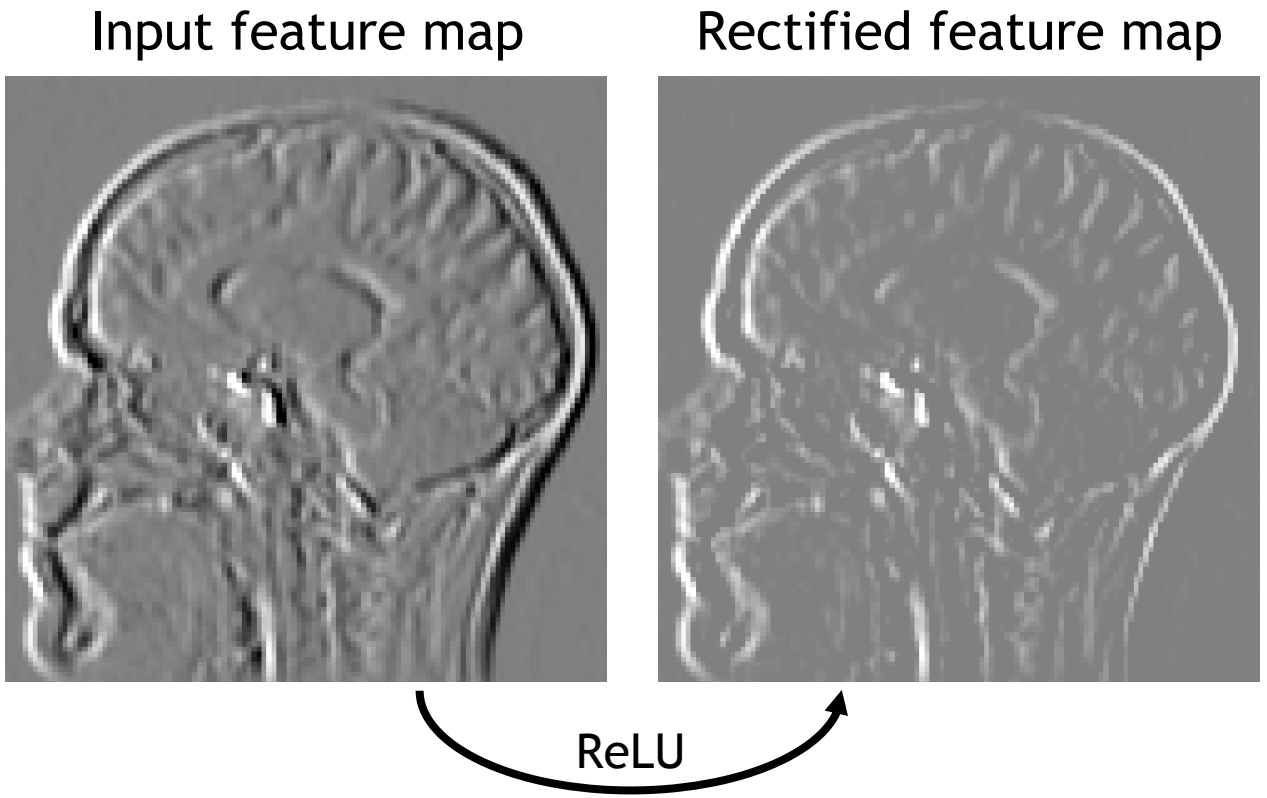
nn.ReLU

Introducing non-linearity

Rectified linear unit (ReLU)



$$g(z) = \max(0, z)$$



Black: negative values - White: positive values

Pooling

Pooling

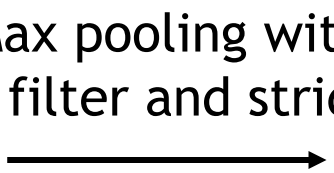
nn.MaxPool2d

nn.MaxPool3d

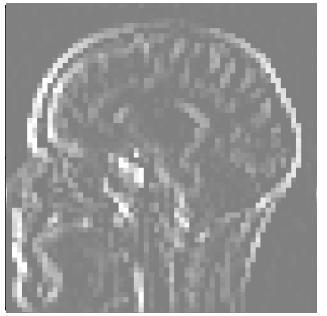
- **Reduce dimensionality while preserving spatial invariance**

Input feature map

Max pooling with
2×2 filter and stride 2

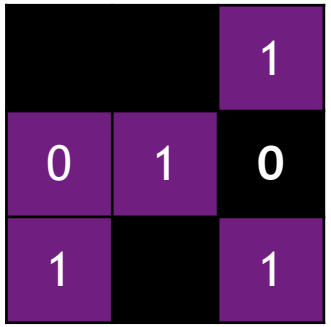


Pooled feature map



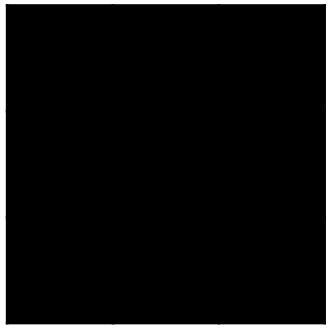
Spatial dropout

- **Dropping out some weights of a given filter would not make sense**



Filter=parameters=weights

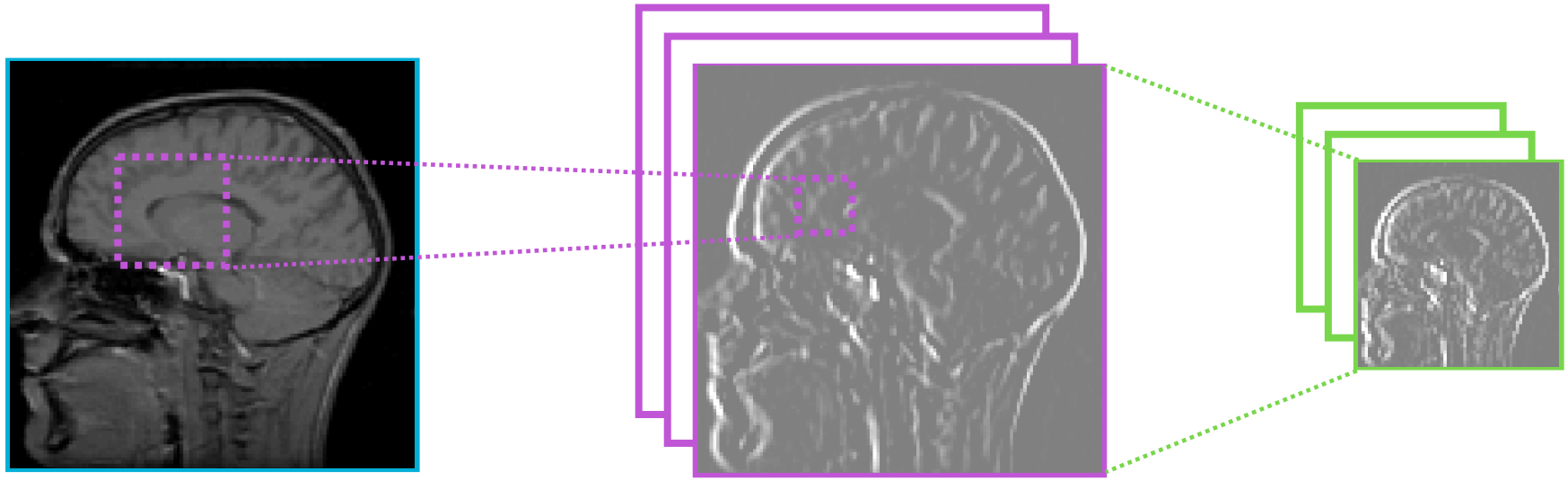
- **Instead, drop out an entire filter**



nn.Dropout2d

nn.Dropout3d

Summary



Input image

Feature maps

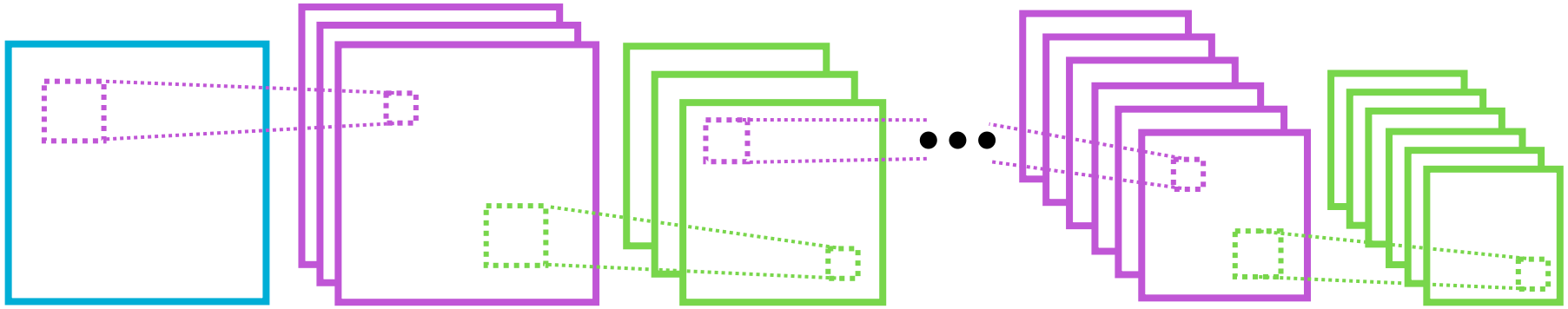
Feature maps

Convolutions + Non-linearity

Pooling

Deep CNN

Operations are repeated to produce a deep network



Input image

Convolution
+
Non-linearity

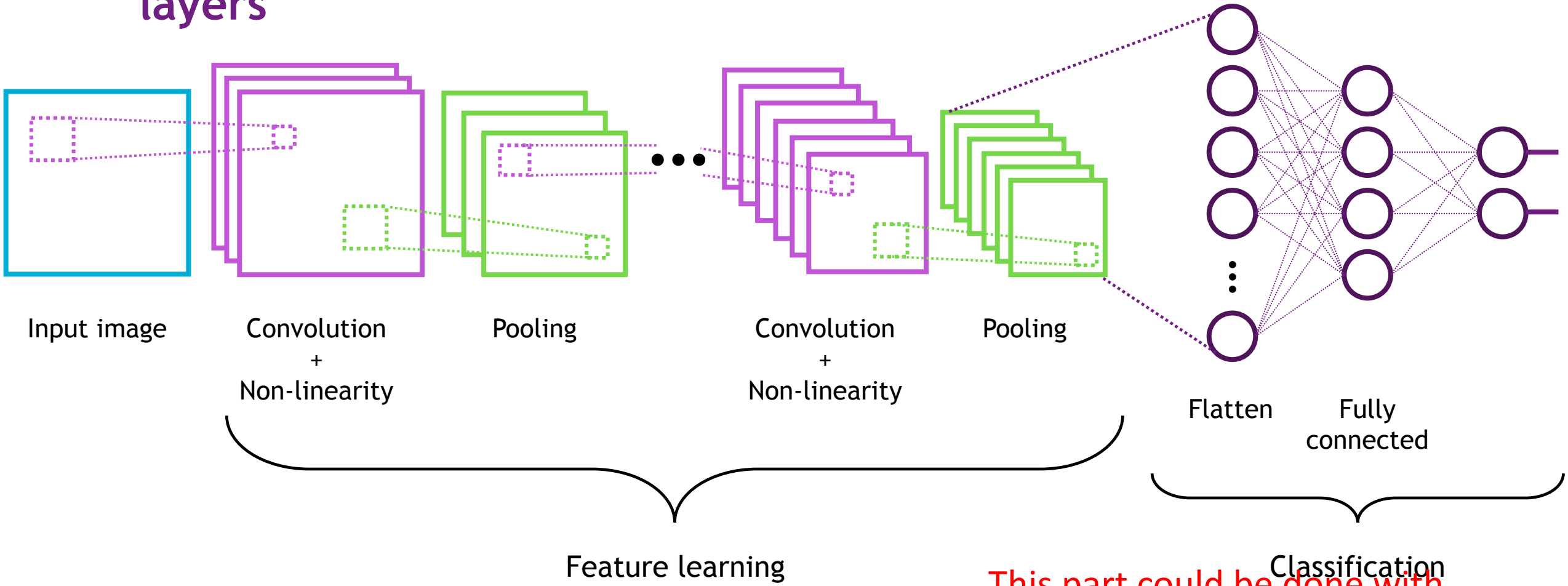
Pooling

Convolution
+
Non-linearity

Pooling

CNN for classification

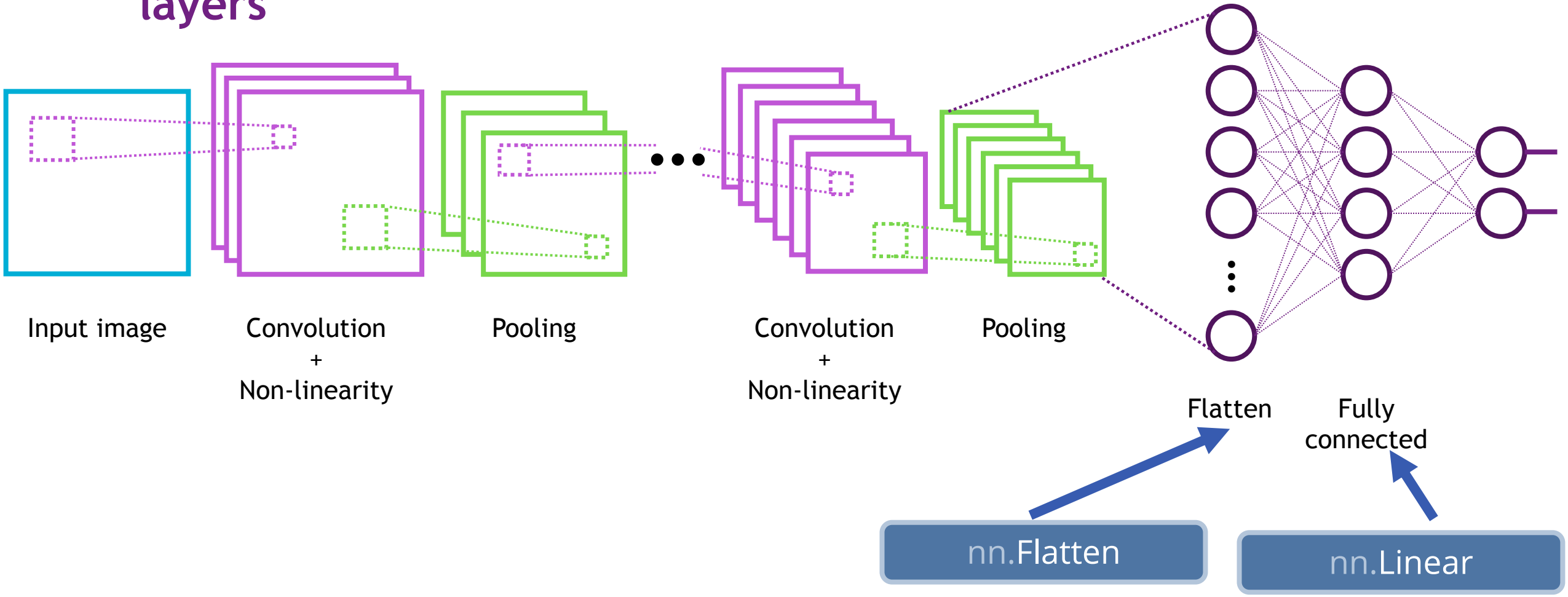
Flatten the final feature maps and add fully connected layers



This part could be done with another classifier (e.g. SVM…)

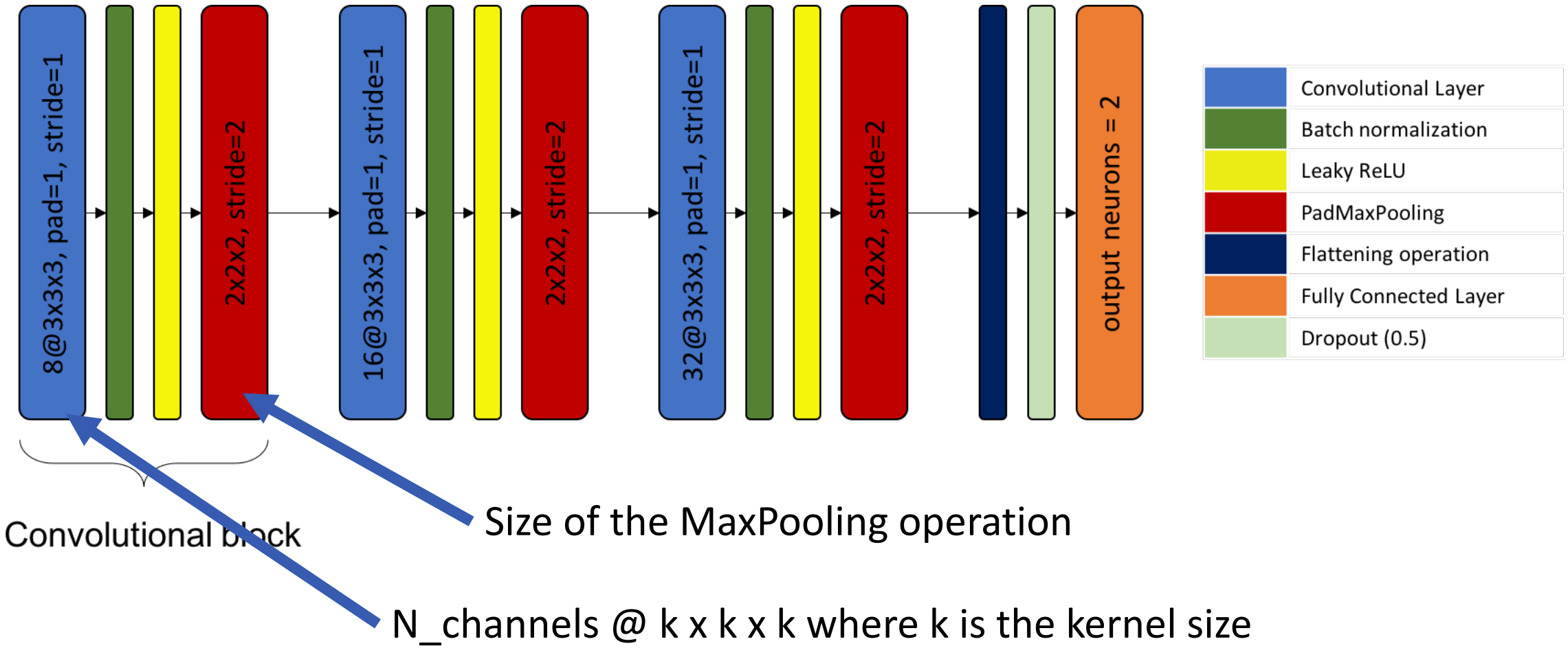
CNN for classification

Flatten the final feature maps and add fully connected layers



Reading CNN diagrams

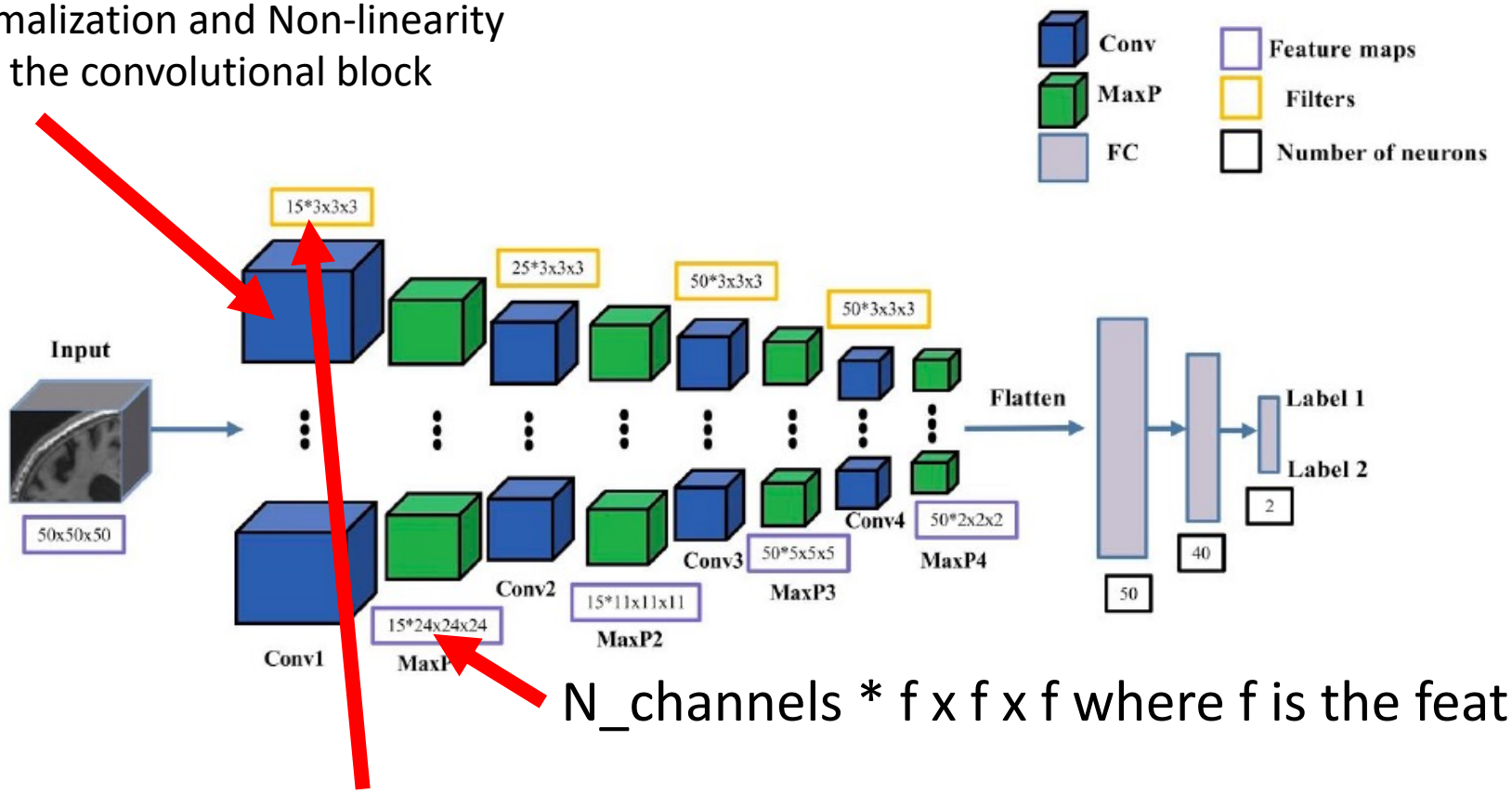
Unfortunately there is no unified notation



Reading CNN diagrams

Unfortunately there is no unified notation

Batch normalization and Non-linearity
are within the convolutional block



N_channels * f x f x f where f is the feature map size

N_channels * k x k x k where k is the kernel size

Part 2 – Classification (and regression)
2.4 Examples of applications with CNNs

Classification of chest radiographs

RESEARCH ARTICLE

Deep learning for chest radiograph diagnosis: A retrospective comparison of the CheXNeXt algorithm to practicing radiologists

Pranav Rajpurkar[1‡*], Jeremy Irvin[1‡], Robyn L. Ball[2], Kaylie Zhu[1], Brandon Yang[1], Hershel Mehta[1], Tony Duan[1], Daisy Ding[1], Aarti Bagul[1], Curtis P. Langlotz[3], Bhavik N. Patel[3], Kristen W. Yeom[3], Katie Shpanskaya[3], Francis G. Blankenberg[3], Jayne Seekins[3], Timothy J. Amrhein[4], David A. Mong[5], Safwan S. Halabi[3], Evan J. Zucker[3], Andrew Y. Ng[1☯], Matthew P. Lungren[3☯]

PLOS MEDICINE

Citation: Rajpurkar P, Irvin J, Ball RL, Zhu K, Yang B, Mehta H, et al. (2018) Deep learning for chest radiograph diagnosis: A retrospective comparison of the CheXNeXt algorithm to practicing radiologists. PLoS Med 15(11): e1002686. https://doi.org/10.1371/journal.pmed.1002686

Classification of chest radiographs

Data

The ChestX-ray14 dataset [14] was used to develop the deep learning algorithm. The dataset is currently the largest public repository of radiographs, containing 112,120 frontal-view (both posteroanterior and anteroposterior) chest radiographs of 30,805 unique patients. Each image in ChestX-ray14 was annotated with up to 14 different thoracic pathology labels that were chosen based on frequency of observation and diagnosis in clinical practice. The labels for each image were obtained using automatic extraction methods on radiology reports, resulting in 14 binary values per image, where 0 indicates the absence of that pathology and 1 denotes the presence (multiple pathologies can be present in each image). We partitioned the dataset into training, tuning, and validation (see S1 Table for statistics of dataset splits used in this study).

The training set was used to optimize network parameters, the tuning set was used to compare and choose networks, and the validation set was used to evaluate CheXNeXt and radiologists. There is no patient overlap among the partitions.

Classification of chest radiographs

Radiologist annotations

A validation set of 420 frontal-view chest radiographs was selected from ChestX-ray14 for radiologist annotation. The set was curated to contain at least 50 cases of each pathology according to the original labels provided in the dataset by randomly sampling examples and iteratively updating the selected examples by sampling from the examples labeled with the underrepresented pathologies. The radiographs in the validation set were annotated by 3 independent board-certified cardiothoracic specialist radiologists (average experience 15 years, range 5–28 years) for the presence of each of the 14 pathologies. The majority vote of their annotations was taken as a consensus reference standard on each image. To compare to the algorithm, 6 board-certified radiologists from 3 academic institutions (average experience 12 years, range 4–28 years) and 3 senior radiology residents also annotated the validation set of 420 radiographs for all 14 labels. All radiologists individually reviewed and labeled each of the images using a freely available image viewer with capabilities for picture archiving and communication system features such as zoom, window leveling, and contrast adjustment. Radiologists did not have access to any patient information or knowledge of disease prevalence in the data. Labels were entered into a standardized data entry program, and the total time to complete the review was recorded. The Stanford International Review Board (IRB) approved this study, and all radiologists consented to participate in the labeling process.

Classification of chest radiographs

Algorithm development

The deep learning algorithm, called CheXNeXt, is a neural network trained to concurrently detect the 14 pathologies in frontal-view chest radiographs. Neural networks are functions with many parameters that are structured as a hierarchy of layers to model different levels of abstraction. In this study, the selected architecture was a convolutional neural network, a particular type of neural network that is specially designed to handle image data. By exploiting a parameter sharing receptive field, convolutional neural networks scan over an image to learn features from local structure and aggregate the local features to make a prediction on the full image. The neural network used in this study is a 121-layer DenseNet architecture [15] in which each layer is directly connected to every other layer within a block. For each layer, the feature maps of all preceding layers are used as inputs, and its own feature maps are passed on to all following layers as inputs.

Once specifying the neural network architecture, the parameters are automatically learned from a large amount of data labeled with the presence or absence of each pathology. The learning process consists of iteratively updating the parameters to decrease the prediction error, which is computed by comparing the network's prediction to the known annotations on each image. By performing this procedure using a representative set of images, the resulting network can make predictions on previously unseen frontal-view chest radiographs.

Classification of chest radiographs

Table 1. Radiologists and algorithm AUC with CIs.

Pathology	Radiologists (95% CI)	Algorithm (95% CI)	Algorithm – Radiologists Difference (99.6% CI)[a]	Advantage
Atelectasis	0.808 (0.777 to 0.838)	0.862 (0.825 to 0.895)	0.053 (0.003 to 0.101)	Algorithm
Cardiomegaly	0.888 (0.863 to 0.910)	0.831 (0.790 to 0.870)	–0.057 (–0.113 to –0.007)	Radiologists
Consolidation	0.841 (0.815 to 0.870)	0.893 (0.859 to 0.924)	0.052 (–0.001 to 0.101)	No difference
Edema	0.910 (0.886 to 0.930)	0.924 (0.886 to 0.955)	0.015 (–0.038 to 0.060)	No difference
Effusion	0.900 (0.876 to 0.921)	0.901 (0.868 to 0.930)	0.000 (–0.042 to 0.040)	No difference
Emphysema	0.911 (0.866 to 0.947)	0.704 (0.567 to 0.833)	–0.208 (–0.508 to –0.003)	Radiologists
Fibrosis	0.897 (0.840 to 0.936)	0.806 (0.719 to 0.884)	–0.091 (–0.198 to 0.016)	No difference
Hernia	0.985 (0.974 to 0.991)	0.851 (0.785 to 0.909)	–0.133 (–0.236 to –0.055)	Radiologists
Infiltration	0.734 (0.688 to 0.779)	0.721 (0.651 to 0.786)	–0.013 (–0.107 to 0.067)	No difference
Mass	0.886 (0.856 to 0.913)	0.909 (0.864 to 0.948)	0.024 (–0.041 to 0.080)	No difference
Nodule	0.899 (0.869 to 0.924)	0.894 (0.853 to 0.930)	–0.005 (–0.058 to 0.044)	No difference
Pleural thickening	0.779 (0.740 to 0.809)	0.798 (0.744 to 0.849)	0.019 (–0.056 to 0.094)	No difference
Pneumonia	0.823 (0.779 to 0.856)	0.851 (0.781 to 0.911)	0.028 (–0.087 to 0.125)	No difference
Pneumothorax	0.940 (0.912 to 0.962)	0.944 (0.915 to 0.969)	0.004 (–0.040 to 0.051)	No difference

[a]The AUC difference was calculated as the AUC of the algorithm minus the AUC of the radiologists. To account for multiple hypothesis testing, the Bonferroni-corrected CI (1 – 0.05/14; 99.6%) around the difference was computed.

The nonparametric bootstrap was used to estimate the variability around each of the performance measures; 10,000 bootstrap replicates from the validation set were drawn, and each performance measure was calculated for the algorithm and the radiologists on these same 10,000 bootstrap replicates. This produced a distribution for each estimate, and the 95% bootstrap percentile intervals (2.5th and 97.5th percentiles) are reported.

Abbreviations: AUC, area under the receiver operating characteristic curve; CI, confidence interval.

Classification of chest radiographs

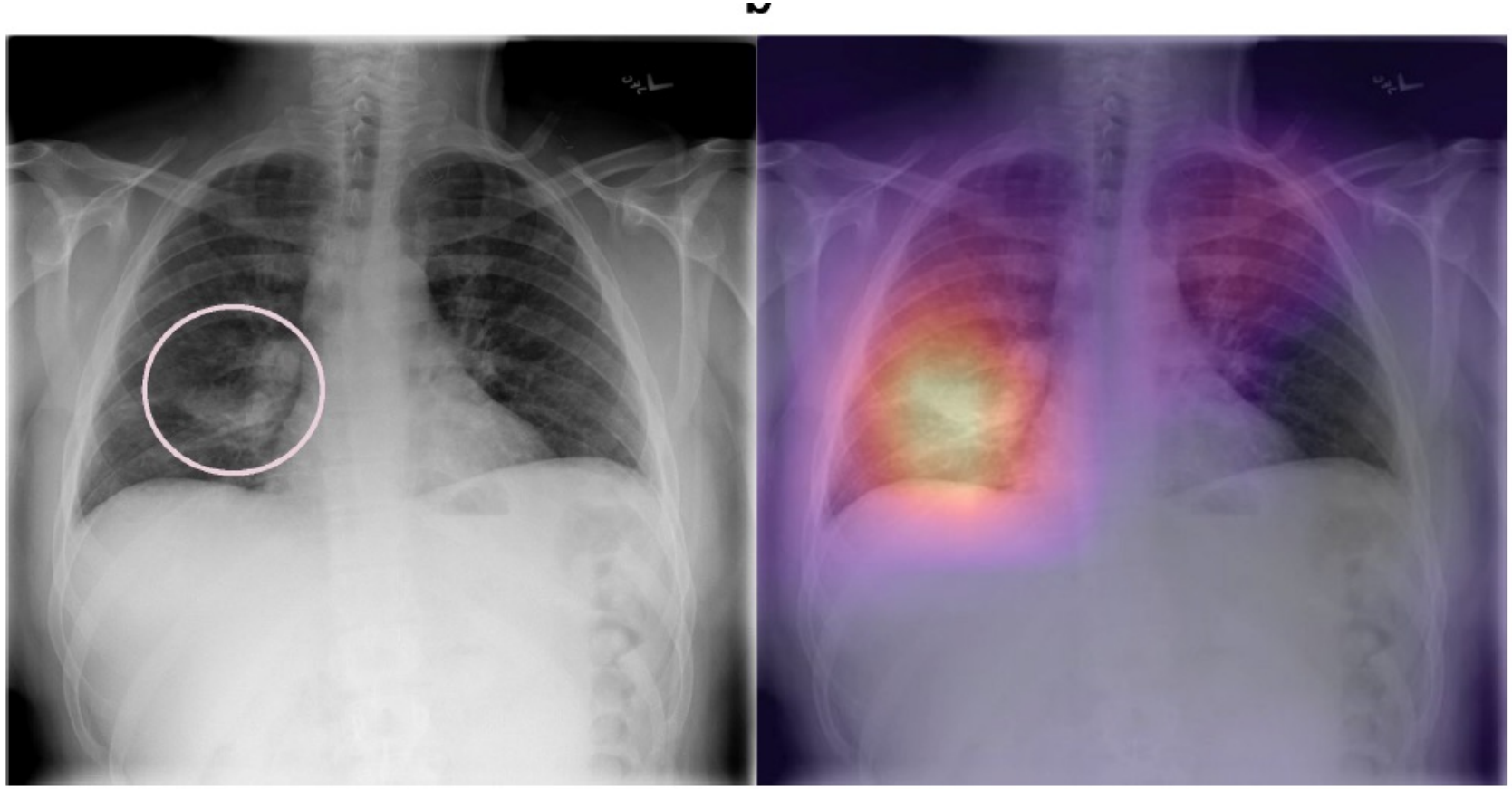
Table 1. Radiologists and algorithm AUC with CIs.

Pathology	Radiologists (95% CI)	Algorithm (95% CI)
Atelectasis	0.808 (0.777 to 0.838)	0.862 (0.825 to 0.895)
Cardiomegaly	0.888 (0.863 to 0.910)	0.831 (0.790 to 0.870)
Consolidation	0.841 (0.815 to 0.870)	0.893 (0.859 to 0.924)
Edema	0.910 (0.886 to 0.930)	0.924 (0.886 to 0.955)
Effusion	0.900 (0.876 to 0.921)	0.901 (0.868 to 0.930)
Emphysema	0.911 (0.866 to 0.947)	0.704 (0.567 to 0.833)

Classification of chest radiographs

Pathology	Advantage
Atelectasis	Algorithm
Cardiomegaly	Radiologists
Consolidation	No difference
Edema	No difference
Effusion	No difference
Emphysema	Radiologists
Fibrosis	No difference
Hernia	Radiologists
Infiltration	No difference
Mass	No difference
Nodule	No difference
Pleural thickening	No difference
Pneumonia	No difference
Pneumothorax	No difference

Classification of chest radiographs

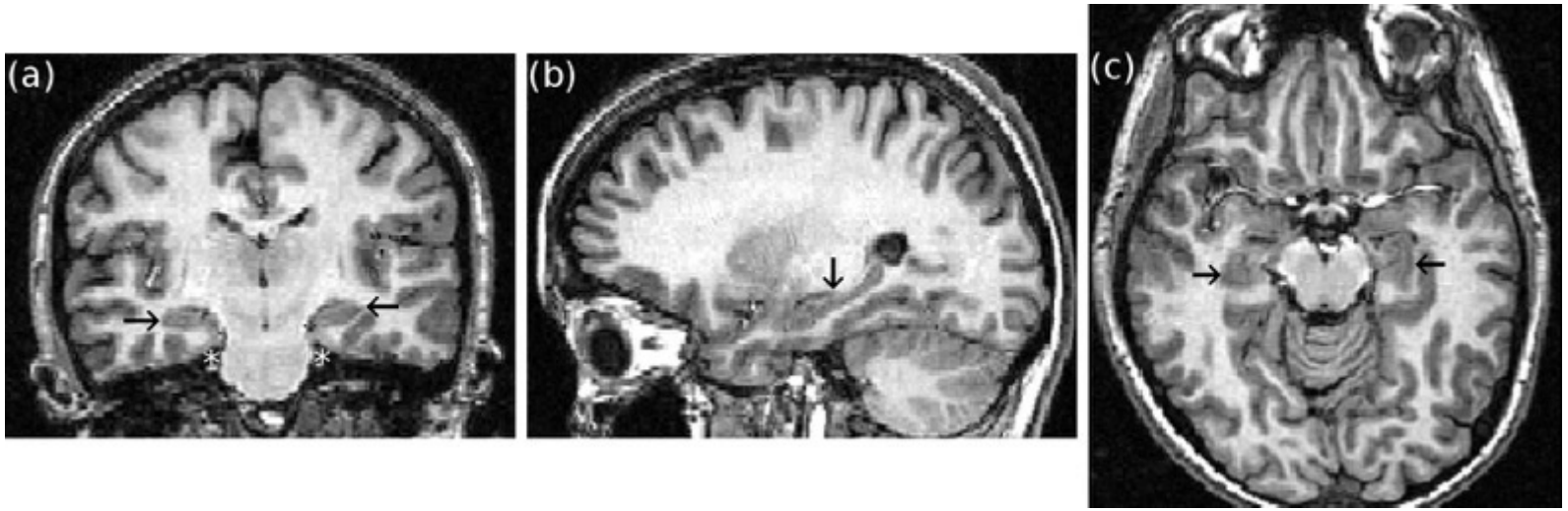


Interpreting network predictions

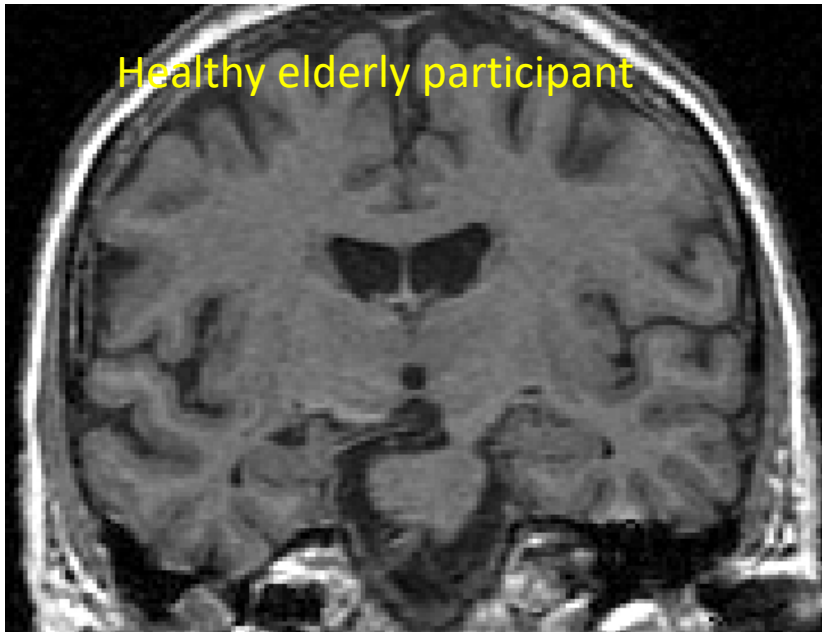
Example in brain image classification

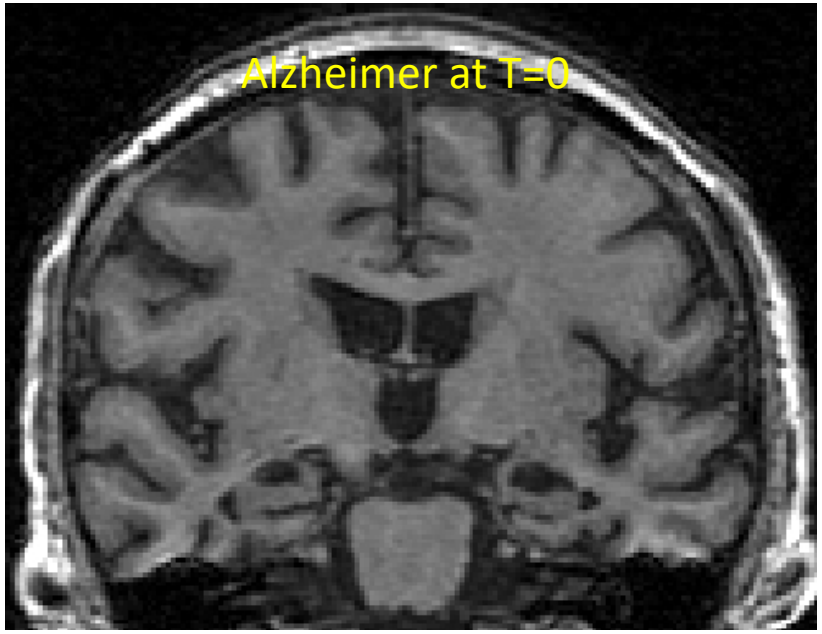
- Alzheimer's disease
  - Neurodegenerative disease
    - Loss of neurons and connections

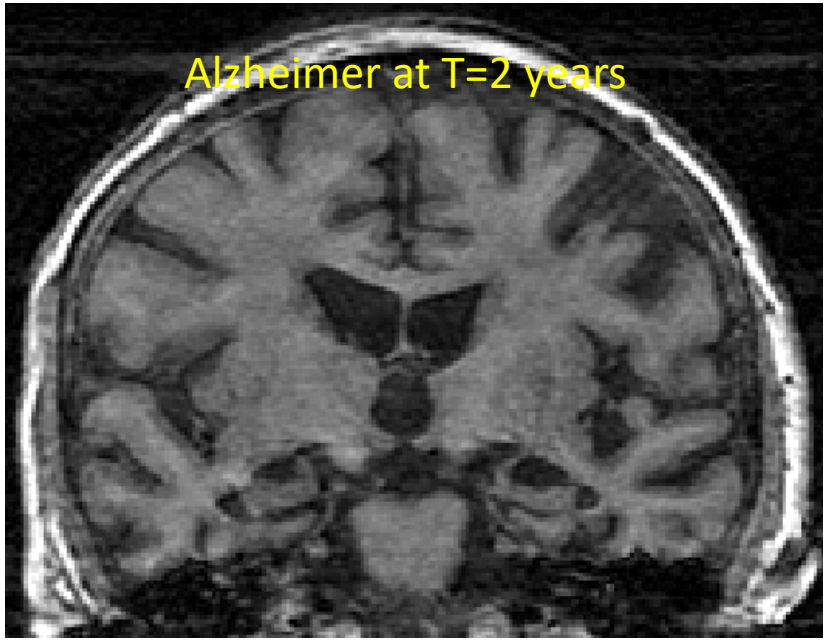
- Anatomical MRI (magnetic resonance imaging)



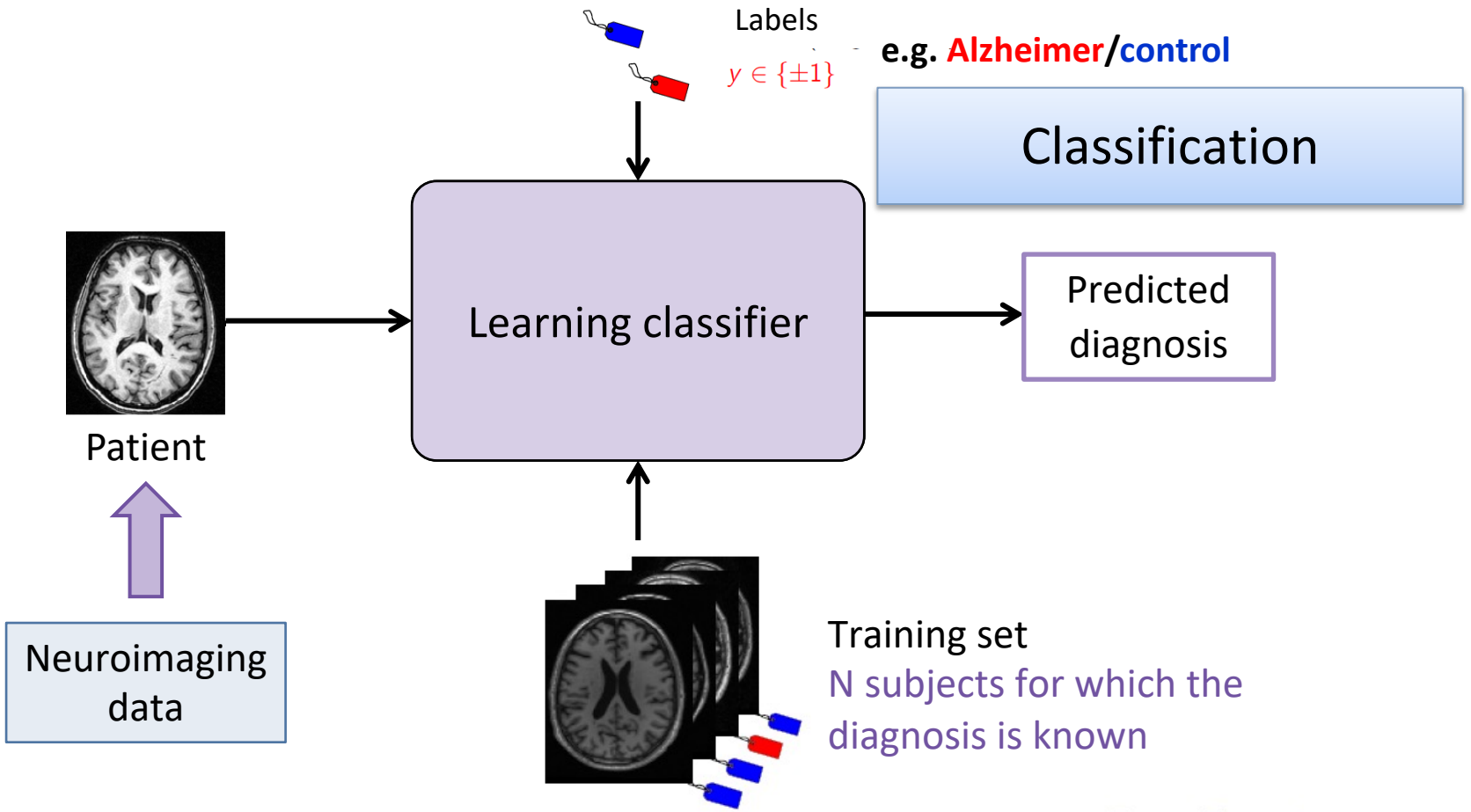
Can we detect that the patient has Alzheimer's disease from MRI?





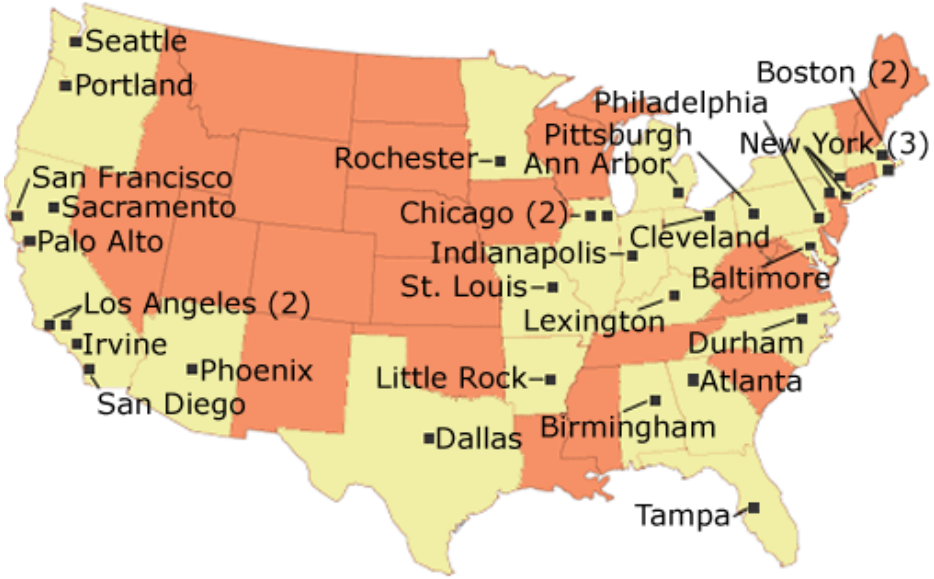


Example in brain image classification



Data

- **Data: ADNI (Alzheimer's Disease NeuroImaging Initiative)**
  - Large multicentric dataset
  - Over 1000 participants
  - Multimodal data: Brain imaging, cognitive scores, biomarkers, genetics
  - Publicly available



Data

Table 2. Summary of participant demographics, mini-mental state examination (MMSE) and global clinical dementia rating (CDR) scores at baseline for ADNI.

	Subjects	Sessions	Age	Gender	MMSE	CDR
CN	330	1 830	74. 4 ± 5.8 [59.8, 89.6]	160 M / 170 F	29.1 ± 1.1 [24, 30]	0: 330
MCI	787	3 458	73.3 ± 7.5 [54.4, 91.4]	464 M / 323 F	27.5 ± 1.8 [23, 30]	0: 2; 0.5: 785
sMCI	298	1 046	72.3 ± 7.4 [55.0, 88.4]	175 M / 123 F	28.0 ± 1.7 [23, 30]	0.5: 298
pMCI	295	865	73.8 ± 6.9 [55.1, 88.3]	176 M / 119 F	26.9 ± 1.7 [23, 30]	0.5: 293; 1: 2
AD	336	1 106	75.0 ± 7.8 [55.1, 90.9]	185 M / 151 F	23.2 ± 2.1 [18, 27]	0.5: 160; 1: 175; 2: 1

Values are presented as mean ± SD [range]. M: male, F: female

3D CNN

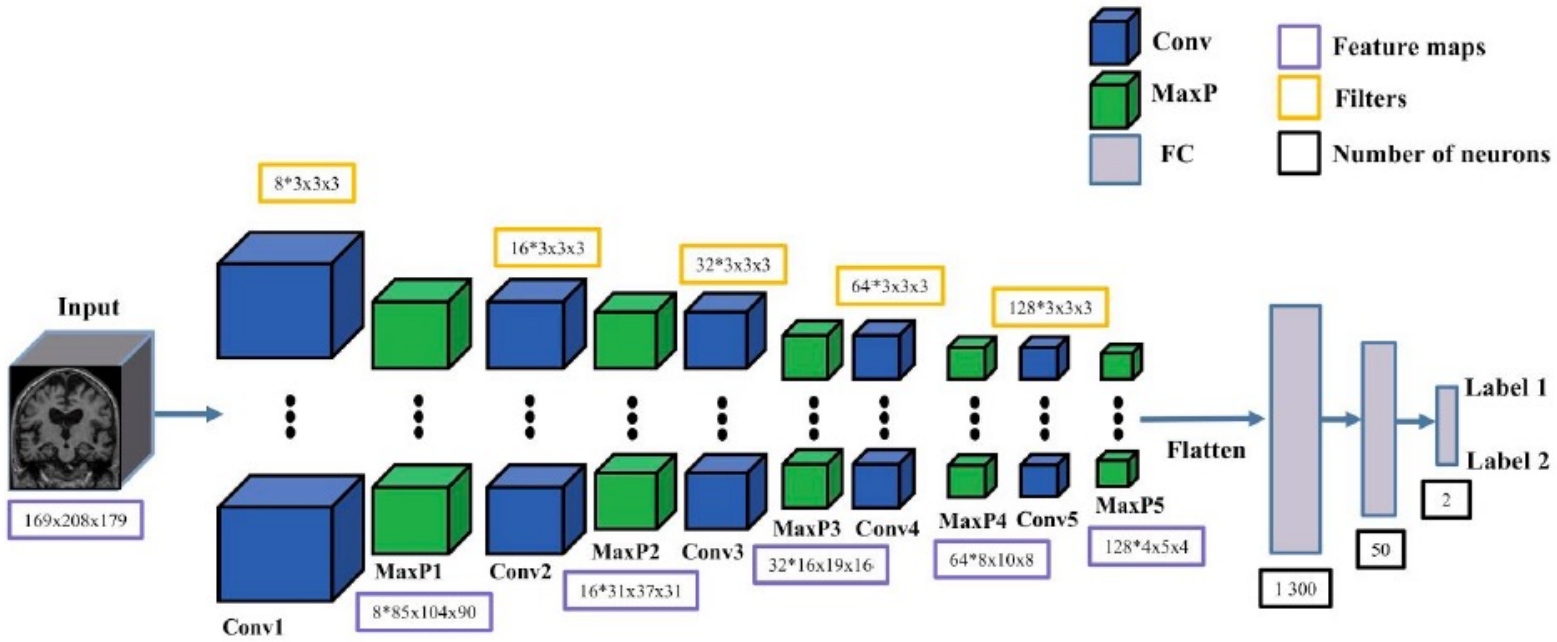


Figure 1: Architecture of the 3D subject-level CNNs. For each convolutional block, we only display the convolutional and max pooling layers. Filters for each convolutional layer represent the number of filters * filter size. Feature maps of each convolutional block represent the number of feature maps * size of each feature map. Conv: convolutional layer; MaxP: max pooling layer; FC: fully connected layer.

3D ROI-based CNN

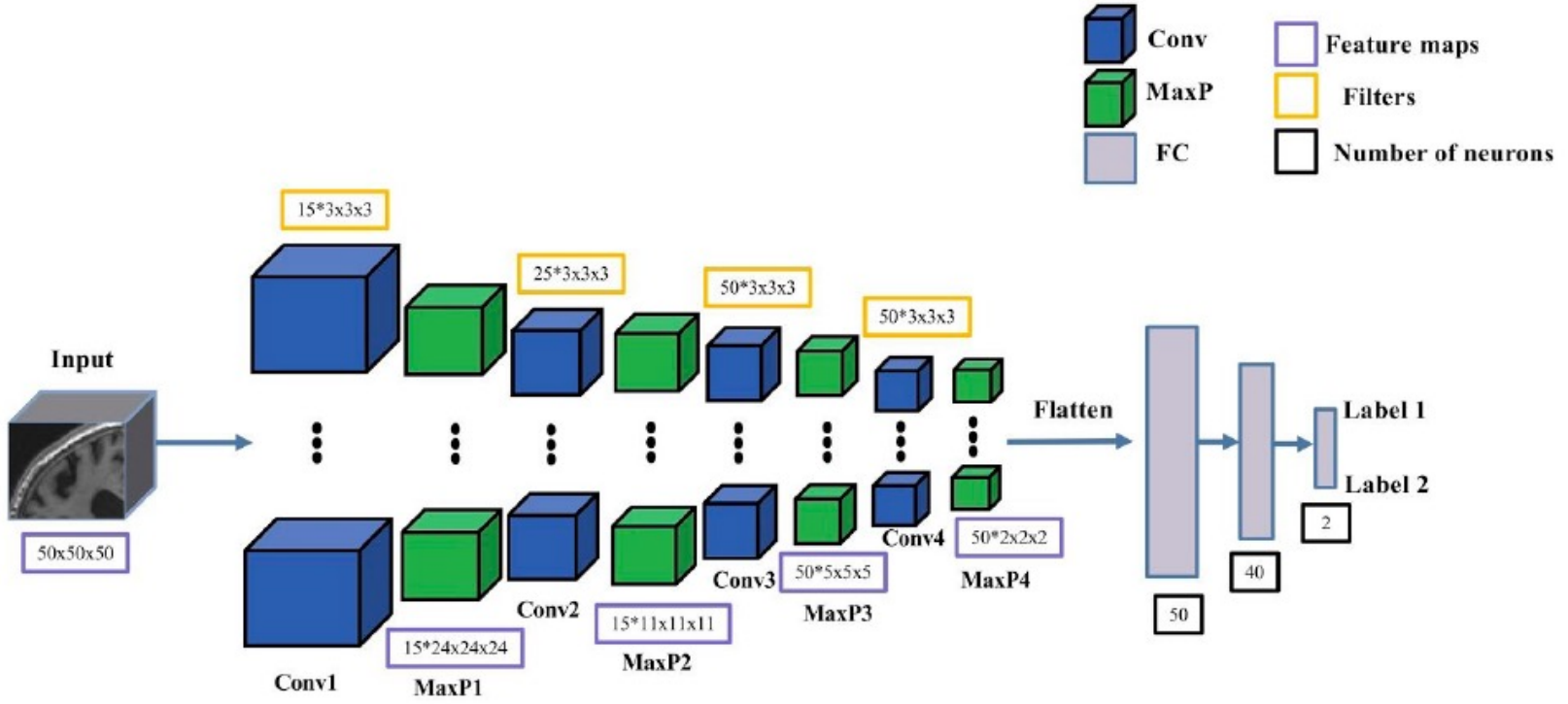


Figure 2: Architecture of the 3D ROI-based and 3D patch-level CNNs. For each convolutional block, we only display the convolutional and max pooling layers. Filters for each convolutional layer represent the number of filters * filter size. Feature maps of each convolutional block represent the number of feature maps * size of each feature map. Conv: convolutional layer; MaxP: max pooling layer; FC: fully connected layer.

2D slice-level CNN (ResNet)

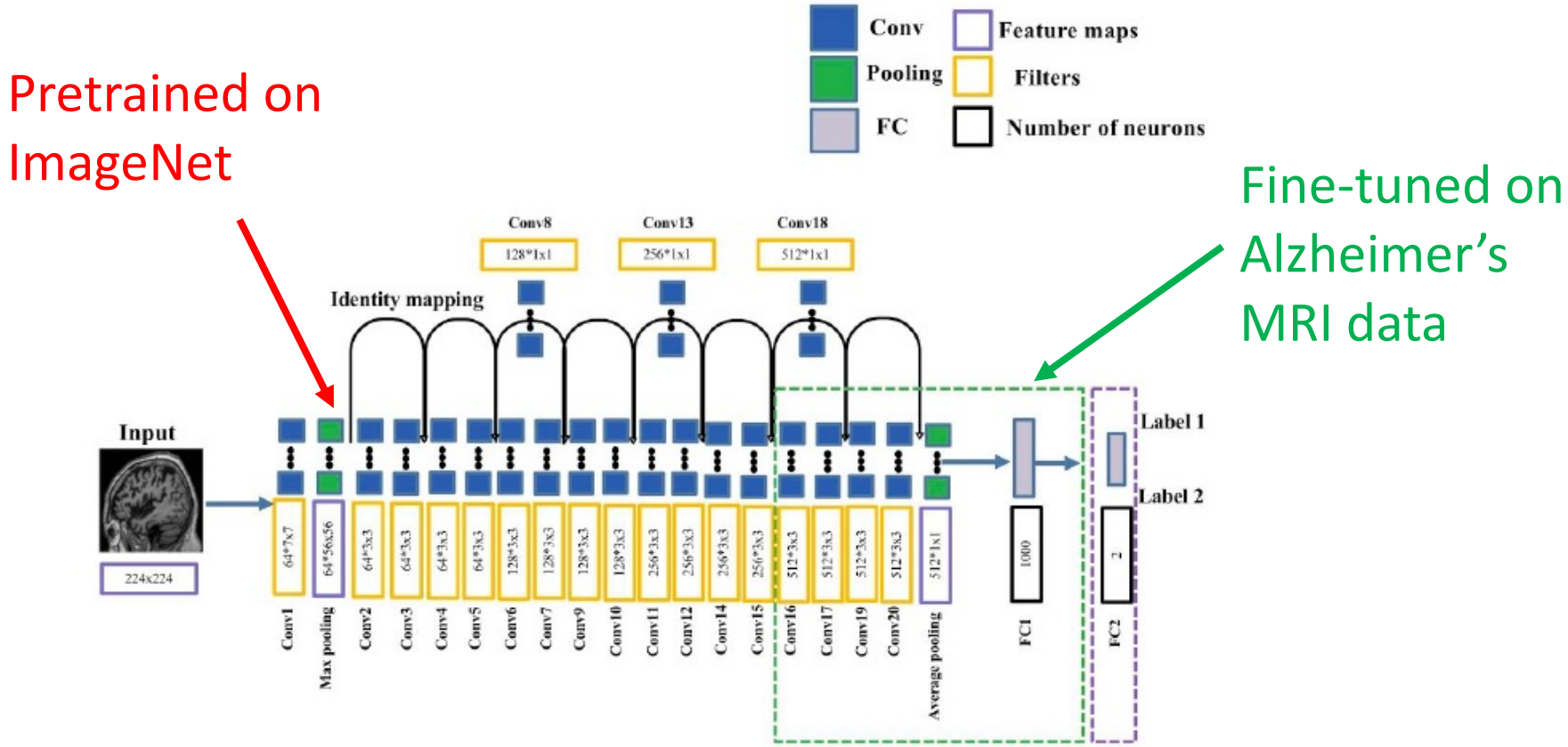
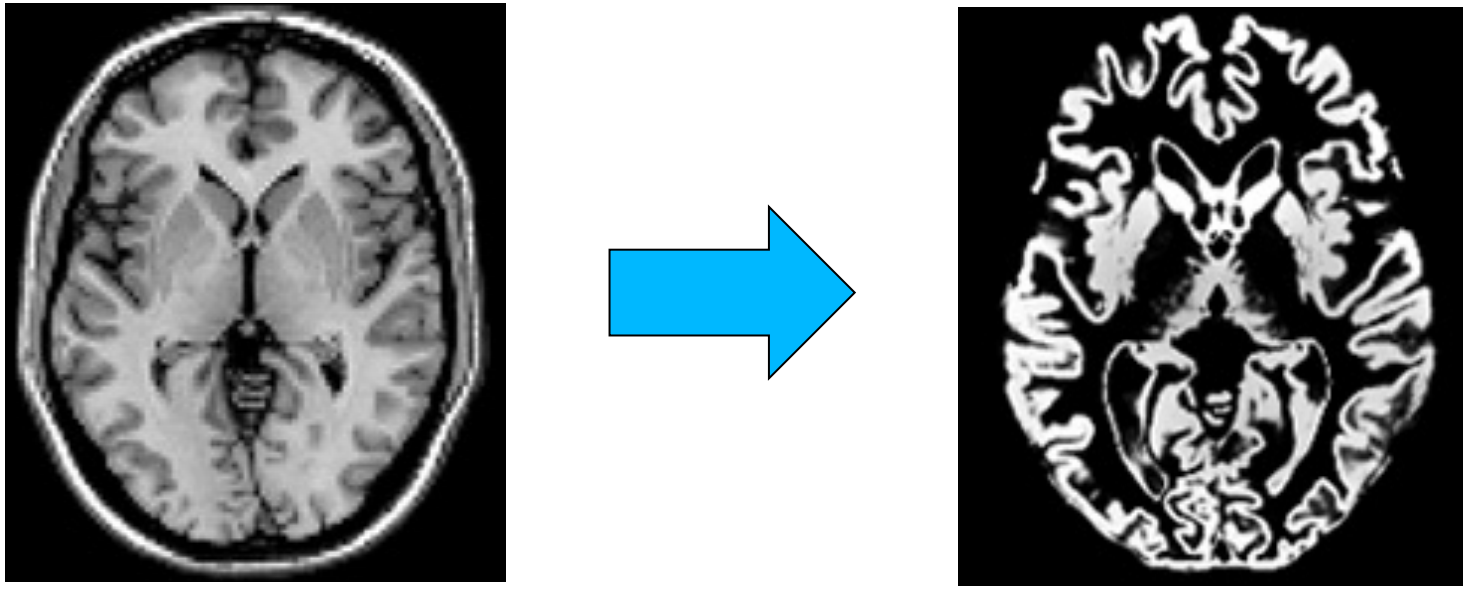


Figure 3: Architecture of the 2D slice-level CNN. An FC layer (FC2) was added on top of the ResNet. The last five convolutional layers and the last FC of ResNet (green dotted box) and the added FC layer (purple dotted box) were fine-tuned and the other layers were frozen during training. Filters for each convolutional layer represent the number of filters * filter size. Feature maps of each convolutional block represent the number of feature maps * size of each feature map. Conv: convolutional layer; FC: fully connected layer.

Feature extraction using domain knowledge

- **Knowledge:** Alzheimer's disease causes the loss of neurons which ultimately results in atrophy of the gray matter of the brain

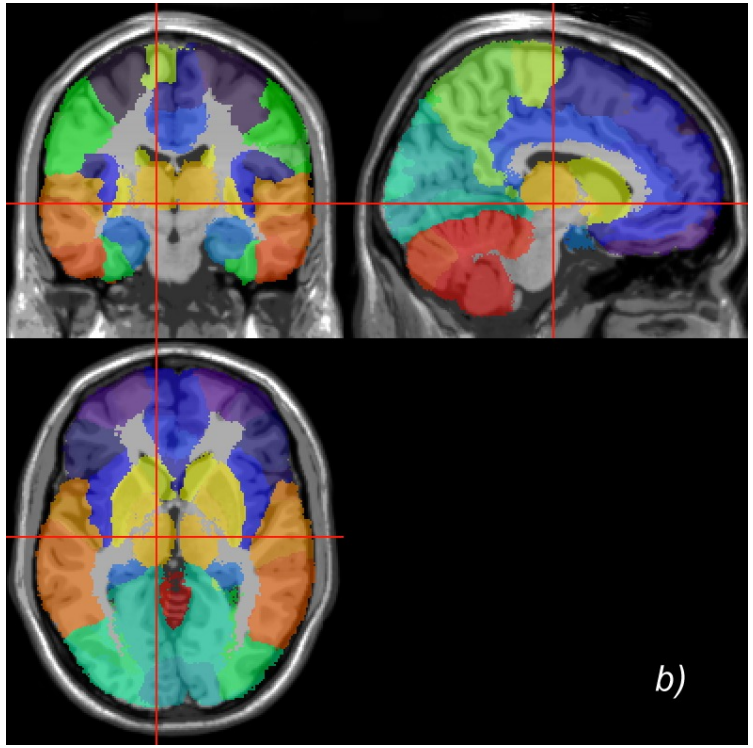
- Idea: quantify the gray matter in the image



Feature extraction using domain knowledge

- **Knowledge:** Alzheimer's disease affects preferentially some specific regions of the brain

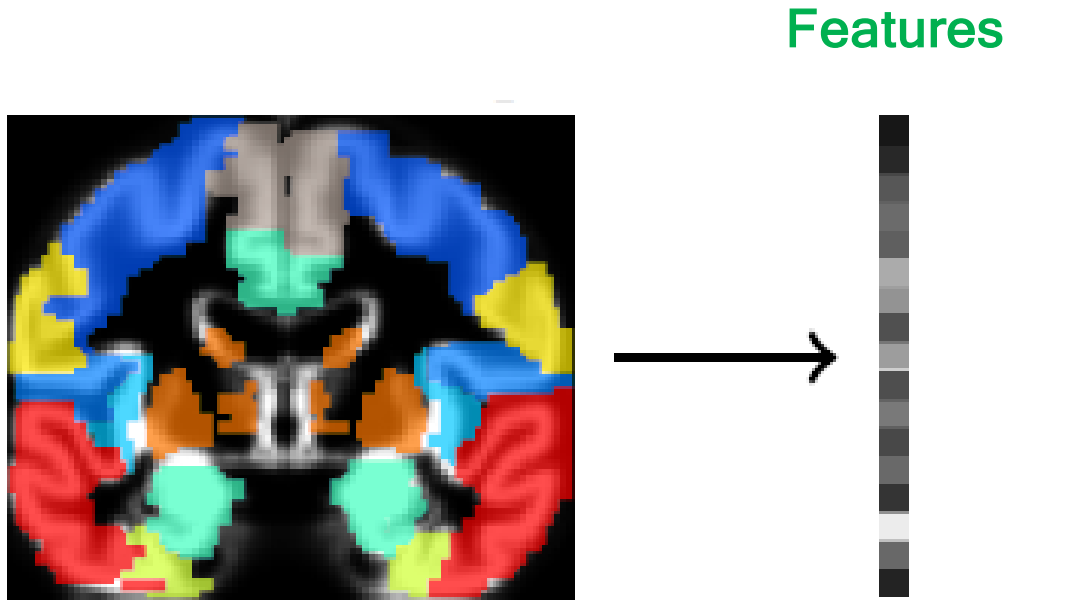
- Idea: parcellate the brain into different regions



All these operations are done automatically by sophisticated image processing algorithms

Feature extraction using domain knowledge

- **Features:** amount of gray matter in each region of the brain

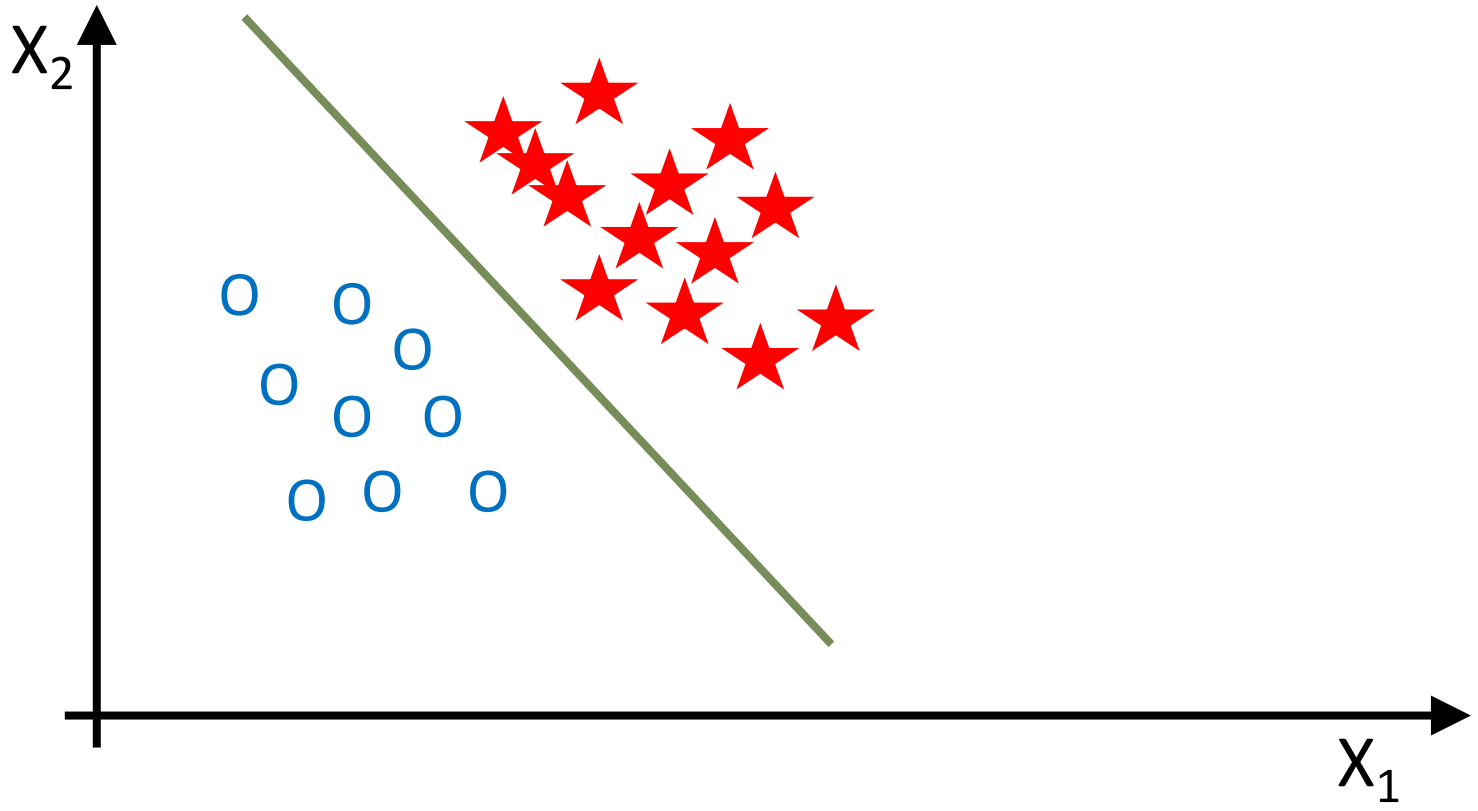


$x \in \mathbb{R}^p$

where p is the number of regions

Classifier

- **Classifier:** linear SVM



Linear SVM vs deep learning

		Alzheimer vs controls	Stable vs progressive MCI
Deep learning	3D Whole-brain CNN	84%	73%
	3D Hippocampus CNN	86%	73%
	2D ResNet	76%	N/A
Linear SVM	Linear SVM	86%	72%

Wen, Thibeau--Sutre et al, *Medical Image Analysis, 2020*

Image classification

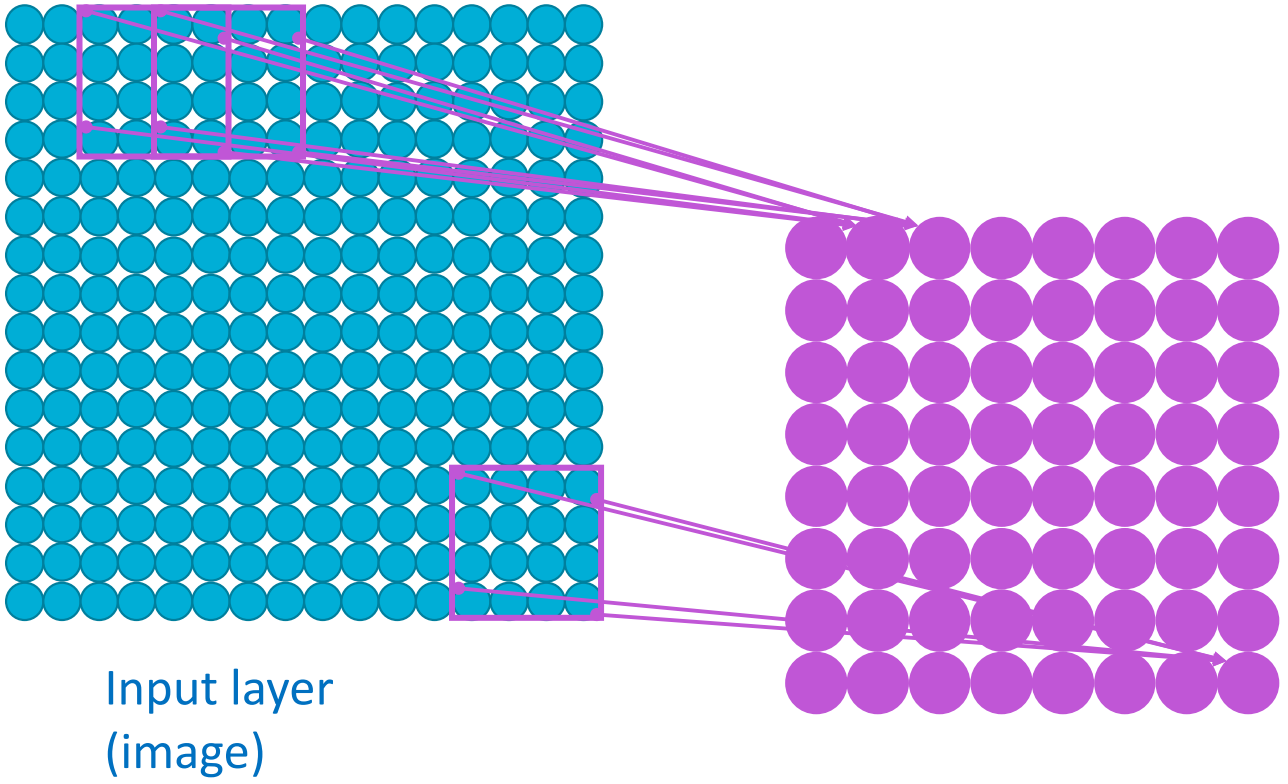
- **A situation very different from computer vision and from other medical image classifications**

- **Where does it come from?**
  - Not enough samples (hundreds vs millions)
  - Brain images are highly constrained objects
  - Image preprocessing and feature extraction are very mature in this field

Part 2 – Classification (and regression)
2.5 Transformers

Introduction

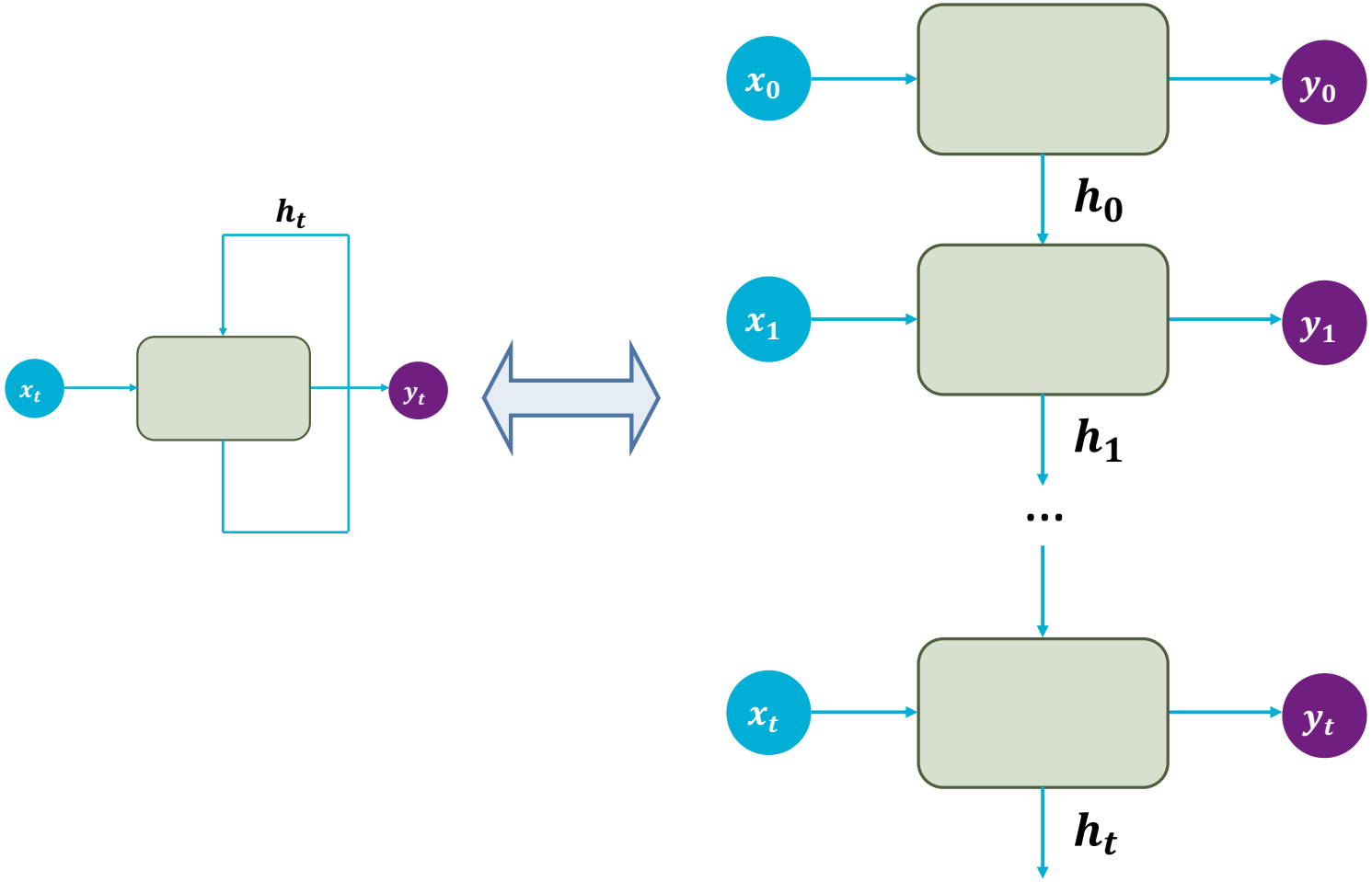
CNNs: Convolutional Neural Networks



Convolution: local operation (even though multiple features can subsequently be combined in the fully connected layers)

Introduction

RNNs: Recurrent Neural Networks



Process input step-by-step: only short range dependencies

Introduction

- **Transformers**
  - Initially developed in NLP
  - Subsequently adapted to other fields (e.g. computer vision)
  - Based on a mechanism called "attention »

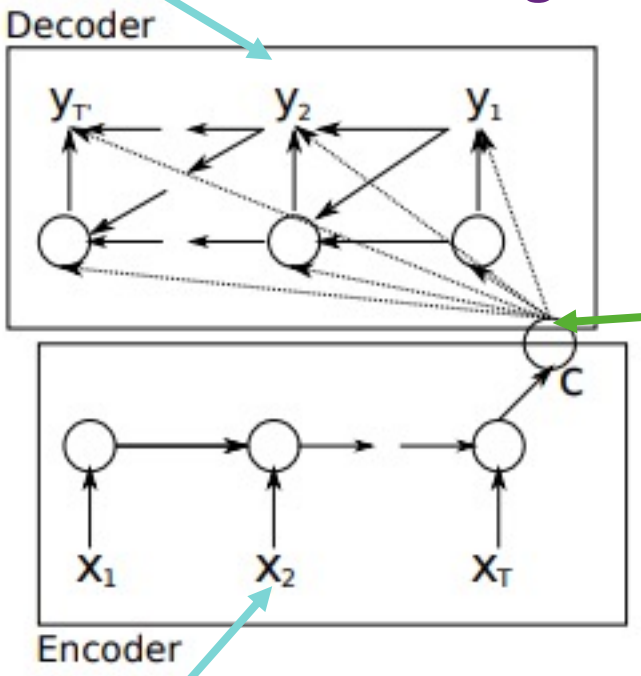
Introduction

- **Some advantages of transformers**
  - Capture of long-range dependencies
    - In RNNs, information is processed step-by-step
    - In CNNs, convolution operations are intrinsically local
  - Parallelization
    - Unlike RNNs which are sequential
  - Easier to train than RNNs
    - This is not an issue with CNNs
  - Attention mechanism
    - Can focus on some parts of the input

RNN Encoder-Decoder

Variable length output

Takes into account context by encoding past and present into a fixed length representation



Fixed length representation

Variable length input

(Cho et al, EMNLP 2014) - https://aclanthology.org/D14-1179.pdf

Attention mechanism

- **Limitations of fixed-length encoding**
  - given that the sizes of sentences vary and as the sentences get longer, a fixed-length vector is a real bottleneck: it gets increasingly difficult not to lose any information in the encoding process

- **Solution:** attention module (Bahdanue et al, 2015)
  - The attention module allows the model to **consider the parts of the sentence that are relevant** to predicting the next token.
  - Facilitates the understanding of **relationships among tokens that are further apart**.

(Courant, Edberg, Dufour and Kalogeiton, 2023) - https://link.springer.com/protocol/10.1007/978-1-0716-3195-9_6

Attention mechanism

Given two lists of tokens, $\boldsymbol{X} \in \mathbb{R}^{N \times d_x}$ and $\boldsymbol{Y} \in \mathbb{R}^{N \times d_y}$, attention encodes information from $\boldsymbol{Y}$ into $\boldsymbol{X}$, where N is the length of inputs $\boldsymbol{X}$ and $\boldsymbol{Y}$, and d_x and d_y are their respective dimensions. For this, we first define three linear mappings: query mapping $\boldsymbol{W}^Q \in \mathbb{R}^{d_x \times d_q}$, key mapping $\boldsymbol{W}^K \in \mathbb{R}^{d_y \times d_k}$ and value mapping $\boldsymbol{W}^V \in \mathbb{R}^{d_y \times d_v}$, where d_q, d_k, and d_v is the embedding dimension in which the query, key, and value are going to be computed, respectively.

Then, we define the query $\boldsymbol{Q}$, key $\boldsymbol{K}$ and value $\boldsymbol{V}$ [4] as:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q$$
$$\boldsymbol{K} = \boldsymbol{Y}\boldsymbol{W}^K$$
$$\boldsymbol{V} = \boldsymbol{Y}\boldsymbol{W}^V$$

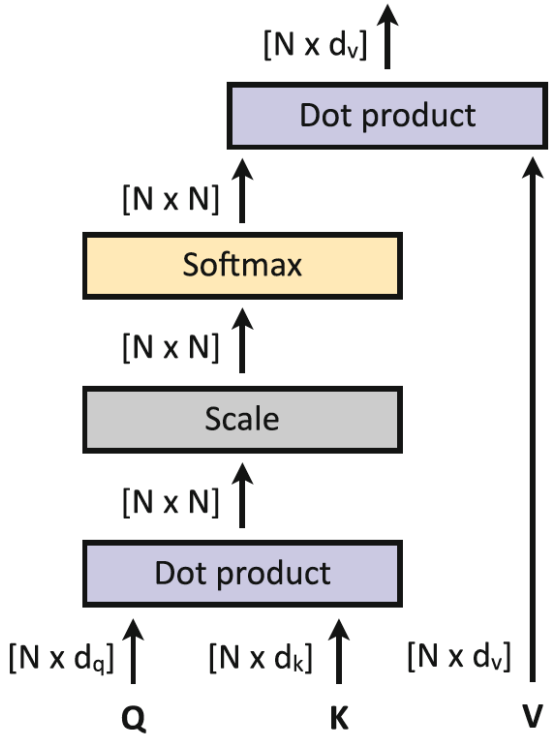
Next, the *attention matrix* is defined as:

$$A(\boldsymbol{Q}, \boldsymbol{K}) = \text{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right) \quad .$$

The resulting $N \times N$ matrix encodes the relationship between $\boldsymbol{X}$ with respect to $\boldsymbol{Y}$: it measures how important a token in $\boldsymbol{X}$ is with respect to another one in $\boldsymbol{Y}$.

Finally, the *attention output* is defined as:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = A(\boldsymbol{Q}, \boldsymbol{K})\boldsymbol{V} \quad . \qquad (2)$$



(Courant, Edberg, Dufour and Kalogeiton, 2023) - https://link.springer.com/protocol/10.1007/978-1-0716-3195-9_6

Transformers

The attention mechanism is positional agnostic

Need for positional encoding

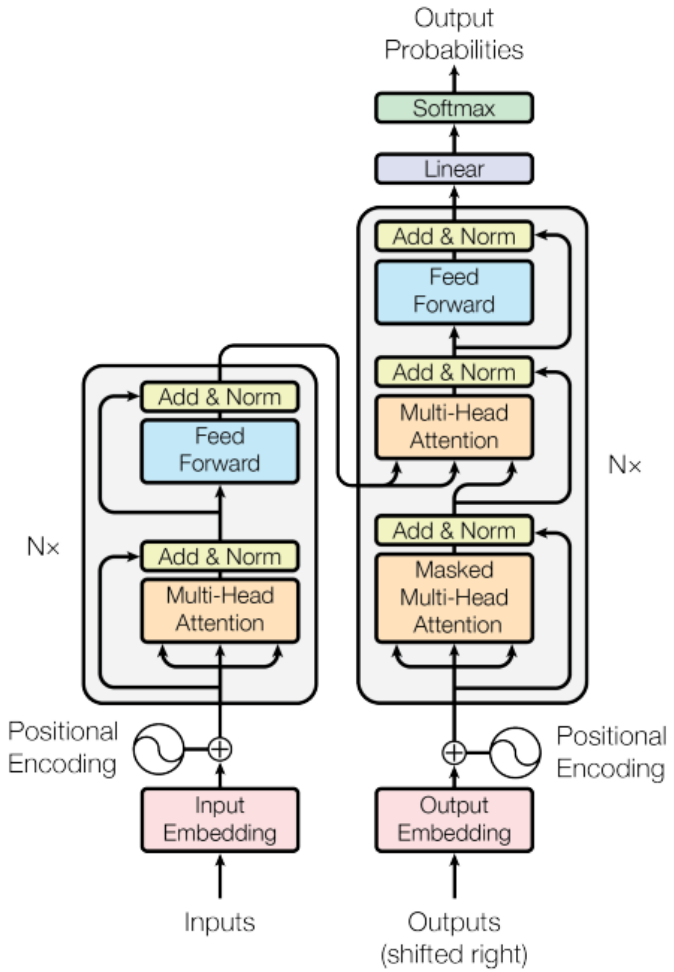


Figure 1: The Transformer - model architecture.

Vision transformers: the ViT architecture

- **Transformers initially developed for NLP**
- Then extended to computer vision

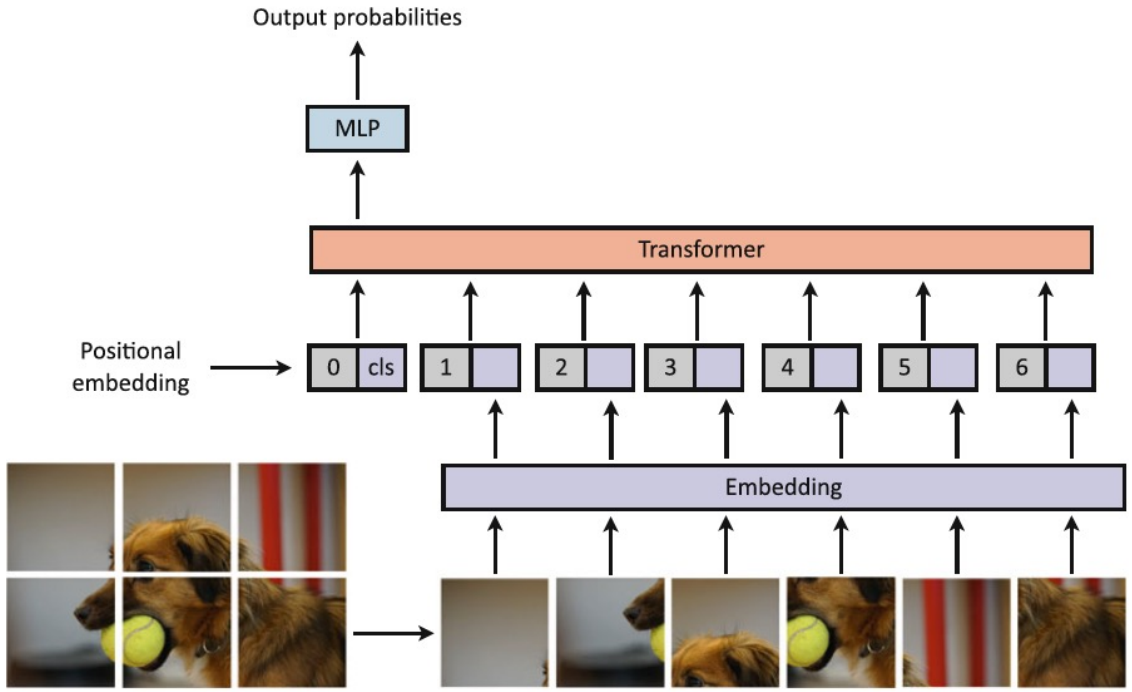


Fig. 4 The vision transformer architecture (ViT). First, the input image is split into patches (bottom), which are linearly projected (embedding), and then concatenated with positional embedding tokens. The resulting tokens are fed into a transformer, and finally the resulting classification token is passed through an MLP to compute output probabilities. Figure inspired from [5]

In self-attention, the tokens of $\boldsymbol{X}$ attend to themselves ($\boldsymbol{X} = \boldsymbol{Y}$). Therefore, it is modelled as follows:

$$\mathrm{SA}(\boldsymbol{X}) = \mathrm{Attention}(\boldsymbol{X}\boldsymbol{W}^{Q}, \boldsymbol{X}\boldsymbol{W}^{K}, \boldsymbol{X}\boldsymbol{W}^{V}) \quad . \tag{3}$$

(Courant, Edberg, Dufour and Kalogeiton, 2023) - https://link.springer.com/protocol/10.1007/978-1-0716-3195-9_6

Vision transformers: the ViT architecture

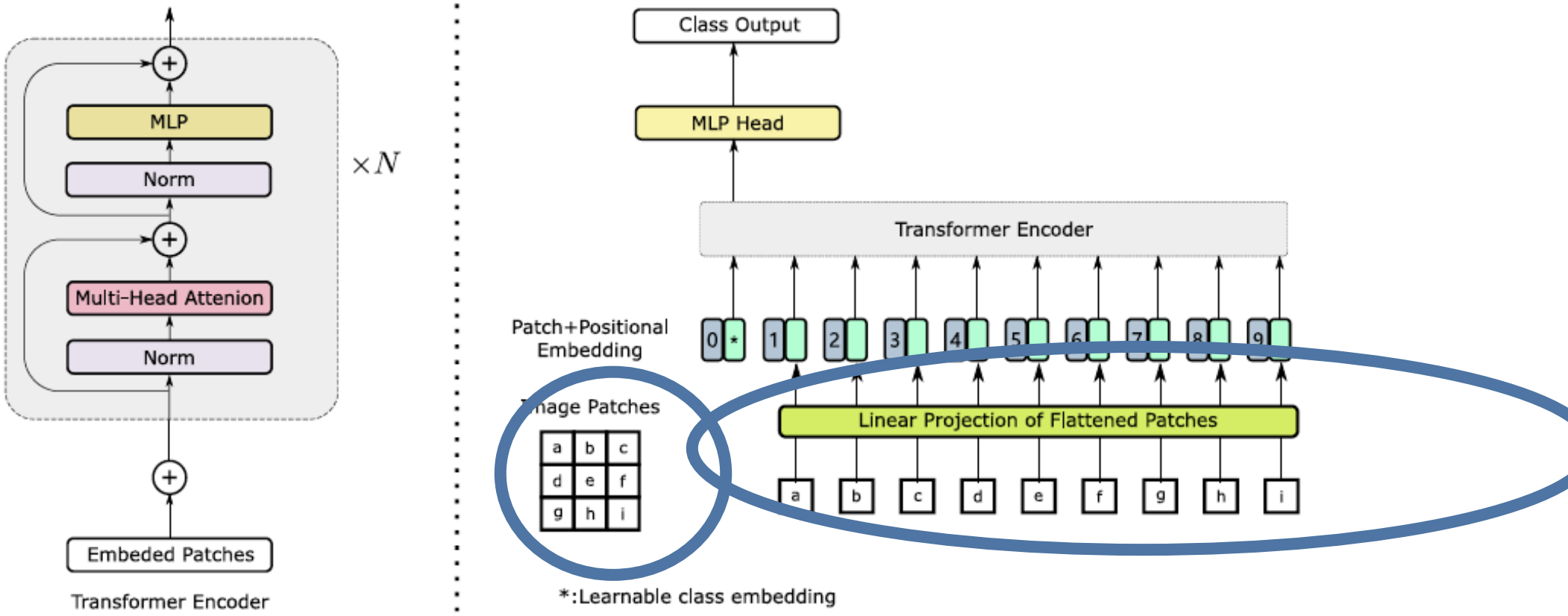


Figure 4. Architecture of ViT, as proposed in [13]. Sequential image patches are used as the input and processed with the transformer encoder, and the class prediction is output by an MLP head. The transformer encoder is constructed using N transformer blocks.

(He et al, Intelligent Medicine, 2023) - https://doi.org/10.1016/j.imed.2022.07.002

Other vision transformers

- **Transformers can be combined with convolutions: need less data for training**

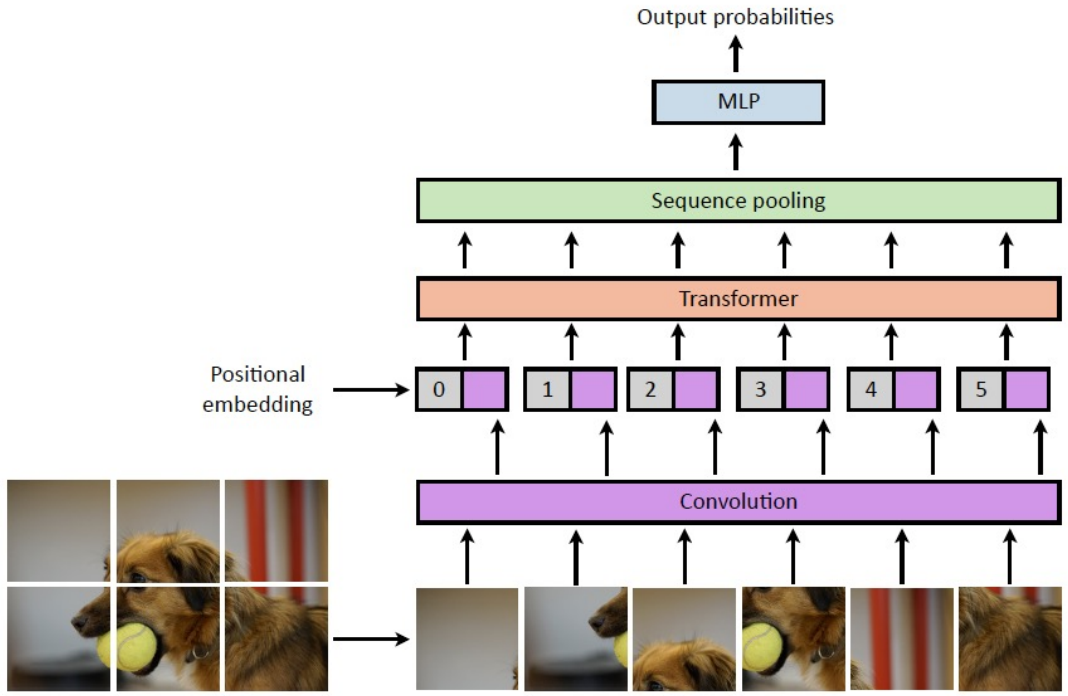


Figure 6: ***Compact Convolutional Transformers.*** *This architecture features a convolutional based patch extraction to leverage a smaller Transformer network, leading to higher data-efficiency. Figure inspired from [23].*

(Courant, Edberg, Dufour and Kalogeiton, 2023) - https://link.springer.com/protocol/10.1007/978-1-0716-3195-9_6

Transformers

- **Advantages**
  - Highly parallelizable and thus scalable
  - Can be trained on extremely massive data (billions of token)
  - And have a huge number of parameters
  - Little inductive bias: learn the structure of the data (unlike e.g. CNNs)
- **Drawbacks**
  - Little inductive bias: need vast amounts of data
    - Less data-hungry architectures have been proposed
    - CNNs are still competitive for many applications

Part 2 – Classification (and regression)
2.6 Examples of applications with transformers

Transformers for medical image classification

- **Various types of approaches**
  - Direct use of ViT (Vision Transformater)
  - Combination with convolutions
  - Review of some applications in medical imaging:
    - (He et al, Intelligent Medicine, 2023) - https://doi.org/10.1016/j.imed.2022.07.002

Transformers for medical image classification

Effective Pancreatic Cancer Screening on Non-contrast CT Scans via Anatomy-Aware Transformers

Yingda Xia[1,*], Jiawen Yao[2], Le Lu[2], Lingyun Huang[3], Guotong Xie[3], Jing Xiao[3], Alan Yuille[1], Kai Cao[4(✉)], Ling Zhang[2]

Classification between patients with pancreatic ductal adenocarcinoma (PDAC) or non-PDAC

common type (about 90% of all pancreatic cancers). This is the main reason that we group all abnormalities into two classes of PDAC and nonPDAC (including nine subtypes [17,25]). The dataset is randomly split into a training and a testing dataset. The training set includes 450 PDACs, 394 nonPDACs, and 477 normal pancreases. The testing set includes 108 PDACs, 80 nonPDACs, 118 normal pancreases. Both PDAC and nonPDAC cases are confirmed by their pathology reports and normal cases by radiology reports and 2-year follow-up. Each patient

(Xia et al, MICCAI 2022)

Transformers for medical image classification

Effective Pancreatic Cancer Screening on Non-contrast CT Scans via Anatomy-Aware Transformers

Yingda Xia[1,*], Jiawen Yao[2], Le Lu[2], Lingyun Huang[3], Guotong Xie[3], Jing Xiao[3], Alan Yuille[1], Kai Cao[4(✉)], Ling Zhang[2]

Classification between patients with pancreatic ductal adenocarcinoma (PDAC) or non-PDAC

	2-class			3-class		
Method	AUC	Sensitivity	Specificity	PDAC	nonPDAC	Normal
S4C with UNet [29]	95.98	91.48	95.76	75.00	73.75	95.76
Hybrid CNN	98.25	94.68	94.91	76.85	80.00	94.91
Hybrid Transformer	**98.37**	**95.21**	**95.76**	**78.70**	**80.00**	**95.76**
Mean radiologists WOTC	-	79.66	87.58	53.63	57.96	87.58

Table 1: Results on two-class classification (PDAC+nonPDAC vs. normal) and three-class classification (PDAC vs. nonPDAC vs. normal). WOTC: without time constraint.

(Xia et al, MICCAI 2022)

Transformers for medical image classification

Global-Local Transformer for Brain Age Estimation

Sheng He, P. Ellen Grant, Yangming Ou

TABLE II
THE INFORMATION OF DATASETS IN THE HEALTHY COHORT USED FOR BRAIN AGE ESTIMATION.

Dataset	$N_{samples}$	Age range	Gender(female/male)
◊ BGSP [46]	1,570	19.0-35.0	905/665
◊ OASIS-3 [47]	1,222	42.0-97.0	750/472
◊ NIH-PD [48]	1,211	0-22.2	626/585
◊ ABIDE-I [49]	567	6.4-56.2	98/469
◊ IXI*	556	19.9-86.3	309/247
◊ DLBS [50]	315	20.5-89.1	198/117
⊕ CMI [51]	1,765	5.0-21.9	1,117/648
⊕ CoRR [52]	1,173	6.0-88.0	591/582
Overall	8,379	0-97.0	4,594/3,785

*https://brain-development.org/ixi-dataset/
◊ Datasets are used for cross-validation.
⊕ Datasets are used for evaluating the generality.

TABLE VI
COMPARISON WITH STATE-OF-THE-ART METHODS FOR BRAIN AGE ESTIMATION BASED ON FIVE-CROSS VALIDATION.

Method	MAE	Pearson Correlation (r)	CS(α=5 years)
ShuffleNet V2 (2.0x) [64]	3.85±0.12	0.9668±0.0036	76.90%±0.78
SqueezeNet [63]	3.71±0.16	0.9710±0.0035	77.05%±1.53
ResNet50 [20]	3.12±0.08	0.9781±0.0027	81.88%±0.73
ResNet101 [20]	3.15±0.13	0.9778±0.0029	81.53%±0.98
WRN-50-2 [61]	3.06±0.10	0.9786±0.0028	82.38%±0.85
WRN-101-2 [61]	3.07±0.10	0.9788±0.0022	81.97%±0.81
DenseNet121 [62]	2.86±0.08	0.9837±0.0017	82.87%±1.01
DenseNet201 [62]	2.80±0.07	0.9836±0.0015	83.72%±0.65
*SFCN 2D [13] (2021)	3.58±0.10	0.9754±0.0023	77.73%±0.95
*SFCN 3D [13] (2021)	3.04±0.06	0.9817±0.0008	81.66%±0.50
**FiA-Net$_{fus}$ 3D [31] (2021)	3.00±0.06	0.9840±0.0000	81.75%±1.20
*DeepBrainNet [8] (2020)	2.97±0.11	0.9815±0.0022	82.87%±0.66
Global-Local Transformer	**2.70±0.03**	**0.9853±0.0020**	**84.53%±0.77**

*Models specifically designed for brain age estimation.
**Results are directly from the paper.

(He et al, IEEE TMI, 2022)