ARAMIS LAB
BRAIN DATA SCIENCE

www.aramislab.fr

ninonburgos.com

10th March 2020

Introduction to deep learning

Ninon Burgos, CNRS Researcher

Aramis Lab, Paris, France

"When you're **fundraising**, it's **AI**

When you're **hiring**, it's **ML**

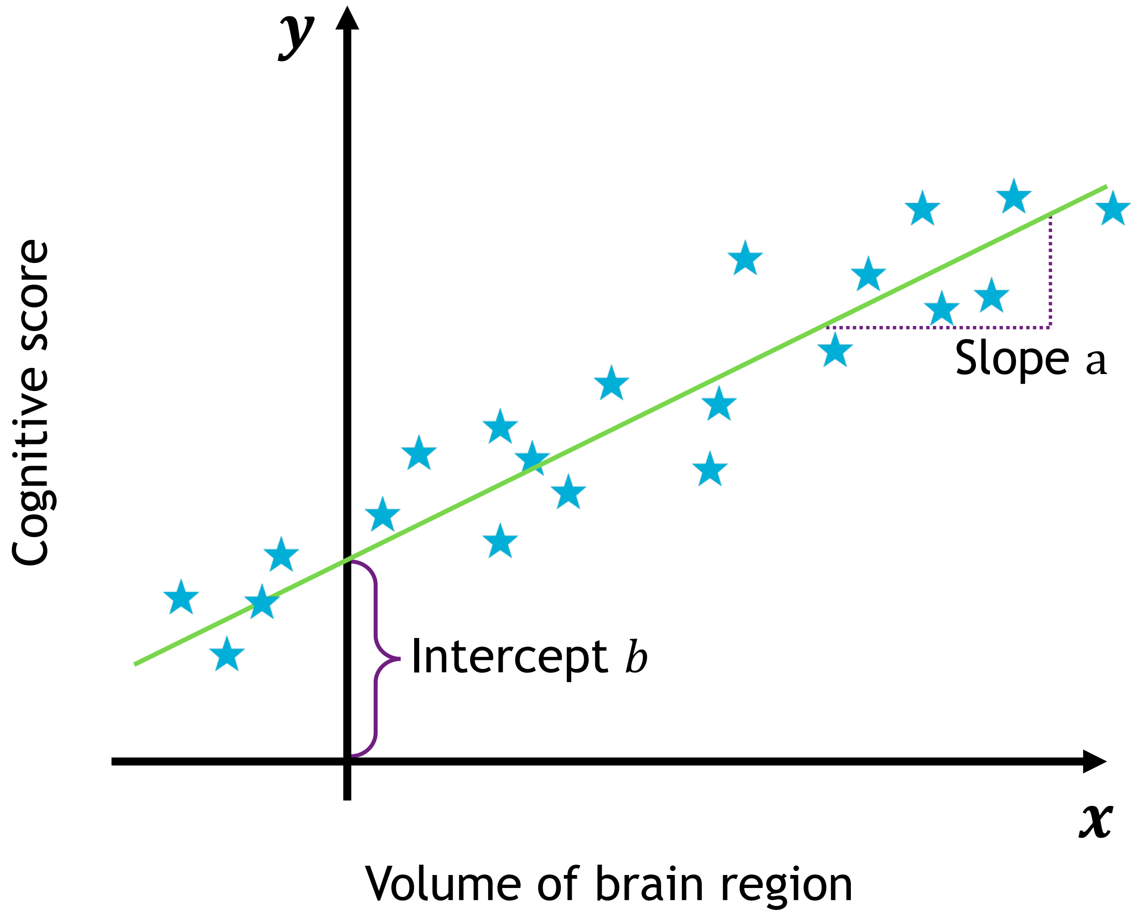
When you're **implementing**, it's **linear regression**"

— Baron Schwartz

The basics

Linear regression

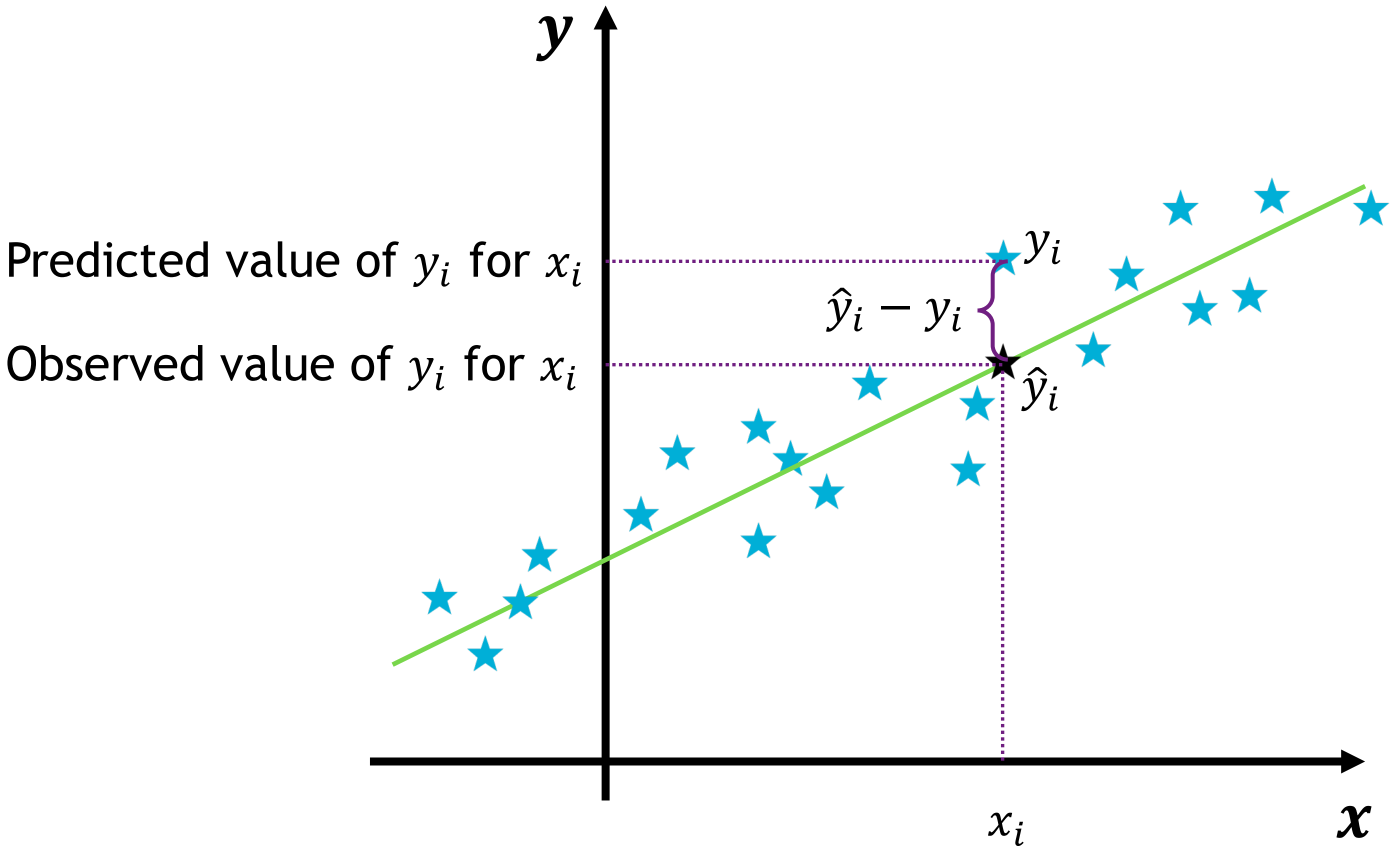
What is linear regression?



$$y = f(x) = ax + b$$

Linear regression

Cost function

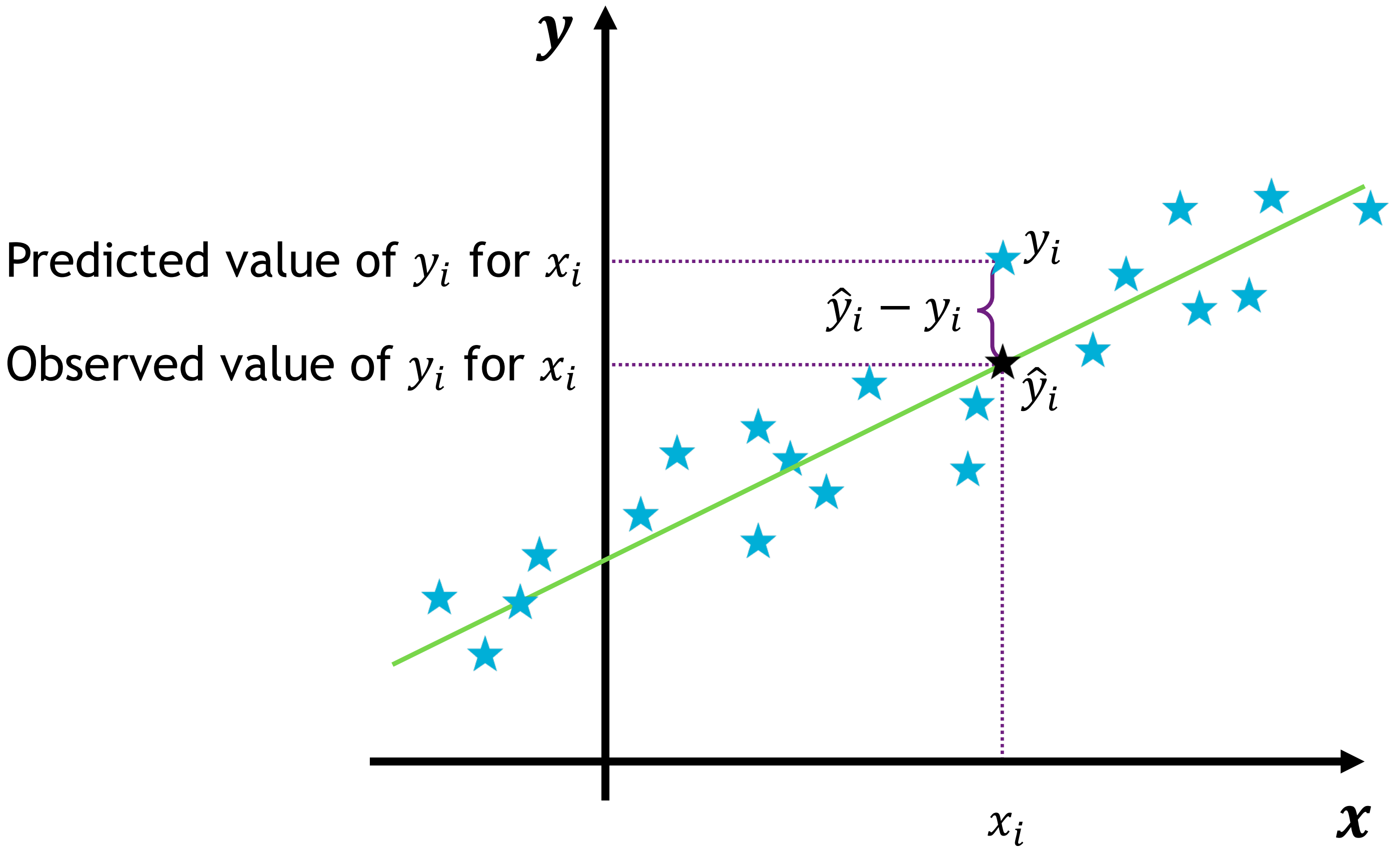


Mean squared error:

$$J = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

Linear regression

Learning

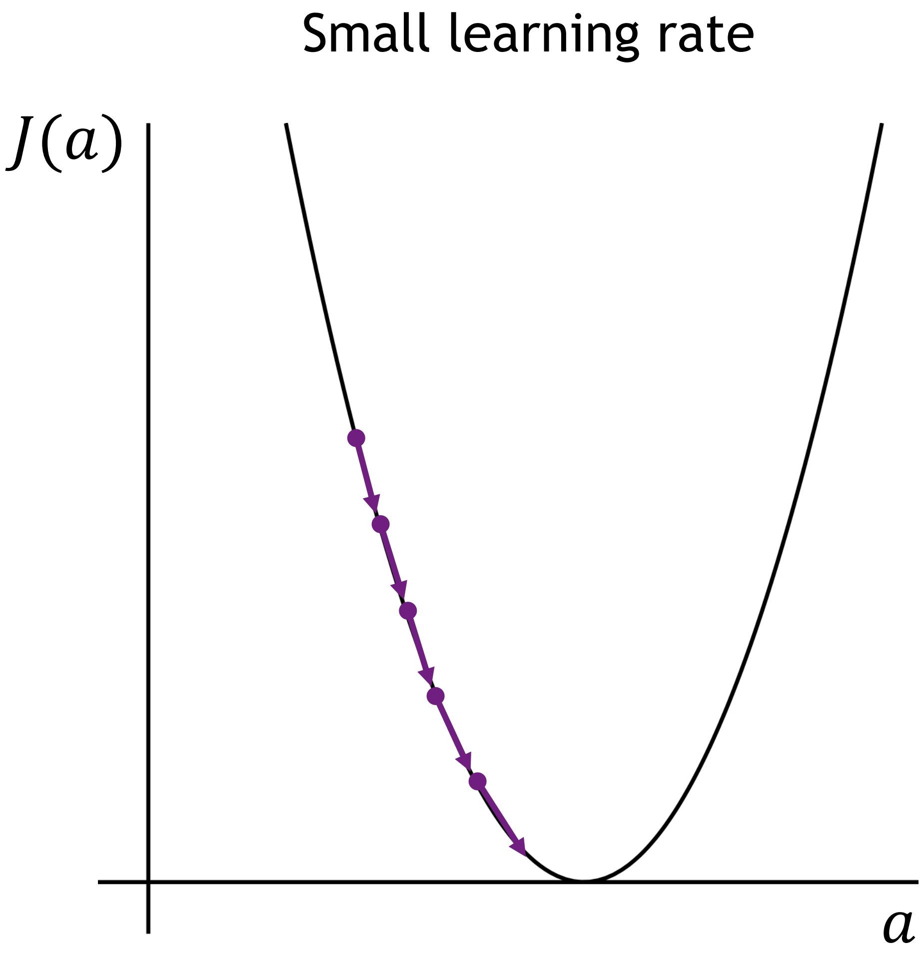


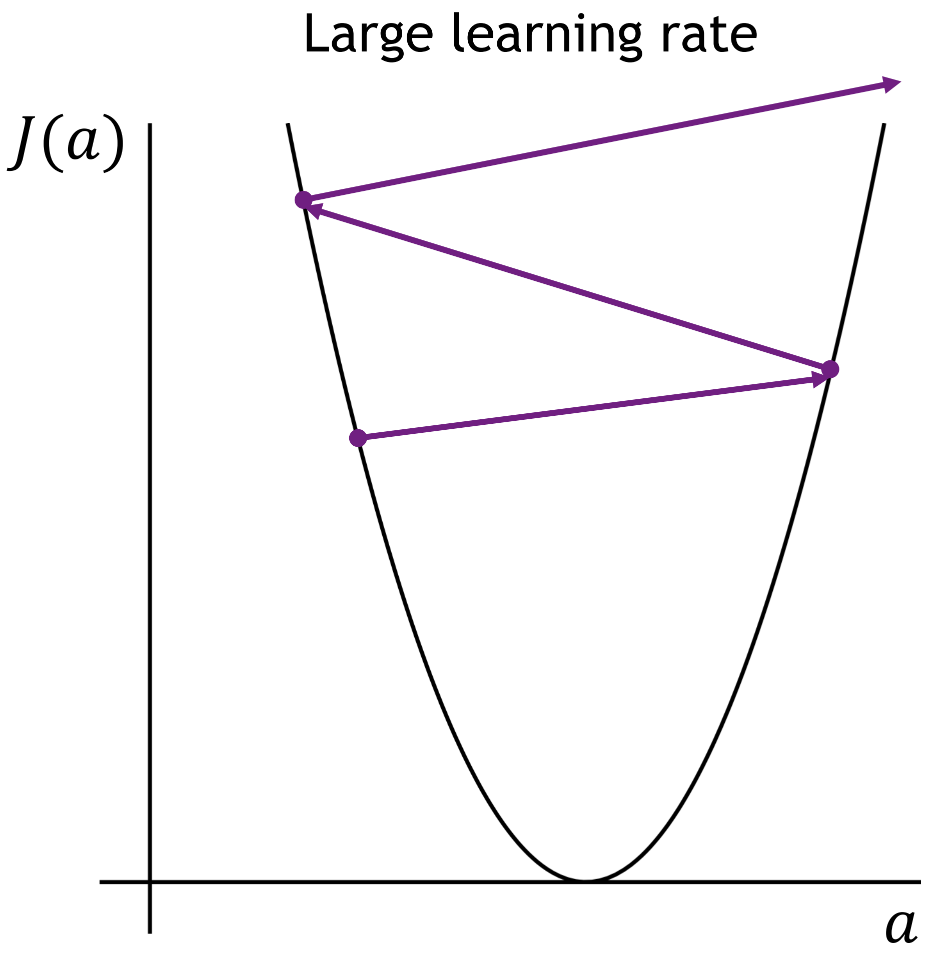
Find the model with the minimal error:

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

Linear regression

Gradient descent





Linear regression

Gradient descent

Cost function:

$$J = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(ax_i + b - y_i)^2$$

Partial derivates:

$$\frac{\partial J}{\partial a} = \frac{2}{n}\sum_{i=1}^{n}(ax_i + b - y_i)\cdot x_i = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)\cdot x_i$$

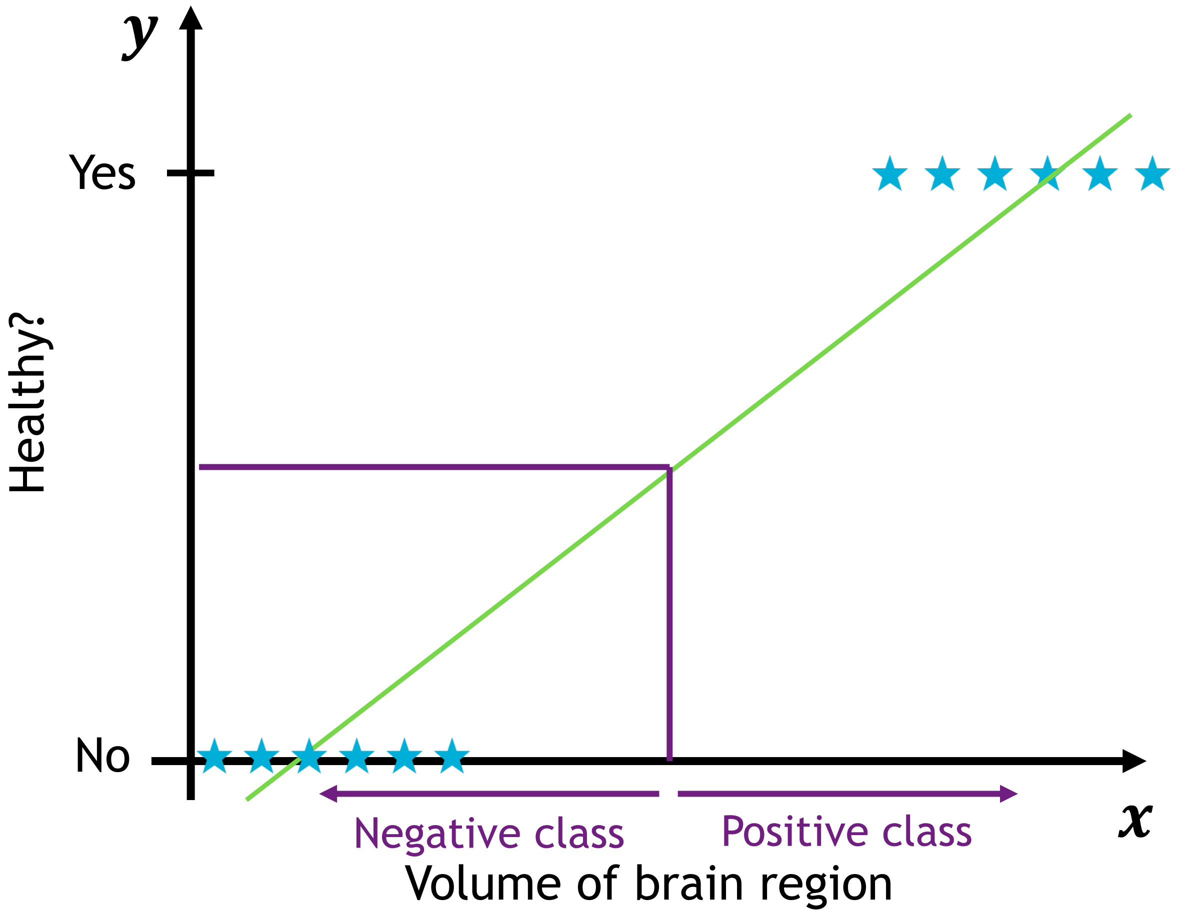
$$\frac{\partial J}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}(ax_i + b - y_i) = \frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)$$

Update of a and b:

$$a \leftarrow a - \eta\frac{\partial J}{\partial a} = a - \eta\frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)$$

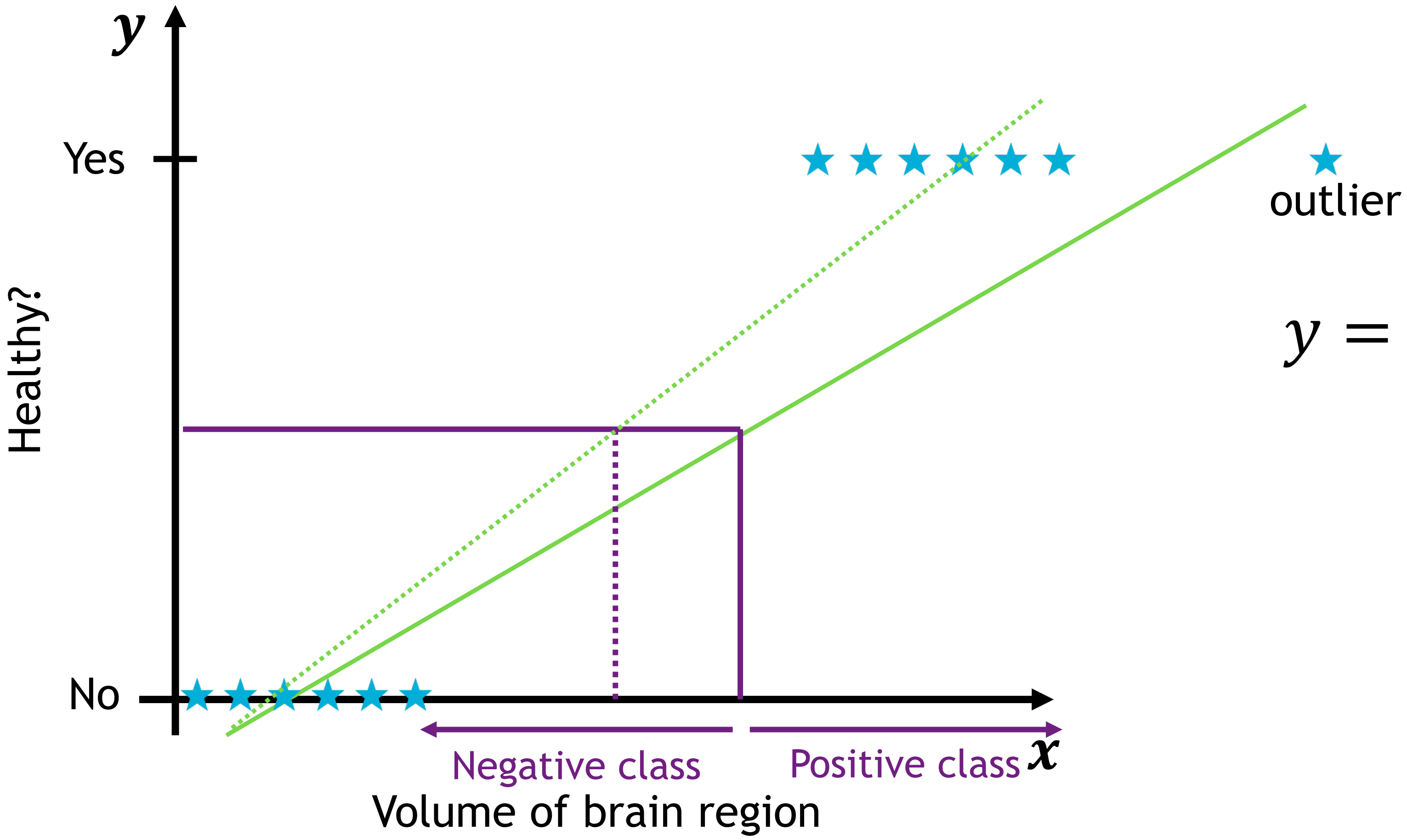
$$b \leftarrow b - \eta\frac{\partial J}{\partial b} = b - \eta\frac{2}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)\cdot x_i$$

Linear function



$$y = ax + b$$

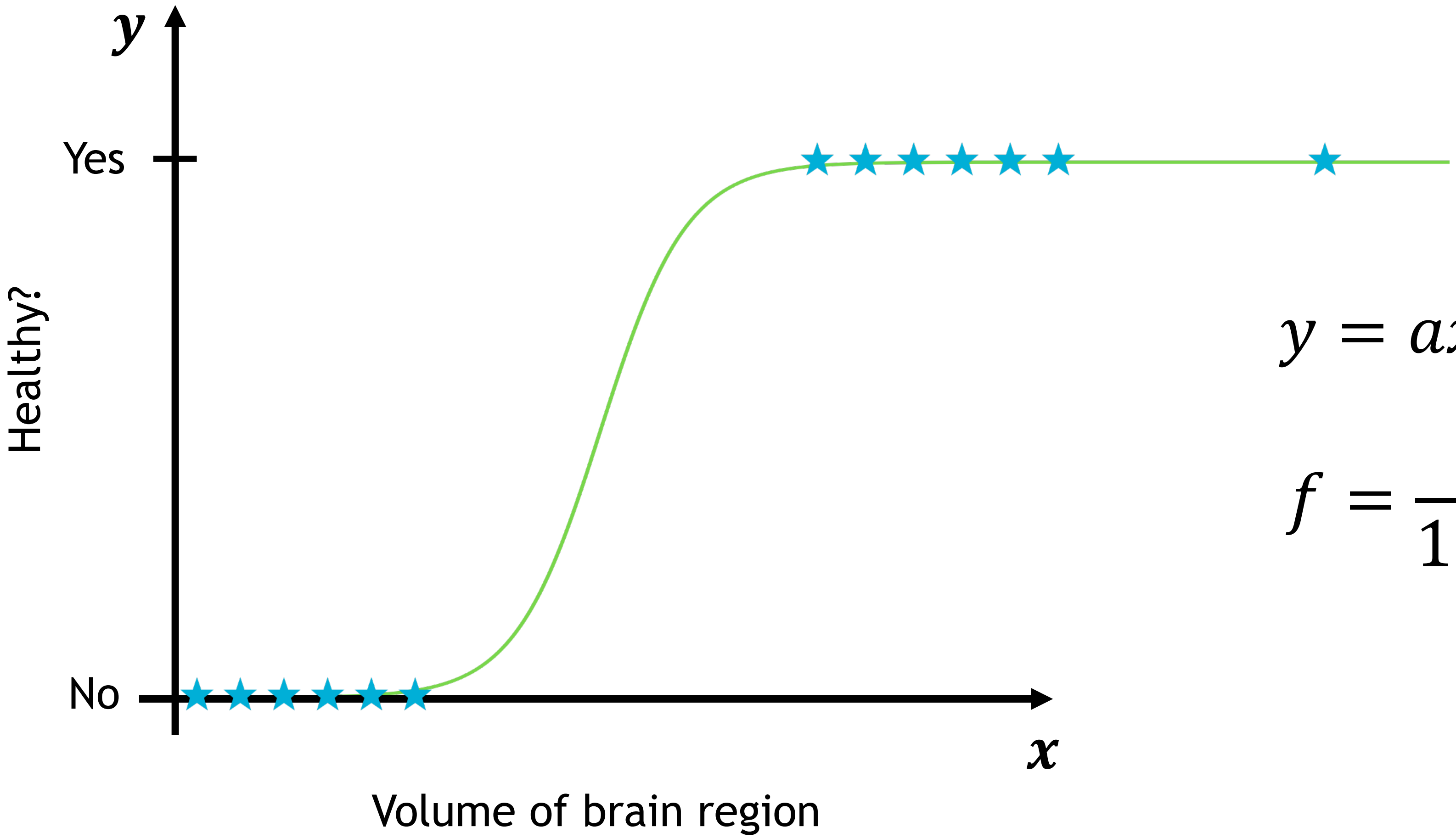
Linear function



$$y = ax + b$$

Logistic regression

Sigmoid function



$$y = ax + b$$

$$f = \frac{1}{1 + e^{-y}} = \frac{1}{1 + e^{-(ax+b)}}$$

Logistic regression

Multiple input variables

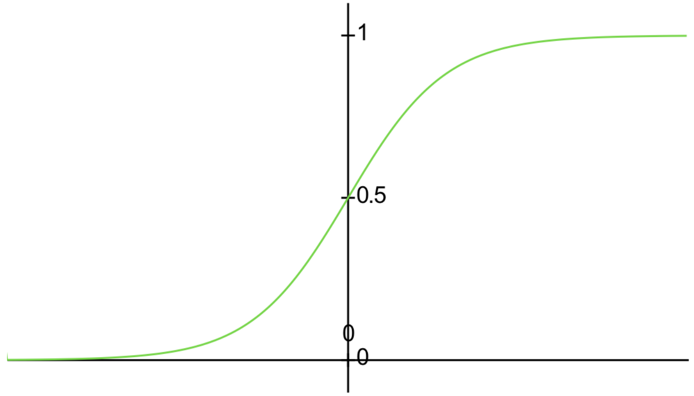
$$z = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_m x_m = w_0 + \sum_{j=1}^{m} x_j w_j$$

$$f = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^{-\left(w_0 + \sum_{j=1}^{m} x_j w_j\right)}}$$

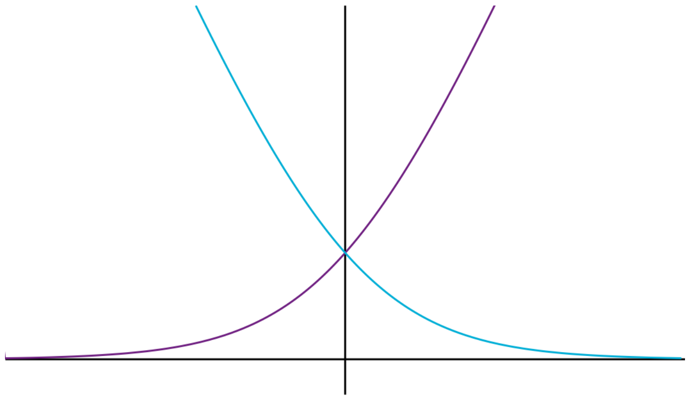
Loss function

- Low if prediction is correct
- High if prediction is wrong

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



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



$$l(f(\boldsymbol{x}), y) = -y \log\big(f(\boldsymbol{x})\big) - (1 - y) \log(1 - f(\boldsymbol{x}))$$

Logistic regression

Cost function

$$J(f) = \frac{1}{n}\sum_{i=1}^{n} l(f(\boldsymbol{x}_i), y_i)$$

$$J(f) = -\frac{1}{n}\sum_{i=1}^{n}\Big(y_i \log\big(f(\boldsymbol{x}_i)\big) + (1 - y_i)\log(1 - f(\boldsymbol{x}_i))\Big)$$

Learning

Find the model with the minimal error:

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

Logistic regression

Gradient descent

Cost function:

$$J(f) = -\frac{1}{n}\sum_{i=1}^{n}\Big(y_i \log\big(f(\boldsymbol{x}_i)\big) + (1 - y_i)\log(1 - f(\boldsymbol{x}_i))\Big)$$

Partial derivates:

$$\frac{\partial J}{\partial w_j} = -\frac{1}{n}\sum_{i=1}^{n}\big(y_{i,j} - f(\boldsymbol{x}_i)\big) \cdot x_{i,j}$$

Update of $\boldsymbol{a}$ and $\boldsymbol{b}$:

$$w_j \leftarrow w_j - \eta \frac{\partial J}{\partial w_j}$$

Summary

Input variables: $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ — Input features

Output: y

Model: $f, y = f(\boldsymbol{x})$ — The "artificial intelligence"

Loss: $l(f(\boldsymbol{x}), y)$ — Quantifies how much the prediction is far from the true output

Cost function: $J(f) = \frac{1}{n}\sum_{i=1}^{n} l(f(\boldsymbol{x}_i), y_i)$ — Quantifies how much the prediction is far from the true output across all training examples

Learning: $\hat{f} = arg \min_{f \in \mathcal{F}} J(f)$ — Find the model with the minimal error

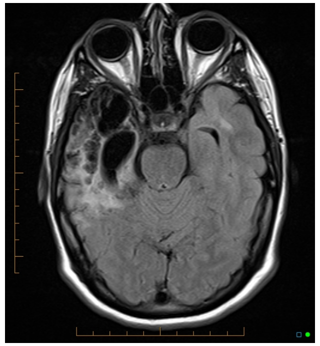
Gradient descent: $w \leftarrow w - \eta \frac{\partial J}{\partial w}$ — Method to find the model with the minimal error

ML vs DL : learning features

ML

Feature extraction

Classification

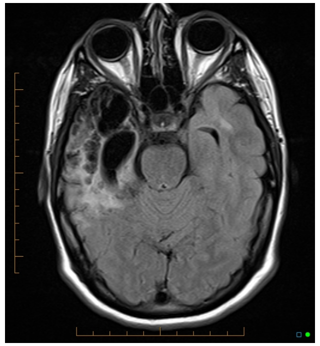


Hand-crafted → Machine learning →

Glioblastoma

Herpes simplex encephalitis

DL



Machine learning → Machine learning →

Glioblastoma

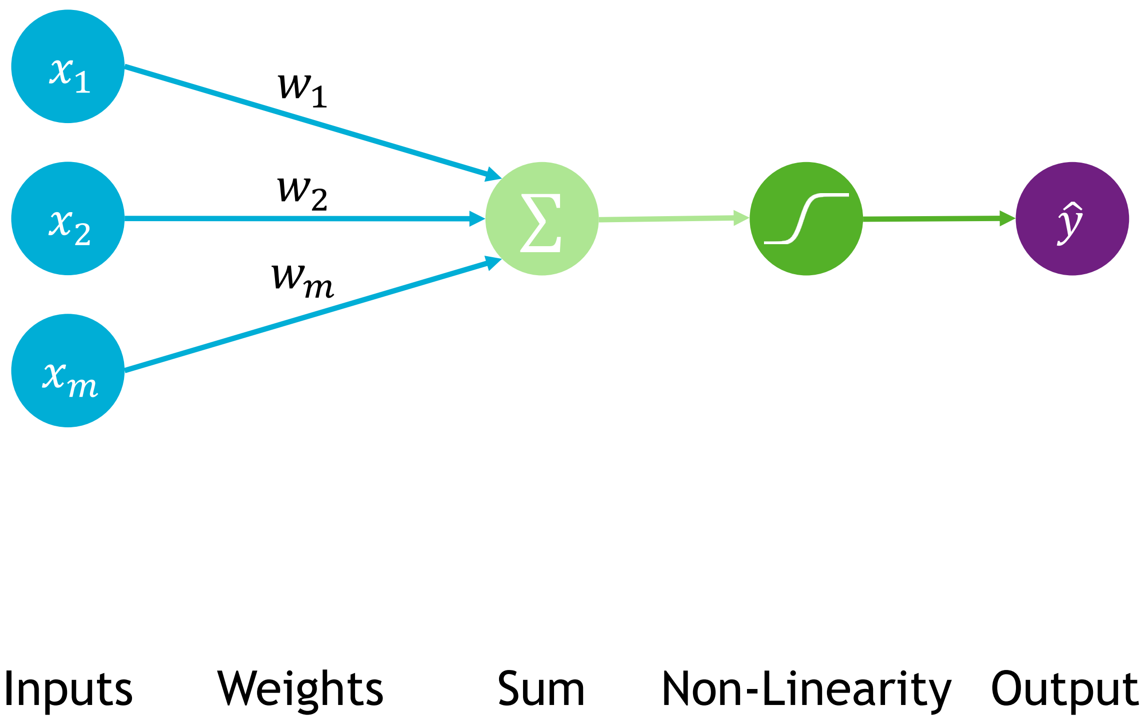
Herpes simplex encephalitis

Case courtesy of Dr Chris O'Donnell, Radiopaedia.org, rID: 27433

Neural networks

Perceptron

Artificial neuron



Output

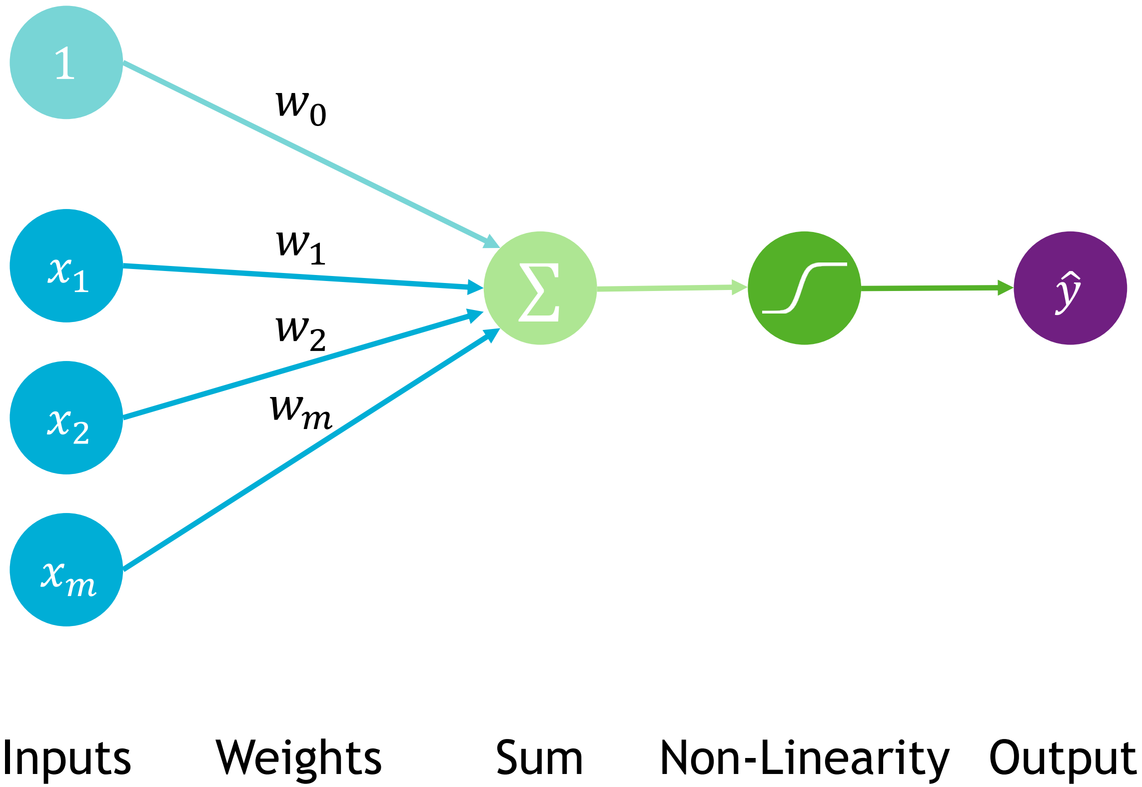
Linear combination of inputs

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

Non-linear activation function

Perceptron

Artificial neuron



Output — Bias — Linear combination of inputs

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

Non-linear activation function

$$\hat{y} = h(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

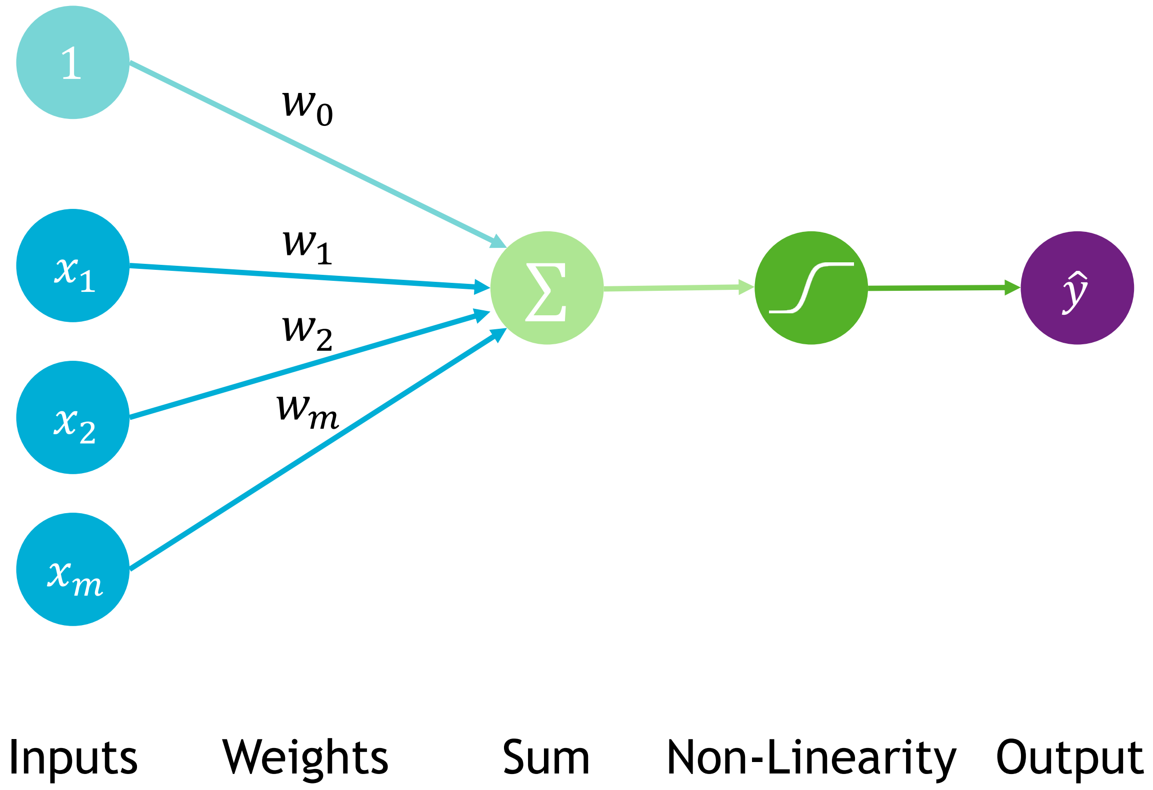
where $\boldsymbol{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$ and $\boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

MIT Introduction to Deep Learning (introtodeeplearning.com)

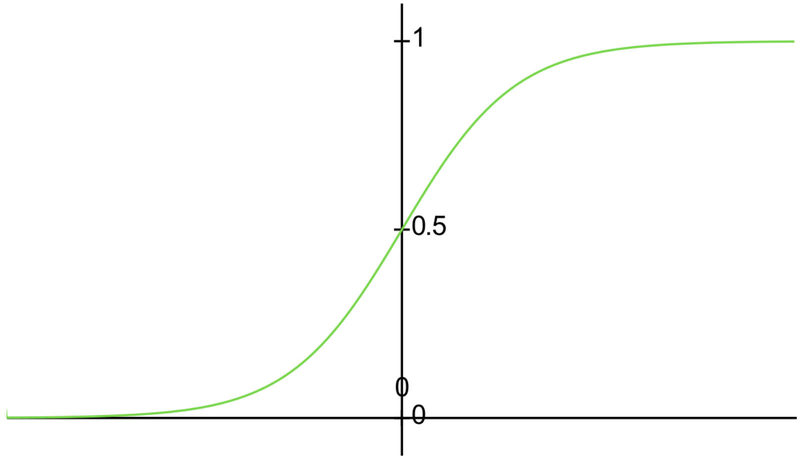
Perceptron

Activation function

$$\hat{y} = h(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$



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

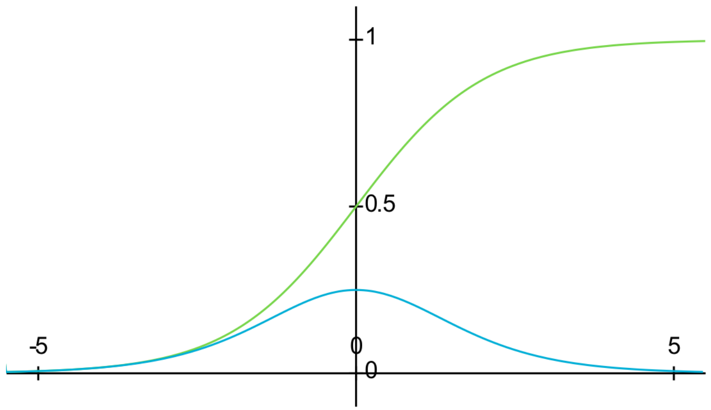


→ Logistic regression

Perceptron

Examples of activation functions

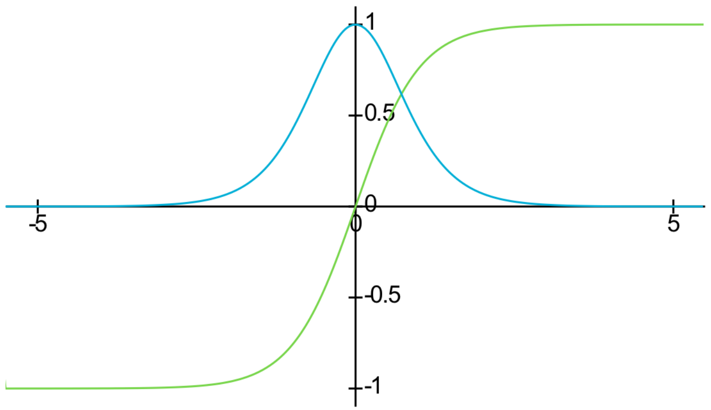
Sigmoid



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

$$h'(z) = h(z)(1 - h(z))$$

Hyperbolic tangent



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

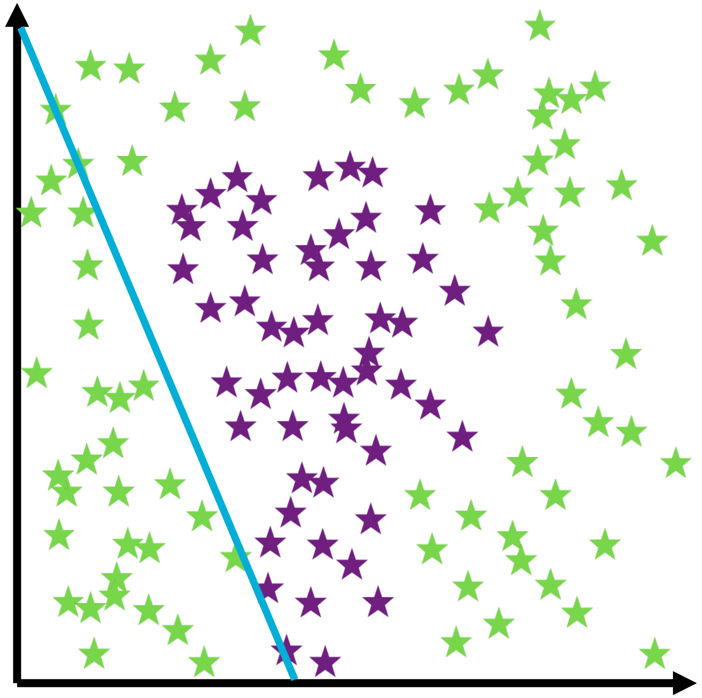
$$h'(z) = 1 - h(z)^2$$

Rectified linear unit (ReLU)

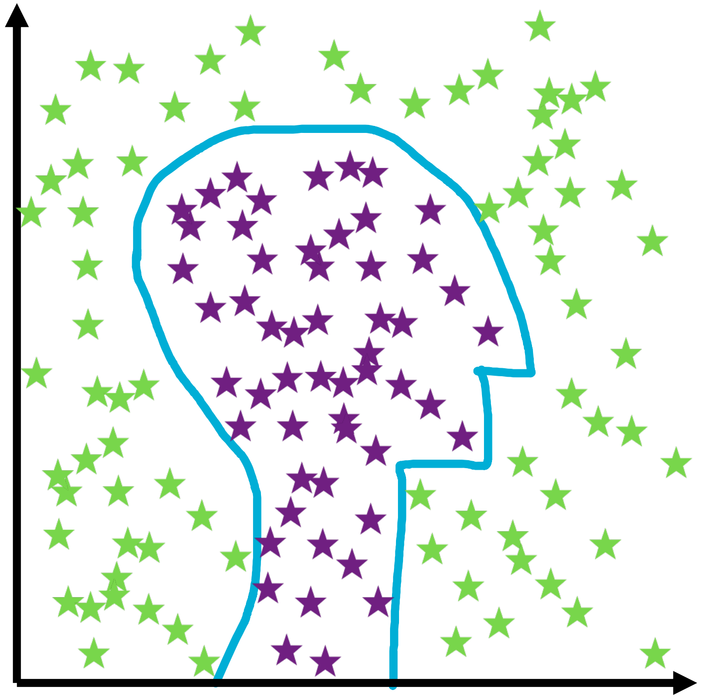
$$h(z) = \max(0, z)$$

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

Importance of non-linear activation functions



Linear activation functions
→ linear decisions

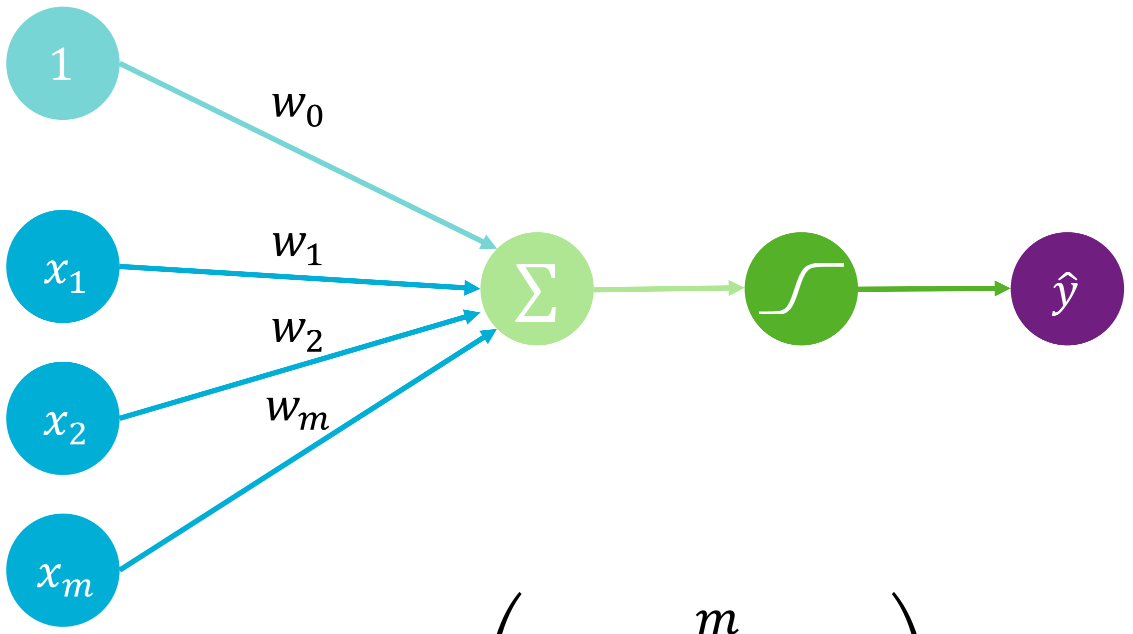


Non-linear activation functions
→ arbitrarily complex decisions

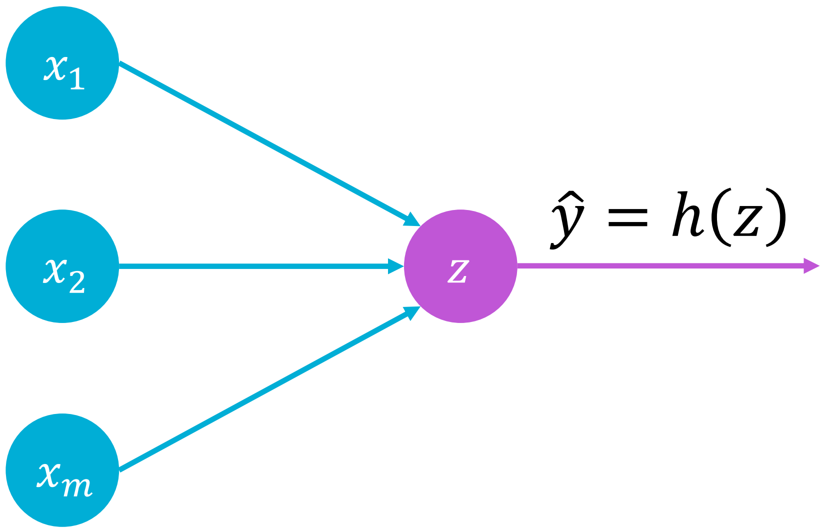
MIT Introduction to Deep Learning (introtodeeplearning.com)

Perceptron

Simplified notation



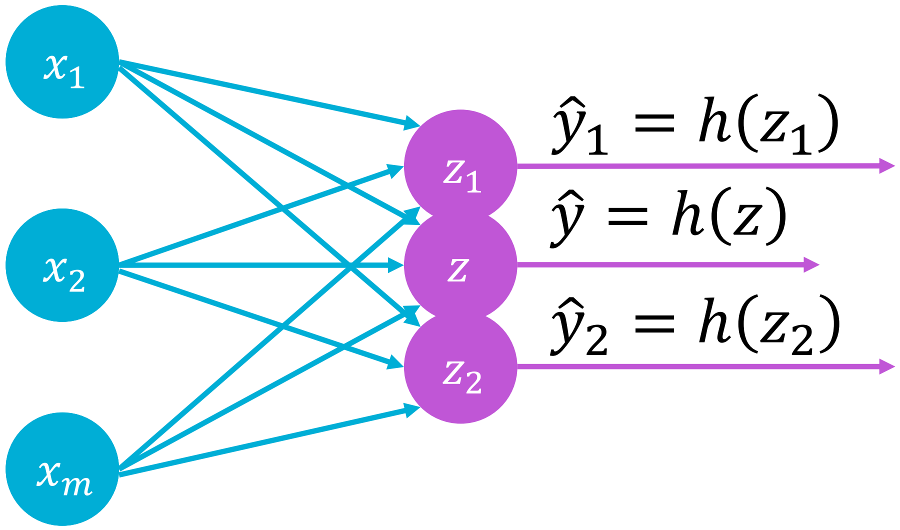
$$\hat{y} = h\left(w_0 + \sum_{j=1}^{m} x_j w_j\right)$$



$$z = w_0 + \sum_{j=1}^{m} x_j w_j$$

Multi output perceptron

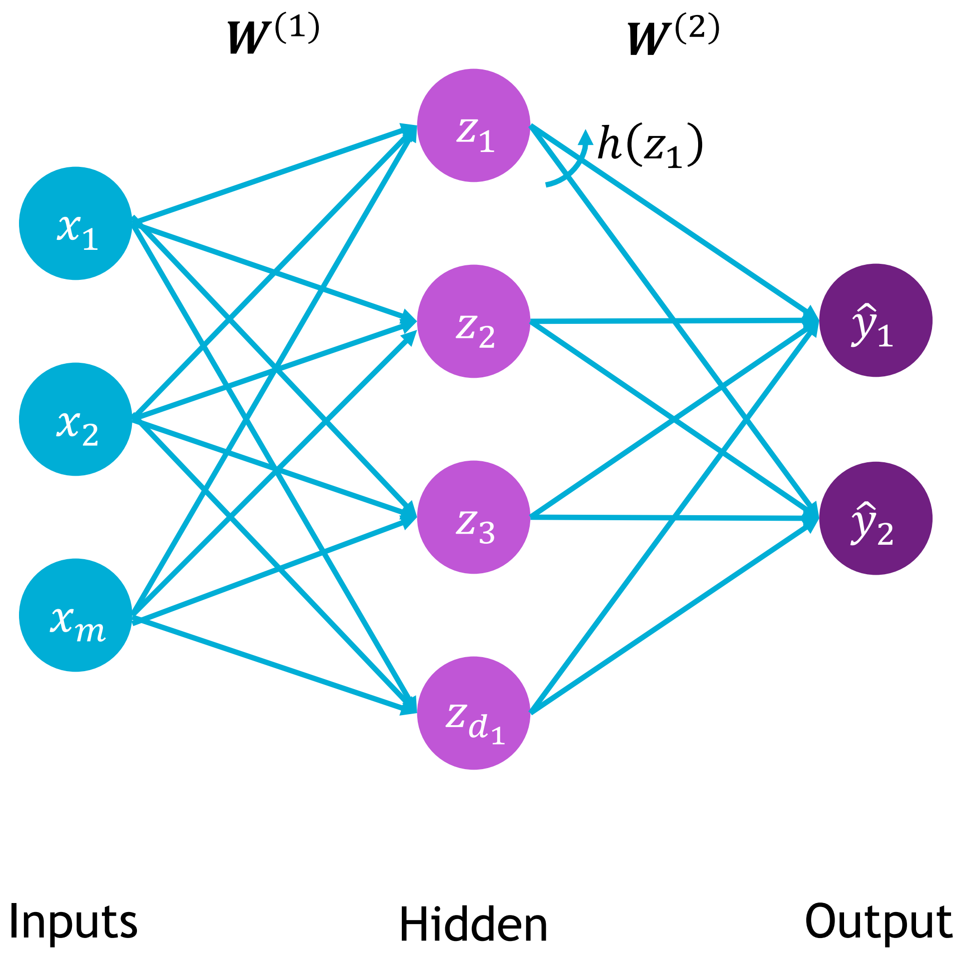
Multi output neural network with dense layers



$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j w_{j,i}$$

MIT Introduction to Deep Learning (introtodeeplearning.com)

Single layer neural network

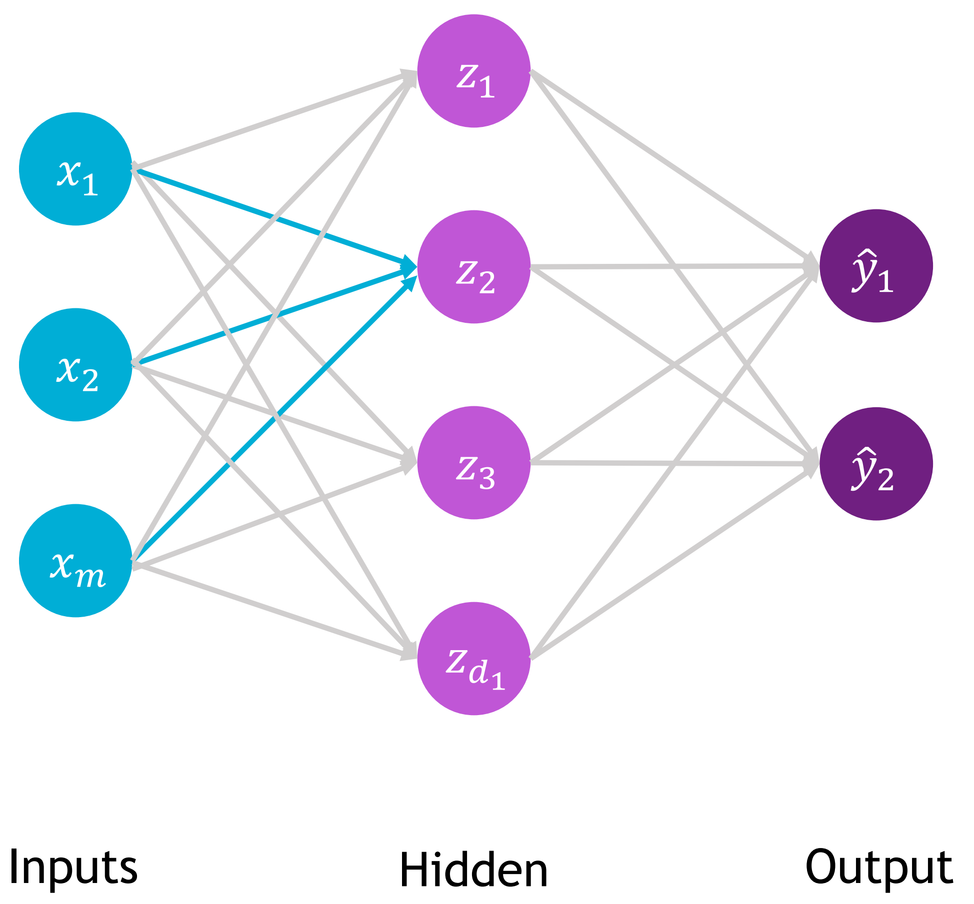


$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j w_{j,i}^{(1)}$$

$$\hat{y}_i = h\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)} \right)$$

MIT Introduction to Deep Learning (introtodeeplearning.com)

Single layer neural network

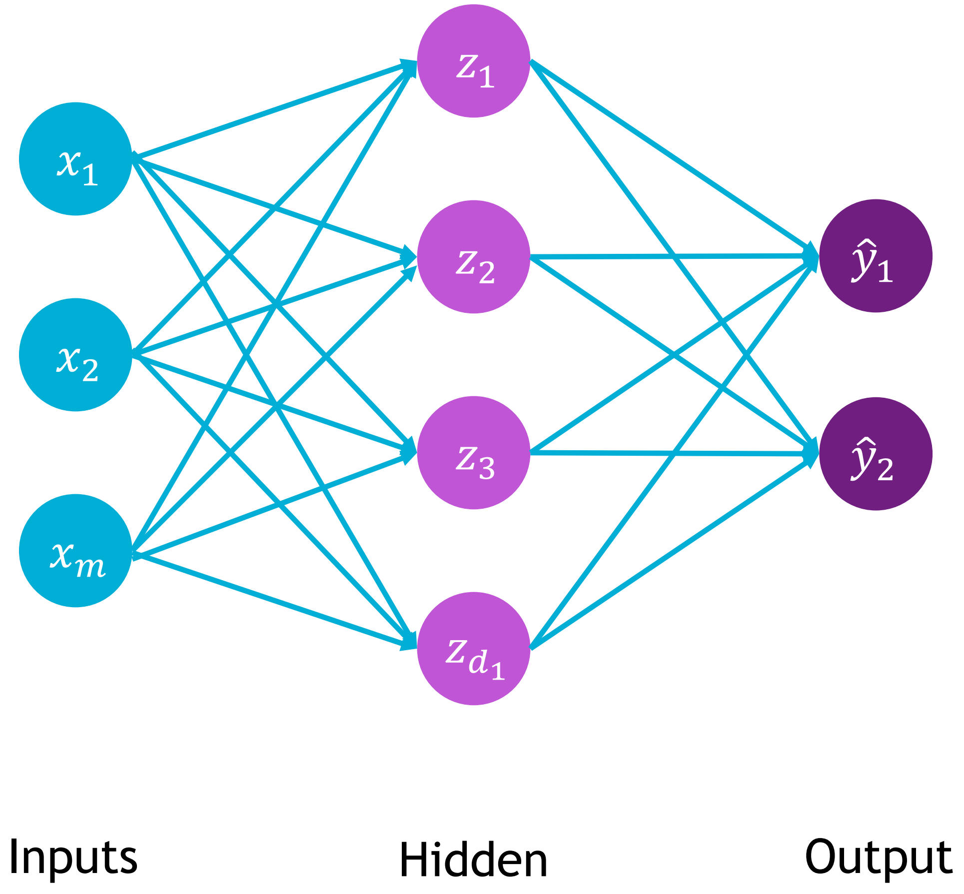


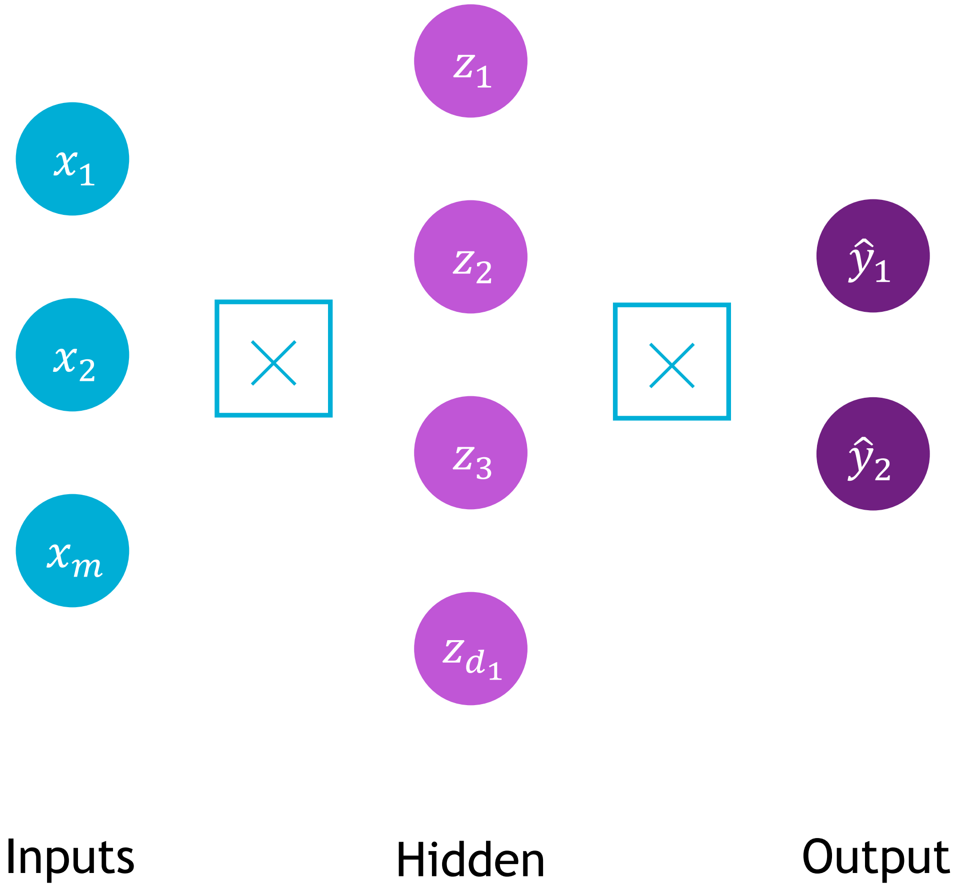
$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j w_{j,2}^{(1)} \text{x}$$

$$= w_{0,2}^{(1)} + x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_m w_{m,2}^{(1)}$$

Single layer neural network

Simplified notation

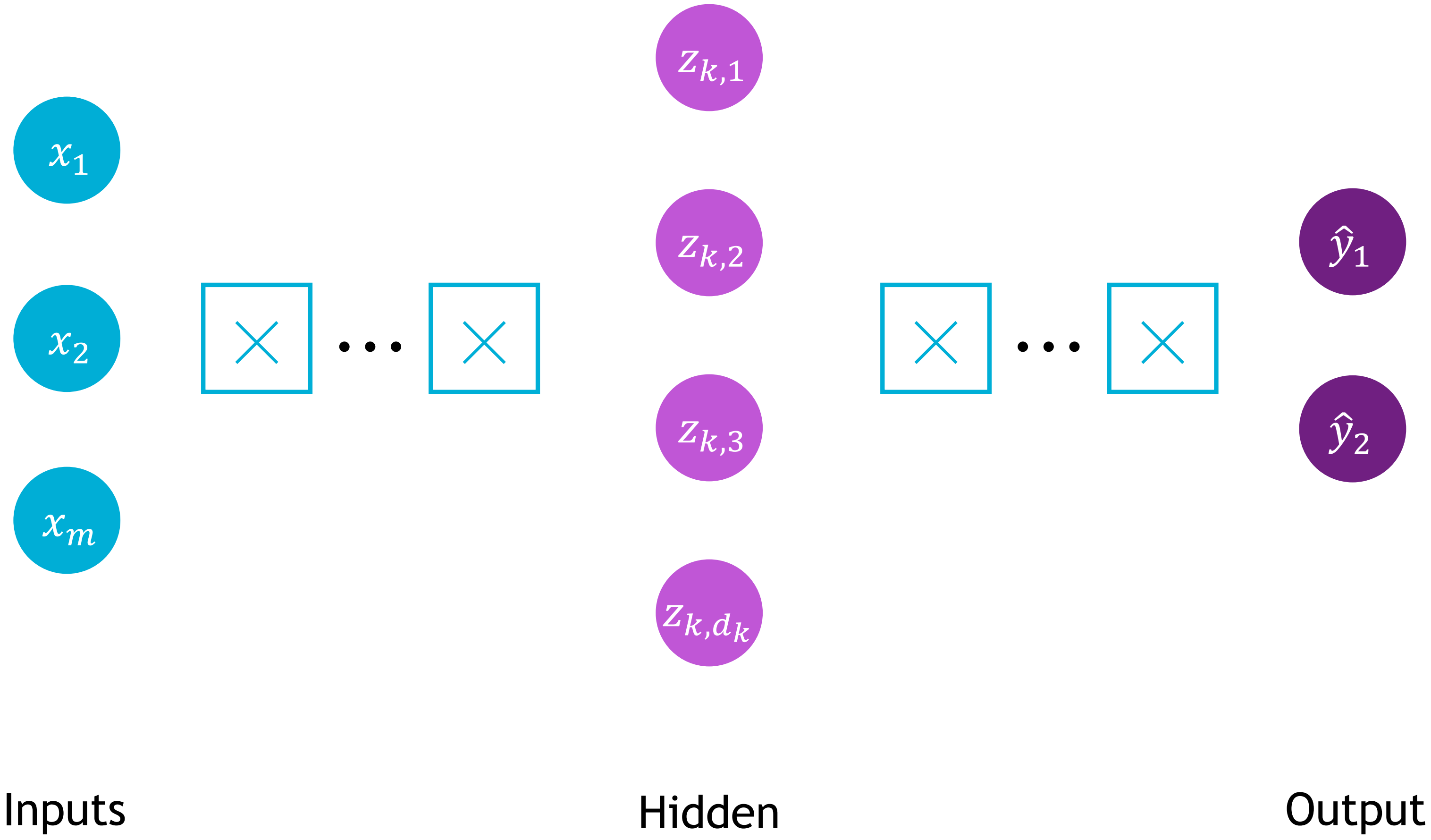




MIT Introduction to Deep Learning (introtodeeplearning.com)

Deep neural network

Stacking layers

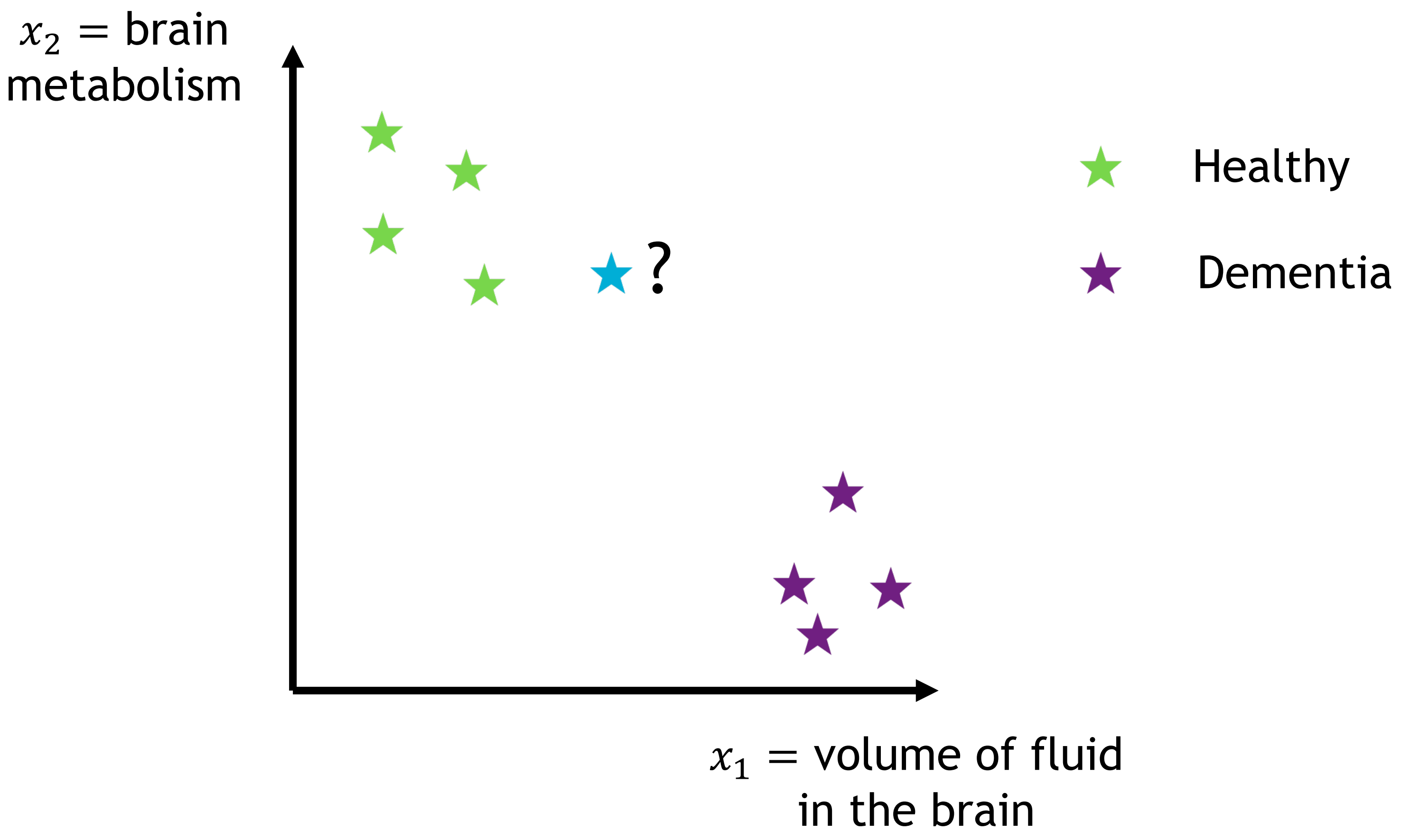


$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} h(z_{k-1,j}) w_{j,i}^{(k)}$$

Applying neural networks

Diagnosis of dementia based on imaging biomarkers

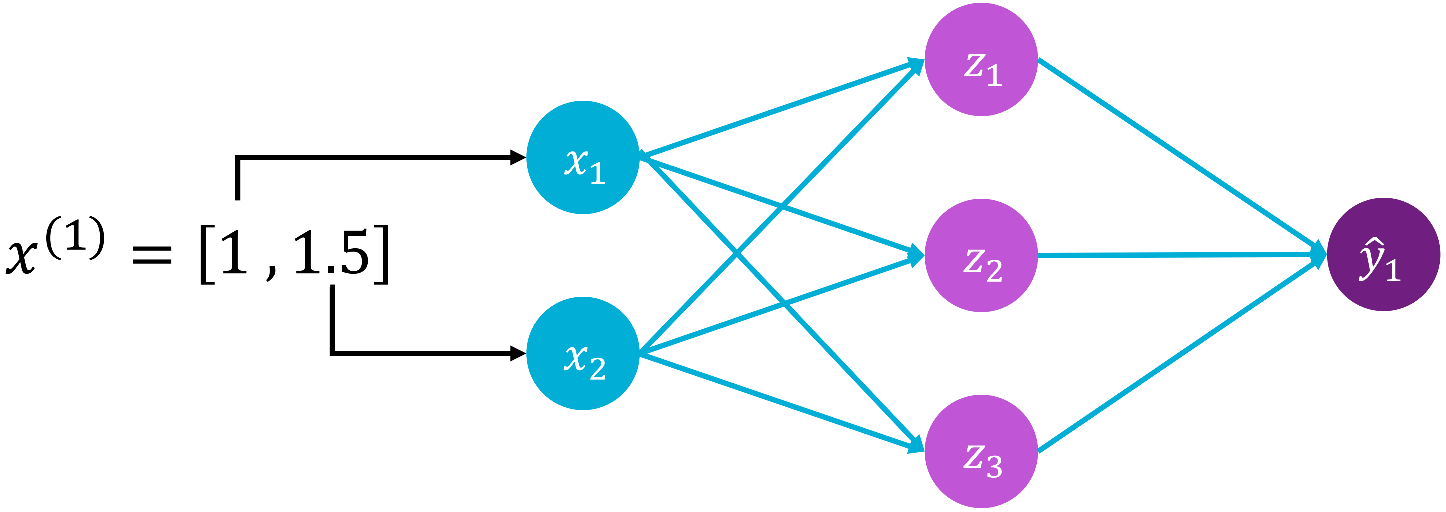
Is this subject healthy?



Applying neural networks

Diagnosis of dementia based on imaging biomarkers

Is this subject healthy?



Predicted: 0.1
Actual: 1

Loss: $l\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$

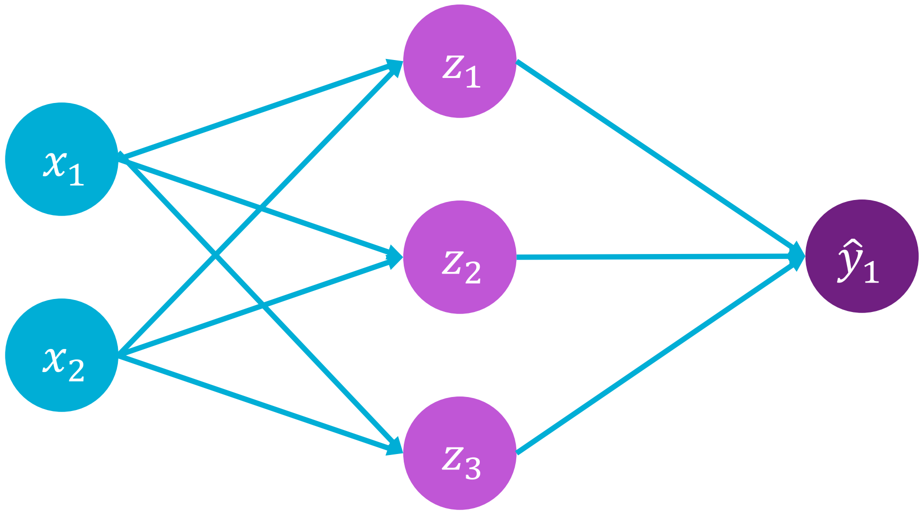
Predicted Actual

Applying neural networks

Diagnosis of dementia based on imaging biomarkers

Is this subject healthy?

$$\boldsymbol{X} = \begin{bmatrix} 1 & 1.5 \\ 3.5 & 1.2 \\ 5 & 0.9 \\ \vdots & \vdots \end{bmatrix}$$



$$f(x) \quad \begin{bmatrix} 0.1 \\ 0.4 \\ 0.6 \\ \vdots \end{bmatrix} \qquad y \quad \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$

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

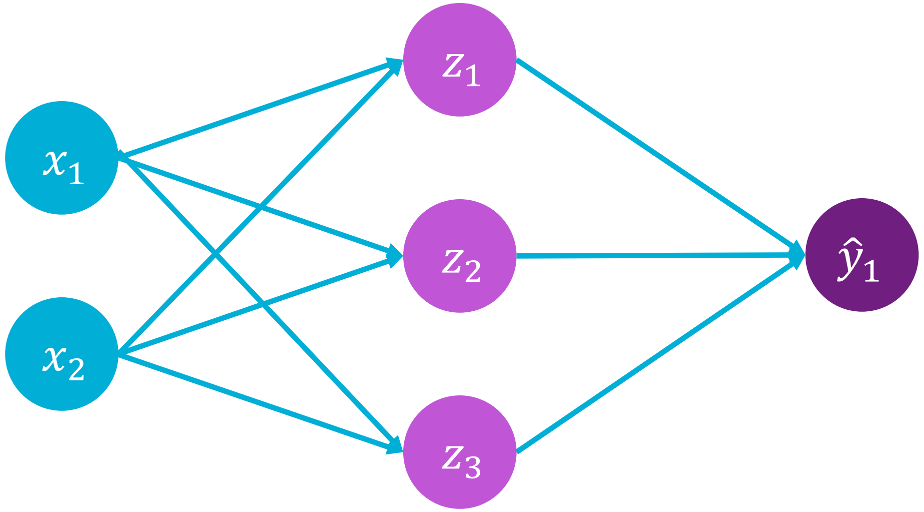
Predicted Actual

MIT Introduction to Deep Learning (introtodeeplearning.com)

Diagnosis of dementia based on imaging biomarkers

Is this subject healthy?

$$\boldsymbol{X} = \begin{bmatrix} 1 & 1.5 \\ 3.5 & 1.2 \\ 5 & 0.9 \\ \vdots & \vdots \end{bmatrix}$$



$$f(x) = \begin{bmatrix} 0.1 \\ 0.4 \\ 0.6 \\ \vdots \end{bmatrix} \qquad y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{bmatrix}$$

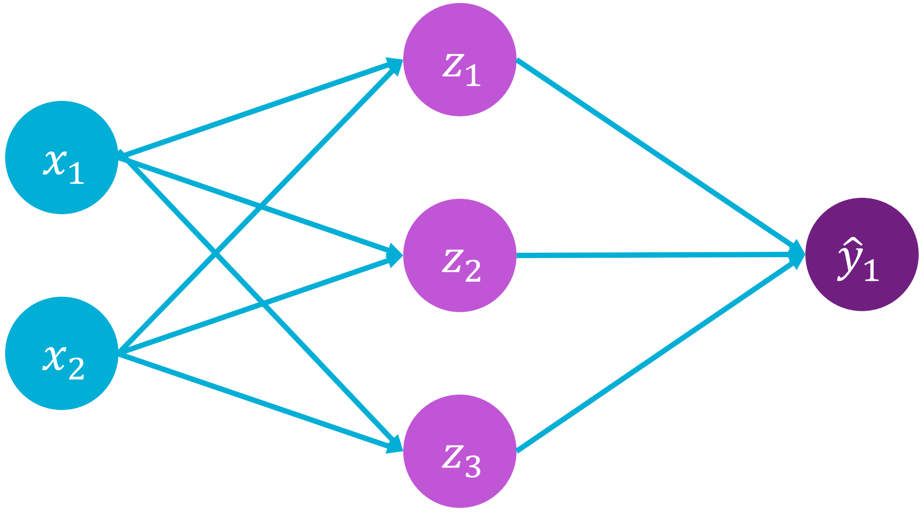
Cost function with cross entropy loss:

$$J(\boldsymbol{W}) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{-y^{(i)}}_{\text{Actual}} \log\left(\underbrace{f\left(x^{(i)}; \boldsymbol{W}\right)}_{\text{Predicted}}\right) - \left(1 - \underbrace{y^{(i)}}_{\text{Actual}}\right) \log\left(1 - \underbrace{f\left(x^{(i)}; \boldsymbol{W}\right)}_{\text{Predicted}}\right)$$

Diagnosis of dementia based on imaging biomarkers

What is the dementia severity?

$$\boldsymbol{X} = \begin{bmatrix} 1 & 1.5 \\ 3.5 & 1.2 \\ 5 & 0.9 \\ \vdots & \vdots \end{bmatrix}$$



$f(x)$ $\begin{bmatrix} 3 \\ 1 \\ 8 \\ \vdots \end{bmatrix}$ y $\begin{bmatrix} 0 \\ 2 \\ 7 \\ \vdots \end{bmatrix}$

Cost function with mean squared error loss:

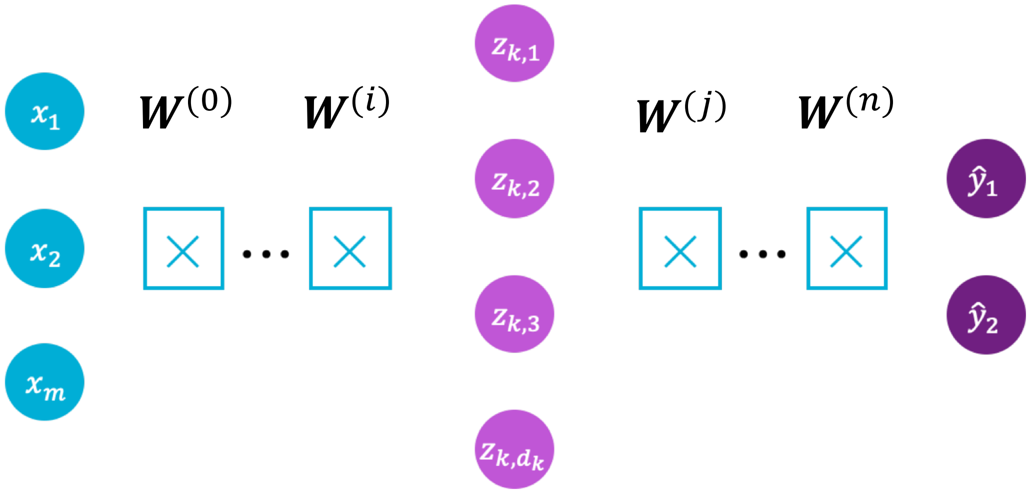
$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\left(\underbrace{y^{(i)}}_{\text{Actual}} - \underbrace{f\left(x^{(i)}; \boldsymbol{W}\right)}_{\text{Predicted}}\right)^2$$

Loss optimisation

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

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

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



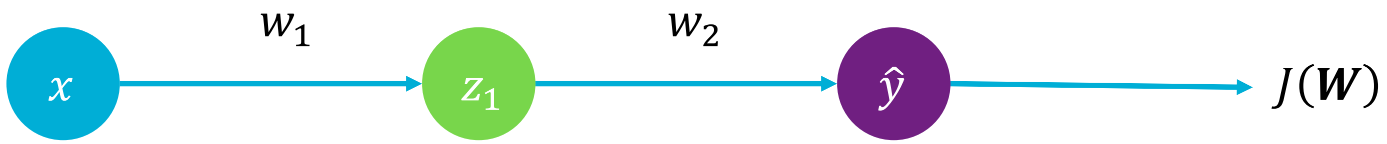
MIT Introduction to Deep Learning (introtodeeplearning.com)

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

Computing gradients: backpropagation



Computing gradients: backpropagation

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

Training Neural Networks

Computing gradients: backpropagation

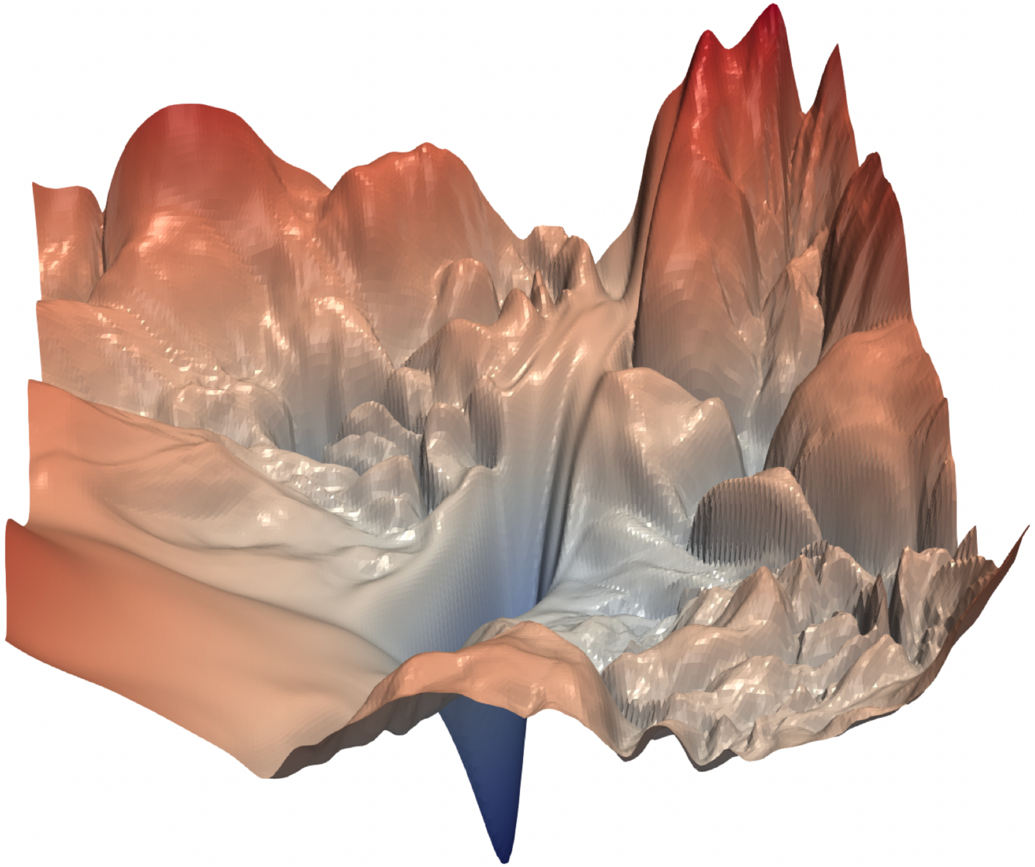
$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_2}$$

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

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

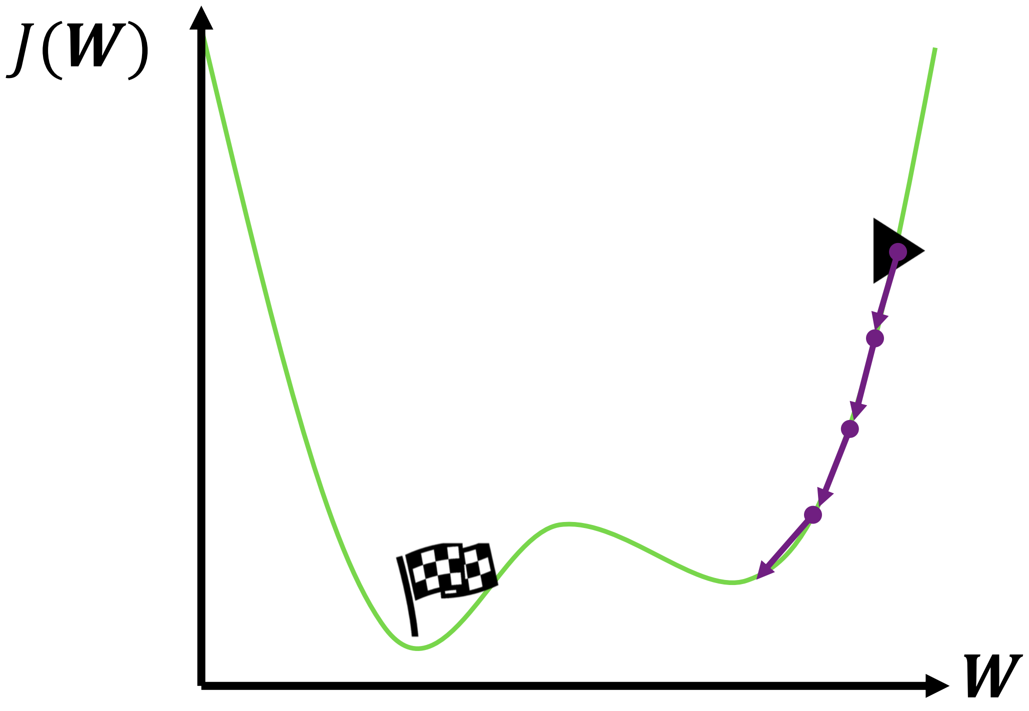


Li et al., Visualizing the Loss Landscape of Neural Nets, NIPS, 2018

Learning rate

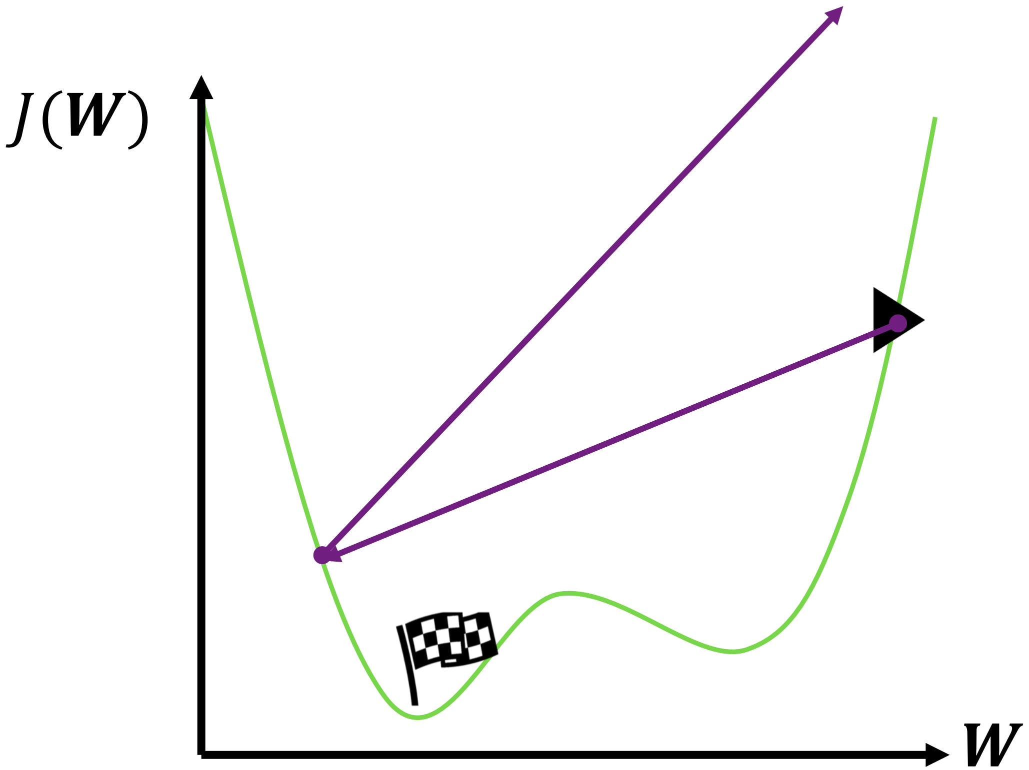
$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

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



Learning rate

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



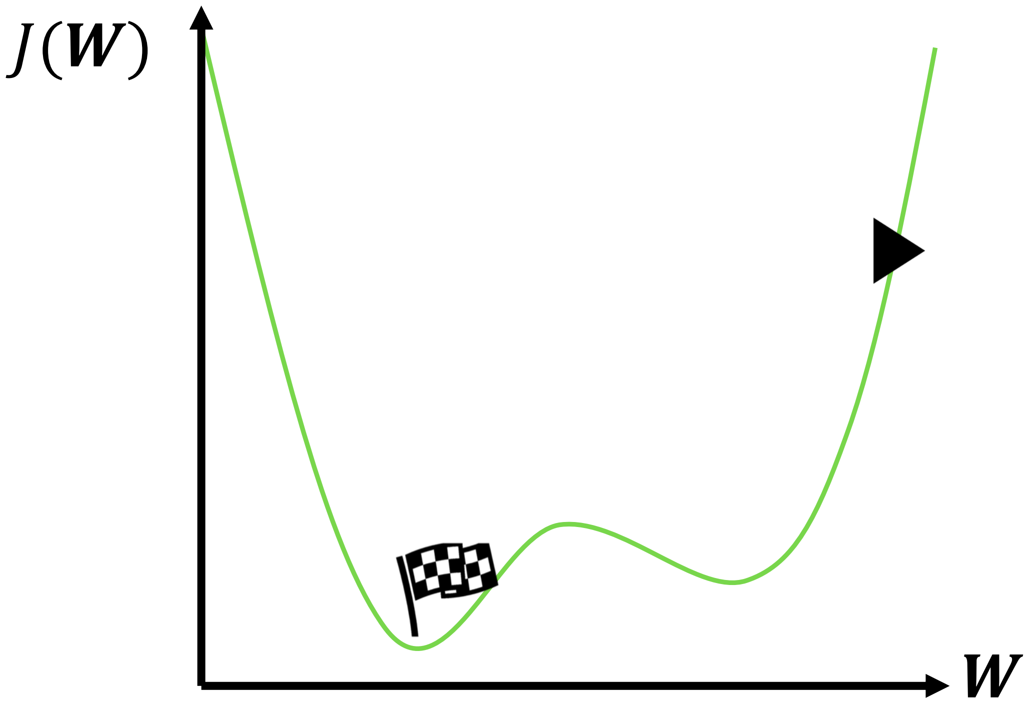
- If η is too small: slow to converge, may be trapped in local minima
- If η is too large: may diverge

Learning rate

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

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

→ Adaptative learning rate



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]

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
   b. Update weights $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
3. Return weights

Stochastic gradient descent with one sample

Algorithm

1. Initialise weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence
   a. Pick single data point i
   b. Compute gradient $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$ Easy 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

Stochastic gradient descent with several samples (mini-batch)

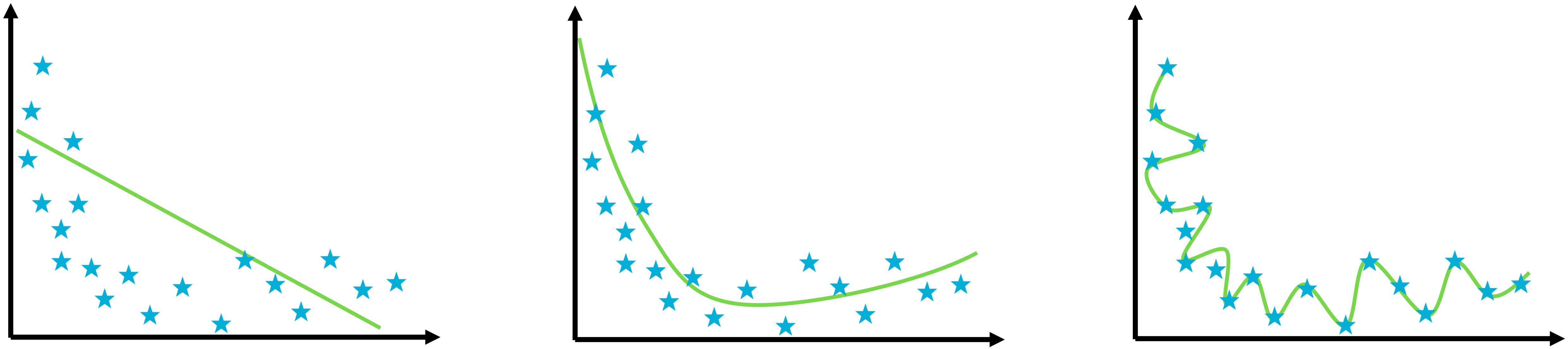
Algorithm

1. Initialise weights randomly $\sim\mathcal{N}(0, \sigma^2)$
2. Loop until convergence
   a. Pick batch of B data points
   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

Fast to compute and good estimate of the true gradient

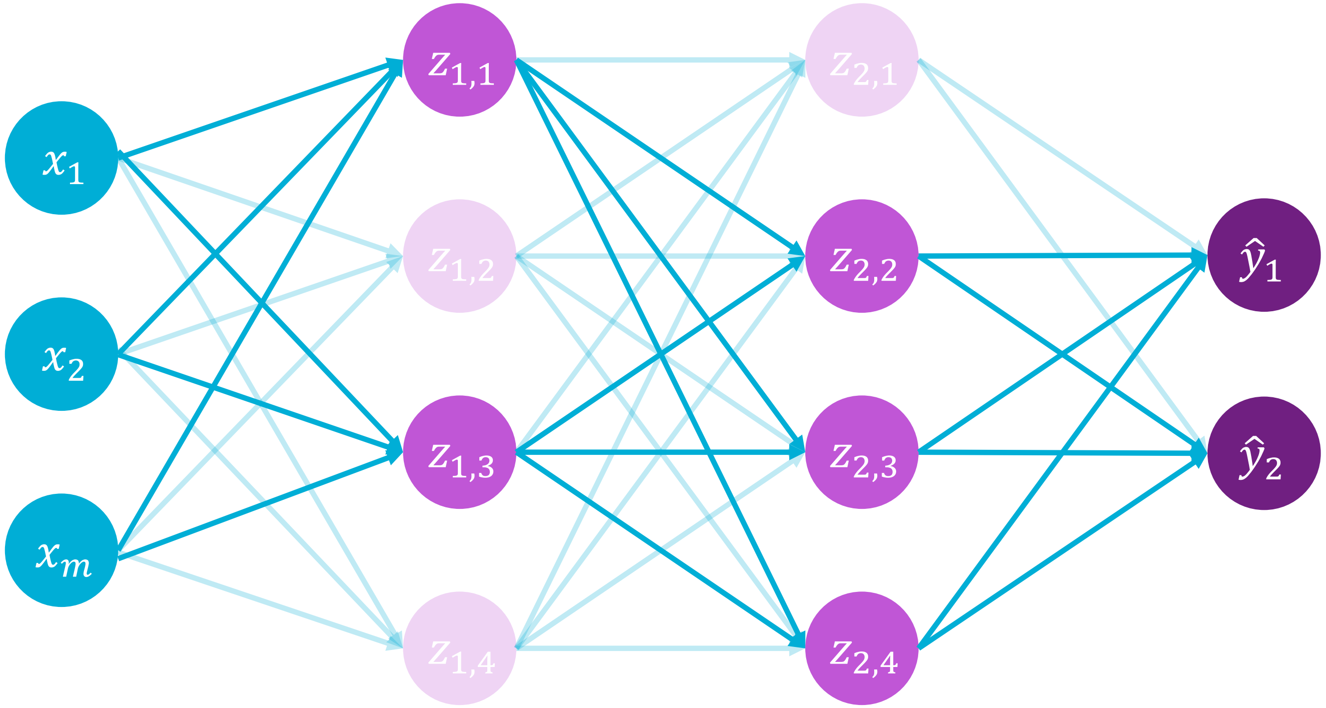
Training Neural Networks

Overfitting



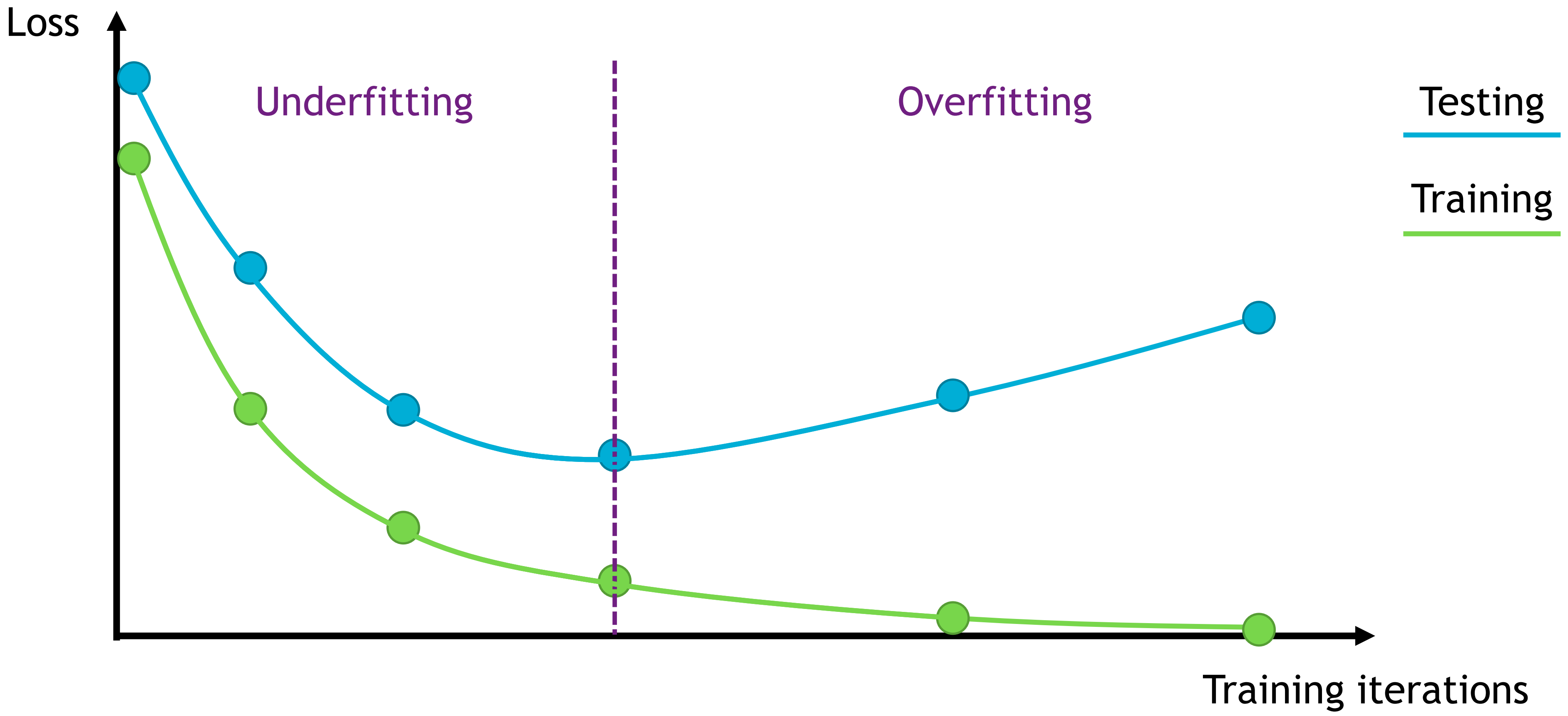
Underfitting   Ideal fit   Overfitting

Regularisation: Dropout



MIT Introduction to Deep Learning (introtodeeplearning.com)

Regularisation: Early stopping

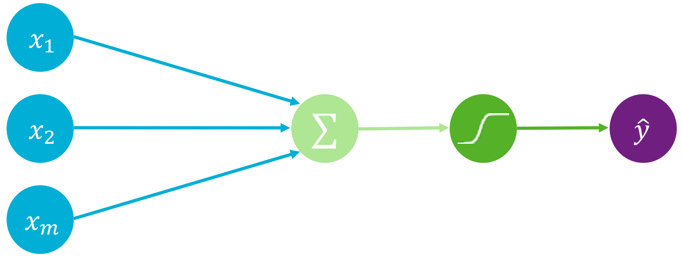


MIT Introduction to Deep Learning (introtodeeplearning.com)

Summary

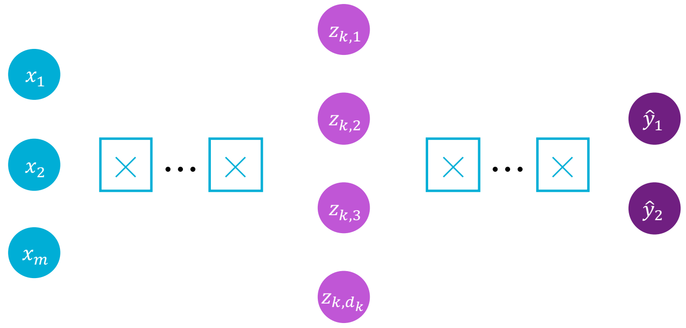
The Perceptron

- Structural building blocks
- Nonlinear activation functions



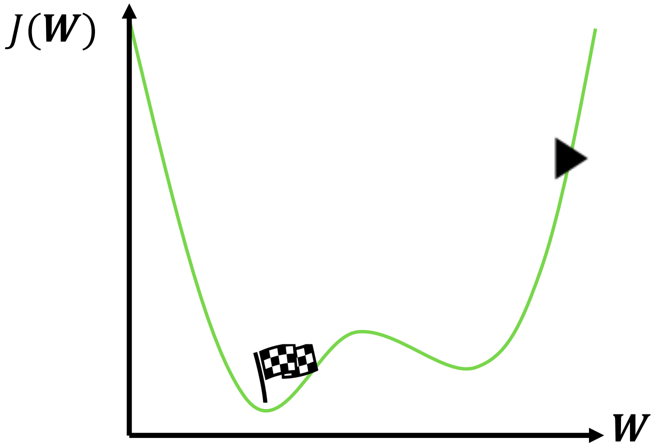
Neural Networks

- Stacking Perceptrons to form neural networks
- Optimisation through backpropagation



Training in Practice

- Adaptive learning
- Batching
- Regularisation

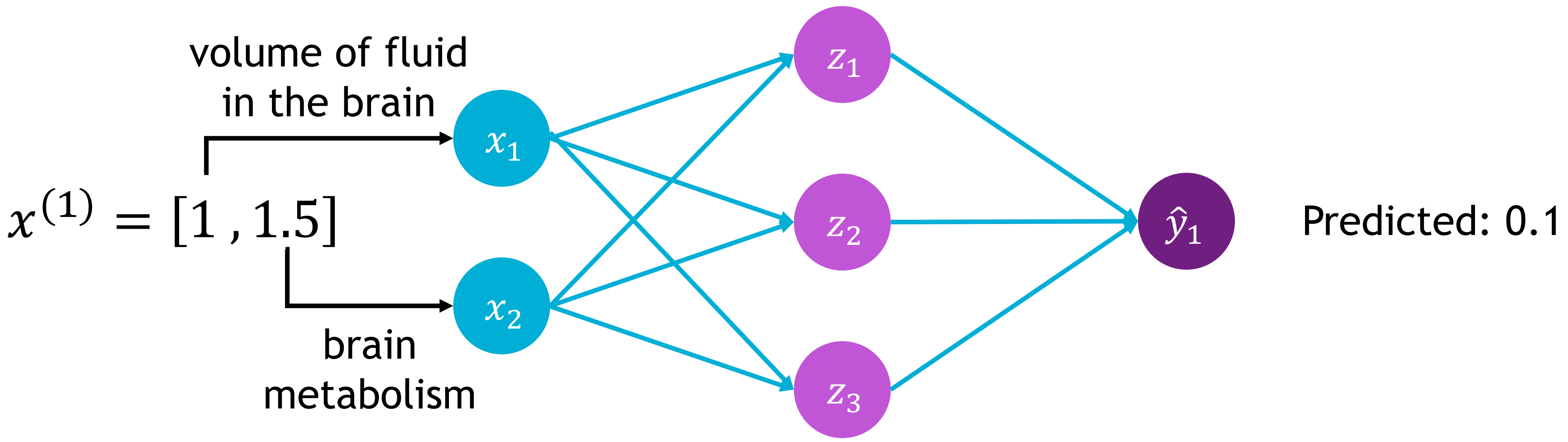


Convolutional neural networks

Applying neural networks

Diagnosis of dementia based on imaging biomarkers

Is this subject healthy?



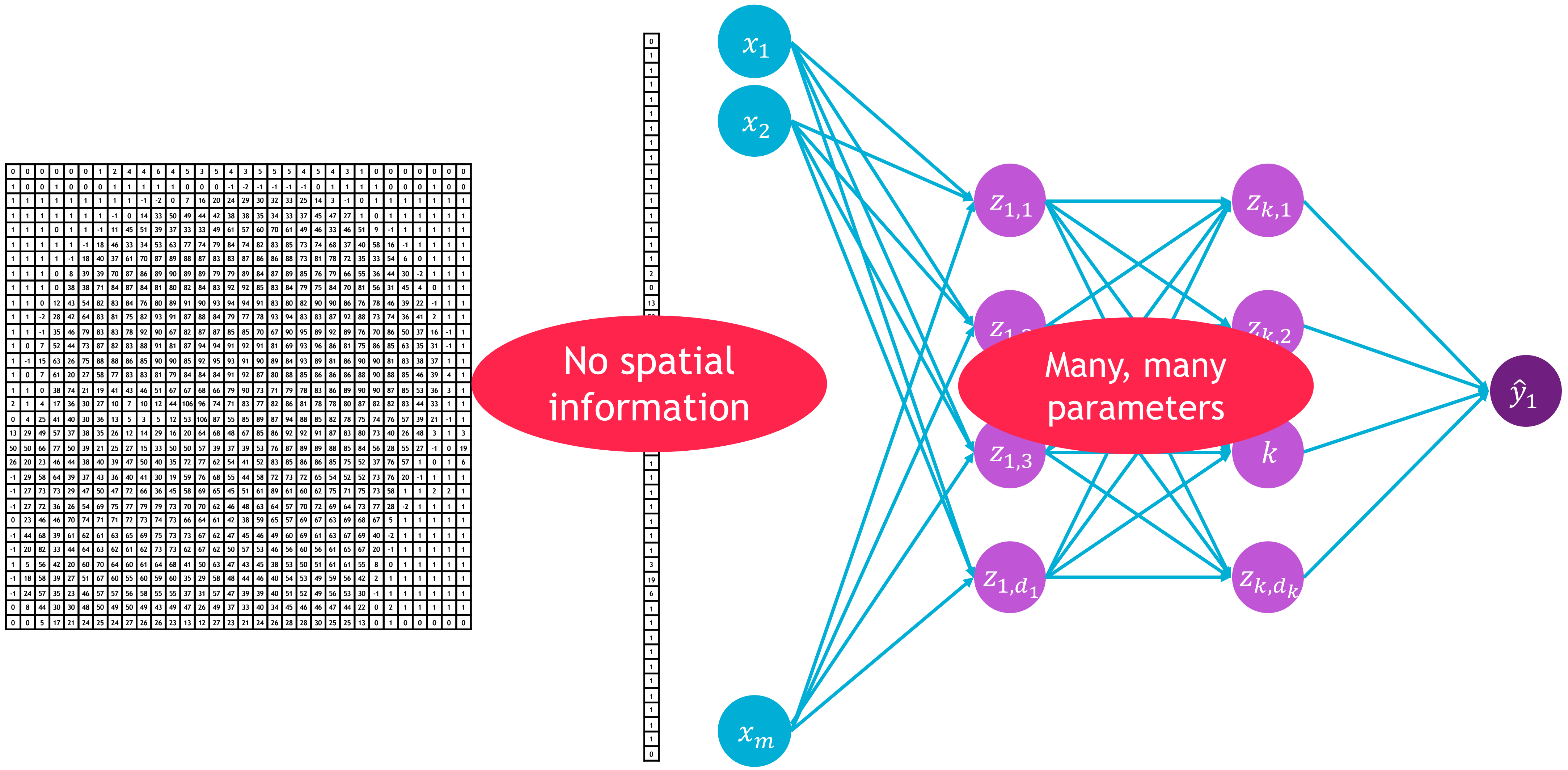
Using an image as input of a neural network

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

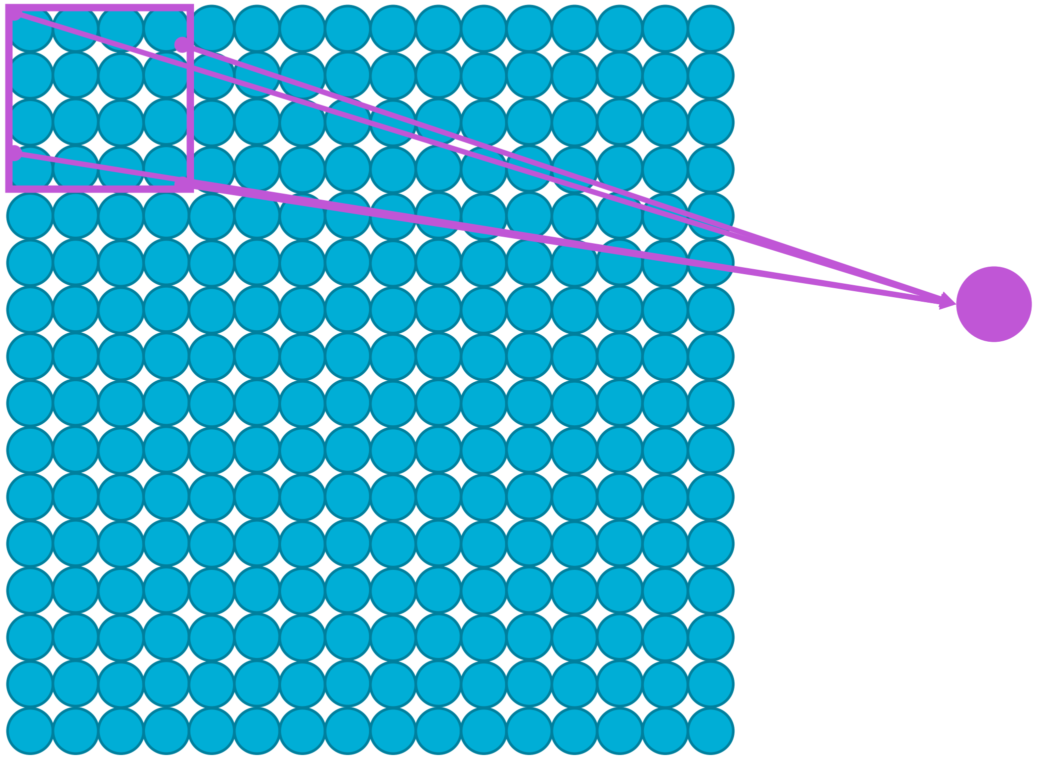
Using an image as input of a neural network

Fully connected neural network



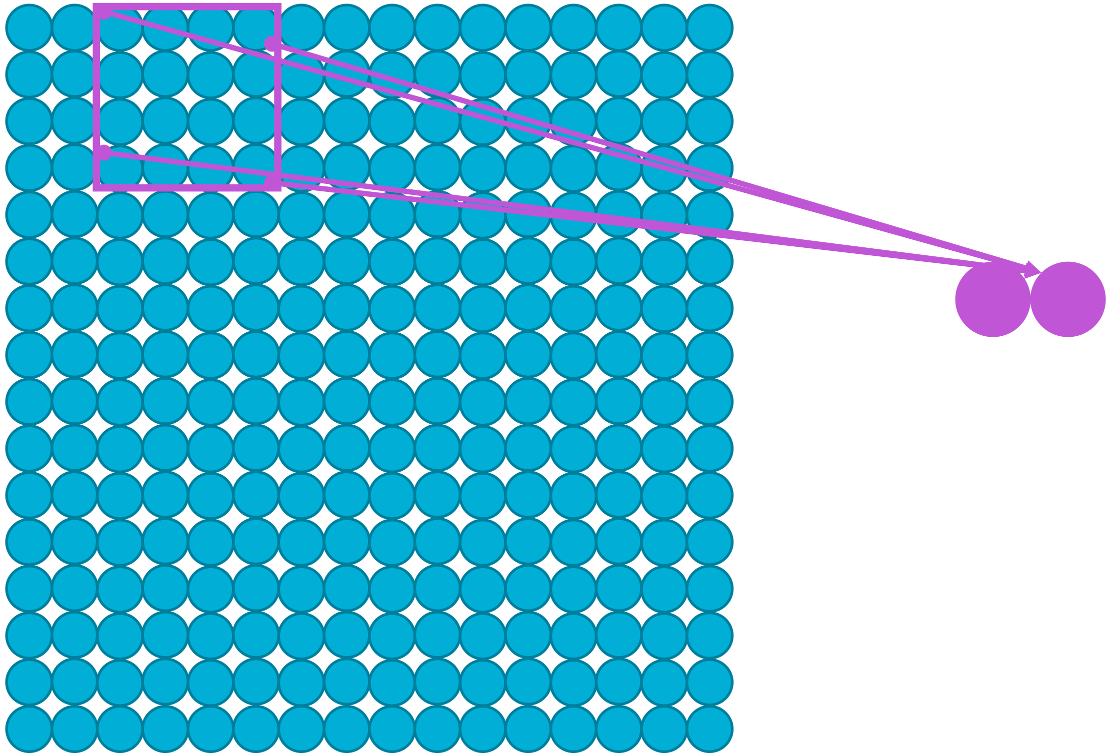
Using an image as input of a neural network

Using spatial features



Idea: connect patches of input to neurons in hidden layer

Using spatial features

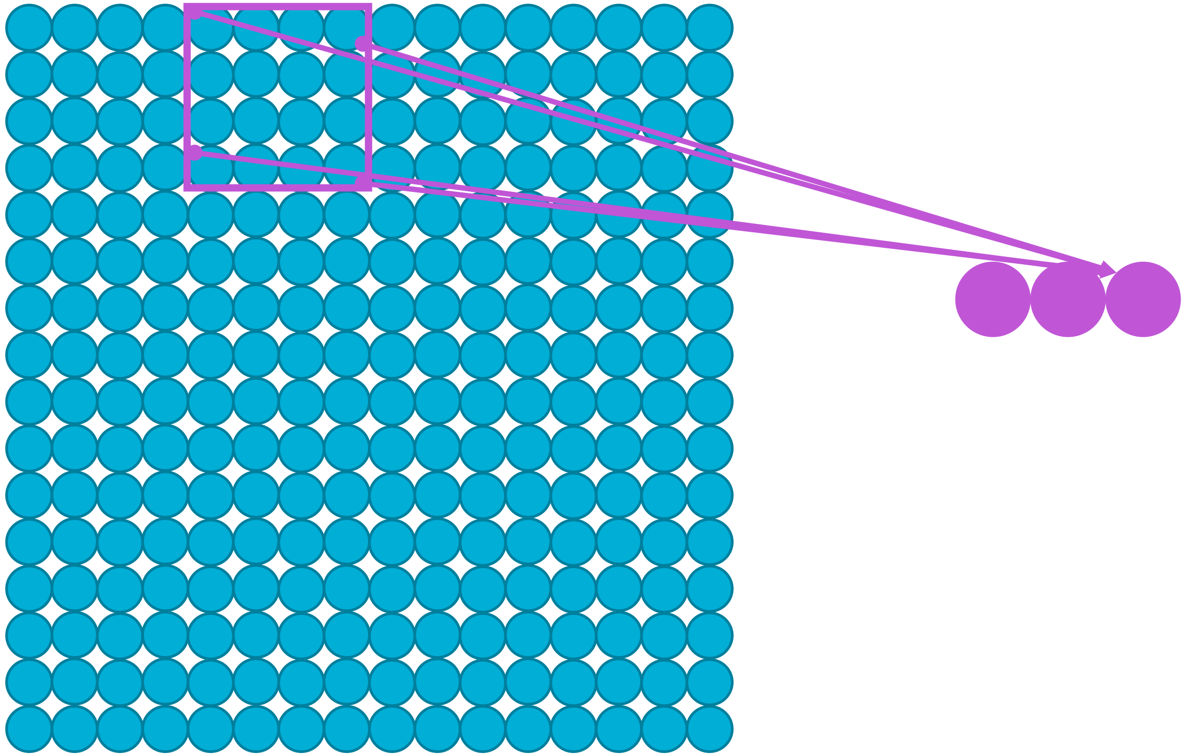


- Slide patch window across input image
- Weight pixels inside the patch
- Apply weighted summation

→ Convolution

Using an image as input of a neural network

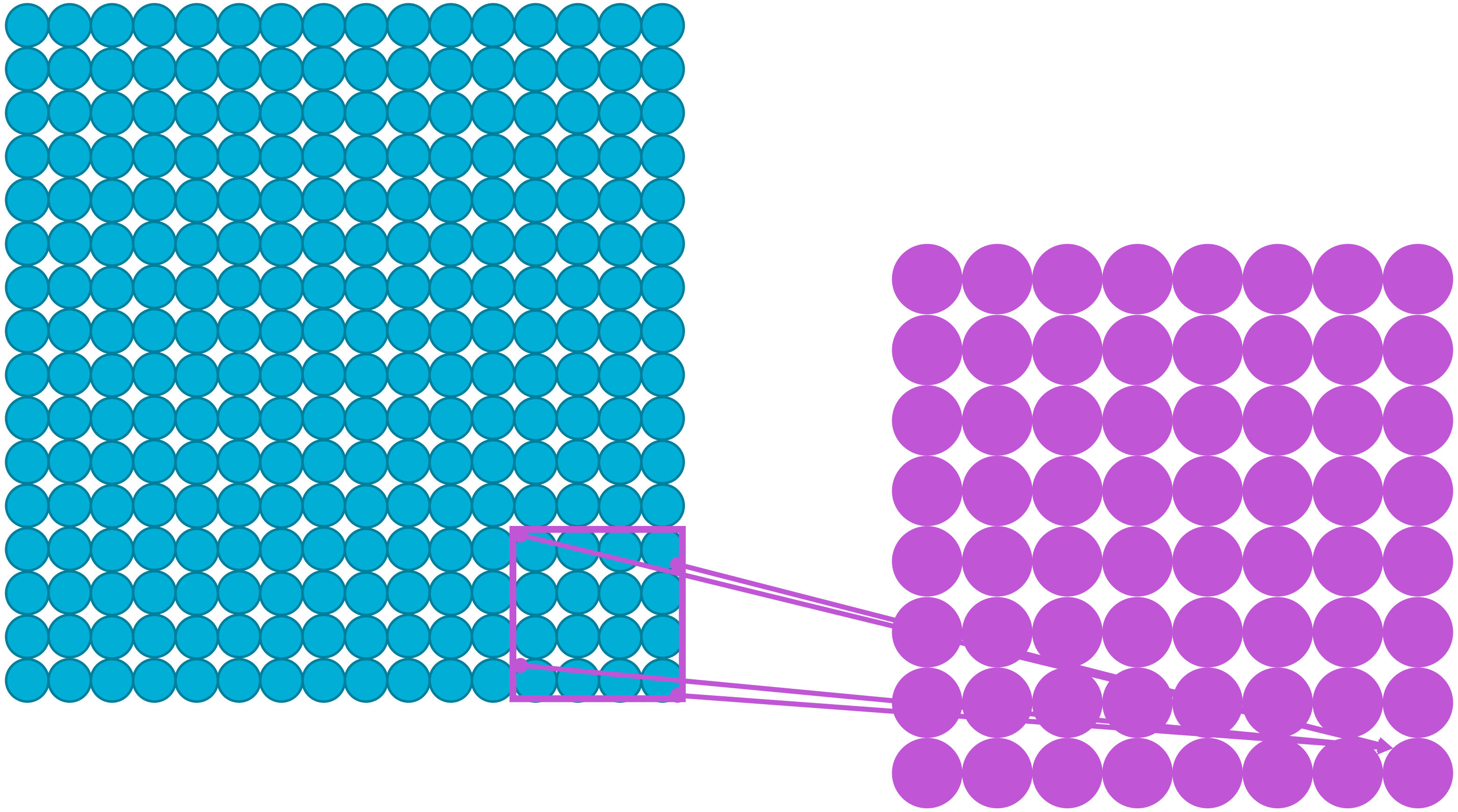
Using spatial features



- Slide patch window across input image
- Weight pixels inside the patch
- Apply weighted summation

→ Convolution

Using spatial features



- Slide patch window across input image
- Weight pixels inside the patch
- Apply weighted summation

→ Convolution

Using an image as input of a neural network

The convolution operation

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

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

$\otimes$

1	0	1
0	1	0
1	0	1

Filter

The convolution operation

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

$$
\begin{aligned}
& 1\times1 + 1\times0 + 1\times1 \\
&+ 0\times0 + 1\times1 + 1\times0 \\
&+ 0\times1 + 0\times0 + 1\times1 \\
&= 4
\end{aligned}
$$

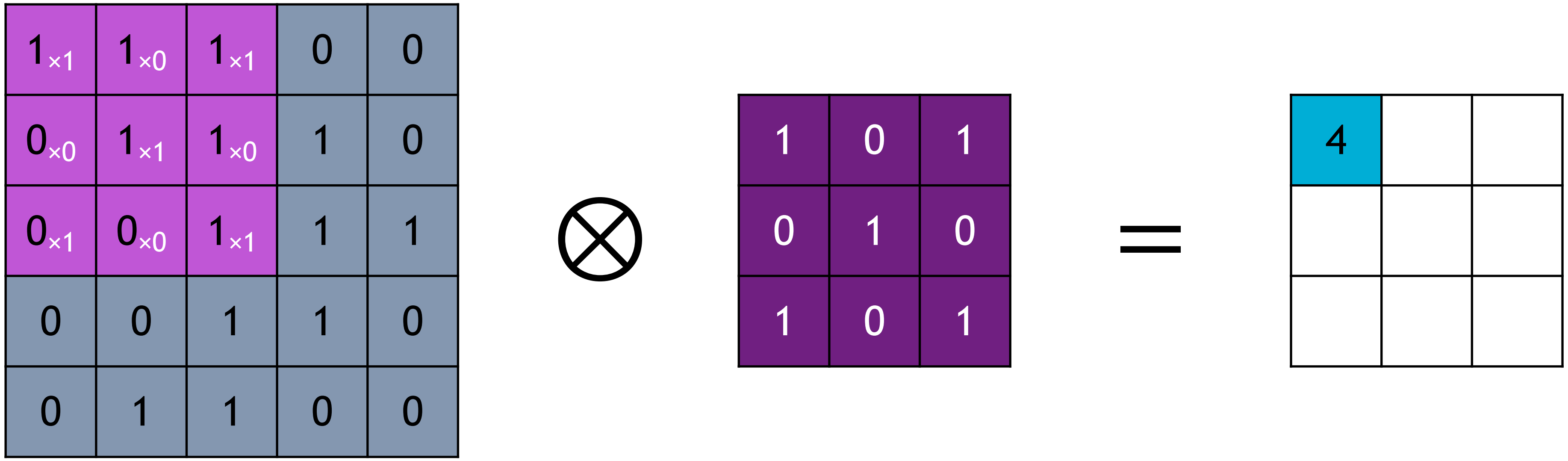


Image   Filter   Feature map

The convolution operation

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

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

1	$1_{\times1}$	$1_{\times0}$	$0_{\times1}$	0
0	$1_{\times0}$	$1_{\times1}$	$1_{\times0}$	0
0	$0_{\times1}$	$1_{\times0}$	$1_{\times1}$	1
0	0	1	1	0
0	1	1	0	0

Image

$\otimes$

1	0	1
0	1	0
1	0	1

Filter

$=$

4	3	

Feature map

MIT Introduction to Deep Learning (introtodeeplearning.com)

Using an image as input of a neural network

The convolution operation

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

$$
\begin{aligned}
& 1\times1 + 1\times0 + 1\times1 \\
& + 1\times0 + 1\times1 + 0\times0 \\
& + 1\times1 + 0\times0 + 0\times1 \\
& = 4
\end{aligned}
$$

1	1	1	0	0
0	1	1	1	0
0	0	$1_{\times1}$	$1_{\times0}$	$1_{\times1}$
0	0	$1_{\times0}$	$1_{\times1}$	$0_{\times0}$
0	1	$1_{\times1}$	$0_{\times0}$	$0_{\times1}$

Image

$\otimes$

1	0	1
0	1	0
1	0	1

Filter

=

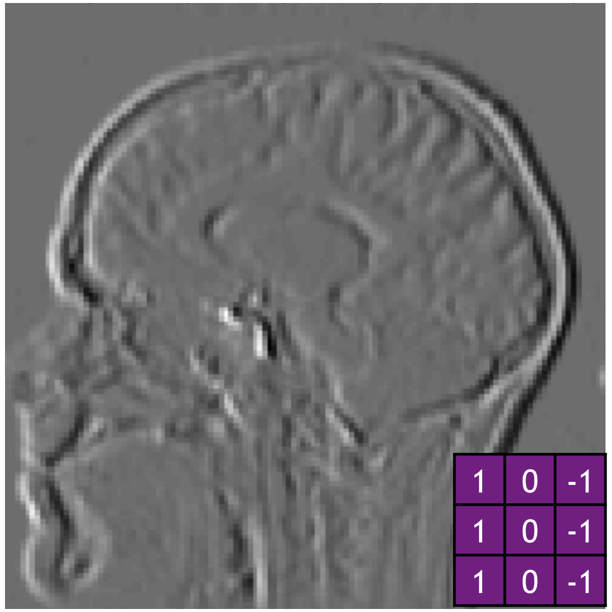
4	3	4
2	4	3
2	3	4

Feature map

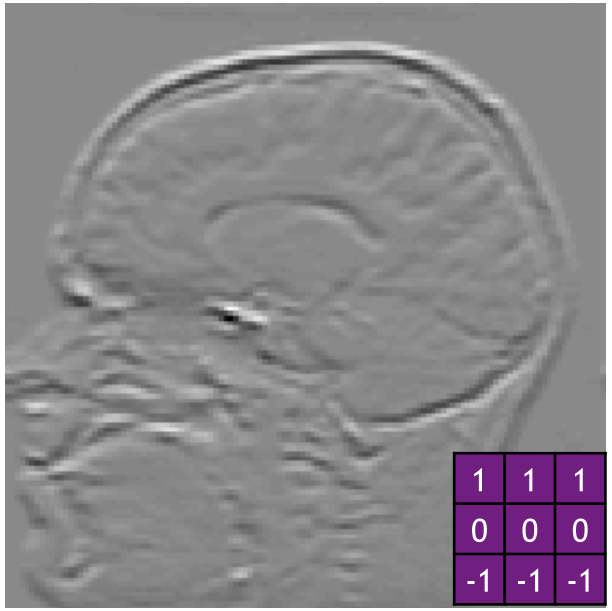
Using an image as input of a neural network

Different filters = different feature maps

Original image

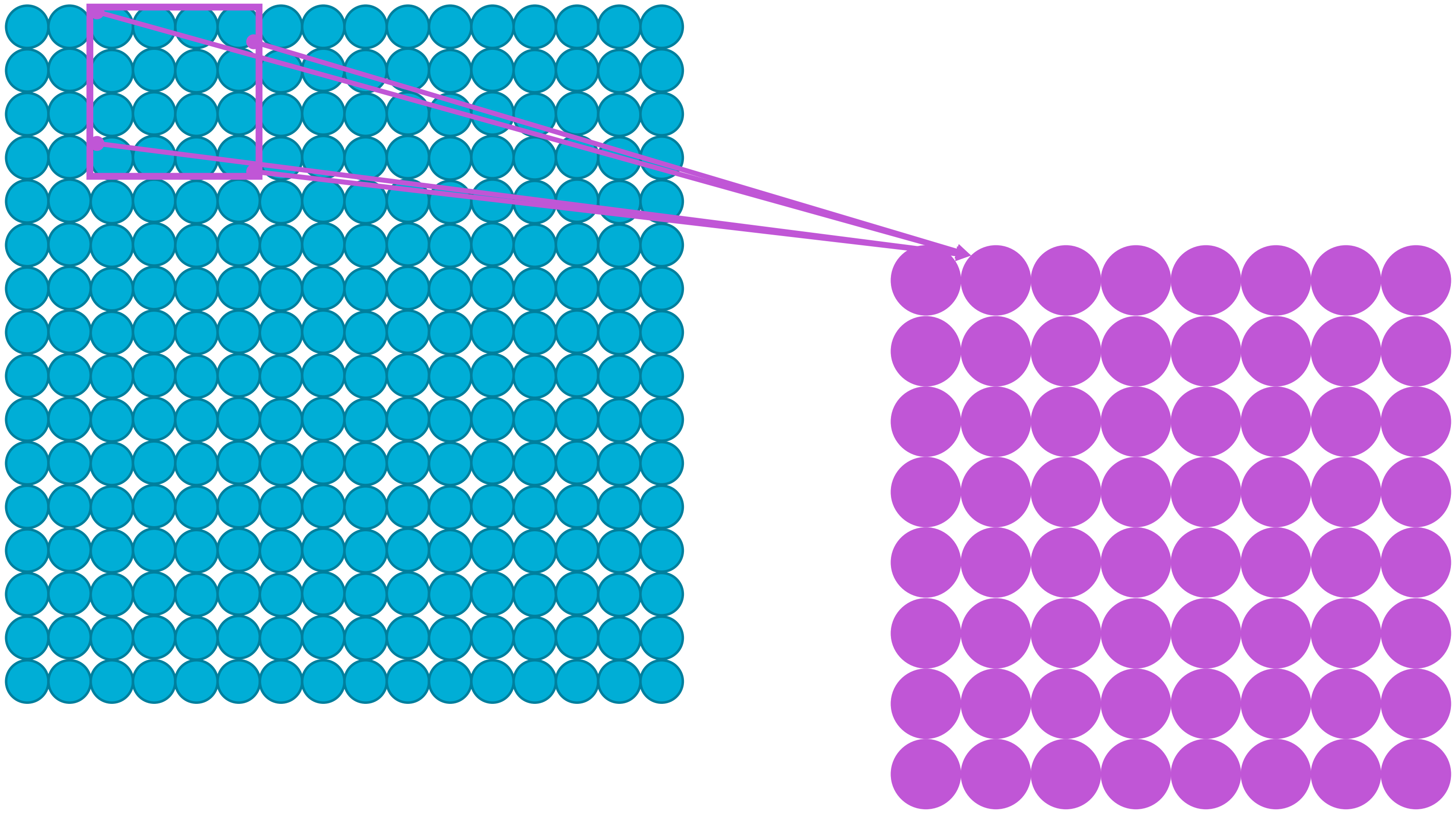


Vertical edge detection



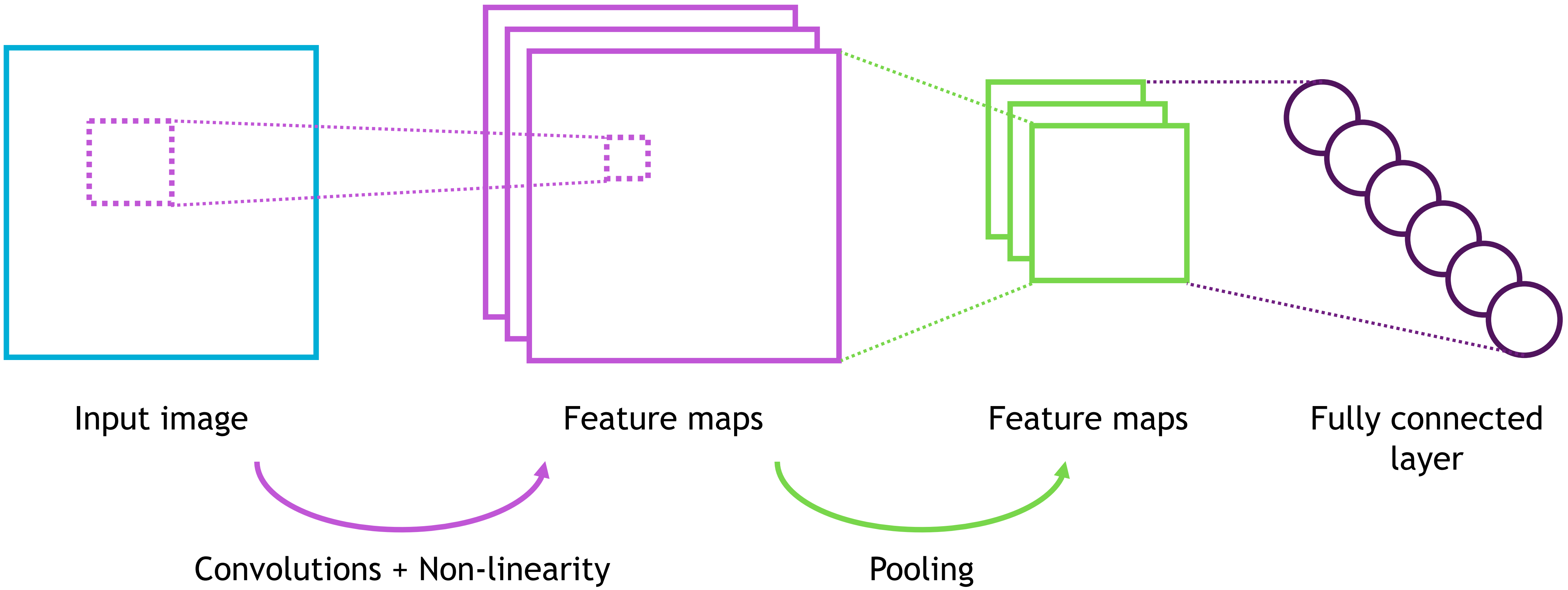
Horizontal edge detection

Using spatial features



- Apply a set of weights – a filter – to extract **local features**
- Use **multiple filters** to extract different features

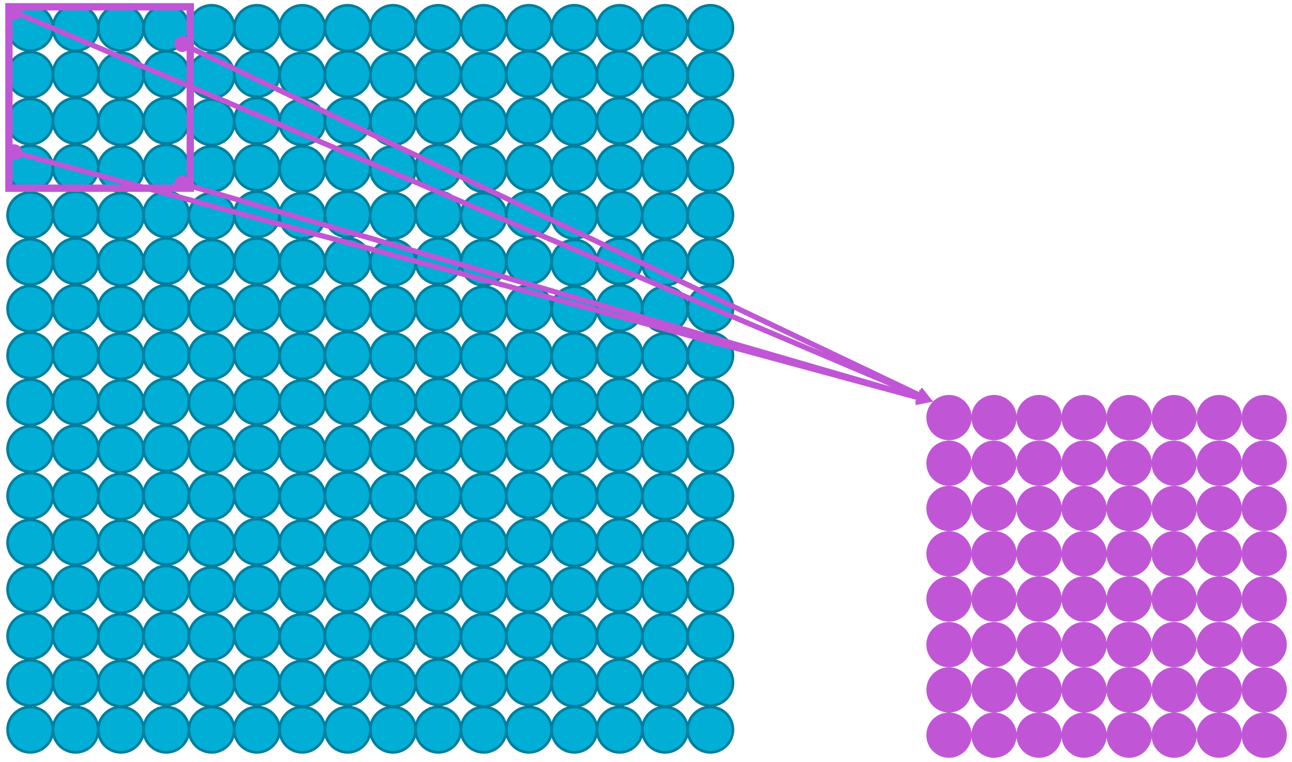
CNNs for classification



MIT Introduction to Deep Learning (introtodeeplearning.com)

Convolutional neural networks

Convolutional layer



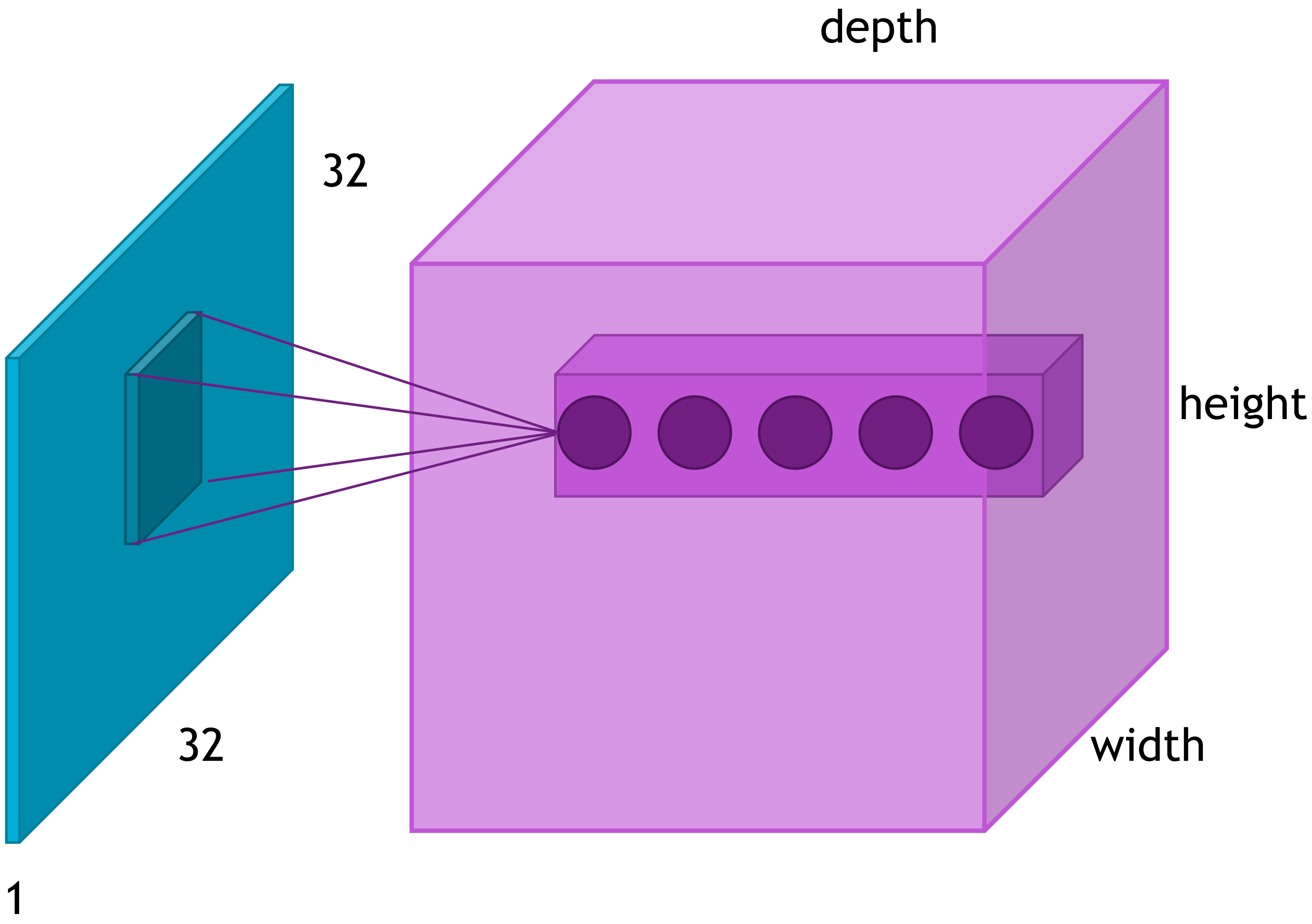
For a neuron in hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias
- Activate with non-linear function

Neuron (p, q) in hidden layer
Filter size 4×4
Weights $w_{i,j}$

$$h\left(\sum_{i=1}^{4}\sum_{j=1}^{4} w_{i,j}x_{i+p,j+q} + b\right)$$

Spatial arrangement of output volume



Layer Dimensions:

$h \times w \times d$

h & w = spatial dimensions
d = number of filters

Stride:

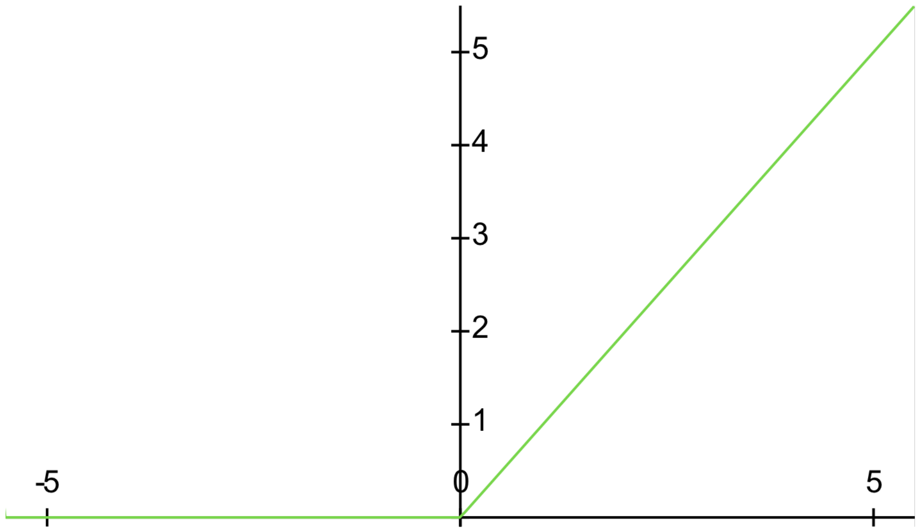
Filter step size

Receptive Field:

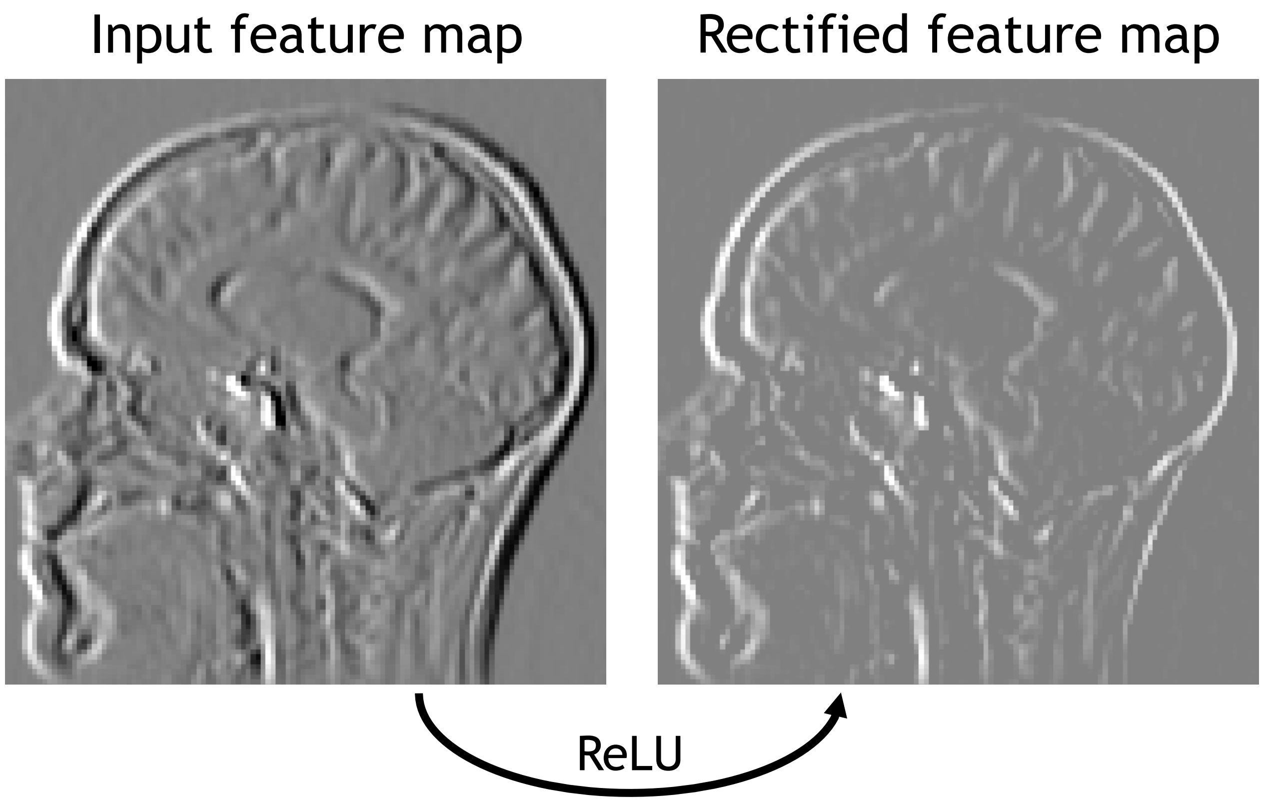
Locations in input image that a node is connected to

Introducing non-linearity

Rectified linear unit (ReLU)



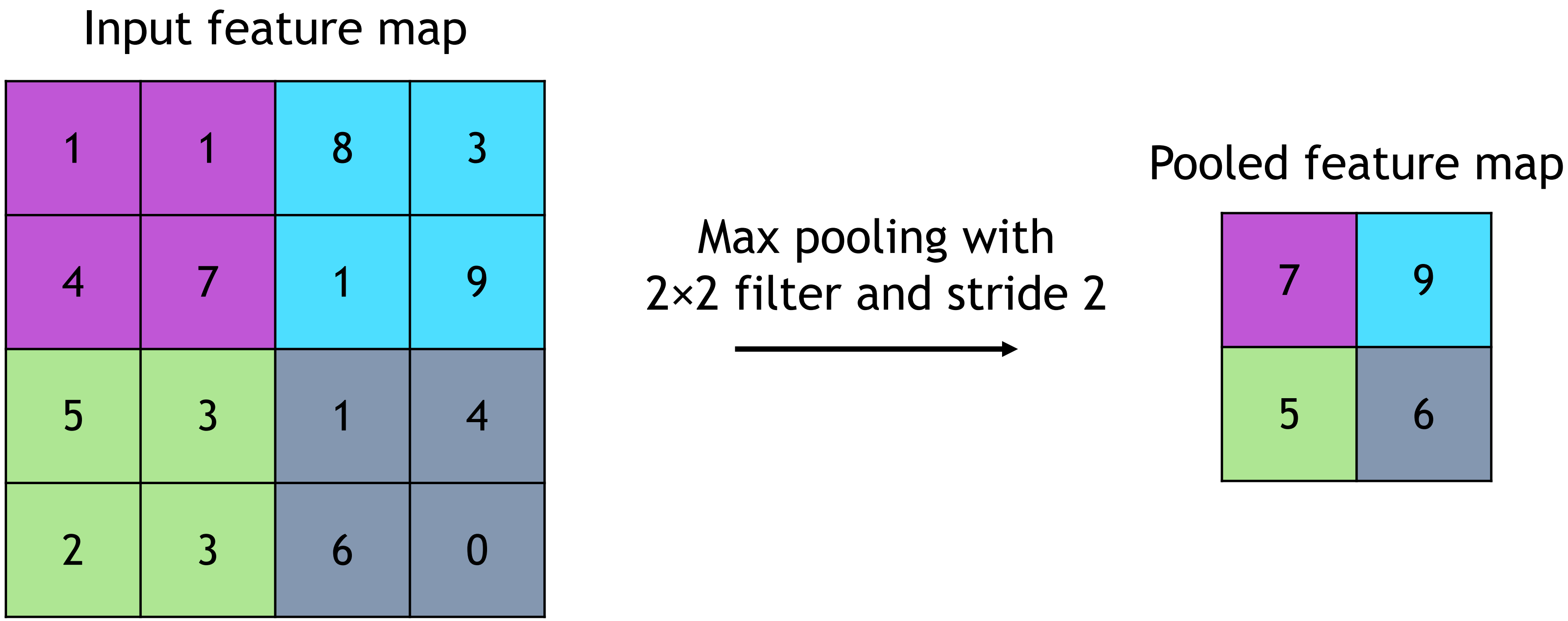
$$h(z) = \max(0, z)$$



Black: negative values - White: positive values

Pooling

- **Reduce dimensionality while preserving spatial invariance**



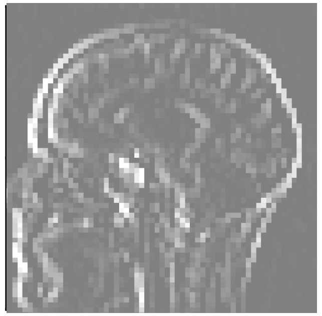
Pooling

- **Reduce dimensionality while preserving spatial invariance**

Input feature map

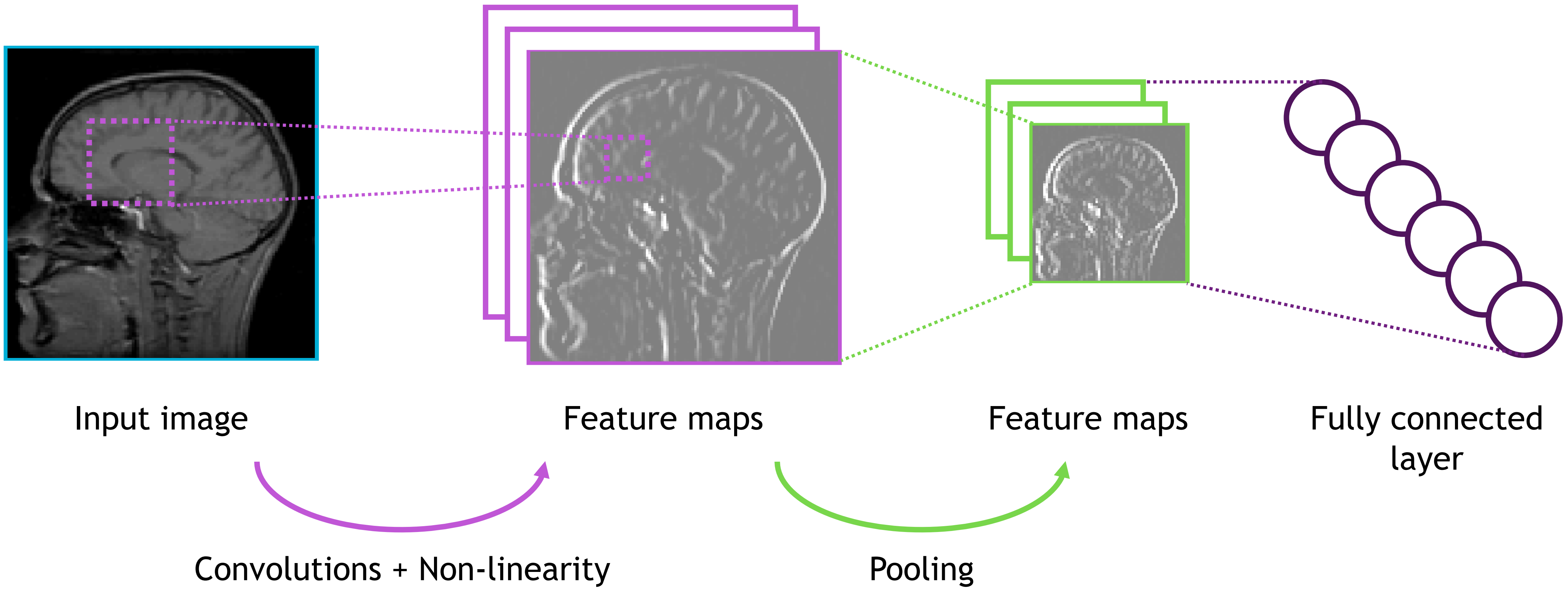
Max pooling with
2×2 filter and stride 2

Pooled feature map



Convolutional neural networks

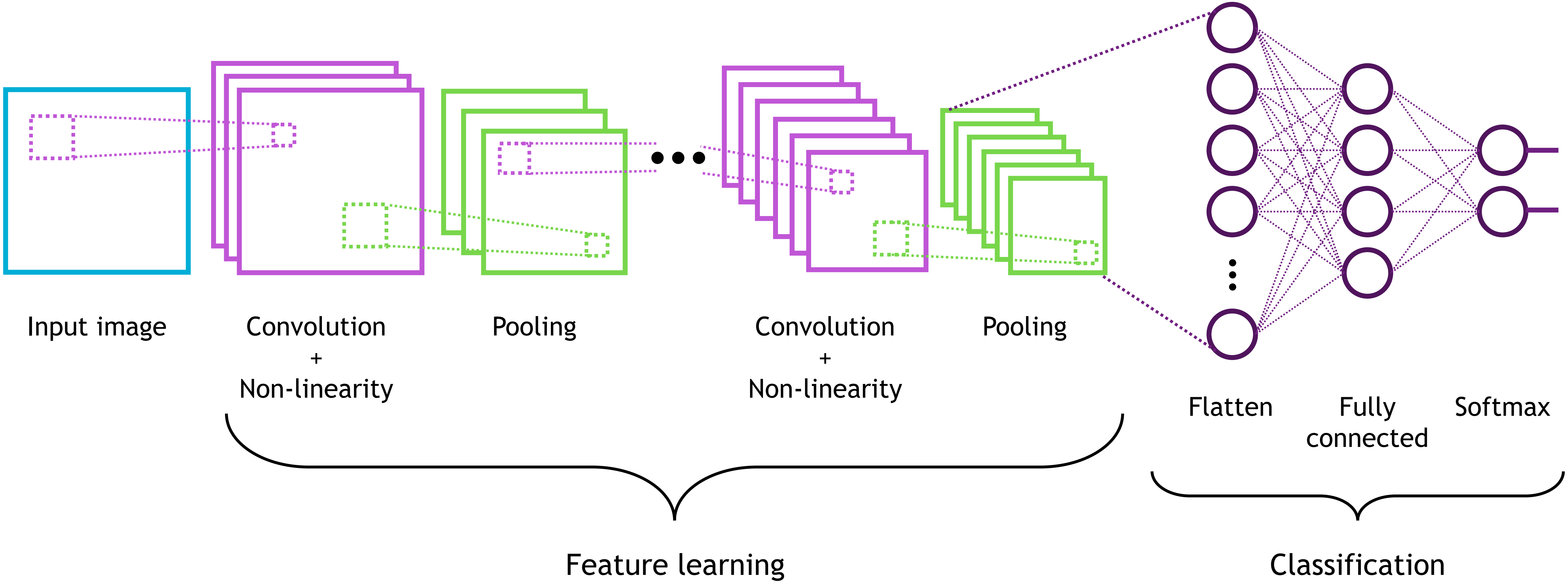
CNNs for classification



MIT Introduction to Deep Learning (introtodeeplearning.com)

Convolutional neural networks

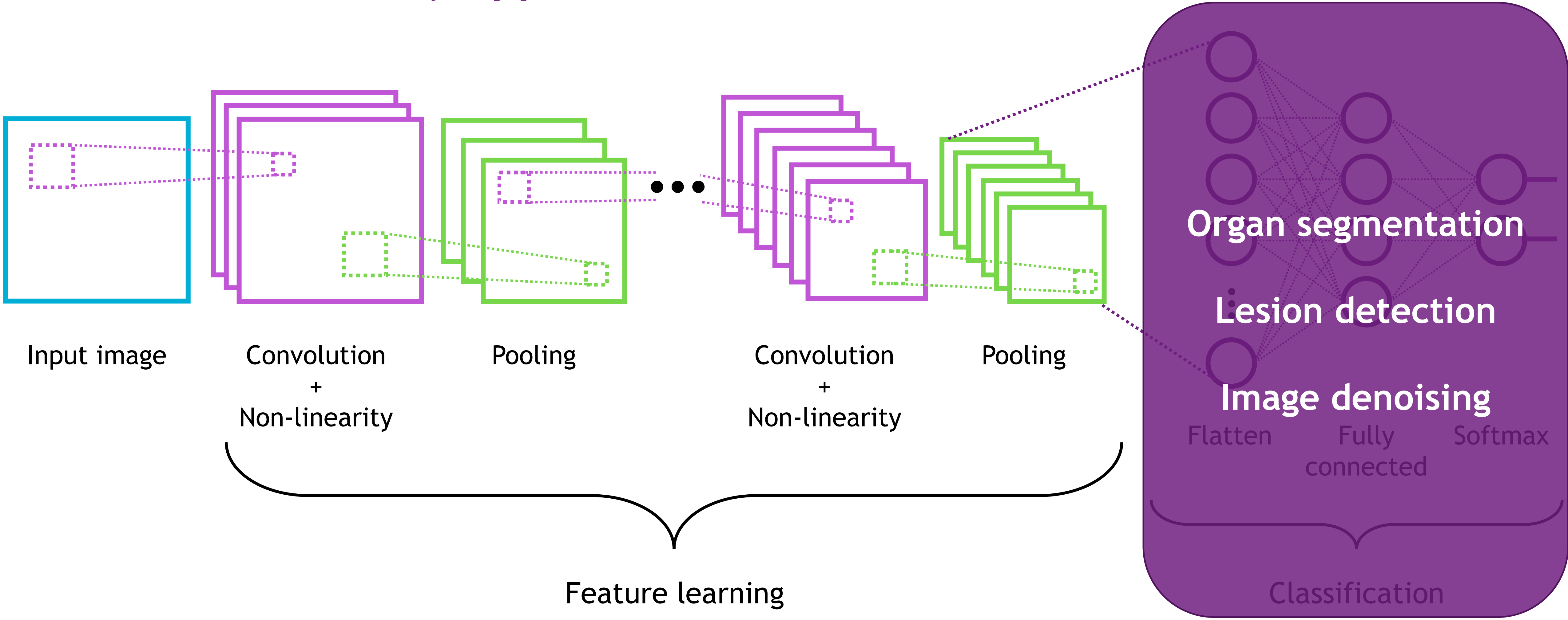
CNNs for classification



MIT Introduction to Deep Learning (introtodeeplearning.com)

Convolutional neural networks

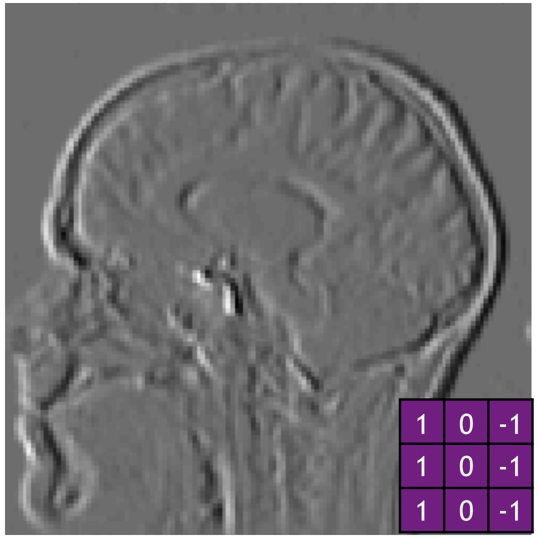
CNNs for many applications

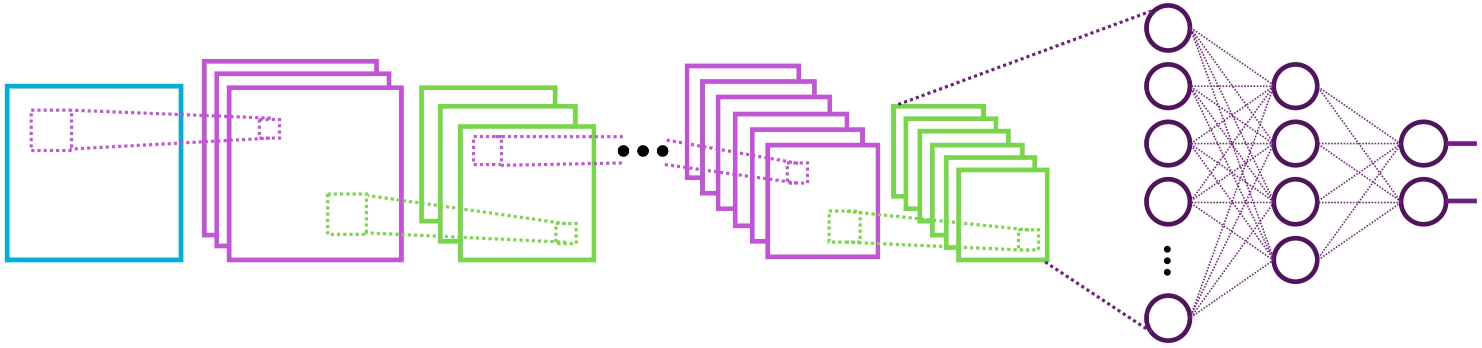


MIT Introduction to Deep Learning (introtodeeplearning.com)

Summary

Images	CNNs	Applications
• Representing images • Convolutions for feature extraction	• Convolution → non-linearity → pooling • Stacking layers	• Classification • Segmentation • Detection



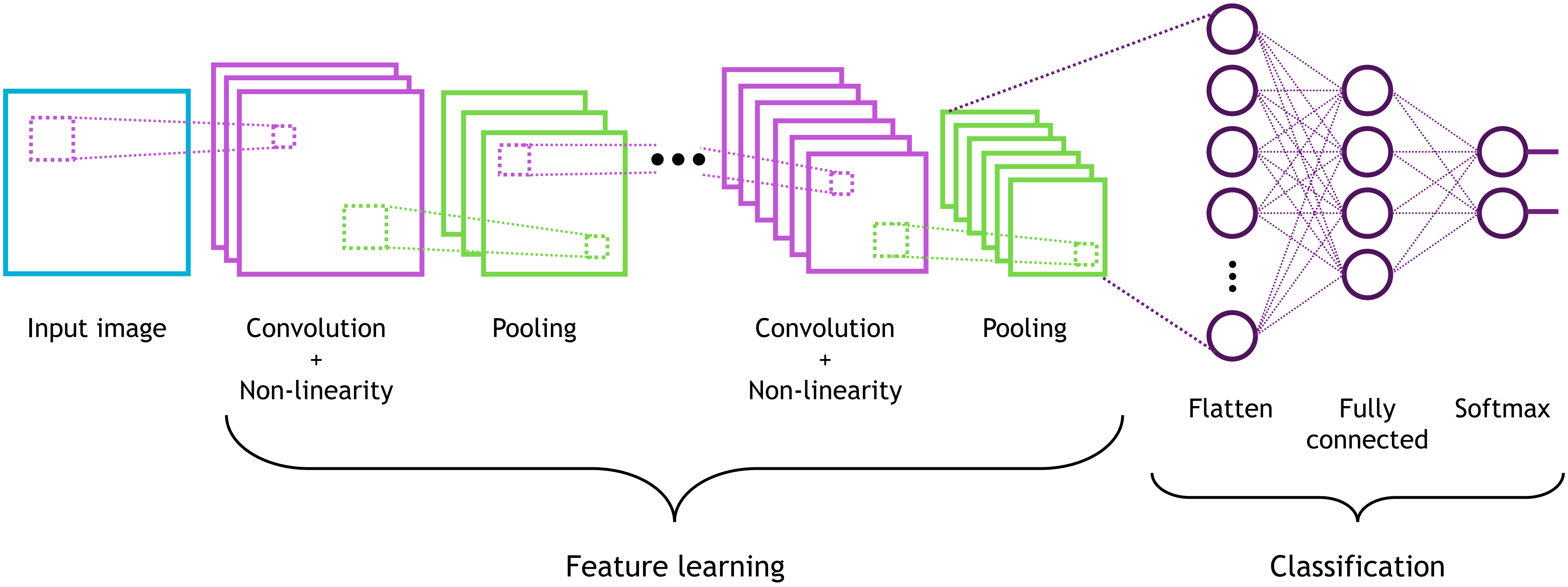


MIT Introduction to Deep Learning (introtodeeplearning.com)

Generative Deep Learning

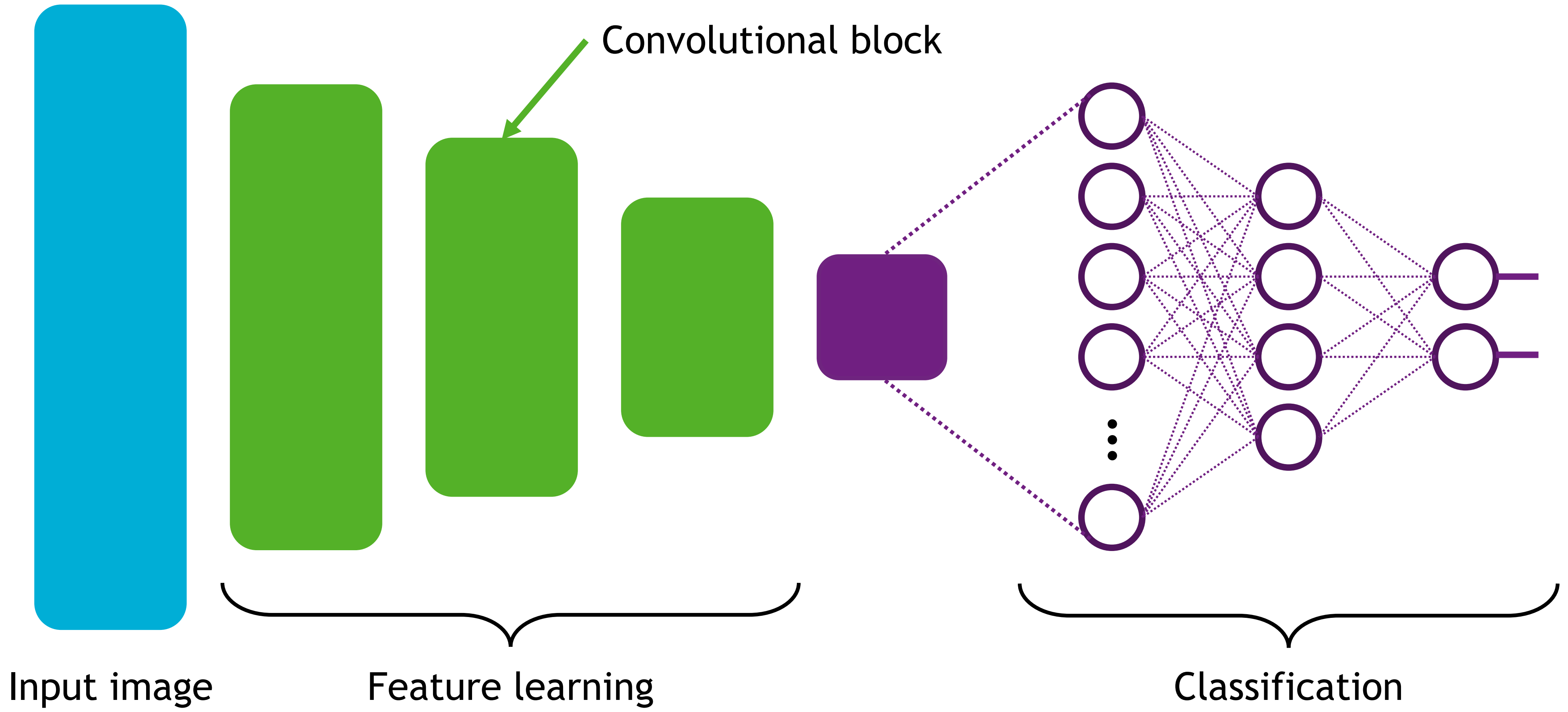
Convolutional neural networks

CNNs for classification



MIT Introduction to Deep Learning (introtodeeplearning.com)

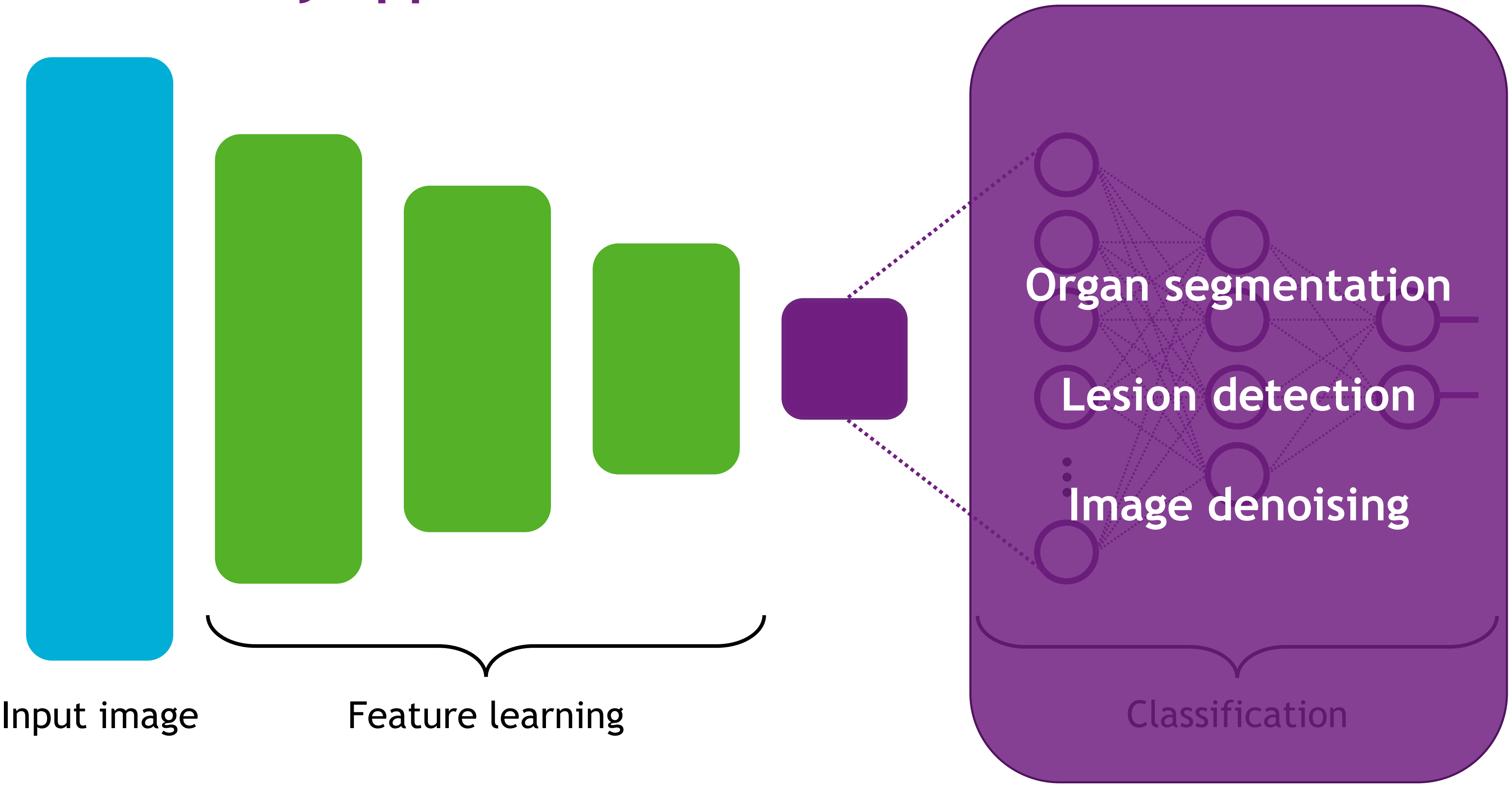
CNNs for classification



MIT Introduction to Deep Learning (introtodeeplearning.com)

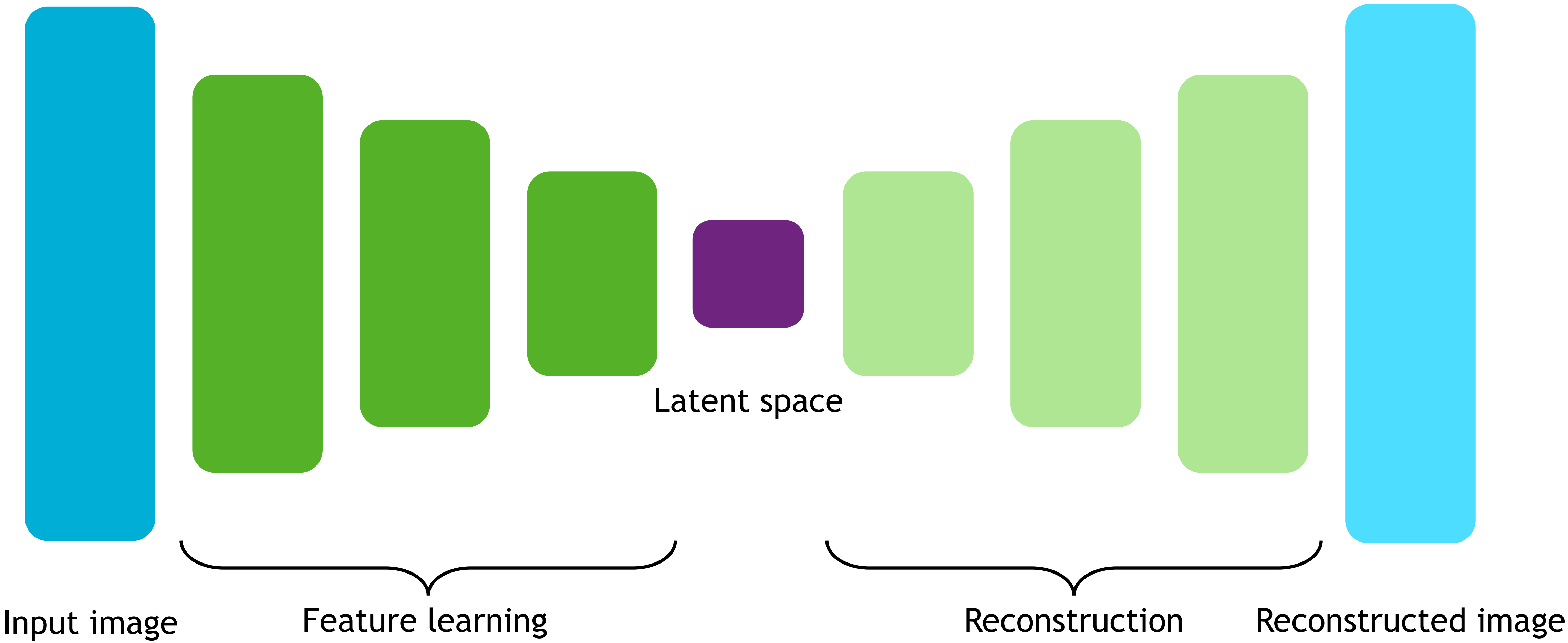
Convolutional neural networks

CNNs for many applications



Convolutional neural networks

CNNs for image generation



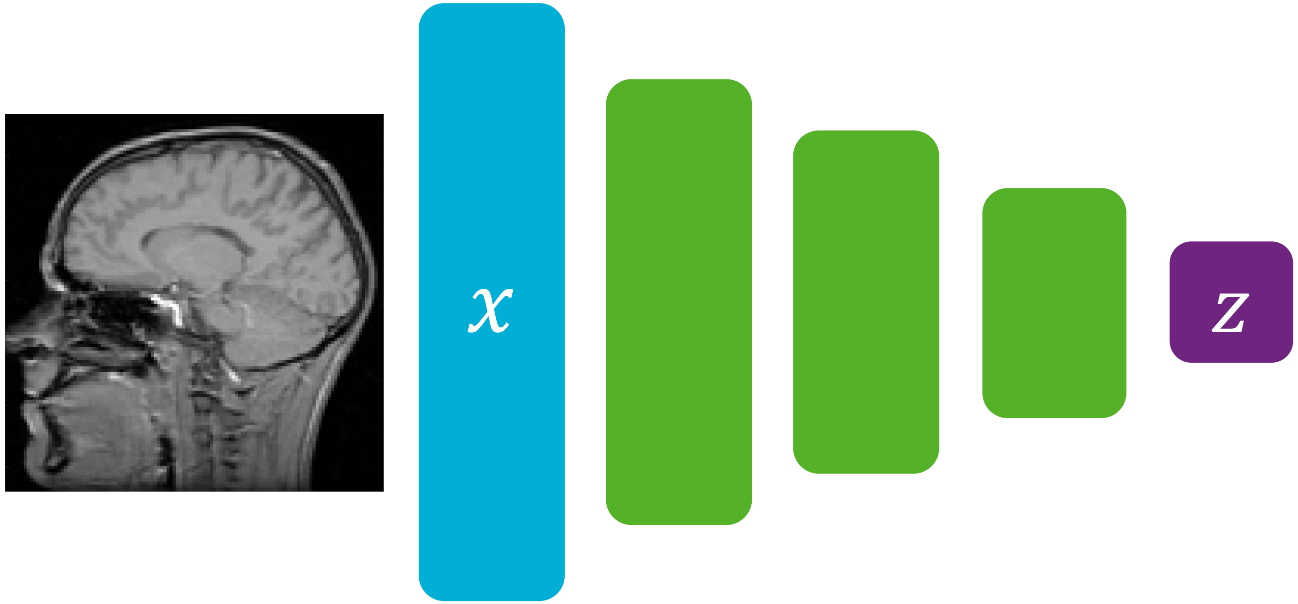
Latent variable

Observed variable

Latent variable

Autoencoders

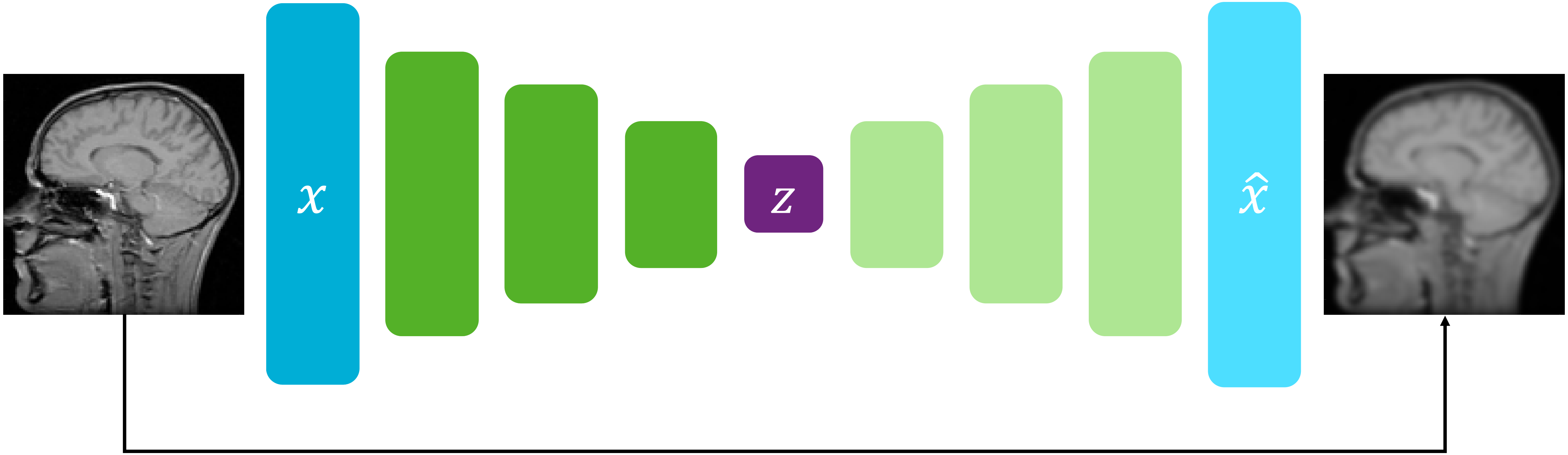
Encoder



Input image
=
observed data

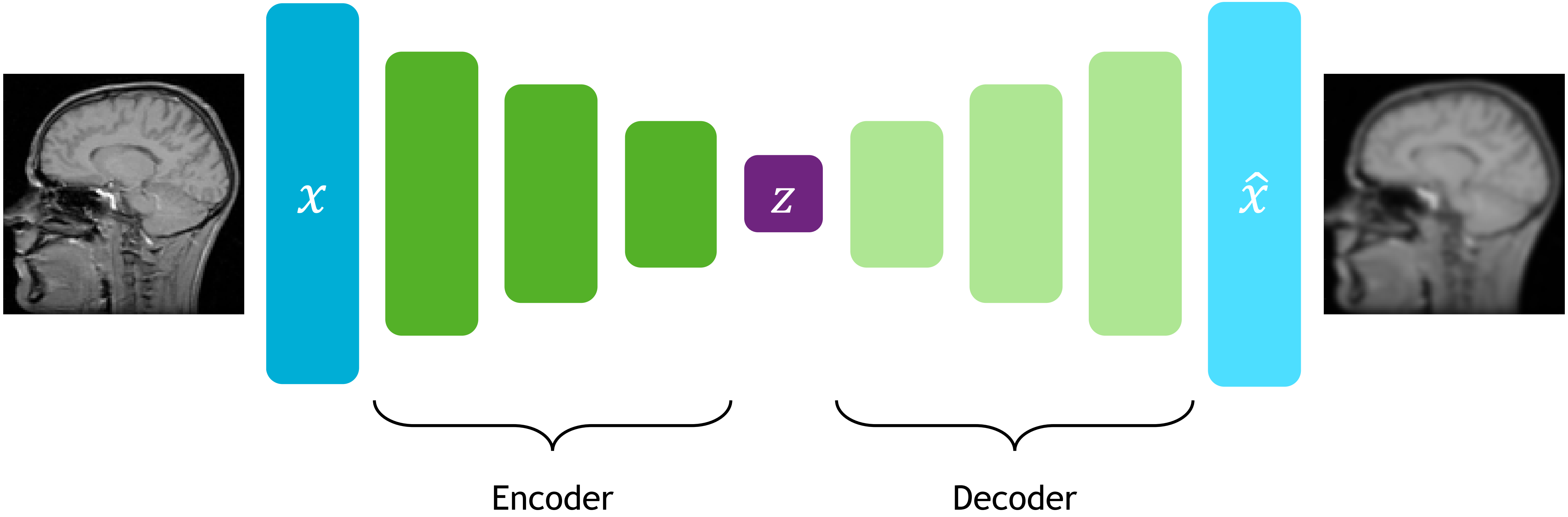
Latent space
=
low dimensional representation
of the observed data

Training autoencoders



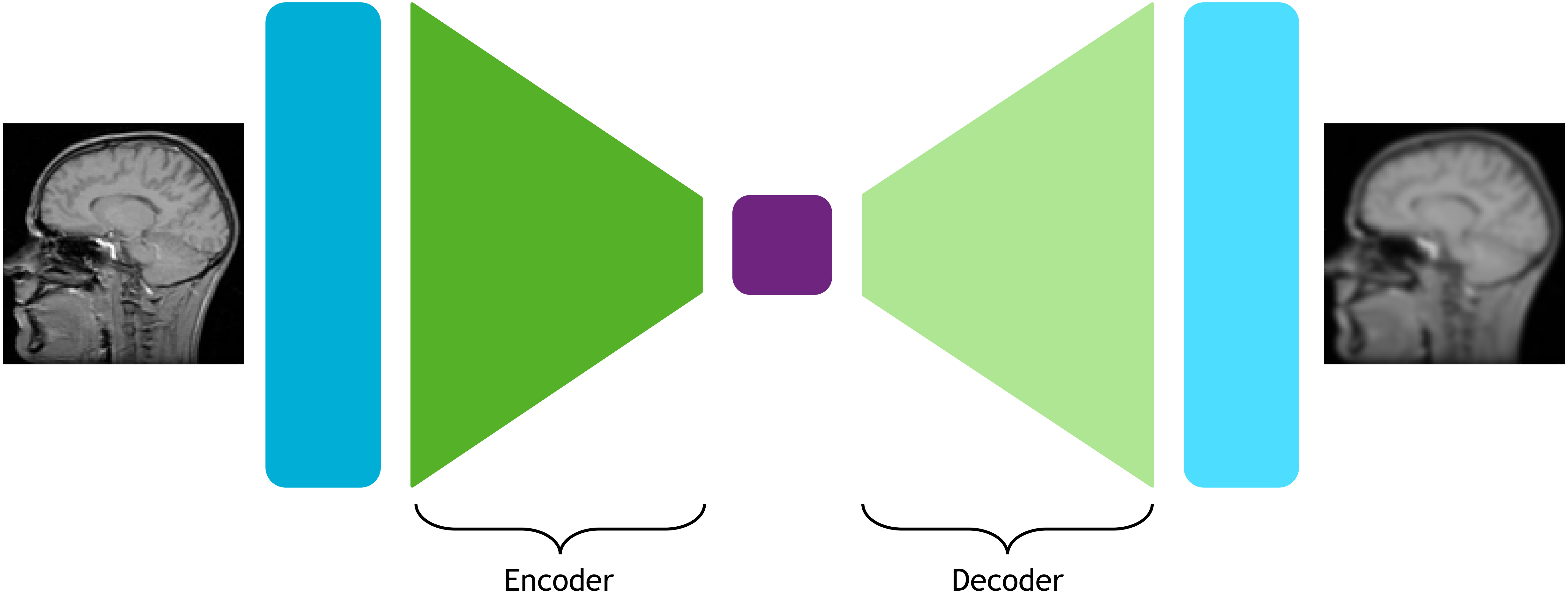
$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

Autoencoders

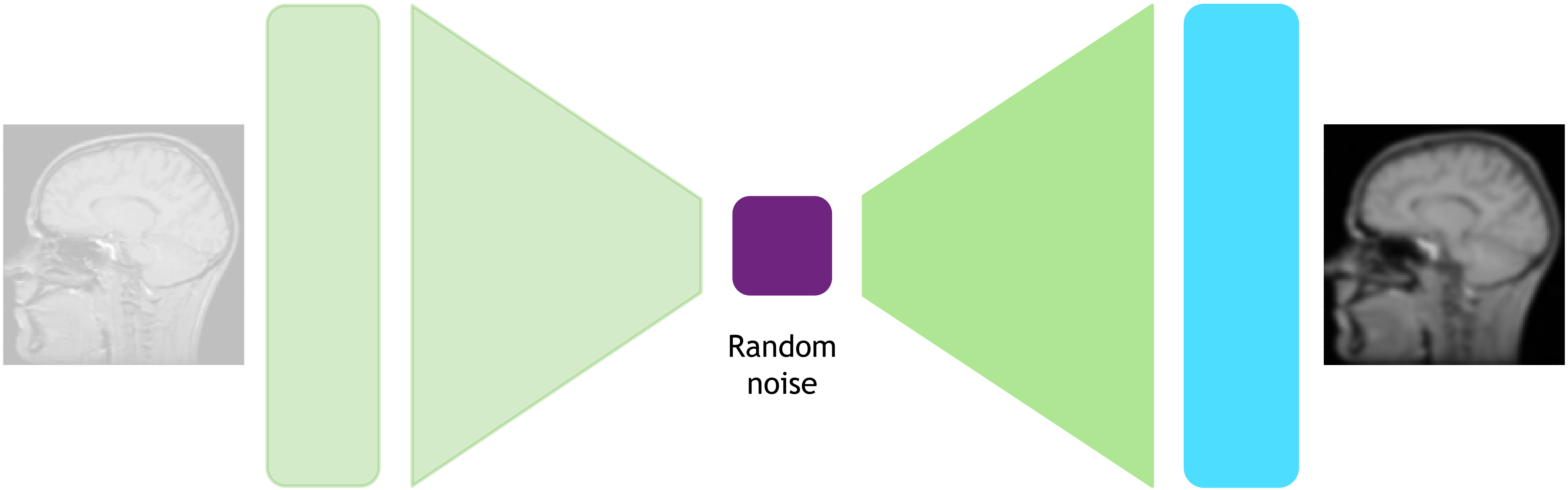


MIT Introduction to Deep Learning (introtodeeplearning.com)

Autoencoders



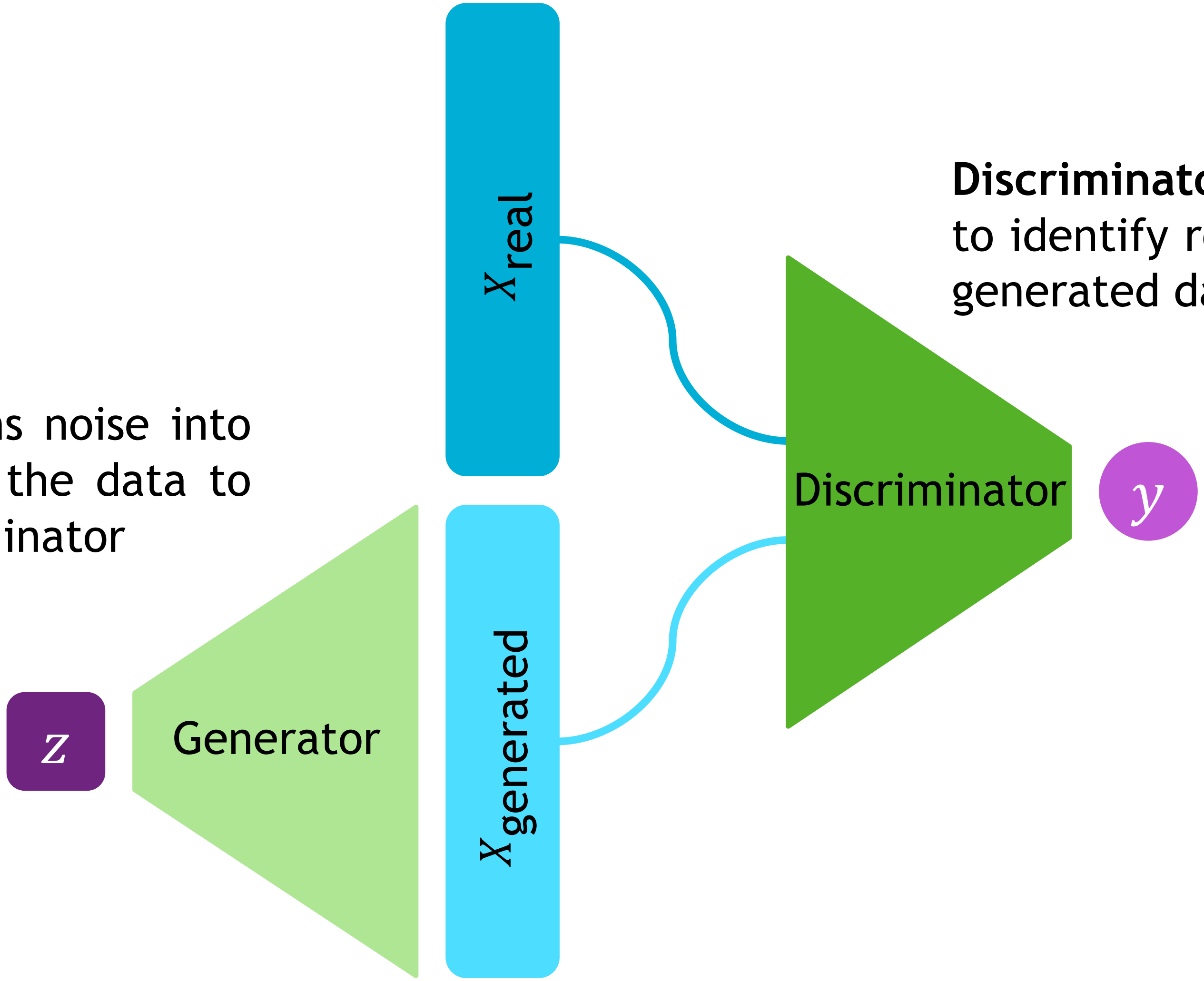
Generating images from scratch



Competing networks

Discriminator: tries to identify real from generated data

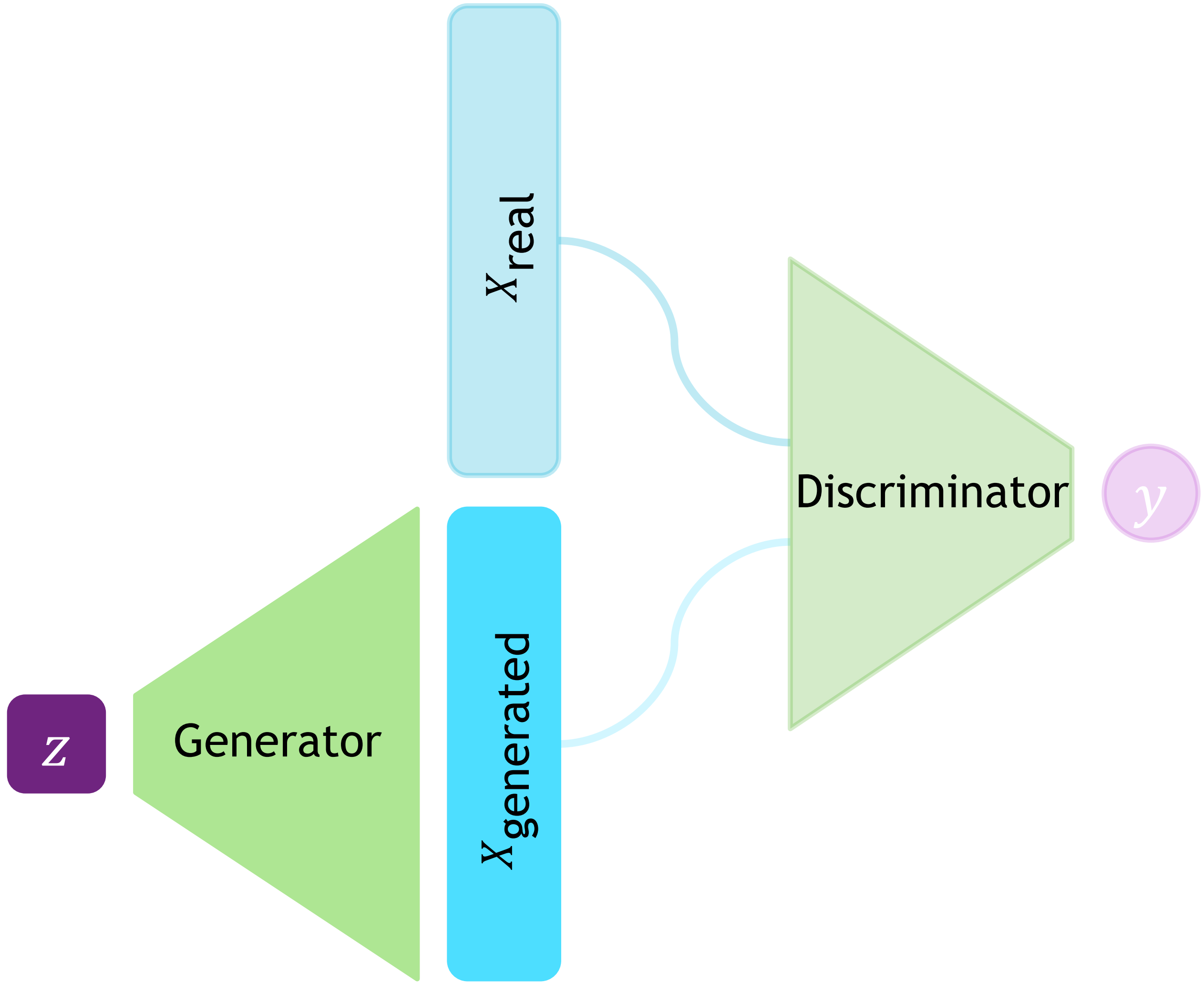
Generator: turns noise into an imitation of the data to trick the discriminator



MIT Introduction to Deep Learning (introtodeeplearning.com)

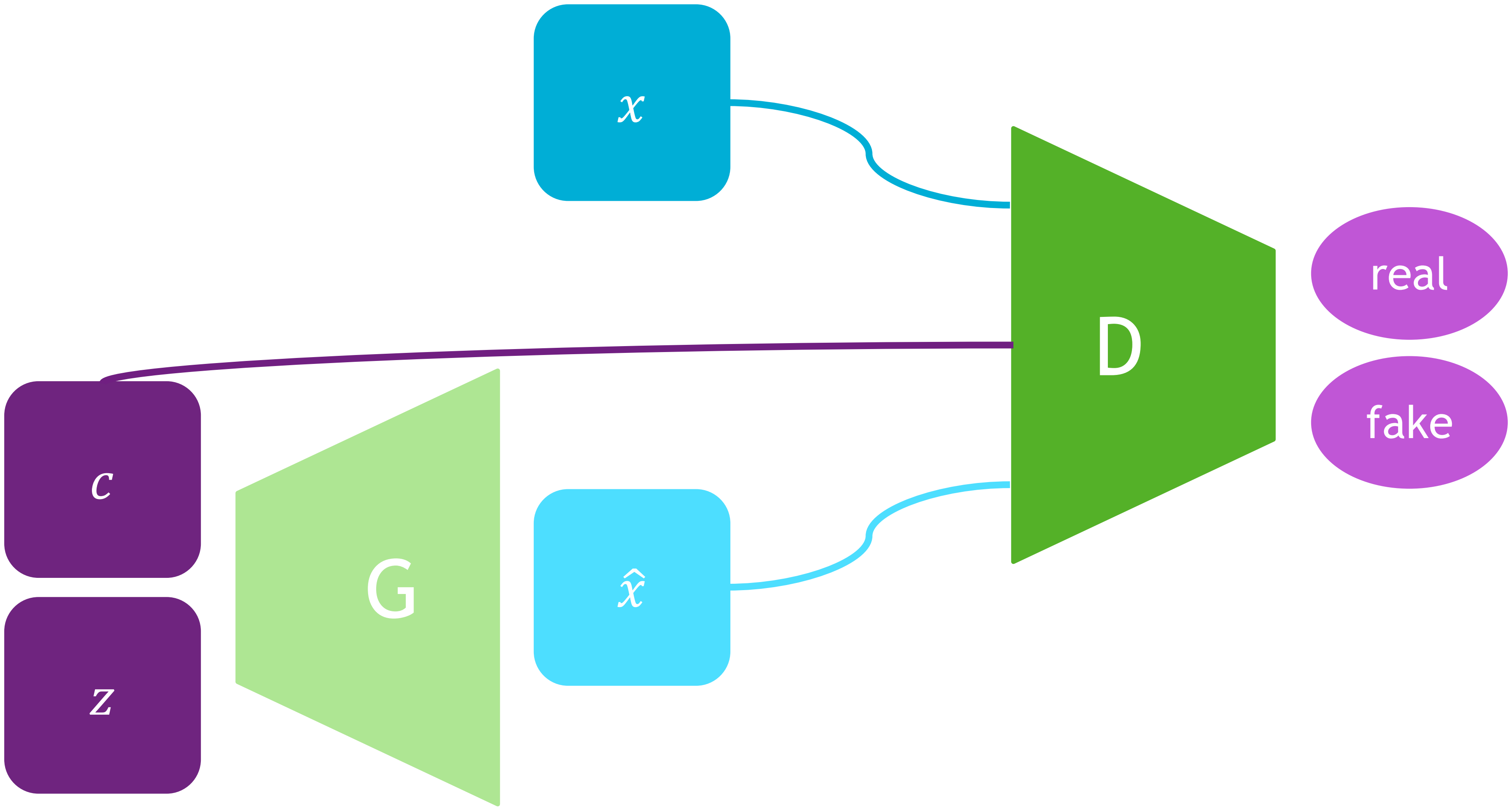
Generative adversarial networks

Generating new images



MIT Introduction to Deep Learning (introtodeeplearning.com)

Image translation with conditional GANs



Generative adversarial networks

Image translation with conditional GANs

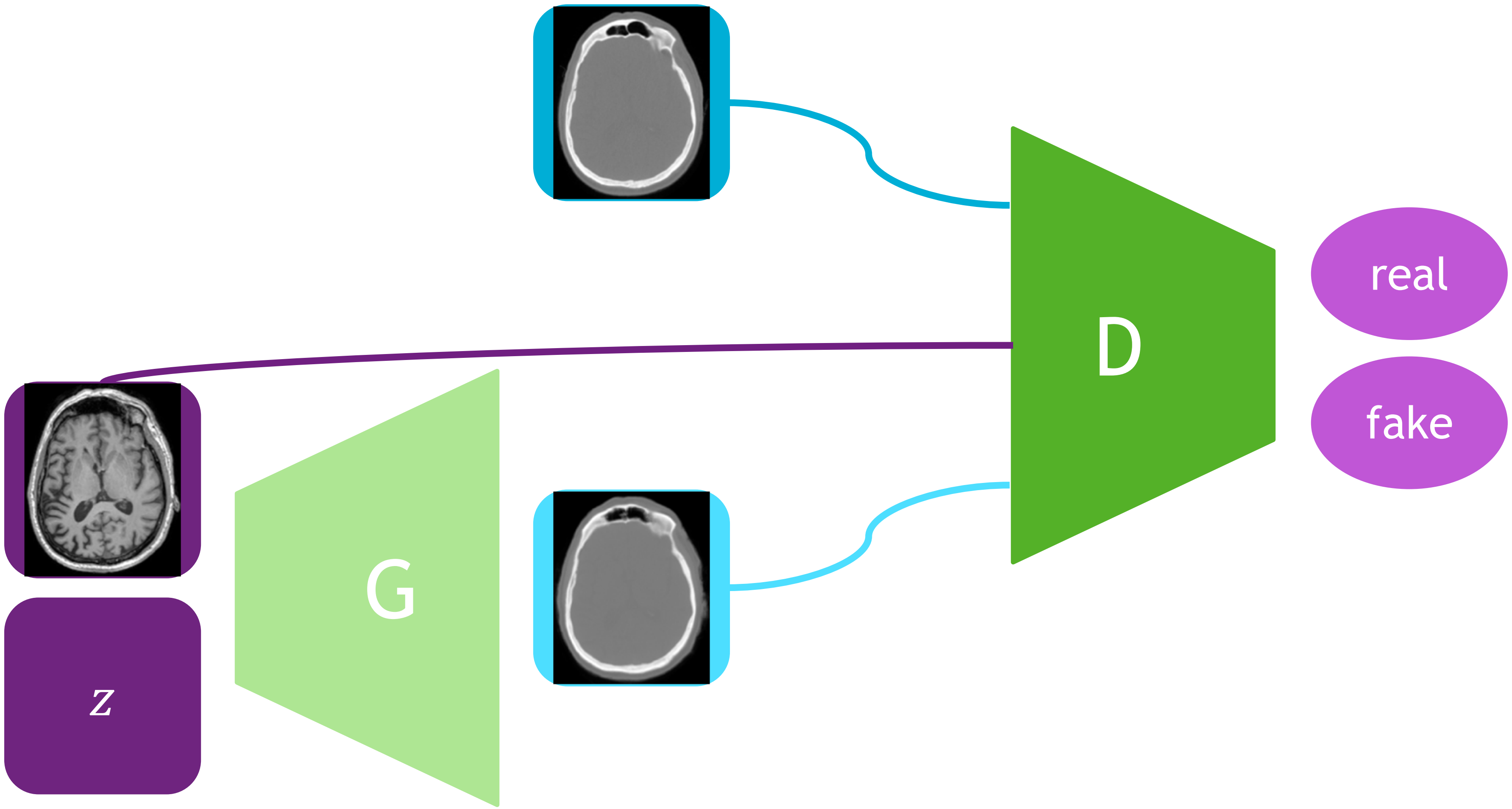
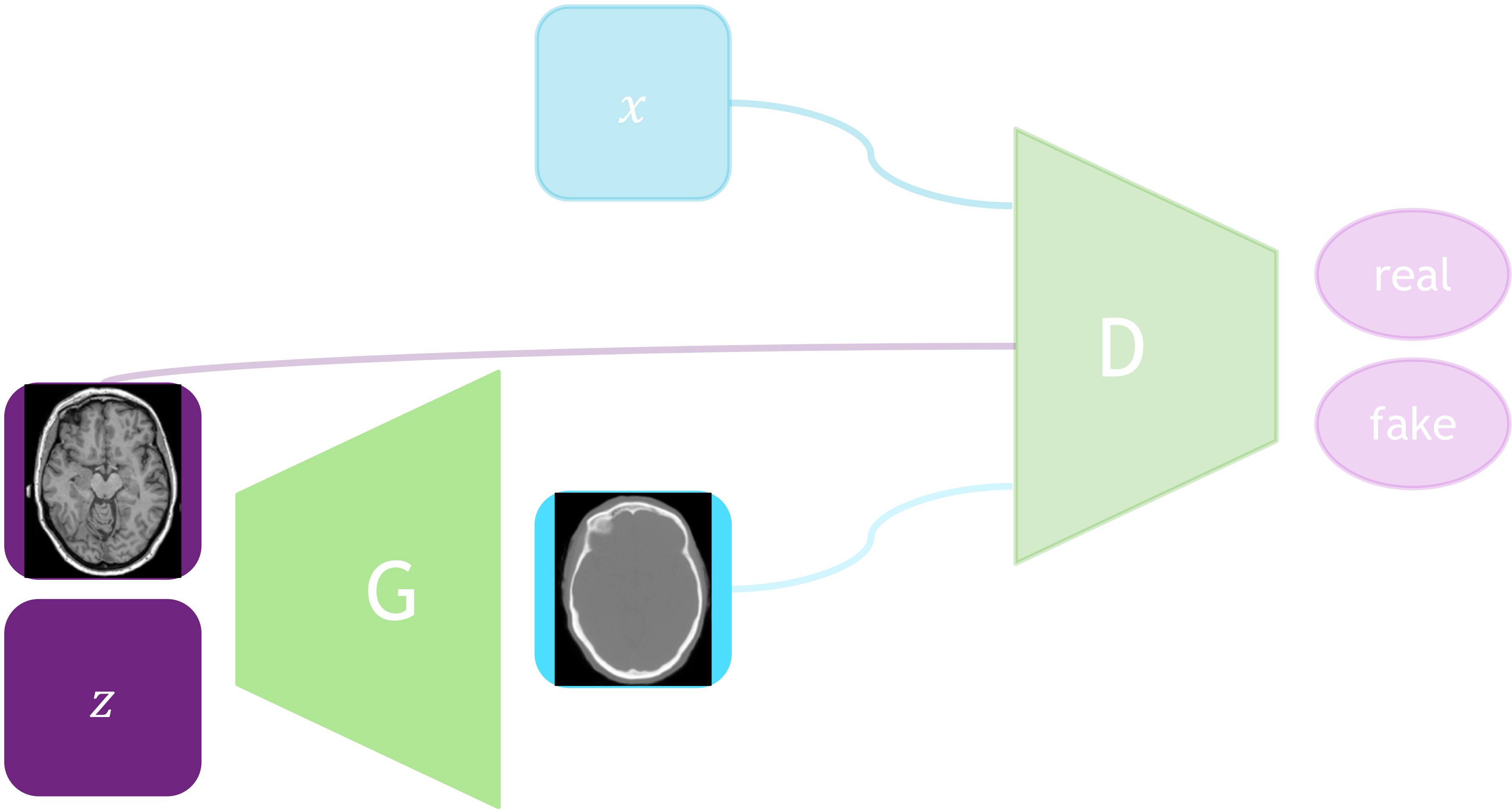


Image translation with conditional GANs



Summary

Autoencoders	GANs	Conditional GANs
• Learn low dimensional latent space	• Competing generator and discriminator networks	• For image translation