



## Week 14 – Part 2 of Neural Networks

Machine Learning and  
Big Data - DATA622

Fall 2023

---

CUNY School of Professional Studies

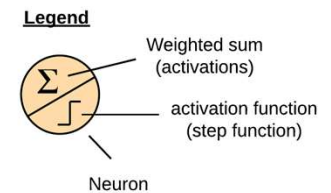
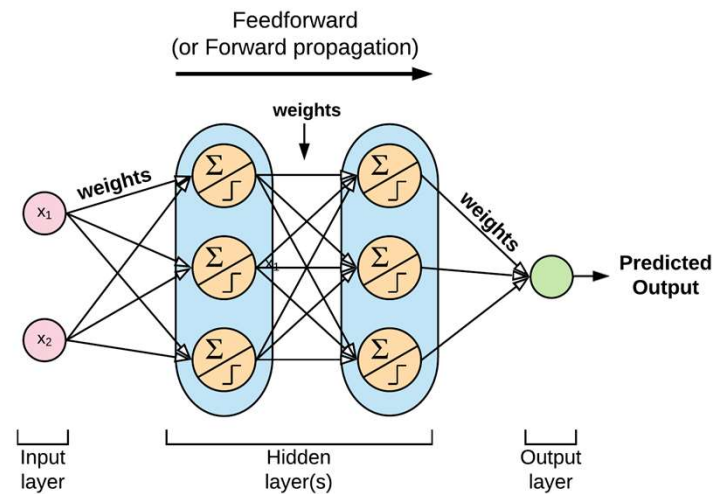
---

# Feedforward

---

**How neural networks make predictions**

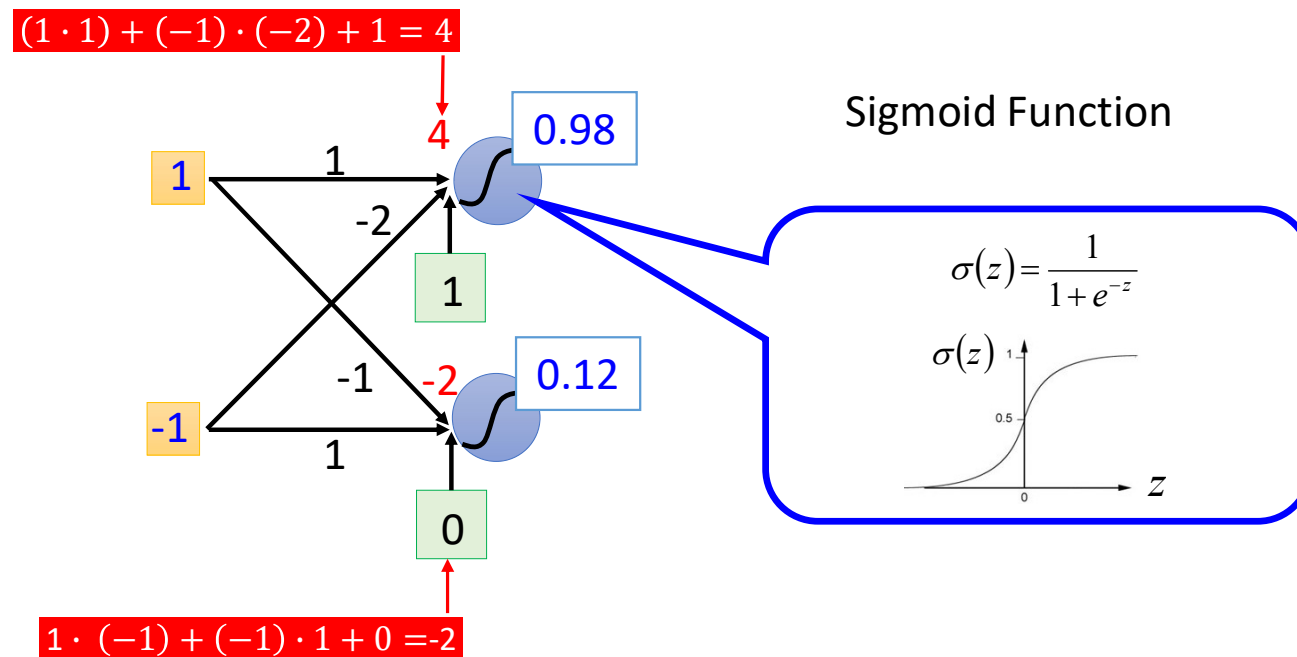
# Feedforward



Source: <https://ekababisong.org/gcp-ml-seminar/deep-learning/>

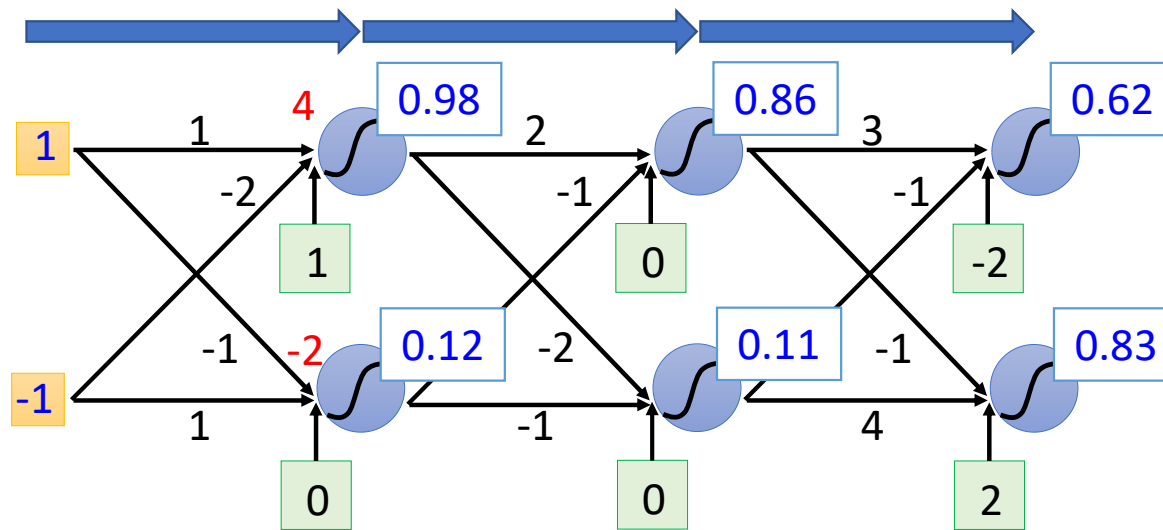
# Feedforward

- A simple network, toy example



Source: Hung-yi Lee – Deep Learning Tutorial

# Feedforward



$$f: \mathbb{R}^2 \rightarrow \mathbb{R}^2 \quad f\left(\begin{bmatrix} 1 \\ -1 \end{bmatrix}\right) = \begin{bmatrix} 0.62 \\ 0.83 \end{bmatrix}$$

Source: Hung-yi Lee – Deep Learning Tutorial

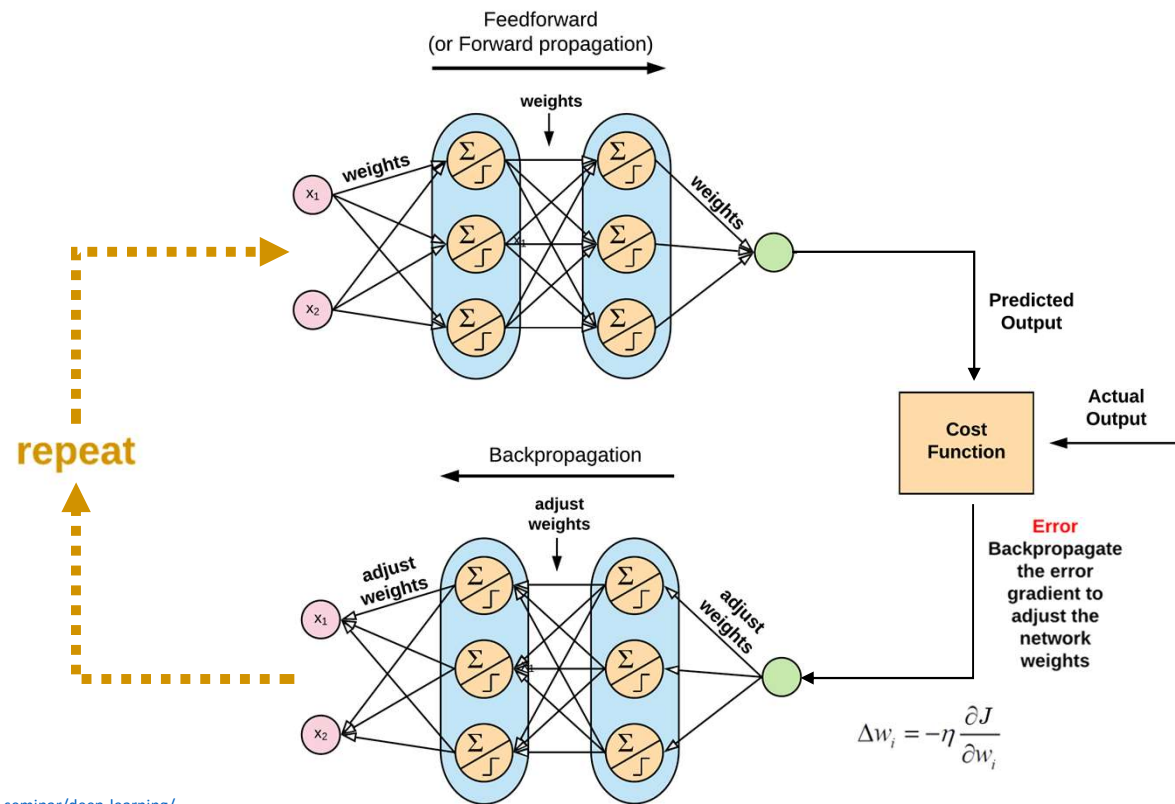
---

# Backpropagation

---

**How neural networks are trained**

# Backpropagation



Source: <https://ekababison.org/gcp-ml-seminar/deep-learning/>

# Backpropagation

---

- Backpropagation is a type of gradient descent (terms used interchangeably)
- Gradient descent refers to the calculation of a gradient on each weight in the neural network for each training element.
  - Because the neural network will not output the expected value for a training element, the gradient of each weight will give you an indication about how to modify each weight to achieve the expected output.
  - If the neural network did output exactly what was expected, the gradient for each weight would be 0, indicating that no change to the weight is necessary.
- How it works:
  - Output of NN is evaluated against desired output
  - If results are not satisfactory, connection (weights) between layers are modified and process is repeated again and again until error is small enough.



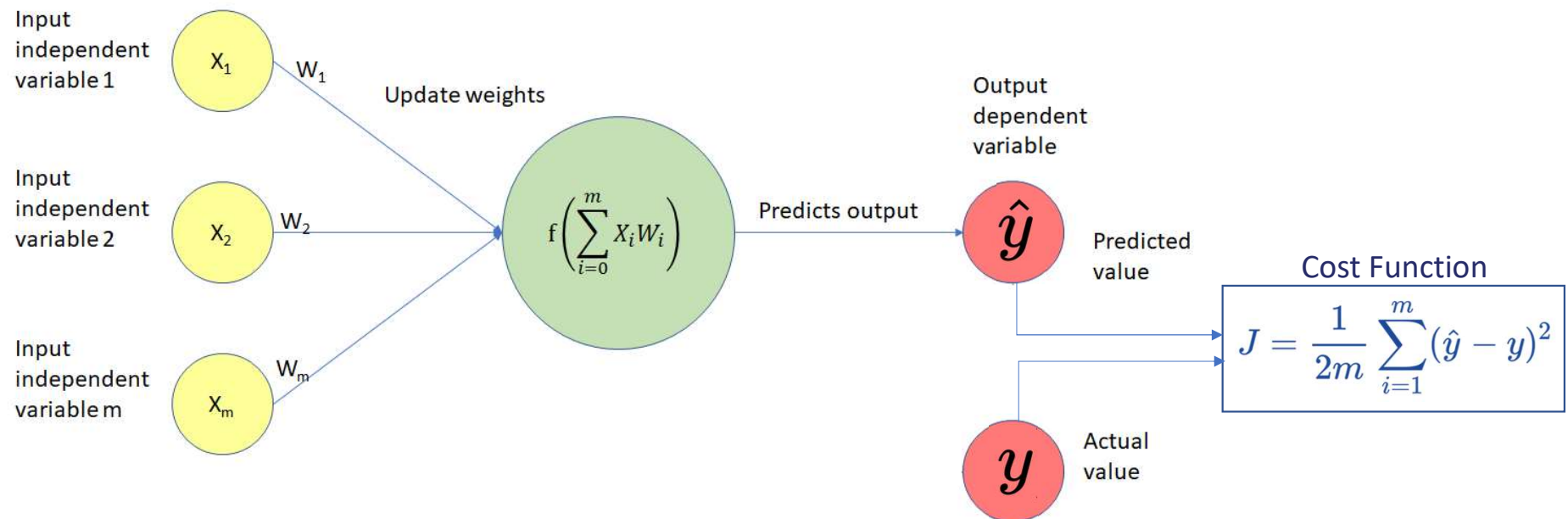
---

# Cost function

---

**Tracking how your training is performing**

# Cost Function (J)



# Cost Function

---

- Cost functions determine how well a model performs for a given dataset
- Cost Functions measure just how wrong the model is in finding a relation between the input and output.
- Comparing other functions:
  - Accuracy functions tell you how well your model does, not how to improve it
  - Error functions measure the difference between the target and the actual values e.g.  $(\hat{y} - y)$
  - Loss functions quantify the cost for a single training example e.g.  $(\hat{y} - y)^2$
  - Cost functions quantify the average across the entire dataset

e.g.

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

- Loss functions quantify the impact of the error i.e. error is objective while loss is subjective e.g. we might adopt a non-symmetric loss function if we may be more negatively affected by an error in a particular direction (e.g., false positive vs. false negative)

# Cost Function

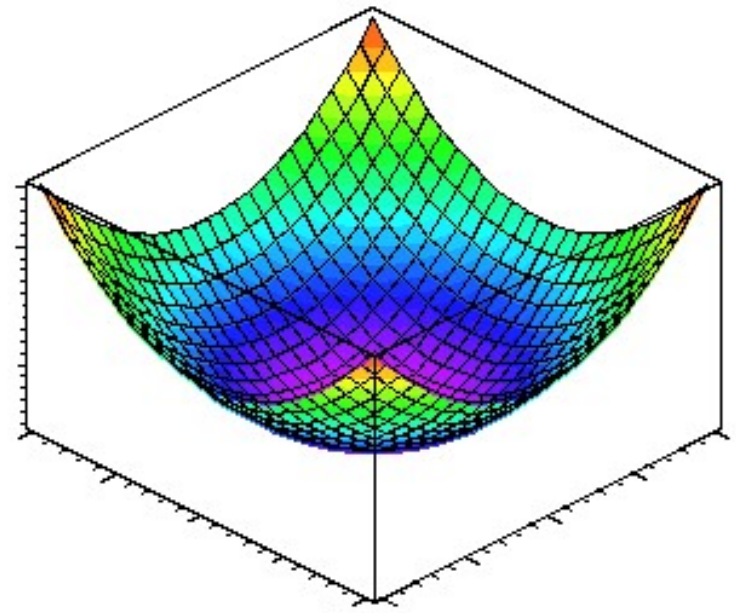
---

- MSE Cost function will always be parabolic (by definition)
  - So it has only one global minimum

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

- SSE – Similar to MSE but not an average

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



Note: The 2 we added in the denominator makes the mathematics cleaner.

# Gradient Descent Rule

---

Gradient descent rule:

$$w_i \leftarrow w_i + \Delta w_i$$

Where:

$$\Delta w_i = -\eta \frac{\partial J}{\partial w_i}$$

$\eta$  is a positive constant called the *learning rate*, and determines step size of gradient descent search

# Cost Function

---

$$SSE = \frac{1}{2} \sum_{j=1}^n (y_j - \hat{y}_j)^2$$

$$SSE = \frac{1}{2} \sum_{j=1}^n \left( y_j - \left( \sum_{i=0}^m \phi(w_i^T \times x_{j,i}) \right) \right)^2$$

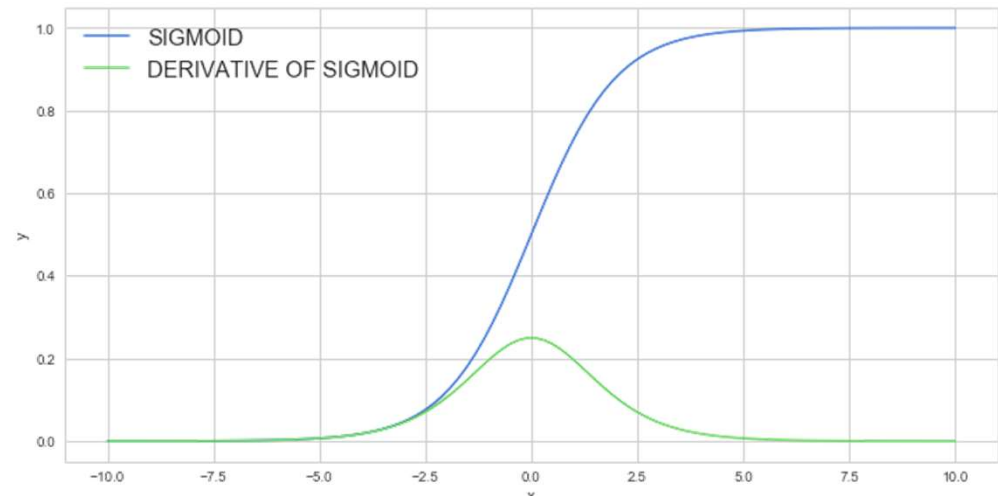
$$\frac{\partial SSE}{\partial w_i} = \sum_{j=1}^n \left( \underbrace{(y_j - \hat{y}_j)}_{\text{error of the output of the weighted sum}} \times \underbrace{-x_{j,i}}_{\text{rate of change of weighted sum with respect to change in } w_i} \right)$$

# Derivative of the Sigmoid Function



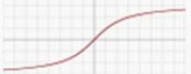




$$y = \frac{1}{1 + e^{-x}}$$

$$\frac{dy}{dx} = -\frac{1}{(1 + e^{-x})^2} (-e^{-x}) = \frac{e^{-x}}{(1 + e^{-x})^2}$$

$$= \frac{1}{1 + e^{-x}} \left( 1 - \frac{1}{1 + e^{-x}} \right) = y(1 - y)$$



# Derivatives of other Activation Functions

Name	Plot	Equation	Derivative
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$



# Cost Function

---

$$SSE = \frac{1}{2} \sum_{j=1}^n (y_j - \hat{y}_j)^2$$

$$SSE = \frac{1}{2} \sum_{j=1}^n \left( y_j - \left( \sum_{i=0}^m \phi(w_i^T \times x_{j,i}) \right) \right)^2$$

$$\frac{\partial SSE}{\partial w_i} = \sum_{j=1}^n \left( \underbrace{(y_j - \hat{y}_j)}_{\substack{\text{error of the} \\ \text{output of the} \\ \text{weighted sum}}} \times \underbrace{-x_{j,i}}_{\substack{\text{rate of change of} \\ \text{weighted sum} \\ \text{with respect to} \\ \text{change in } w_i}} \right)$$

# Gradient Descent Rule or Sigmoid

---

$$w_i^{t+1} = w_i^t + \left( \eta \times \sum_{j=1}^n \underbrace{(y_j^t - \hat{y}_j^t) \times (\hat{y}_j^t \times (1 - \hat{y}_j^t))}_{\substack{\text{Derivative of the sigmoid} \\ \text{activation function with} \\ \text{respect to the weighted sum}}} \times x_{j,i}^t \right)$$

Error gradient for  $w_i$

# Cost Function

---

$$w_i^{t+1} = w_i^t + \left( \eta \times \underbrace{\sum_{j=1}^n ((y_j^t - \hat{y}_j^t) \times x_{j,i}^t)}_{\text{error gradient for } w_i} \right)$$

---

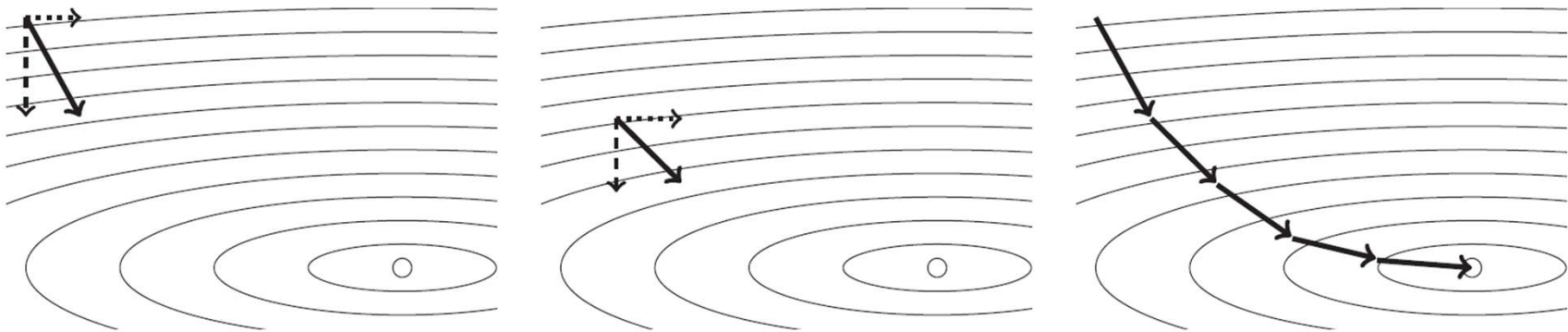
# Learning Rate

---

**You need to adjust the learning rate as you train**

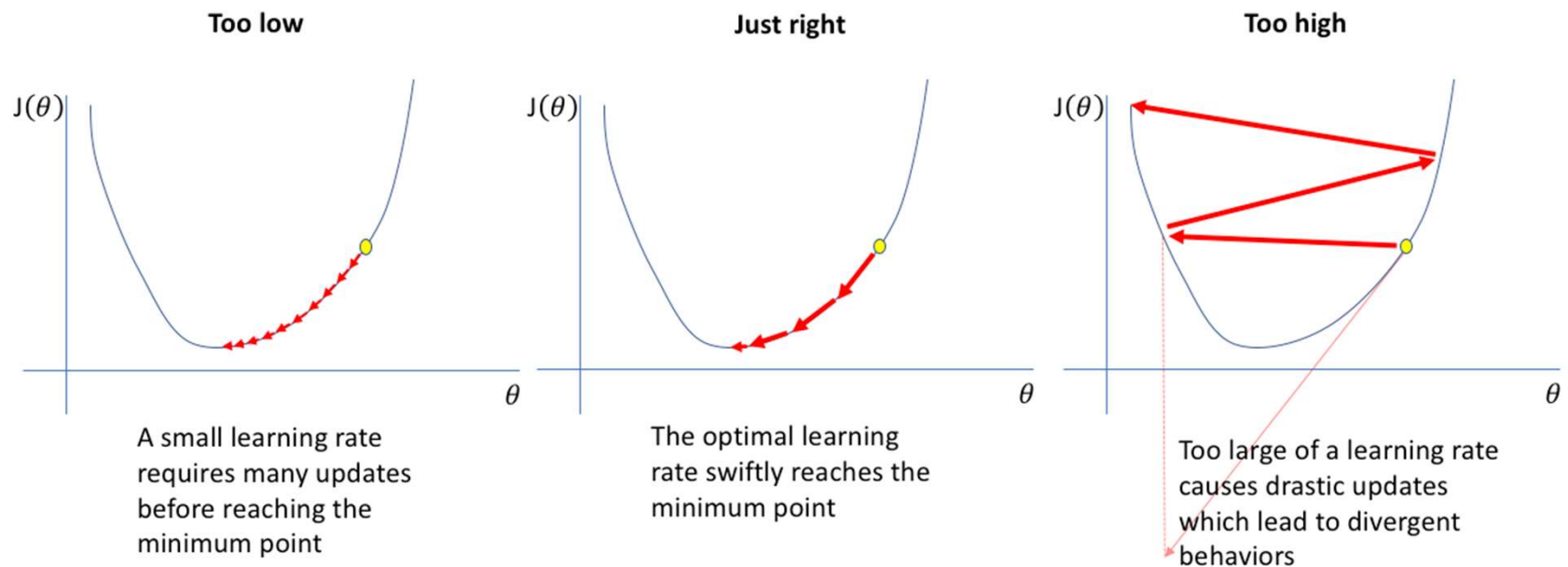
# Gradient descent

The gradient tells us the direction in which the loss has the steepest rate of increase, but it does not tell us how far we should step



# Learning Rate

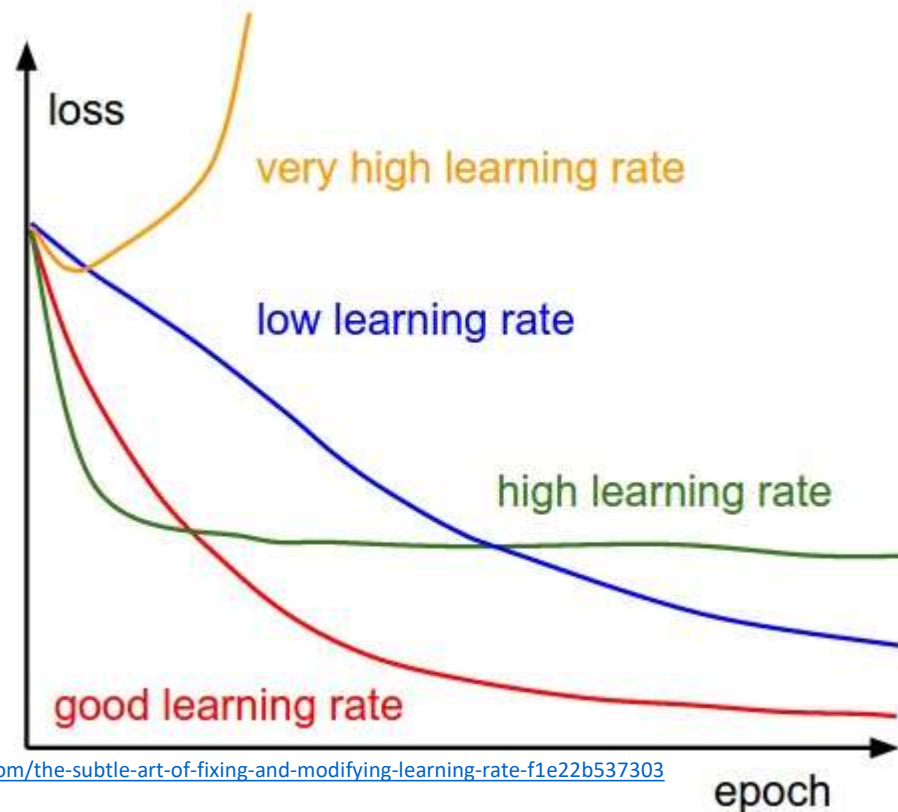
Learning rate (step size) is a hyper-parameter which defines the speed of 'learning'



Source: <https://www.jeremyjordan.me/nn-learning-rate/>

# Impact of different learning rates

- High learning rate: loss increases or plateaus too quickly
- Low learning rate: loss decreases too slowly: too many epochs to reach a solution



Source: <https://towardsdatascience.com/the-subtle-art-of-fixing-and-modifying-learning-rate-f1e22b537303>

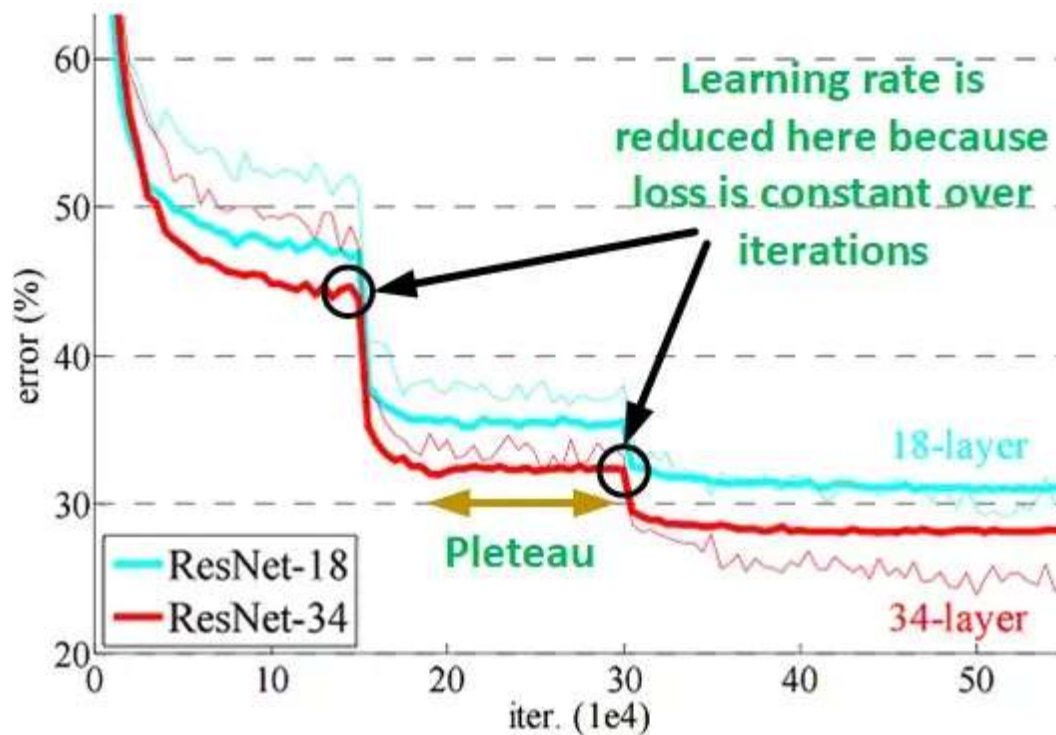
# Adjusting learning rates

---

- Learning rates must be decayed during training
- Approach 1: Automatic Decay
  - Reduce the learning rate by some factor every few epochs
  - Typical values: reduce the learning rate by a half every 5 epochs, or by 10 every 20 epochs
  - Exponential decay reduces the learning rate exponentially over time
  - These numbers depend heavily on the type of problem and the model
- Approach 2: Plateau-driven
  - Reduce the learning rate by a constant (e.g., by half) when the validation loss stops improving
  - In TensorFlow: `tf.keras.callbacks.ReduceLROnPlateau()`
  - Monitor: validation loss
  - Factor: 0.1 (i.e., divide by 10)
  - Patience: 10 (how many epochs to wait before applying it)
  - Minimum learning rate:  $1e-6$  (when to stop)



# Adjusting learning rates



Source: <https://towardsdatascience.com/the-subtle-art-of-fixing-and-modifying-learning-rate-f1e22b537303>

---

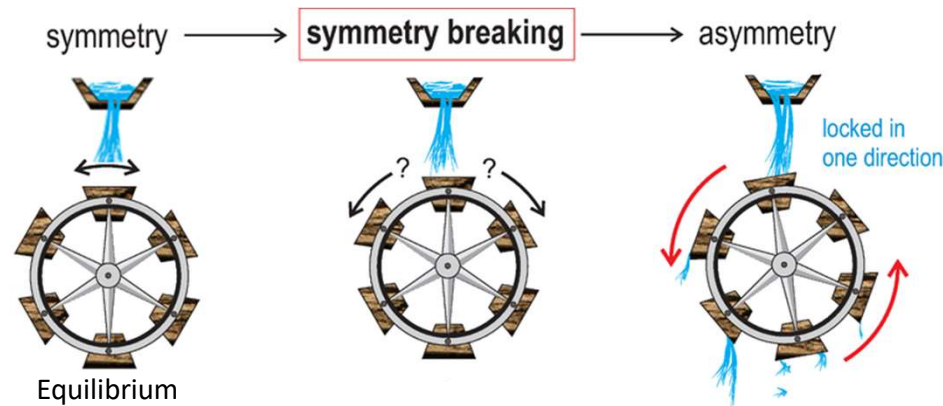
# Lottery ticket hypothesis

---

**Breaking symmetry of neural networks with random initialization**

# Breaking symmetry

- Initializing weights & biases to zeroes (or ones) is problematic



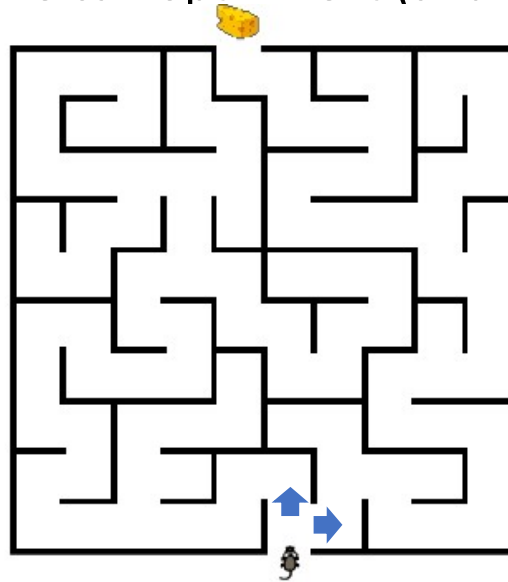
- If all weights are zeros, subsequent hidden layers will get zero signal
- If weights are all the same value (e.g. 1), then subsequent hidden layers will get exactly the same signal i.e. if all weights are initialized to 1, each unit gets signal equal to sum of inputs (and outputs  $\text{sigmoid}(\text{sum}(\text{inputs}))$ ).
- No matter what was the input - if all weights are the same, all units in hidden layer will be the same too. This is why you should initialize weights randomly.

# Mouse Maze

---

Consider a mouse trying to find a way through a maze:

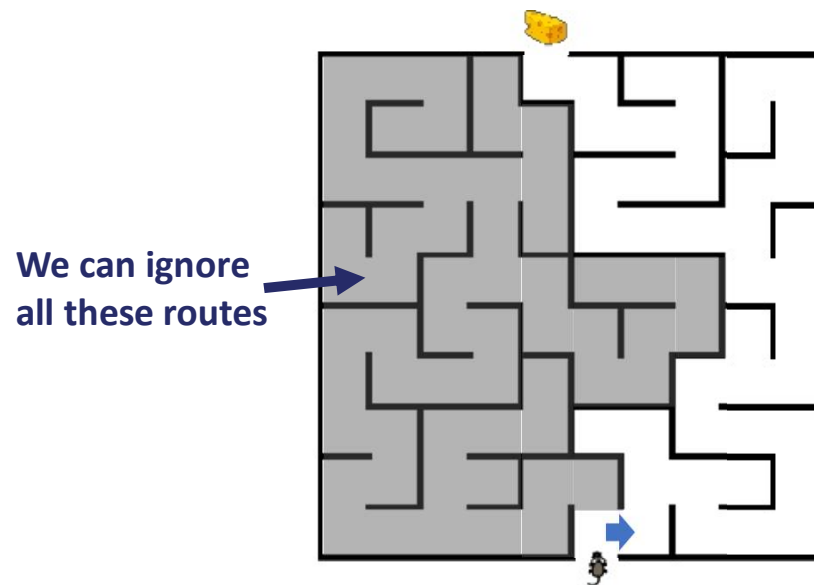
1. The initial routing will be critical in terms of how quickly a solution is found
2. Once a route is found, we can optimize it (and other routes are ignored)



# Mouse Maze

---

If we have a favorable start and found a route, then the rest of the maze is irrelevant



# Lottery ticket hypothesis

---

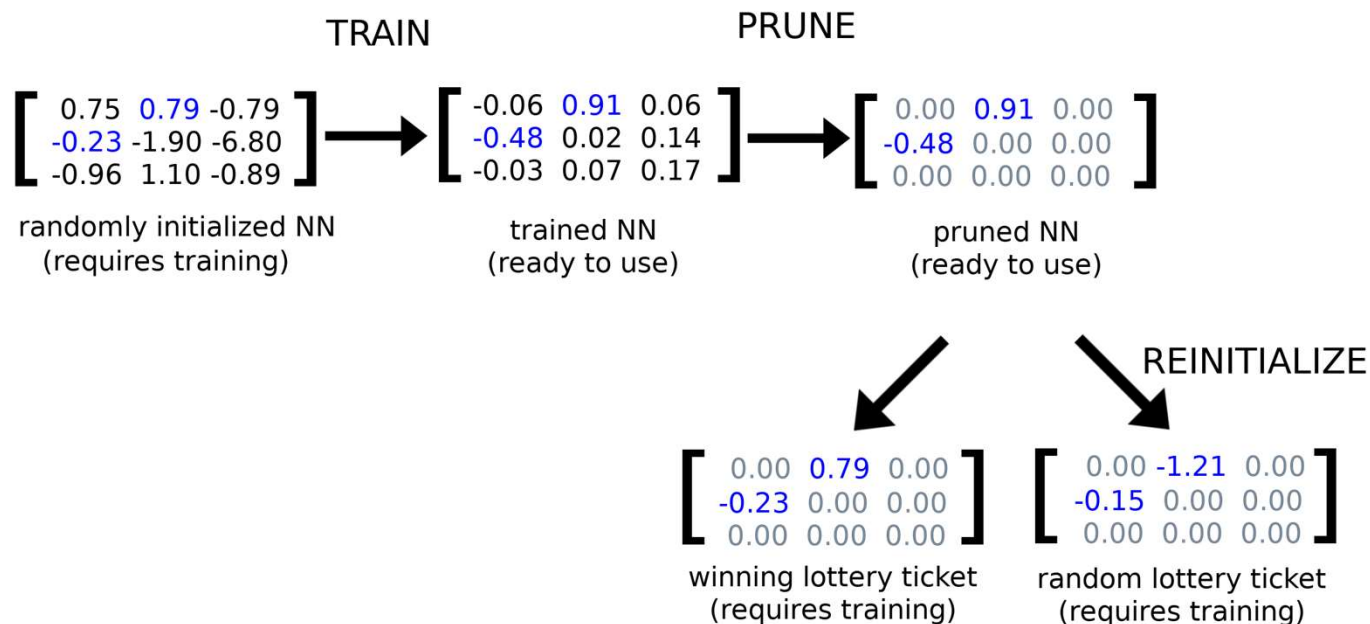
“A randomly-initialized, dense neural network contains a subnetwork that is initialized such that-when trained in isolation-it can match the test accuracy of the original network after training for at most the same number of iterations.”

The subnetwork is referred to as a winning ticket (it has won the initialization lottery!)

Lottery ticket hypothesis: <https://arxiv.org/pdf/1803.03635.pdf>

# Lottery ticket hypothesis

- A neural network is typically sparse (most of the weights are zero)
- Pruning the network doesn't change accuracy but makes the model faster
- The lottery ticket (initial optimal weights) will regenerate the optimal model



Lottery ticket hypothesis: <https://okl1m3k.github.io/lottery-ticket-hypothesis/>

---

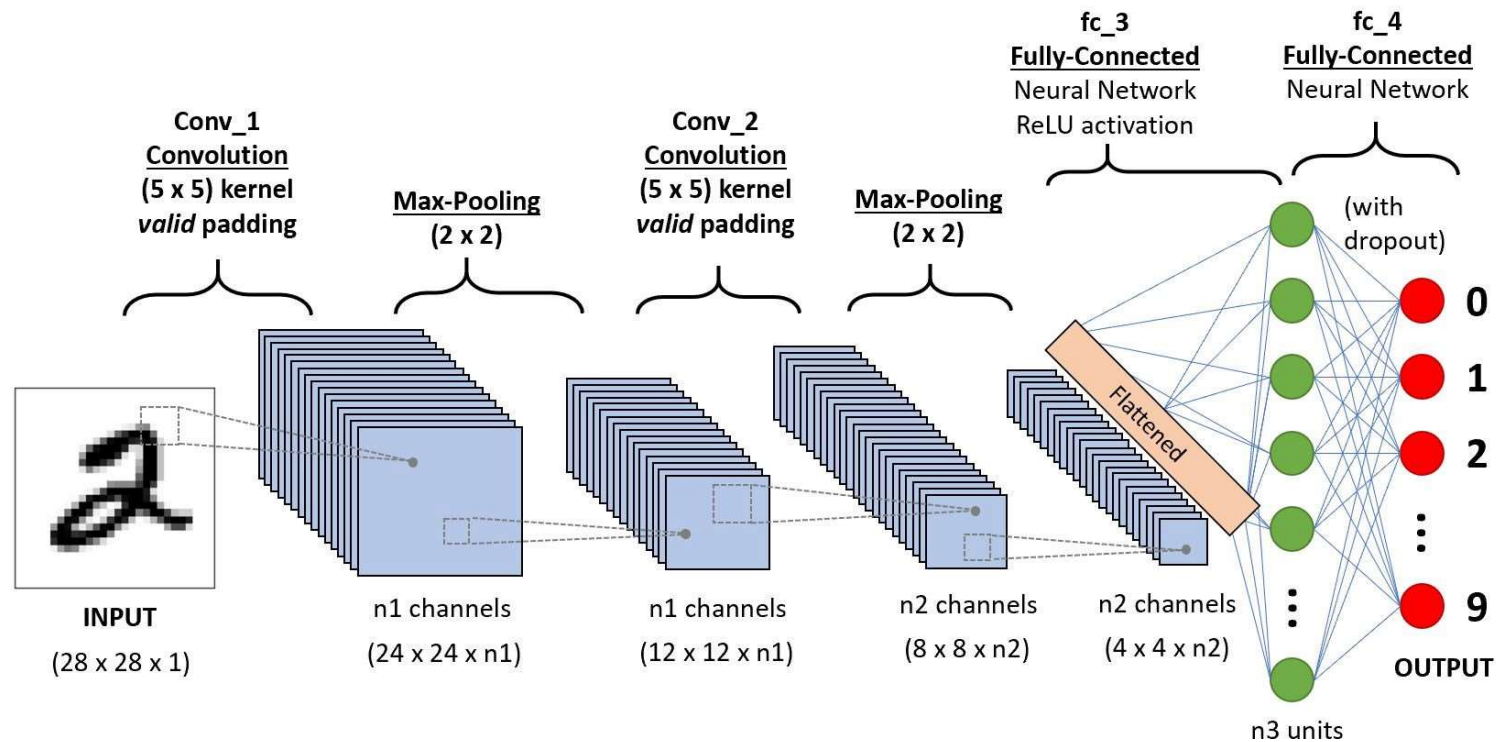
# Convolutional Neural Networks

---

**Image processing (inspired by nature)**



# Convolutional Neural Networks (CNN)



Source: Jonas Bokstaller

# Convolutions

0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

↓  
308

+

↓  
-498

+

↓  
164

↑  
Bias = 1

+ 1 = -25

Output

-25				...
				...
				...
				...
...	...	...	...	...

At each slide, we perform an element-wise multiplication and sum everything to get a single value, with the kernel is sliding over the whole input.

# Pooling

## Max Pooling

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

## Average pooling

7	3	5	2
8	7	1	6
4	9	3	9
0	8	4	5

$$\frac{7 + 3 + 8 + 7}{4}$$

Average pooling

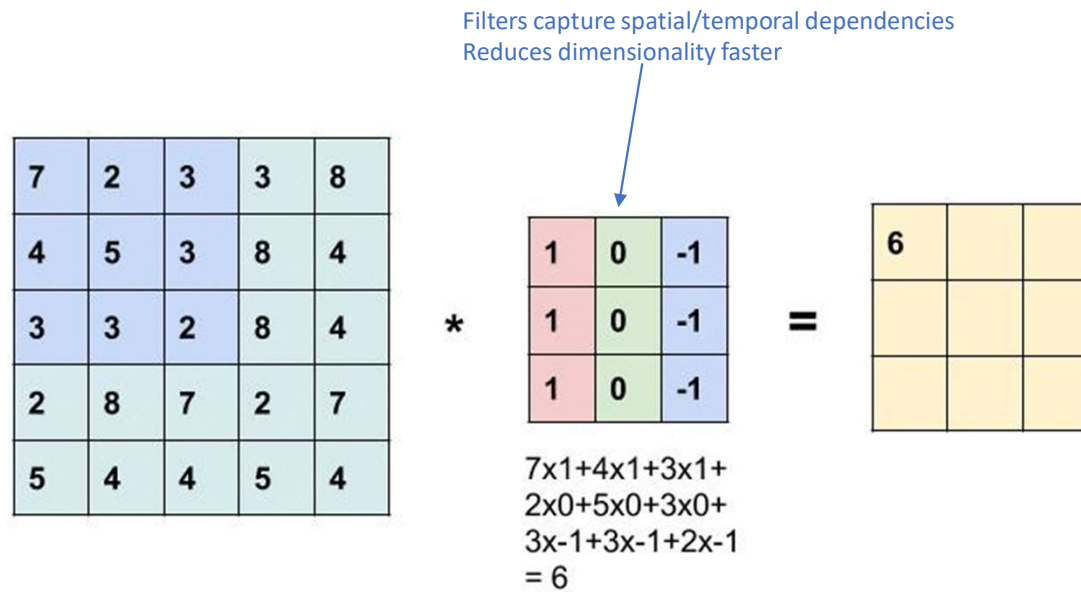
6.25	

The goal of pooling is to reduce the height and width of our image but not the number of channels by using a stride > 1

Source: <https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/>





# Convolutions

---



# Some examples of kernels

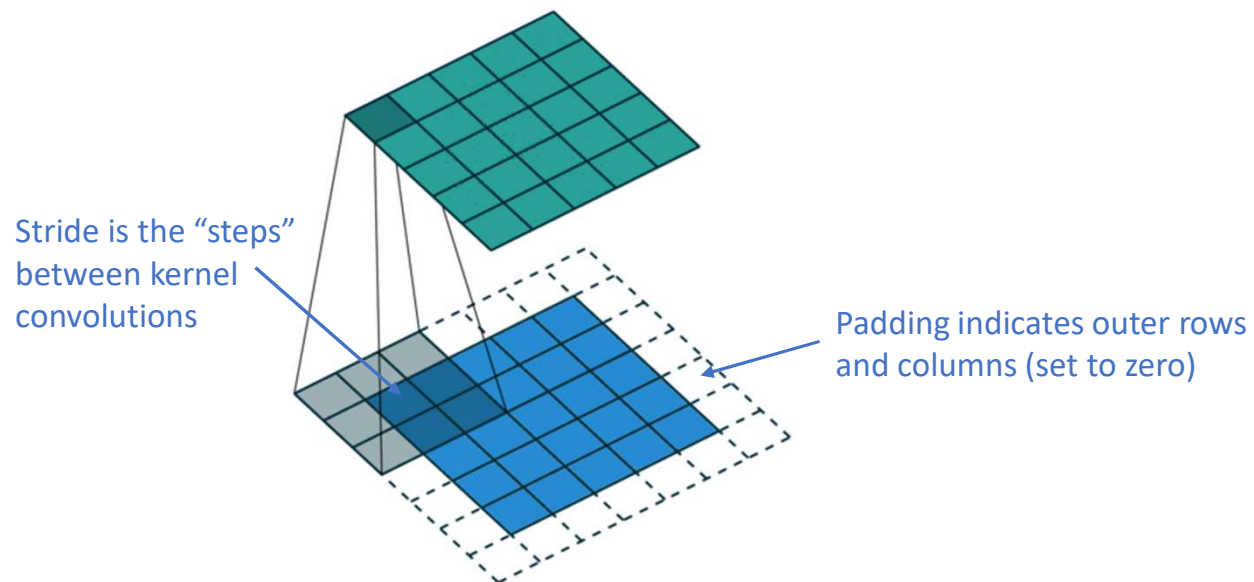
---

Original	Gaussian Blur	Sharpen	Edge Detection
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
			

Source: <https://medium.com/analytics-vidhya/understanding-convolution-operations-in-cnn-1914045816d4>

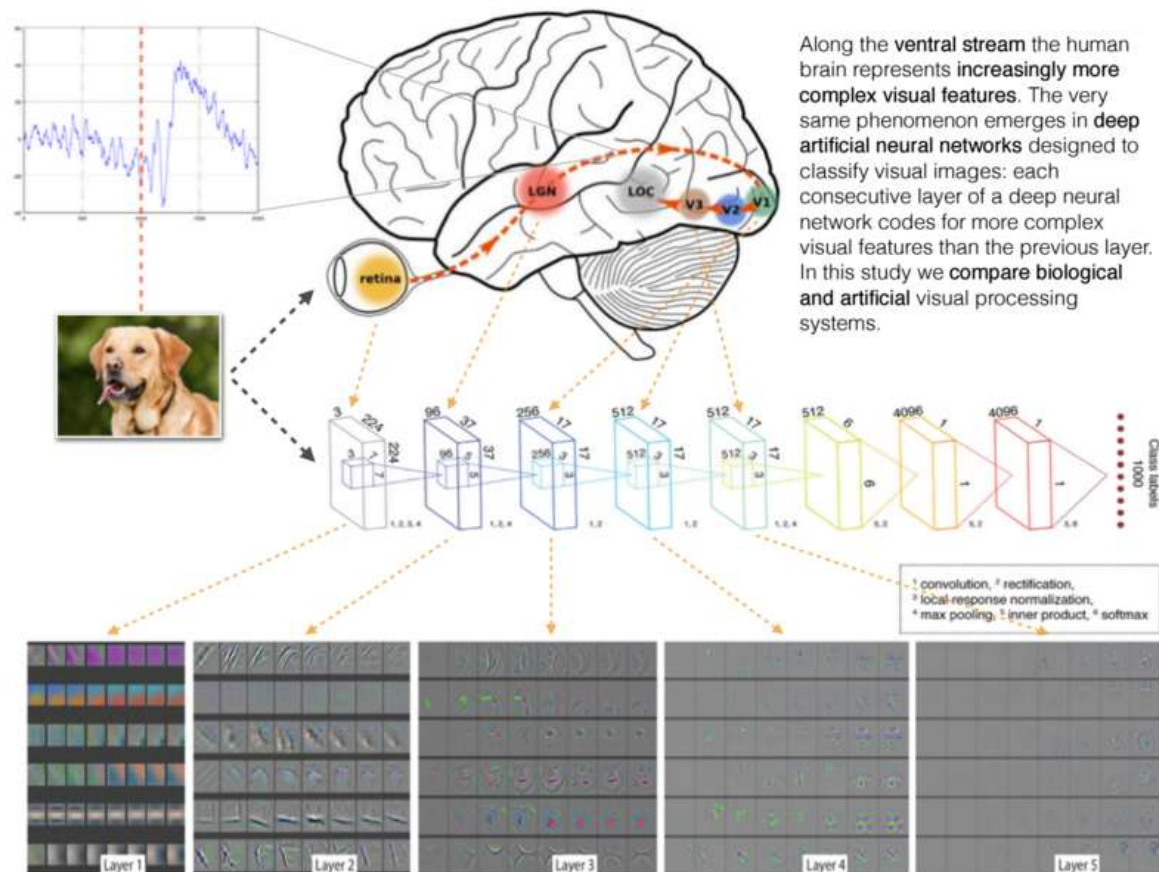
# Padding and stride

---



Source: <https://medium.datadriveninvestor.com/convolutional-neural-networks-3b241a5da51e>

# The Brain's visual cortex



# CNN example

