Artificial Intelligence

CS3AI18/ CSMAI19

Lecture - 7/10: Deep Learning

DR VARUN OJHA

Department of Computer Science

University of Reading

Learning objectives

By the end of this week, you will be able to:

- Learn basic concepts of reinforcement learning
- Learn basic concepts Supervised learning
- Learn neural network and issues with training a neural network
- Learn concepts of convolutional neural network
- Workout an example problem of convolutional neural network

Content of this week

- Part 1: Reinforcement Leaning
- Part 2: Supervised Learning
  - Linear and non-linear regression
  - Cost functions (Classification and Regression)
- Part 3: Neural Network
  - Shallow network
  - Activation functions
  - Deep learning
- Part 4: Convolutional Neural network (CNN)
- Part 5: Practical Exercise (CNN)
- Quiz

Artificial Intelligence

CS3AI18/ CSMAI19

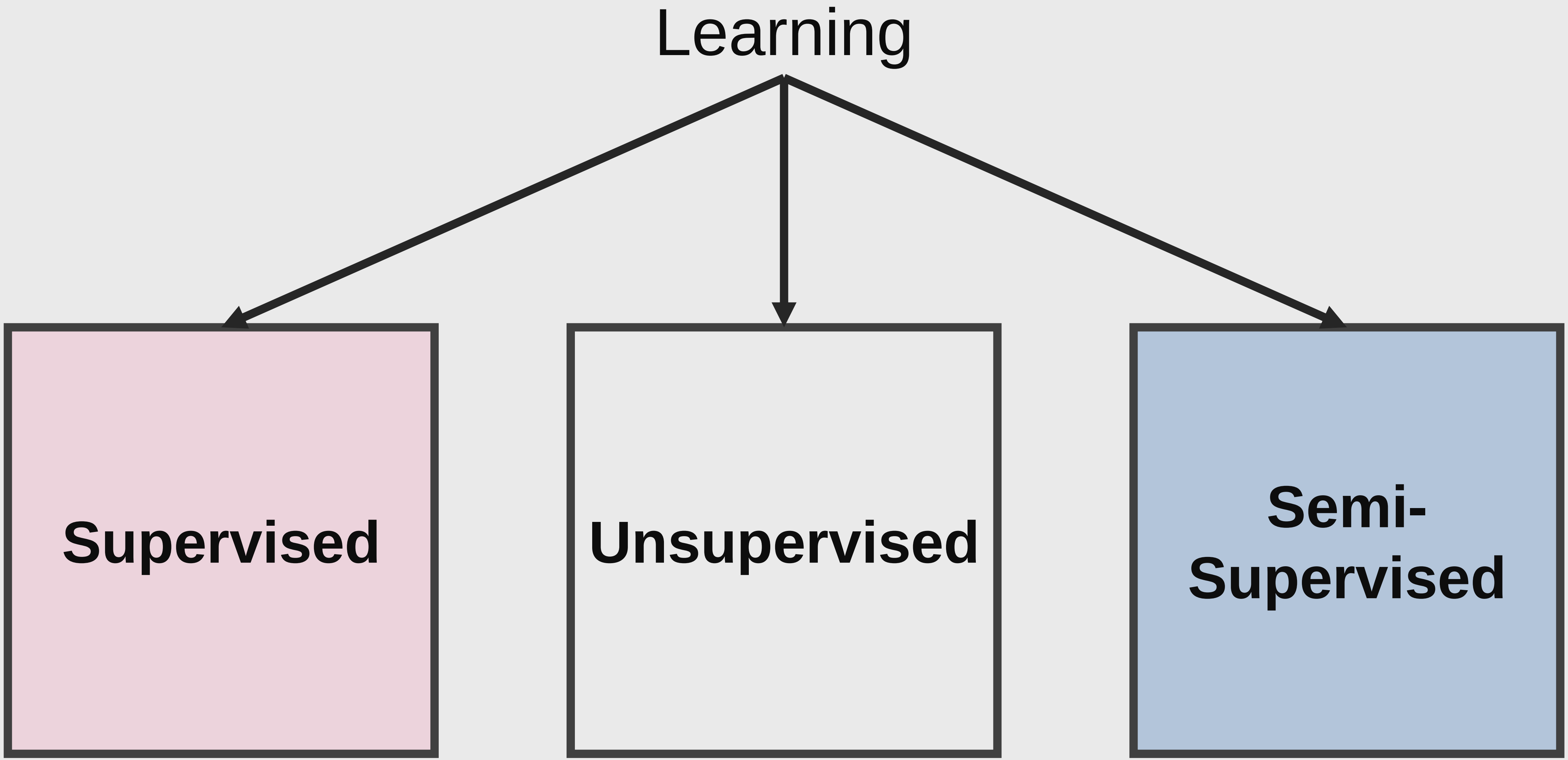
Lecture - 6/10: Learning

Part 1
Reinforcement Learning

DR VARUN OJHA

Department of Computer Science

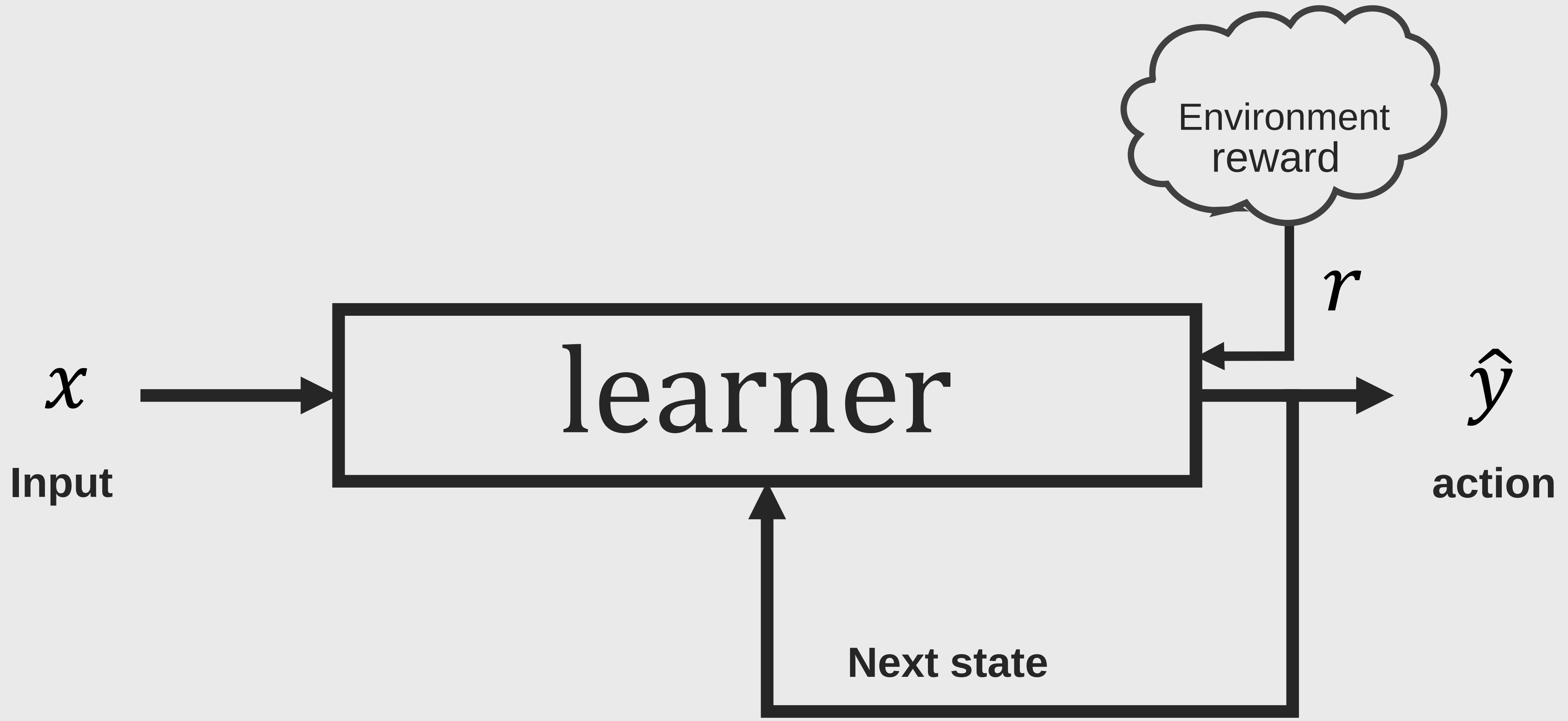
University of Reading


Learning
Supervised
Unsupervised
Semi-
Supervised

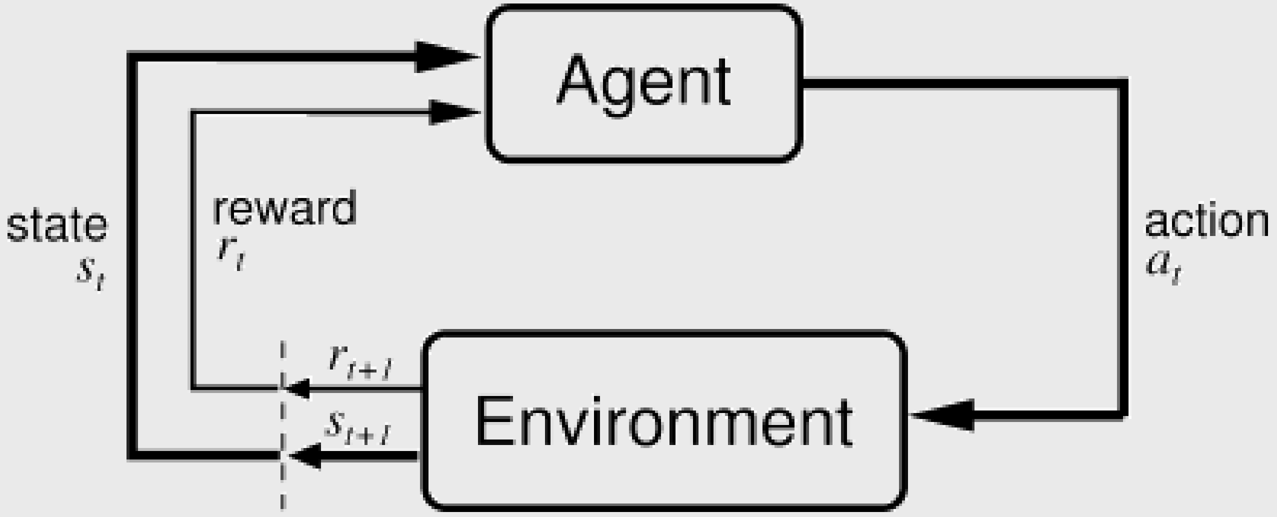
Part 1

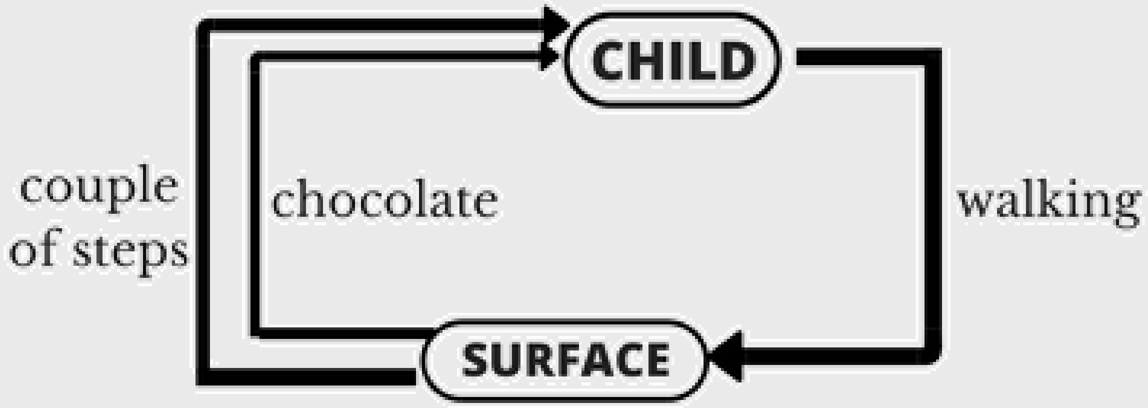
Reinforcement Learning

Semi-Supervised

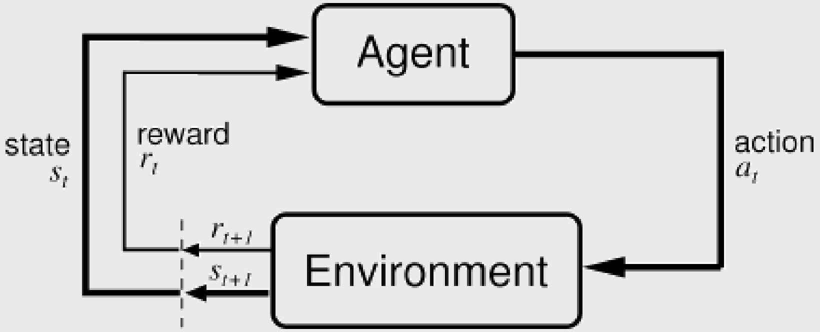


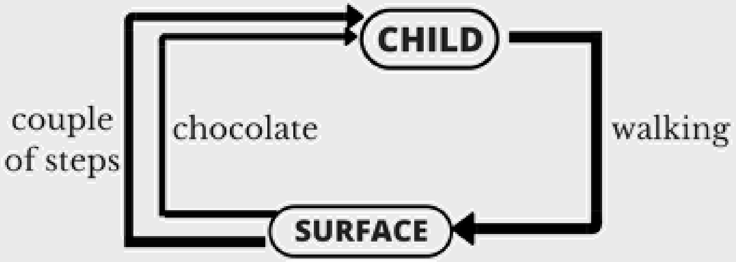
Reinforcement Learning





Source:
https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/





- the **problem statement**: to walk,
- the **agent**: a child
- the **environment:** a surface on which to walk
- an **actions:** walking one step to another
- a **reward:** a chocolate
  - child receives a chocolate for taking a step (a **positive reward**)
  - child do not receive a chocolate for not taking a step (a **negative reward**)

Source: https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/

Reinforcement Learning: Implementation

Markov Decision Process details are taught in lecture 6

- The **mathematical framework** for defining a solution in reinforcement learning scenario is called **Markov Decision Process**. Which can be designed as:
  - Set of states, S
  - Set of actions, A
  - Reward function, R
  - Policy, π – a set of actions taken defines the policy (π)
  - Value, V – rewards given on an action defines the value V

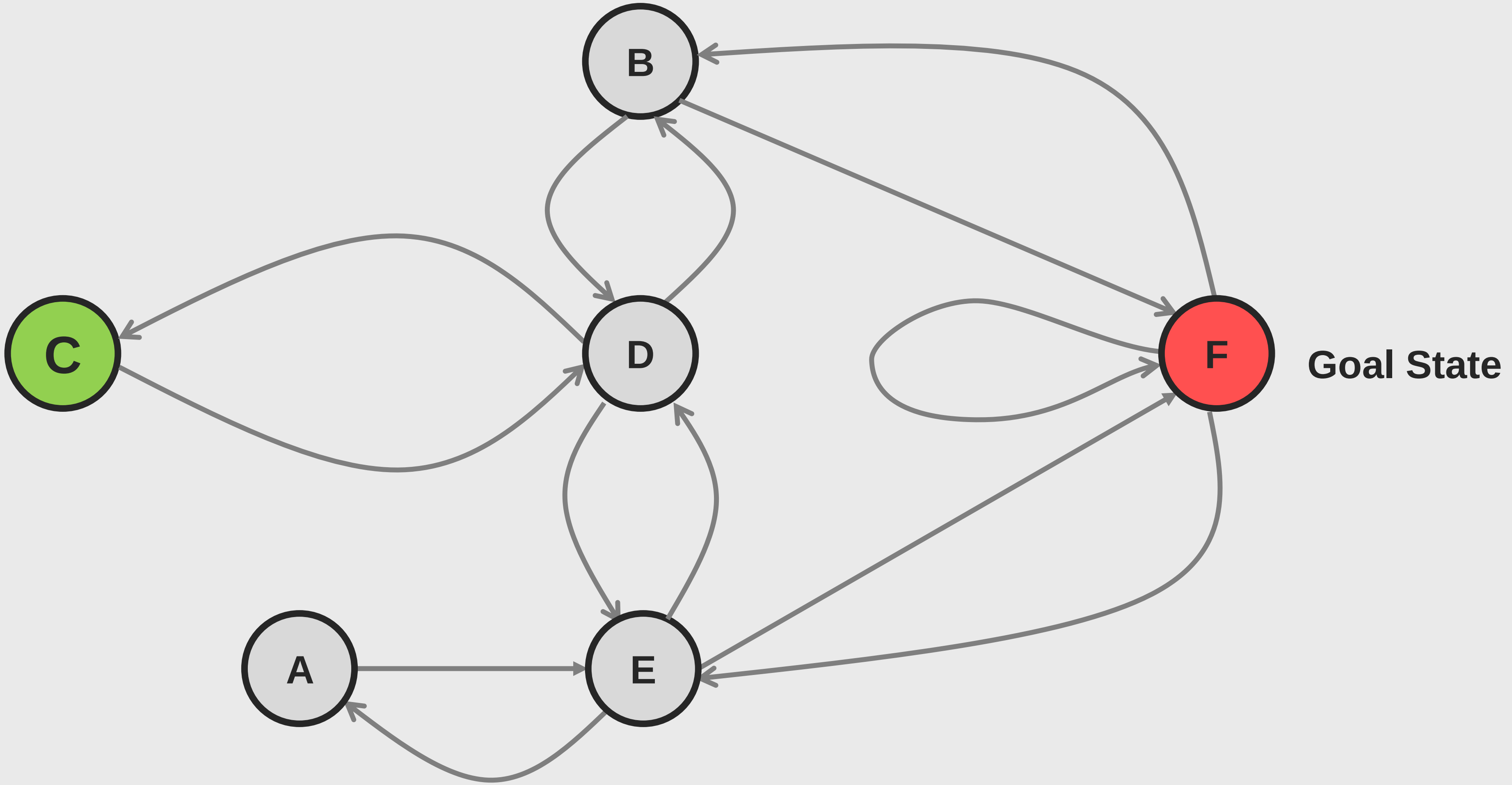
Reinforcement Learning: Implementation

Markov Decision Process details are taught in lecture 6

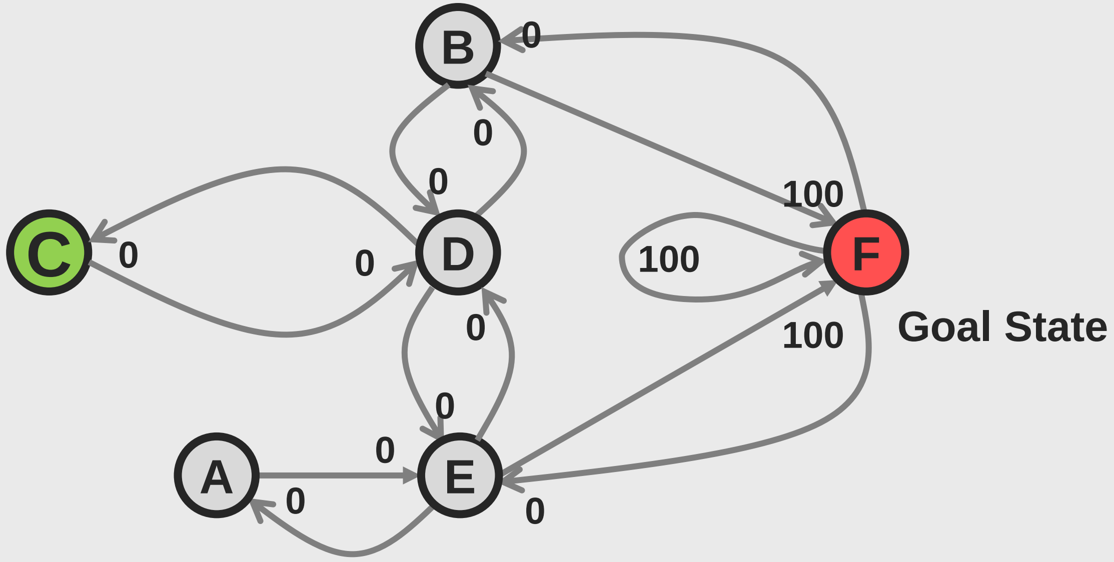
- For a:
  - Set of states, S
  - Set of actions, A
  - Reward function, R
  - Policy, π – a set of actions taken defines the policy (π)
  - Value, V – rewards given on an action defines the value V
- **Reinforcement Learning aims at maximizing**

$$E(r_t \mid \pi, s_t)$$

Shortest Path Problem



Source of example: http://mnemstudio.org/path-finding-q-learning-tutorial.htm



Source: https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/

Epsilon greedy algorithm

Set of states $\boldsymbol{S}$ are the nodes, e.g.,
$\boldsymbol{S} = \{A, B, C, D, E, F\}$.

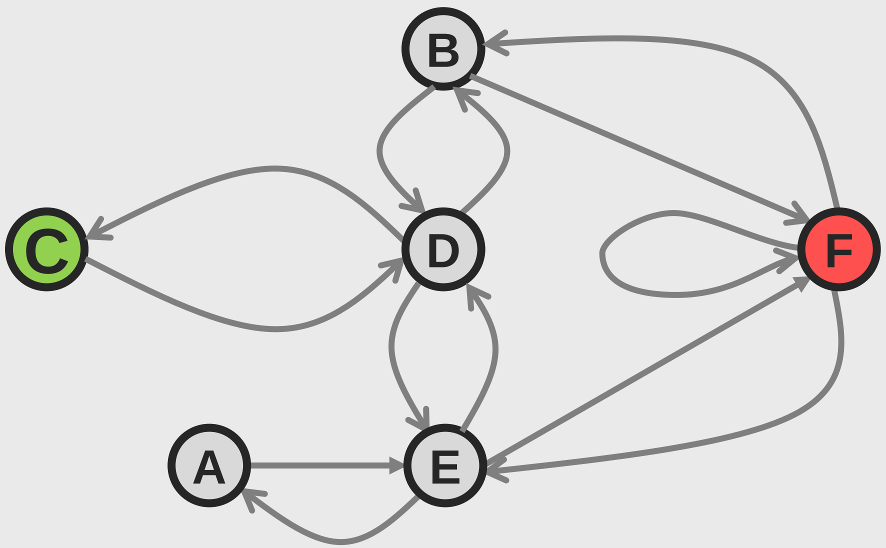
Set of actions $\boldsymbol{A}$ are going from one node to another e.g.,
$\boldsymbol{A} = \{A \rightarrow E, C \rightarrow D, E \rightarrow F, etc\}$.

The reward function is the value represented by edge, e.g. the cost of an action:

$A \rightarrow D \Rightarrow -1$ (costly move – negative reward)

$B \rightarrow F \Rightarrow +100$ (promising move – positive reward)

The **policy** is the "way" to complete the task, e.g.,
$\pi = \{A \rightarrow E \rightarrow F\}$
Or
$\pi = \{B \rightarrow D \rightarrow E \rightarrow F\}$



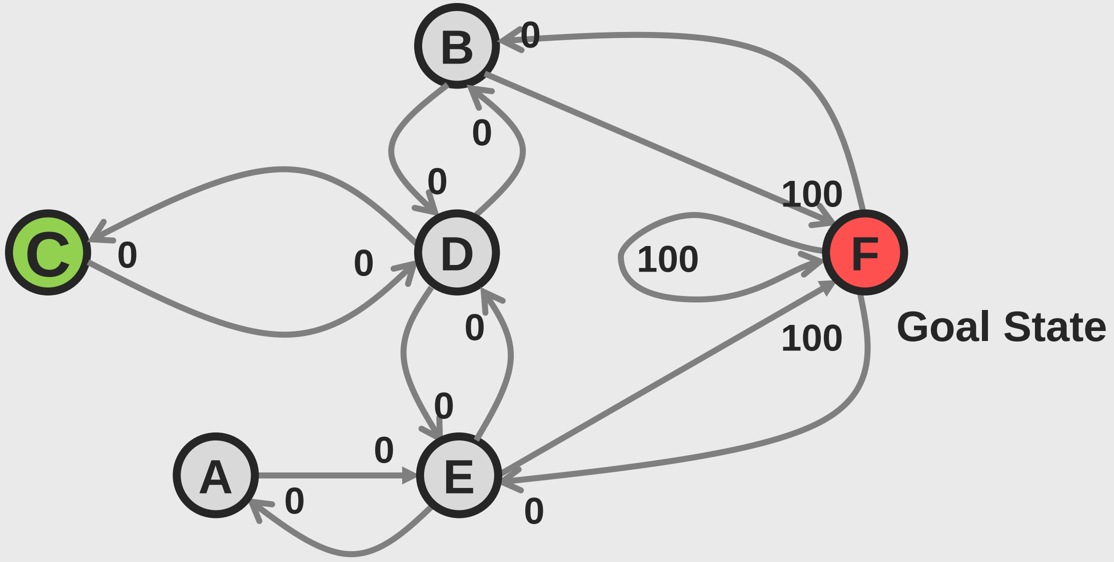
Epsilon greedy algorithm is not **optimal** since its only exploit current state, i.e., pure exploitation

pure exploration vs pure exploitation

We need to explore and exploit both

Policy based learning

Source: https://www.analyticsvidhya.com/blog/2017/01/introduction-to-reinforcement-learning-implementation/



Policy based learning

Policy based,
where our focus is to find ***optimal policy***

Value based,
where our focus is to find ***optimal value***, i.e. cumulative reward

Action based,
where our focus is on what ***optimal actions*** to take at each step

Q-Learning: a policy-based learning

1. **Initialize** the **Values** table $Q(s, a)$ and learning rate γ.
2. Observe the current state $s = s_t$.
3. Choose an action a_t for state s_t based on one of the action selection policies (e.g. epsilon greedy)
4. Take the action, and observe the reward r_t as well as the new state s_{t+1} .
5. Update the **Value** for the state using the observed reward and the maximum reward possible for the next state as per**.**

Set the state to the new state $s = s_{t+1}$, and repeat the process until a terminal state s_{goal} is reached.

Q-Learning: A policy-based learning

1. **Initialize** the **Values** table $Q(s,a)$. Also, called **Q Table**

   Observe the current state $s = s_t$.

2. **Choose** an action a_t for state s_t based on one of the action selection policies (e.g. epsilon greedy)

3. **Perform** the action,

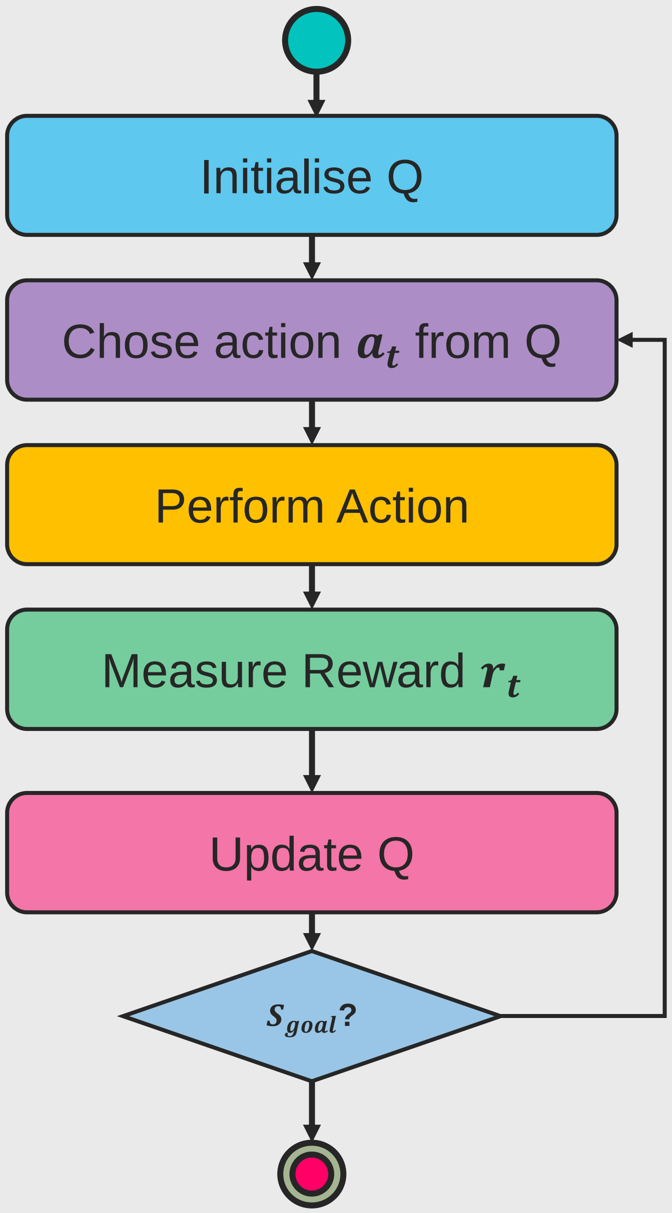
4. **Measure** the reward r_t as well as the new state s_{t+1} .

5. **Update Q** the **Value** for the state using the observed reward and the maximum reward possible for the next state as per

$$Q(s,a) = R(s,a) + \gamma \max_{a' \in A} \{Q(s',a')\}$$

// Q(state, action) = R(state, action) + learning rate * Max[Q(next state, all actions)]

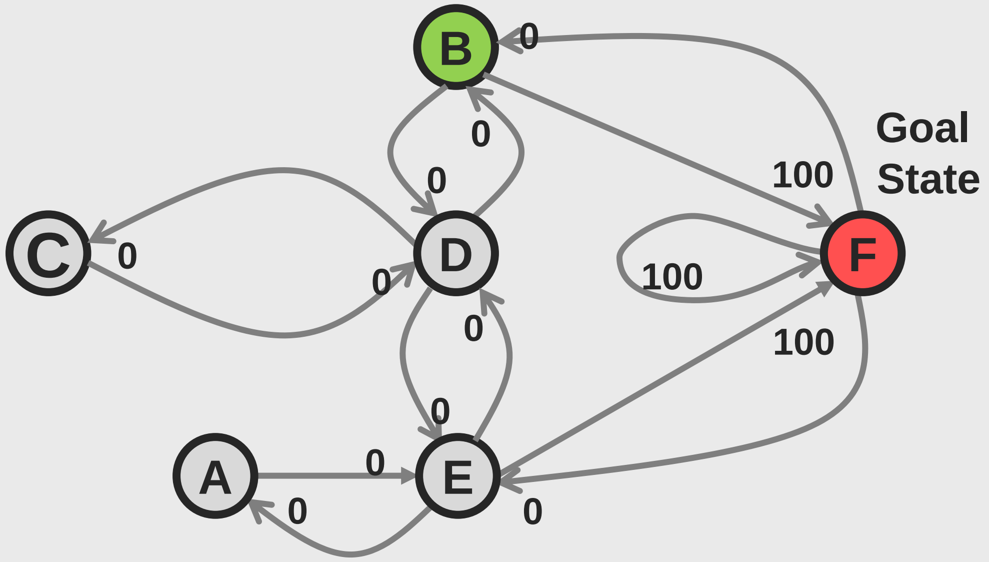
1. Set the state to the new state $s = s_{t+1}$, and repeat the process until a terminal state s_{goal} is reached.



Initialize R Table (Input Problem)

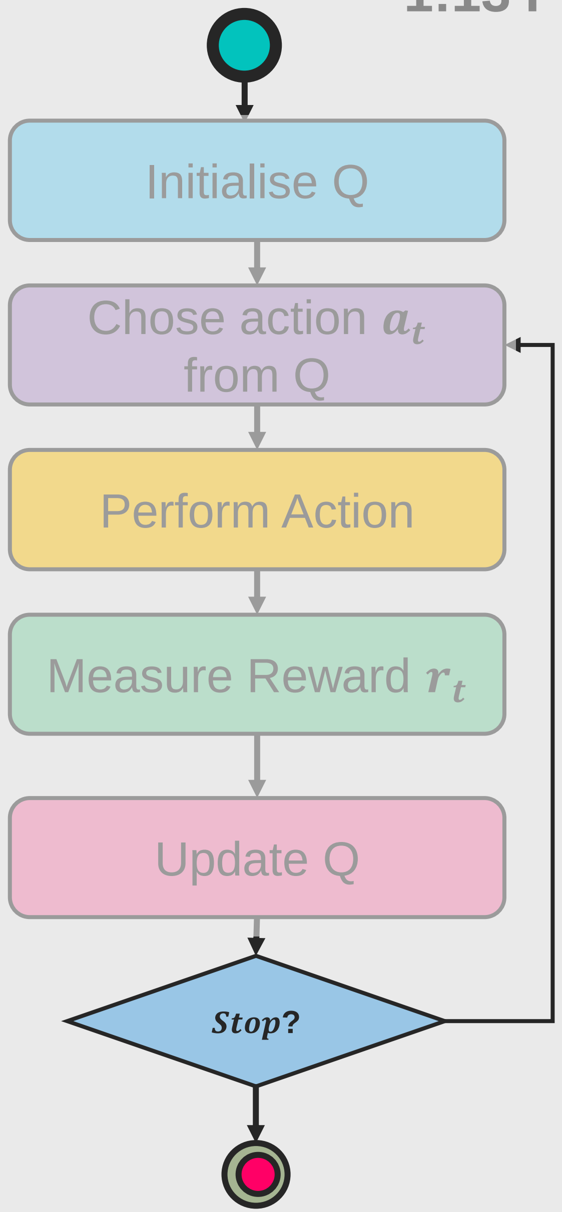
(typically given by problem definition, given $\gamma = 0.8$)

1:13 PM



R =

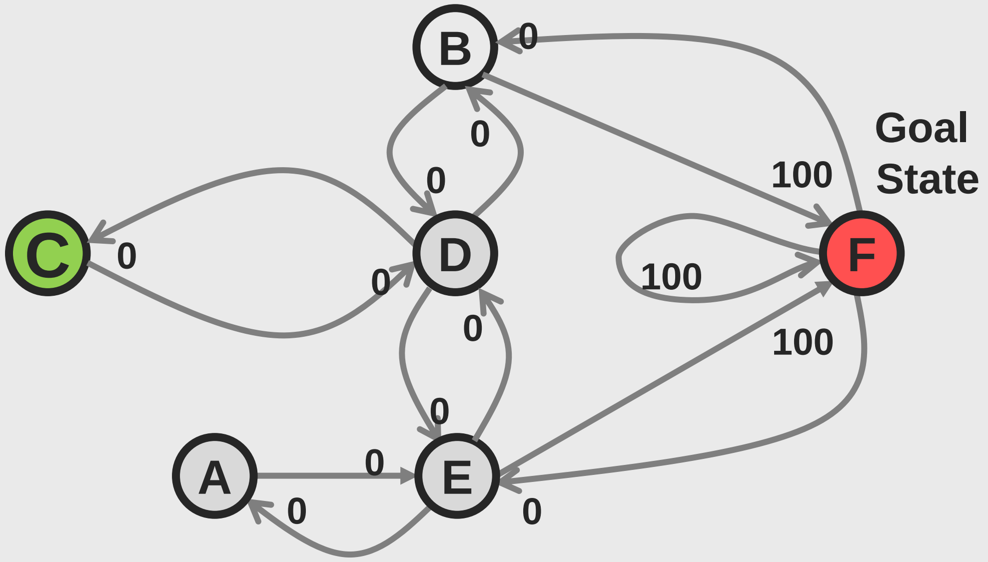
	A	B	C	D	E	F
A	-1	-1	-1	-1	**0**	-1
B	-1	-1	-1	**0**	-1	**100**
C	-1	-1	-1	**0**	-1	-1
D	-1	**0**	**0**	-1	**0**	-1
E	0	-1	-1	**0**	-1	**100**
F	-1	**0**	-1	-1	**0**	**100**

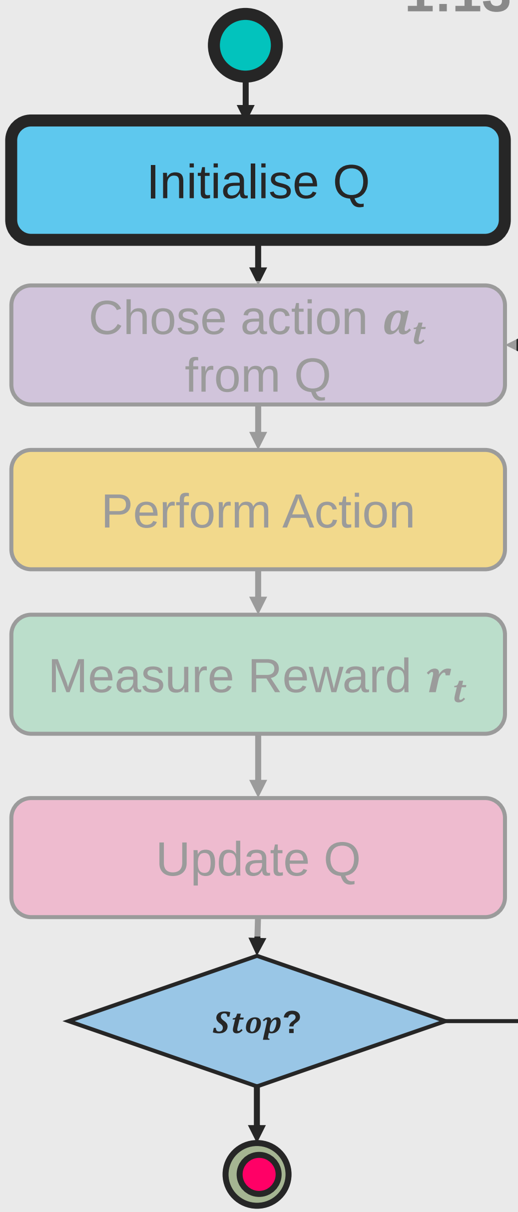


Source of example: http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Initialize Q Table

(given $\gamma = 0.8$, typically initial Q value set to Zero)





R =

	A	B	C	D	E	F
A	-1	-1	-1	-1	**0**	-1
B	-1	-1	-1	**0**	-1	**100**
C	-1	-1	-1	**0**	-1	-1
D	-1	**0**	**0**	-1	**0**	-1
E	0	-1	-1	**0**	-1	**100**
F	-1	**0**	-1	-1	**0**	**100**

Q =

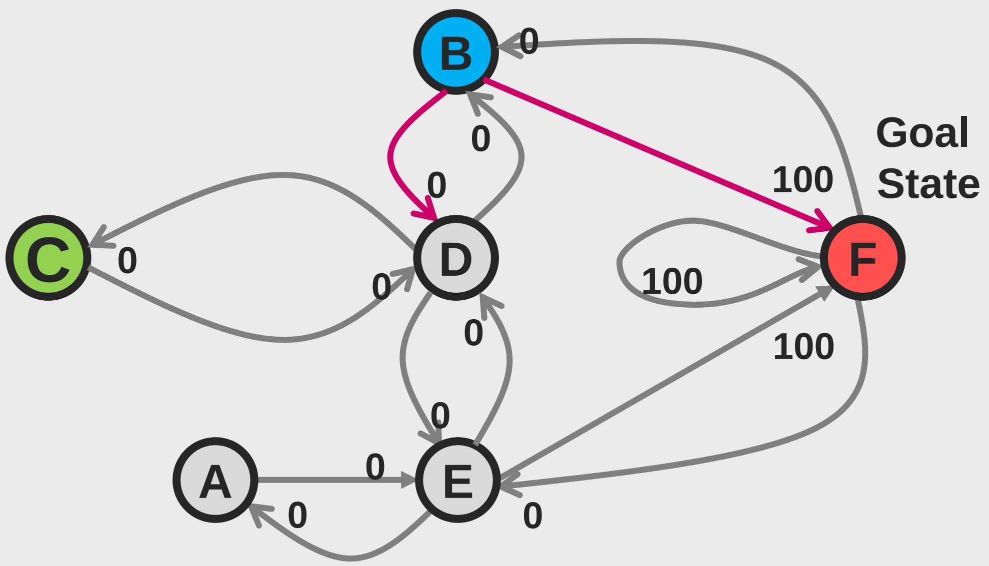
	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Source of example: http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Chose an Action

1:13 PM

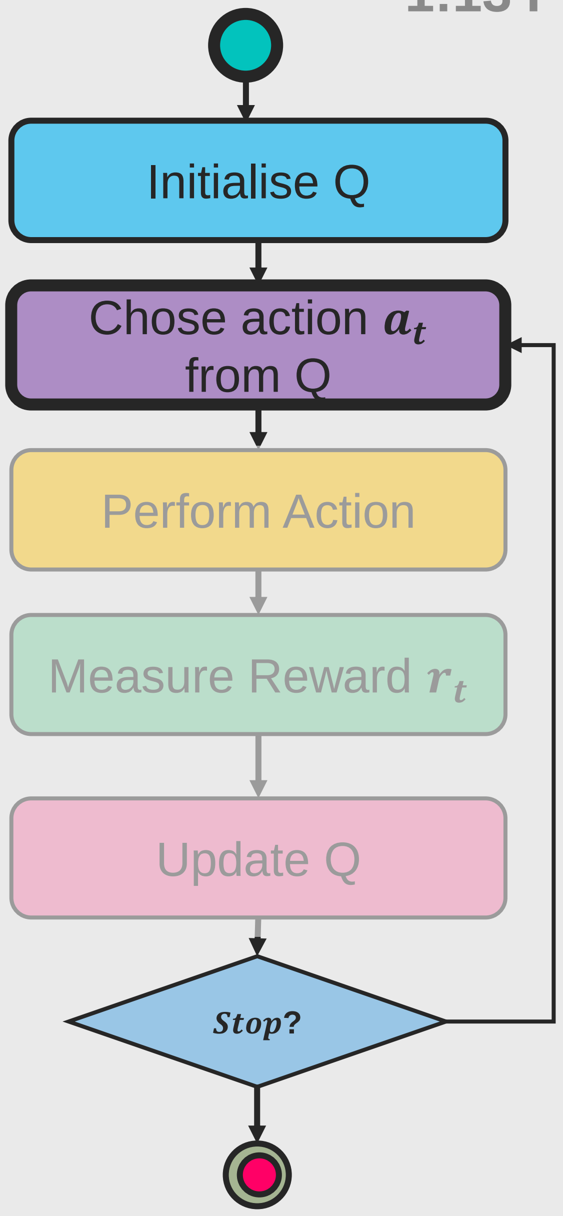
(given $\gamma = 0.8$, say,
let choose a **random state**, say its B



we have possible actions

$\pi_{a_{B \to D}} = \{B \to D\}$

$\pi_{a_{B \to F}} = \{B \to F\}$



R =

	A	B	C	D	E	F
A	-1	-1	-1	-1	0	-1
B	-1	-1	-1	0	-1	100
C	-1	-1	-1	0	-1	-1
D	-1	0	0	-1	0	-1
E	0	-1	-1	0	-1	100
F	-1	0	-1	-1	0	100

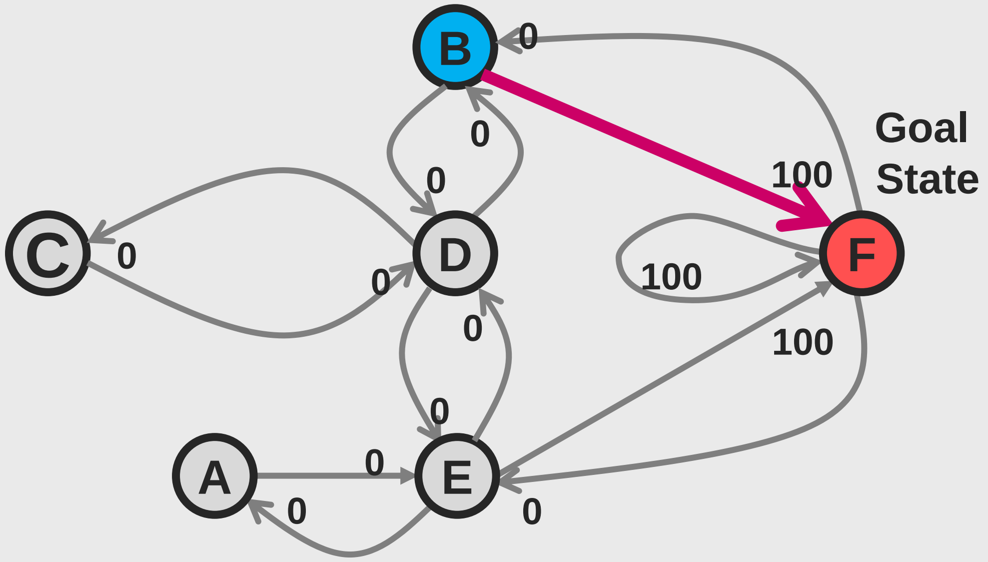
Q =

	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Source of example: http://mnemstudio.org/path-finding-q-learning-tutorial.htm

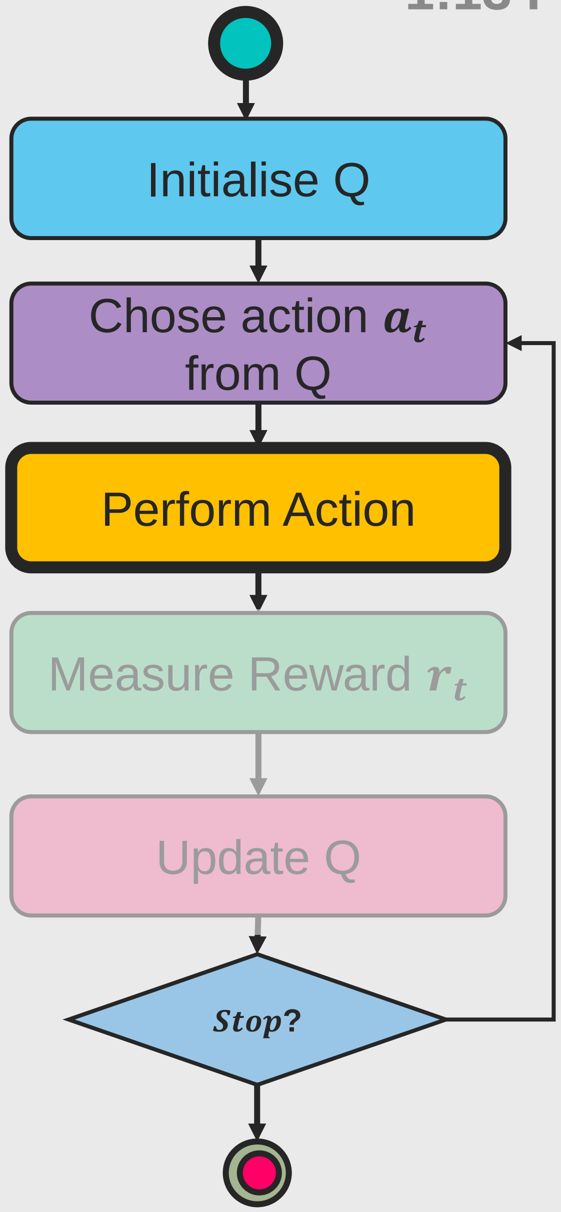
Perform Action

(given $\gamma = 0.8$, say, select a random action chosen from B



Say we performed action by chance

$$\pi_{a_{B \to F}} = \{B \to F\}$$



R =

	A	B	C	D	E	F
A	-1	-1	-1	-1	**0**	-1
B	-1	-1	-1	**0**	-1	**100**
C	-1	-1	-1	**0**	-1	-1
D	-1	**0**	**0**	-1	**0**	-1
E	0	-1	-1	**0**	-1	**100**
F	-1	**0**	-1	-1	**0**	**100**

Q =

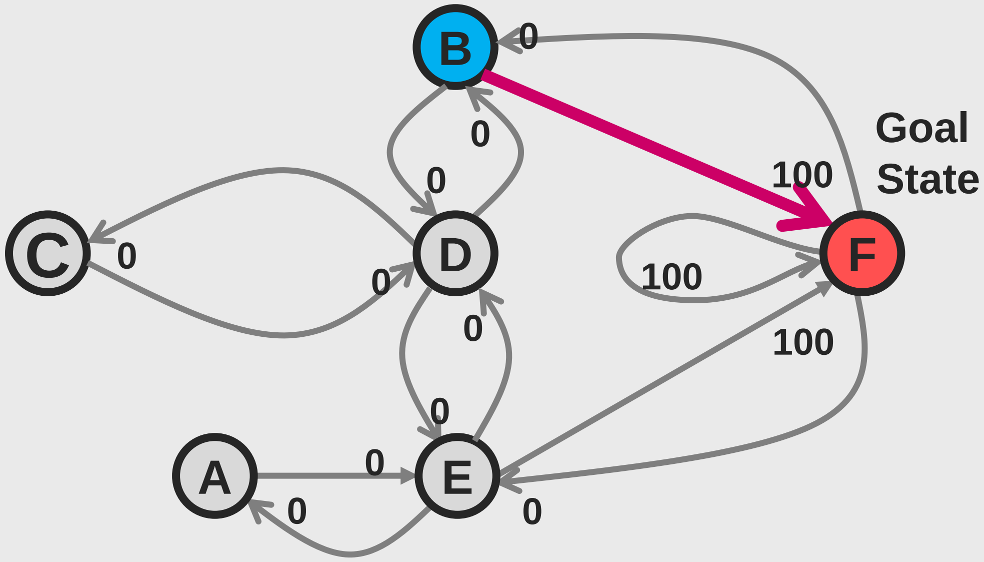
	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Source of example: http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Measure Reward

(given $\gamma = 0.8$)

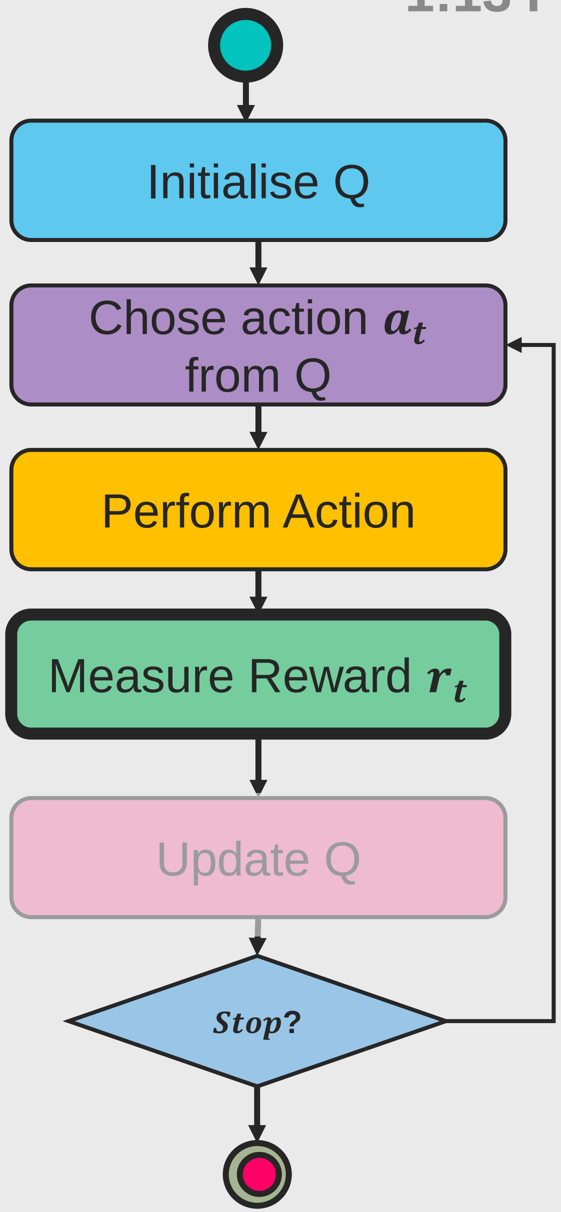
1:13 PM



Reward for action:

$$\pi_{a_{B\to F}} = \{B \to F\}$$

$$R(s, \pi_{a_{B\to F}}) = 100$$



R =

	A	B	C	D	E	F
A	-1	-1	-1	-1	0	-1
B	-1	-1	-1	0	-1	100
C	-1	-1	-1	0	-1	-1
D	-1	0	0	-1	0	-1
E	0	-1	-1	0	-1	100
F	-1	0	-1	-1	0	100

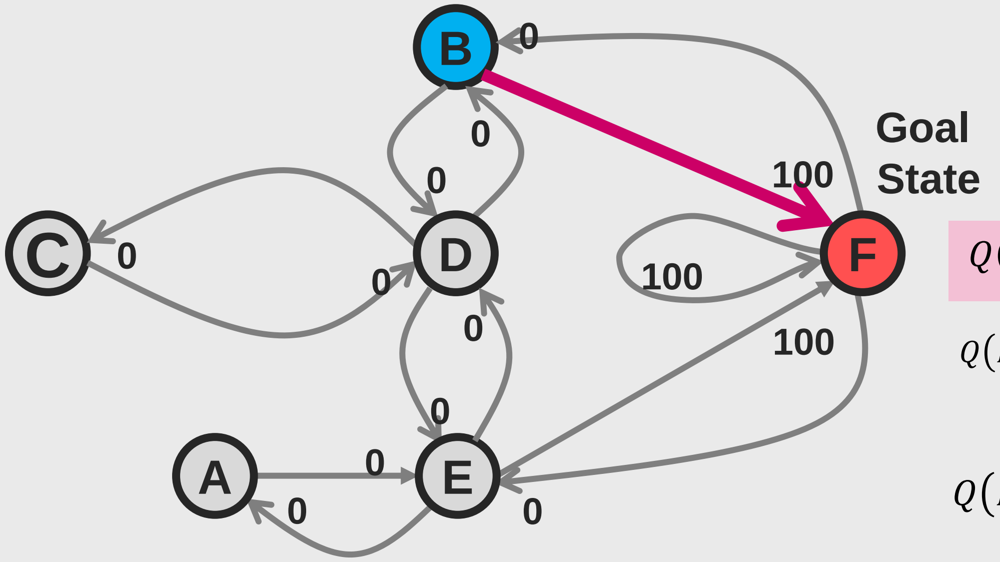
Q =

	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	0
C	0	0	0	0	0	0
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Source of example: http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Update Q

(given $\gamma = 0.8$, say, current state is $S = B$



Reward for action:

$$\pi_{a_{B\to F}} = \{B \to F\}$$

$$R(s, \pi_{a_{B\to F}}) = \mathbf{100}$$

$$Q(s,a) = R(s,a) + \gamma \max_{a' \in A} \{Q(s', a')\}$$

$$Q(B, \pi_{a_{B\to F}}) = R(B, \pi_{a_{B\to F}}) + 0.8 \max\{Q(F, \pi_{a_{F\to B}}), Q(F, \pi_{a_{F\to E}}), Q(F, \pi_{a_{F\to F}})\}$$

$$Q(B, \pi_{a_{B\to F}}) = 100 + 0.8 \max\{0,0,0\} = 100 + 0.8 * 0 = 100$$

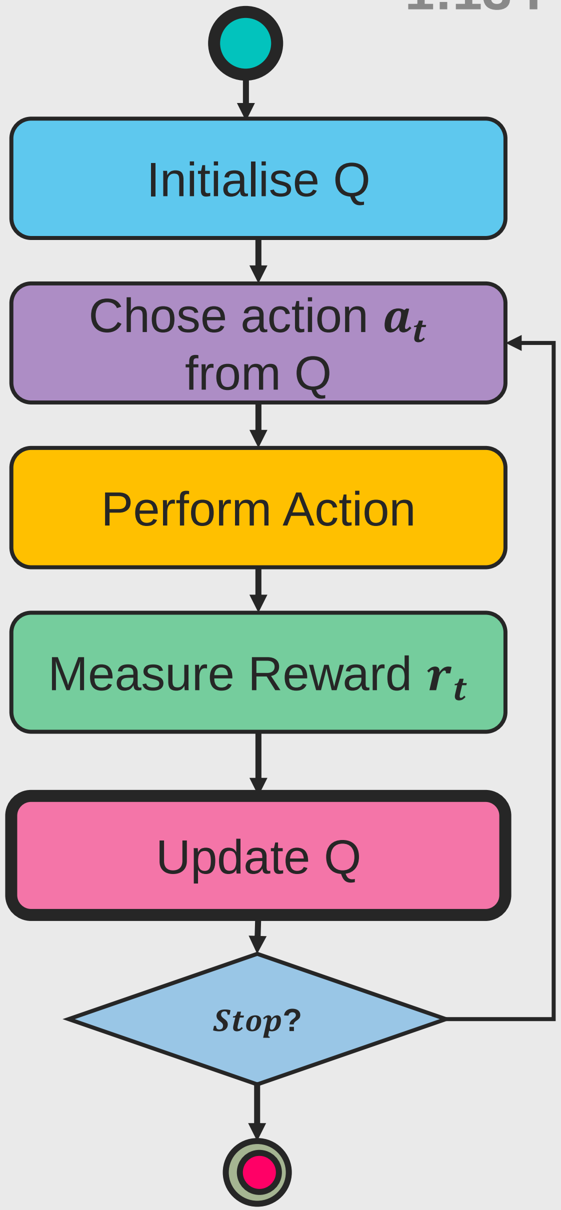
R =

	A	B	C	D	E	F
A	-1	-1	-1	-1	0	-1
B	-1	-1	-1	0	-1	100
C	-1	-1	-1	0	-1	-1
D	-1	0	0	-1	0	-1
E	0	-1	-1	0	-1	100
F	-1	0	-1	-1	0	100

Q =

	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	100
C	0	0	0	0	0	0
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

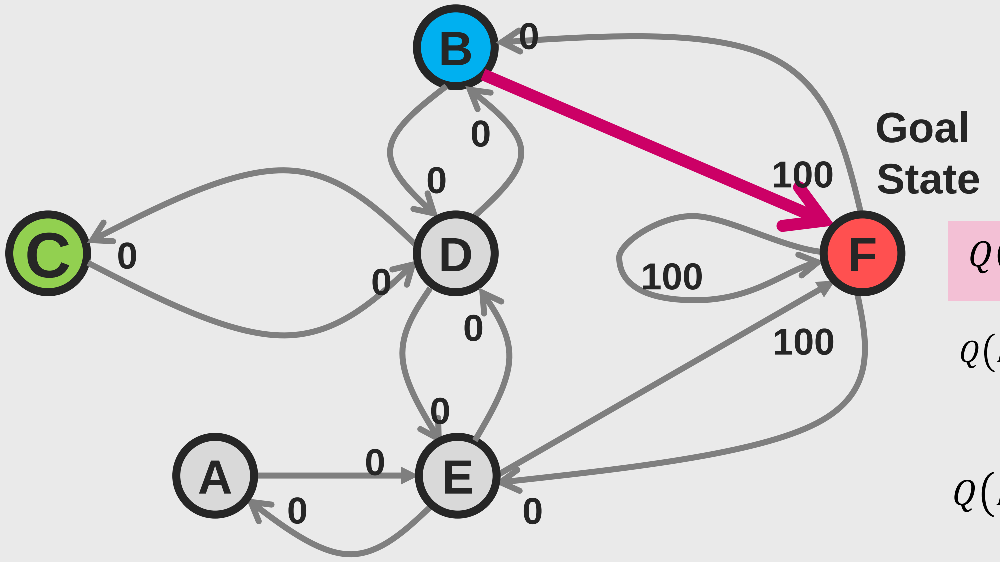
1:13 PM



Source of example: http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Goal State?

(given $\gamma = 0.8$, say, may be not,
we have not find best path



Reward for action:

$$\pi_{a_{B\to F}} = \{B \to F\}$$

$$R(s, \pi_{a_{B\to F}}) = \mathbf{100}$$

$$Q(s,a) = R(s,a) + \gamma \max_{a' \in A} \{Q(s', a')\}$$

$$Q(B, \pi_{a_{B\to F}}) = R(B, \pi_{a_{B\to F}}) + 0.8 \max \{Q(F, \pi_{a_{F\to B}}), Q(F, \pi_{a_{F\to E}}), Q(F, \pi_{a_{F\to F}})\}$$

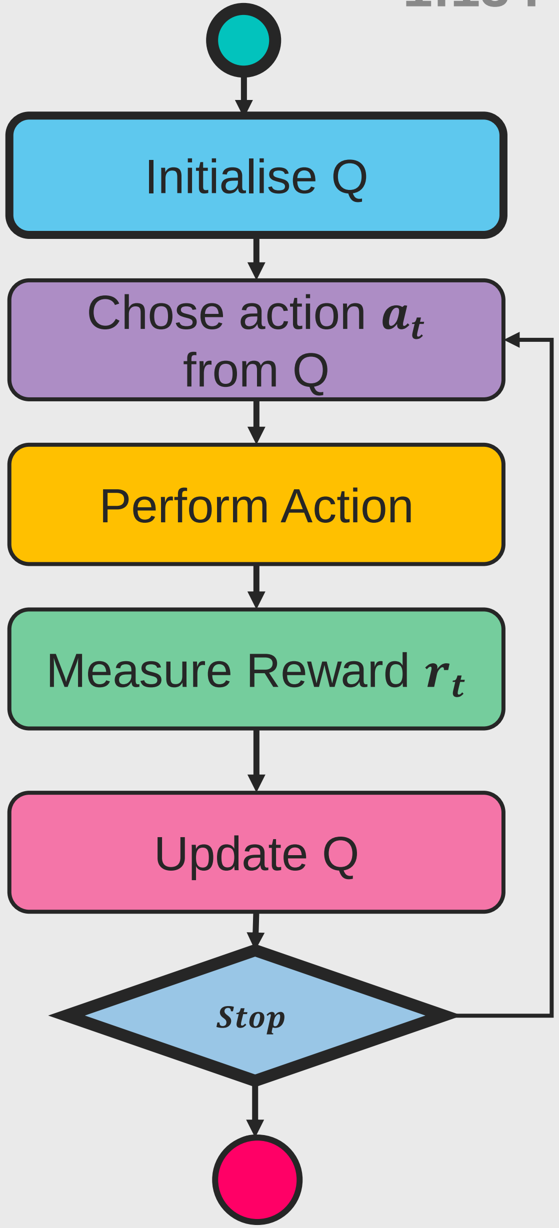
$$Q(B, \pi_{a_{B\to F}}) = 100 + 0.8 \max\{0,0,0\} = 100 + 0.8 * 0 = 100$$

R =

	A	B	C	D	E	F
A	-1	-1	-1	-1	0	-1
B	-1	-1	-1	0	-1	100
C	-1	-1	-1	0	-1	-1
D	-1	0	0	-1	0	-1
E	0	-1	-1	0	-1	100
F	-1	0	-1	-1	0	100

Q =

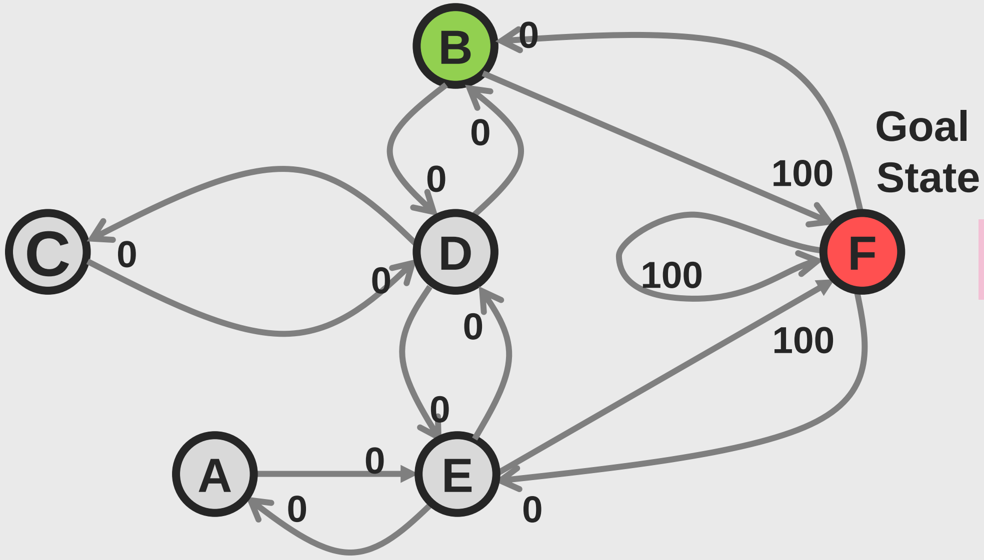
	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	**100**
C	0	0	0	0	0	0
D	0	0	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0



Source of example: http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Next Iteration

(if goal is to find more path)



Chose a random state say "D"

$\boldsymbol{\pi_{a_{D\to B}} = \{D \to B\}}$

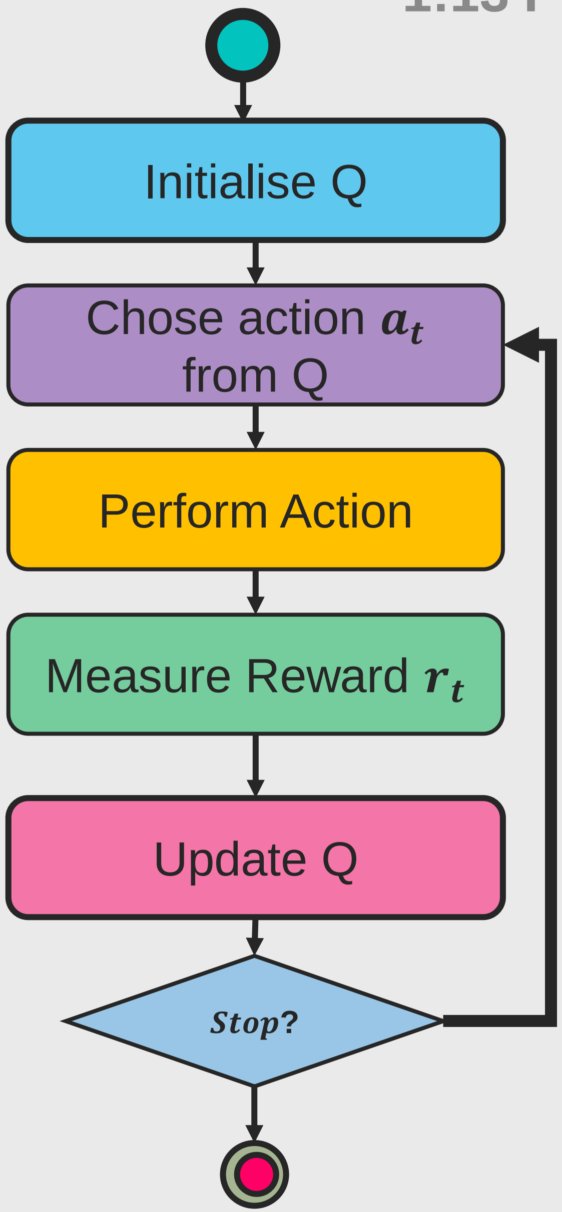
$\pi_{a_{D\to C}} = \{D \to C\}$

$\pi_{a_{D\to E}} = \{D \to E\}$

$$Q(s,a) = R(s,a) + \gamma \max_{a' \in A} \{Q(s',a')\}$$

$$Q(D,\pi_{a_{D\to B}}) = R(D,\pi_{a_{D\to B}}) + 0.8 \max\{Q(B,\pi_{a_{B\to D}}), Q(B,\pi_{a_{B\to F}})\}$$

$$Q(B,\pi_{a_F}) = 0 + 0.8 \max\{0,100\} = 0 + 0.8 * 100 = 80$$



R =

	A	B	C	D	E	F
A	-1	-1	-1	-1	0	-1
B	-1	-1	-1	0	-1	100
C	-1	-1	-1	0	-1	-1
D	-1	0	0	-1	0	-1
E	0	-1	-1	0	-1	100
F	-1	0	-1	-1	0	100

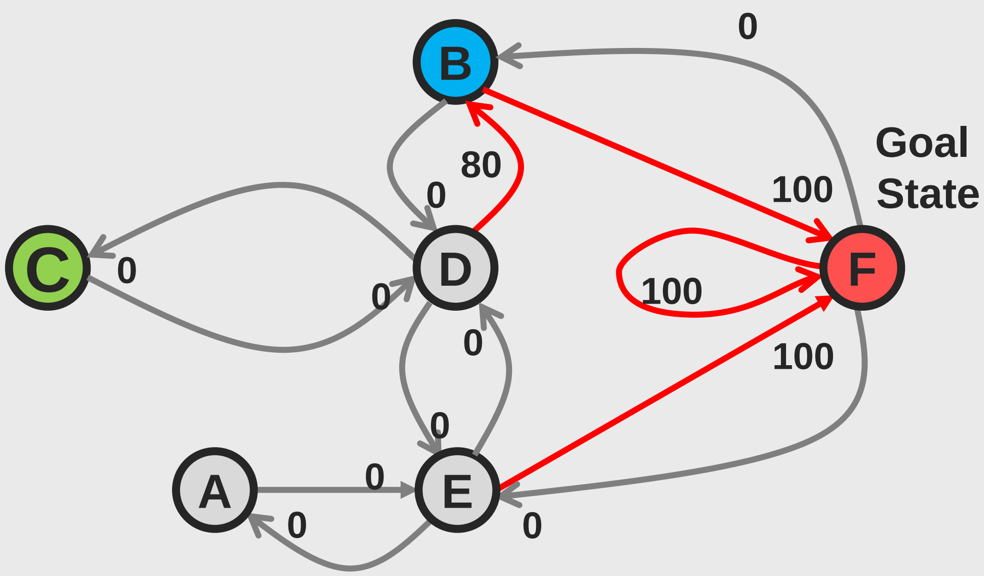
Q =

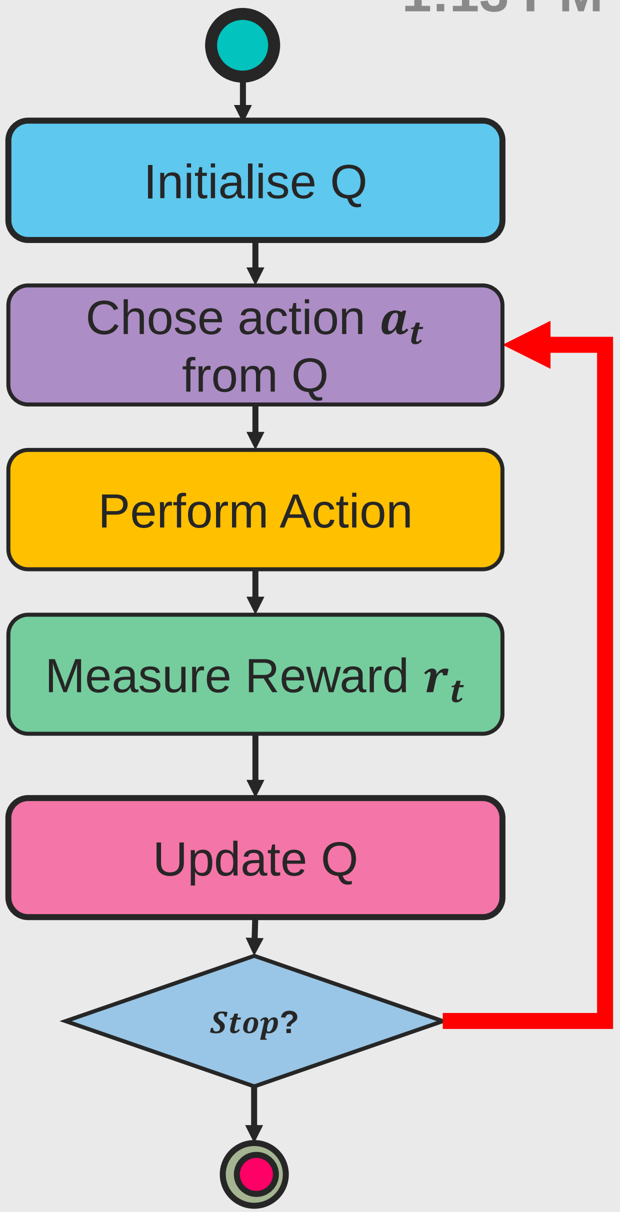
	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	100
C	0	0	0	0	0	0
D	0	80	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

Source of example: http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Stable State?

(if we want to find optimal path from – we continue until Q Table converge





R =

	A	B	C	D	E	F
A	-1	-1	-1	-1	0	-1
B	-1	-1	-1	0	-1	100
C	-1	-1	-1	0	-1	-1
D	-1	0	0	-1	0	-1
E	0	-1	-1	0	-1	100
F	-1	0	-1	-1	0	100

Q =

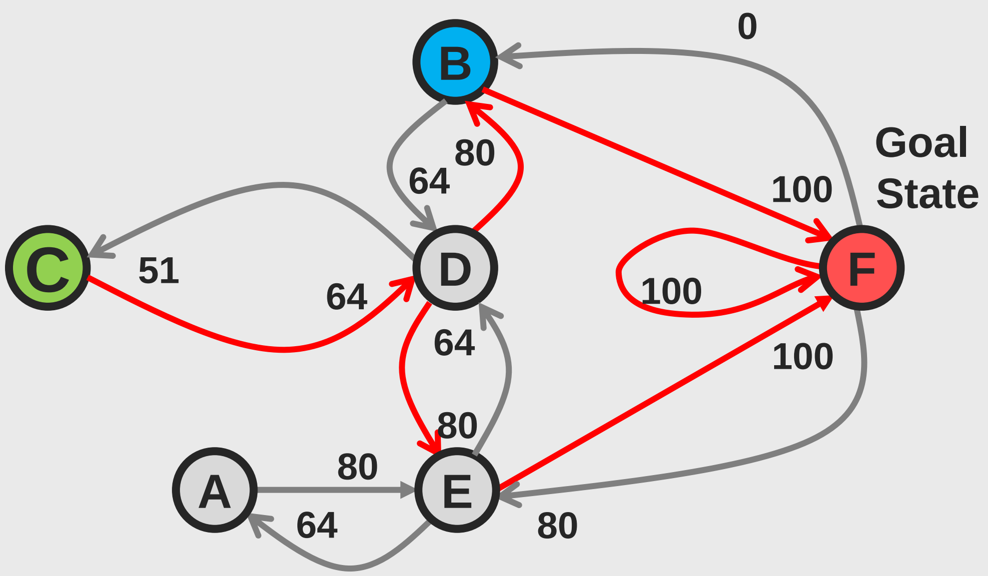
	A	B	C	D	E	F
A	0	0	0	0	0	0
B	0	0	0	0	0	100
C	0	0	0	0	0	0
D	0	80	0	0	0	0
E	0	0	0	0	0	0
F	0	0	0	0	0	0

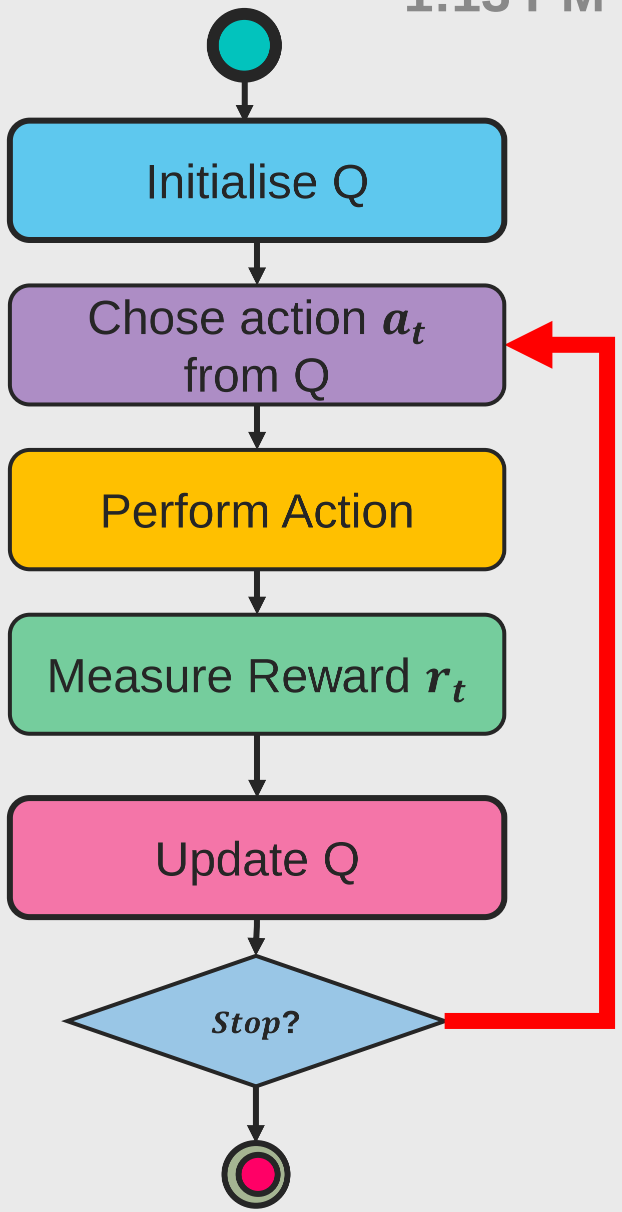
Source of example: http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Let Say After Many Iterations

(Q Table converge)

1:13 PM





R =

	A	B	C	D	E	F
A	-1	-1	-1	-1	**0**	-1
B	-1	-1	-1	**0**	-1	**100**
C	-1	-1	-1	**0**	-1	-1
D	-1	**0**	**0**	-1	**0**	-1
E	0	-1	-1	**0**	-1	**100**
F	-1	**0**	-1	-1	**0**	**100**

Q =

	A	B	C	D	E	F
A	0	0	0	0	**80**	0
B	0	0	0	**64**	0	**100**
C	0	0	0	**64**	0	0
D	0	**80**	**51**	0	**80**	0
E	**64**	0	0	**64**	0	0
F	0	**80**	0	0	**80**	0

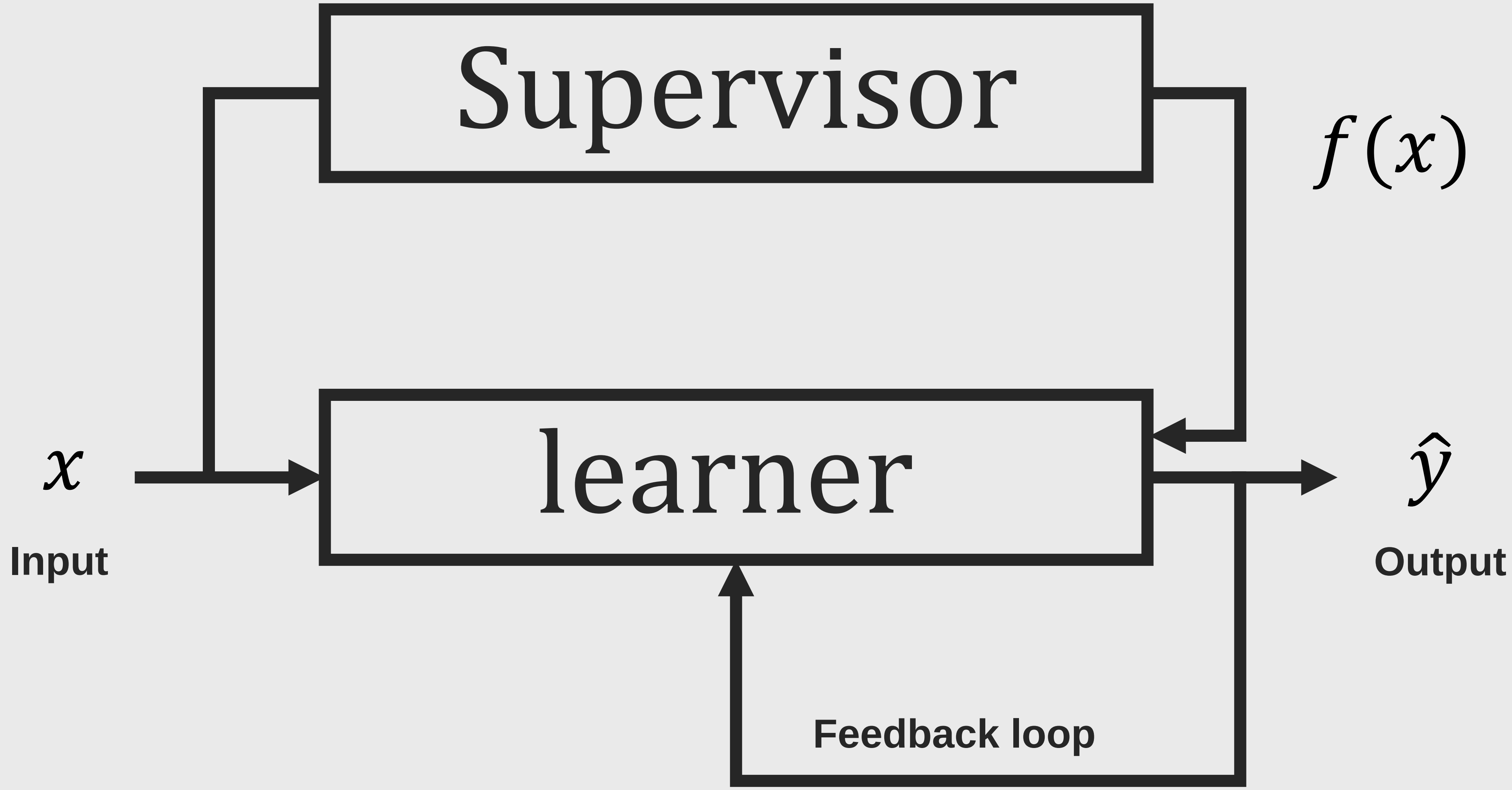
Source of example: http://mnemstudio.org/path-finding-q-learning-tutorial.htm

Reinforcement Leaning Example : Atari Game

https://www.youtube.com/watch?v=V1eYniJ0Rnk

Part 2

Supervised Learning


Supervisor
$f(x)$
x
Input
learner
$\hat{y}$
Output
Feedback loop

Regression and Classification

Class/Traget attribute

	#	Inputs Attributes (Independent)		Target/Class/Output Attributes (Dependent)
		A1	**A2**	**A3**
Records	**Ex. 0**	$A1_0$	$A2_0$	$A3_0$
	Ex. 1	$A1_1$	$A2_1$	$A3_1$
	Ex. 2	$A1_2$	$A2_2$	$A3_2$
	Ex. 3	$A1_3$	$A2_3$	$A3_3$
	Ex. 4	$A1_4$	$A2_4$	$A3_4$
	Ex. 5	$A1_5$	$A2_5$	$A3_5$
	Ex. 6	$A1_6$	$A2_6$	$A3_6$
	Ex. 7	$A1_7$	$A2_7$	$A3_7$
	Ex. 8	$A1_8$	$A2_8$	$A3_8$
	Ex. 9	$A1_9$	$A2_9$	$A3_9$

Target (Class) Attributes (A3)

Regression

Continuous **(Numerical)** labeled data

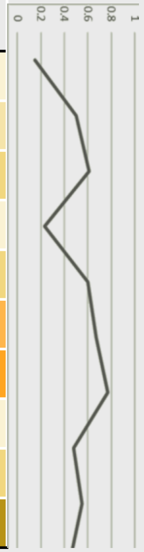
Classification

Discrete **(Categorical)** labeled data

Tasks: Regression and Classification

Continuous
labeled data

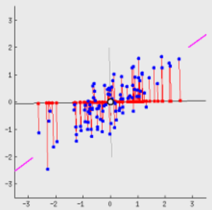
#	Inputs (X)		Target (Y)	
	Area (m^2)	Distance(mile)	Price (£Bn)	
Ex. 0	76.85	17.27	0.15	
Ex. 1	76.97	19.54	0.5	
Ex. 2	77.10	18.51	0.76	
Ex. 3	85.28	46.09	0.23	
Ex. 4	85.42	35.83	0.6	
Ex. 5	88.02	2.59	0.67	
Ex. 6	77.25	6.34	0.89	
Ex. 7	77.49	6.98	0.2	
Ex. 8	85.81	12.18	0.55	
Ex. 9	98.81	2.18	9.45	



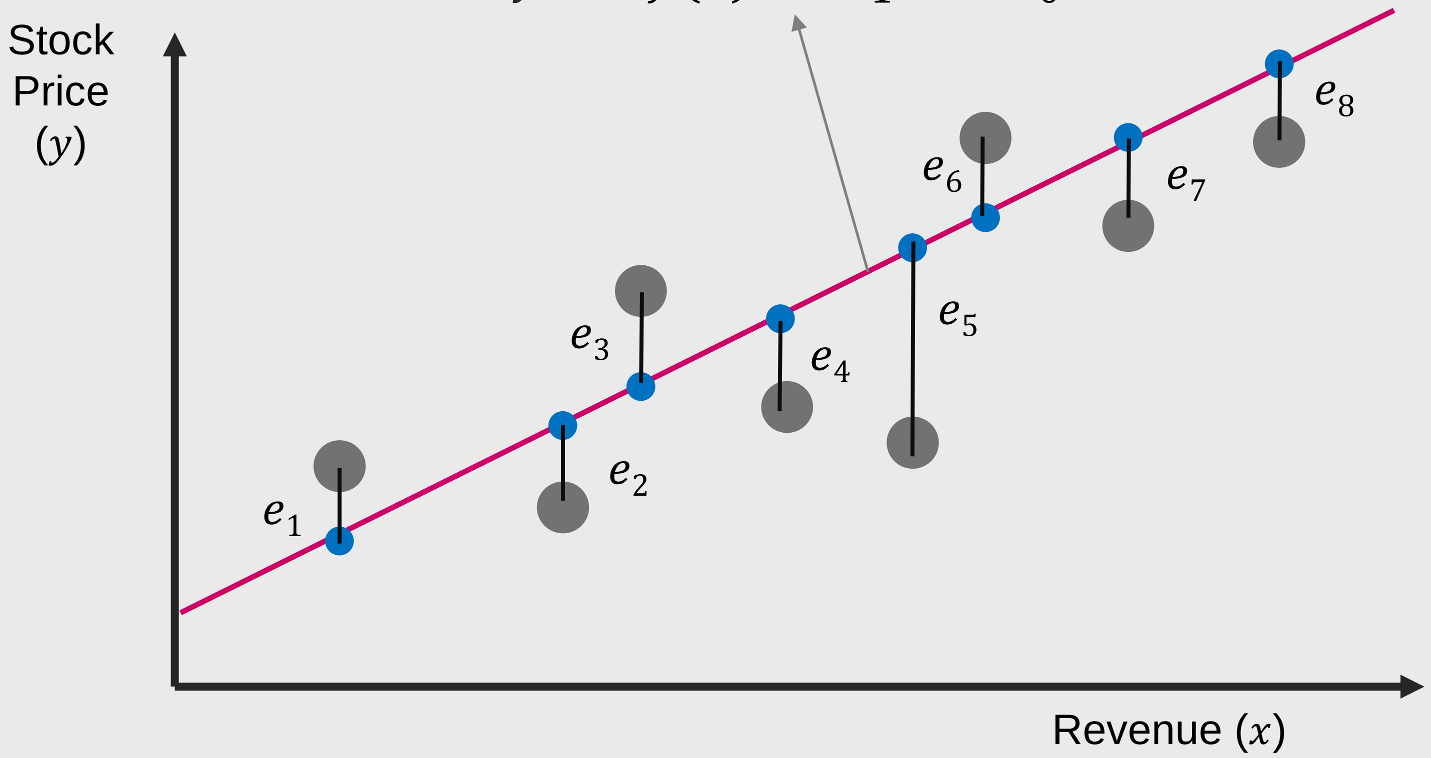
Discrete
labeled data

#	Inputs (X)		Class (Y)	
	Length (cm)	Weight (kg)	Sales	
Ex. 0	23.2	3.2	Good	
Ex. 1	70.9	19.5	Bad	
Ex. 2	60.5	18.51	Bad	
Ex. 3	24.5	4.6	Good	
Ex. 4	110.0	35.83	Bad	
Ex. 5	23.8	3.7	Good	
Ex. 6	25.8	4.5	Good	
Ex. 7	24.7	4.9	Good	
Ex. 8	85.8	25.6	Bad	
Ex. 9	78.8	20.33	Bad	

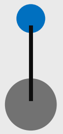
Regression: Linear function



$$\hat{y} = f(x) = w_1 x + w_0$$



- ✓ Best Fit
- ✓ Find the line (parameters of a line equation) that minimize the norm of the *y* errors
- ✓ (sum of the squares)

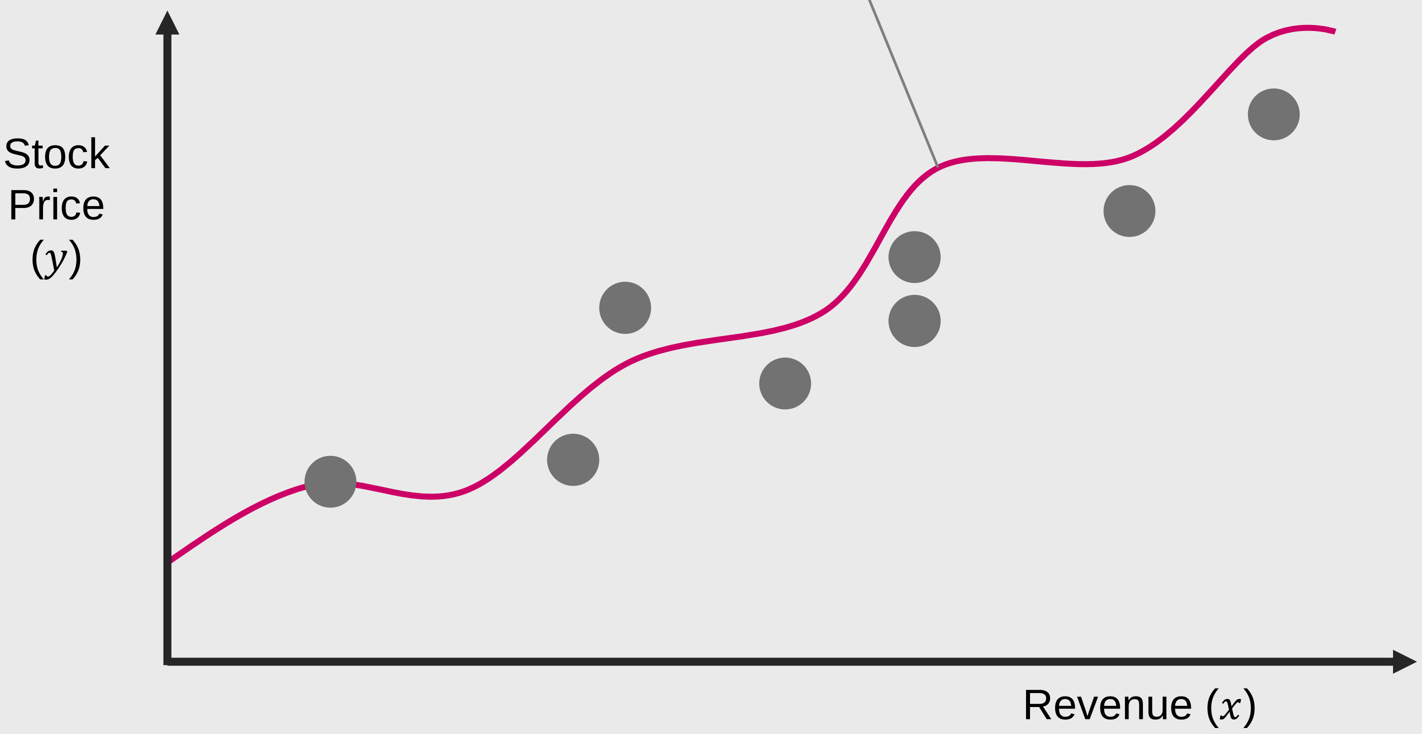

Error

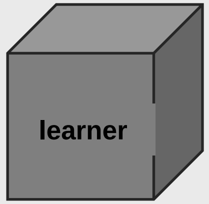
$$e_i = \hat{y}_i - y_i$$

$$\boldsymbol{e} = \sum_{i=1}^{8} (\hat{y}_i - y_i)^2$$

Regression: Non-Linear function

$$\hat{y} = f(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_m x^m$$





Find values of the weight: $w_1, w_2, \dots w_m$

Loss function: Mean Squared Error, $\boldsymbol{E}$

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

$\hat{y}_i$ - predicted output
y_i - target output
n - number of examples in training/test set

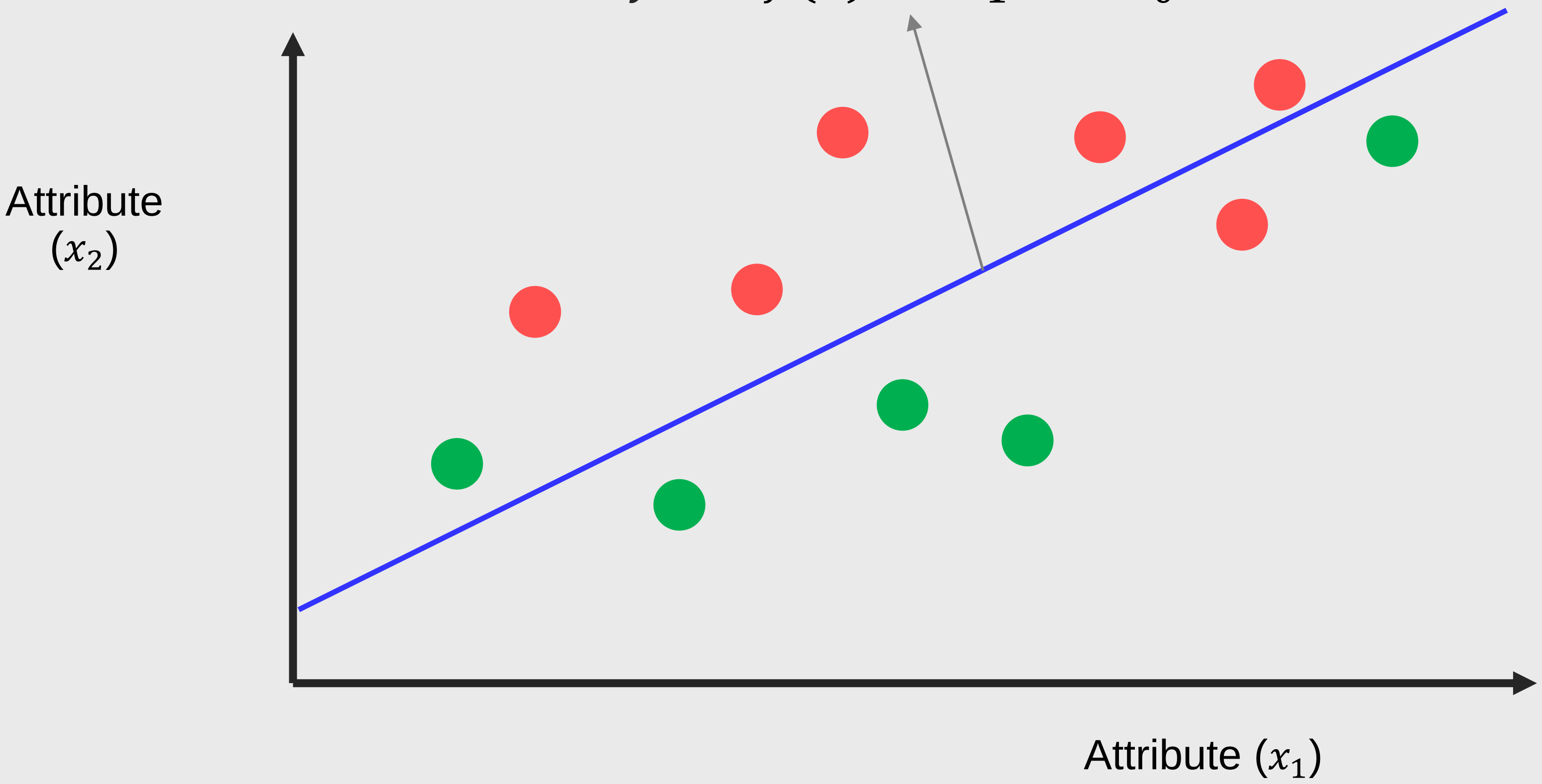
Loss function: Mean Absolute Error, $\boldsymbol{E}$

$$\boldsymbol{E} = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

$\hat{y}_i$ - predicted output
y_i - target output
n - number of examples in training/test set

Classification: Linear function

$$\hat{y} = f(x) = w_1 x + w_0$$

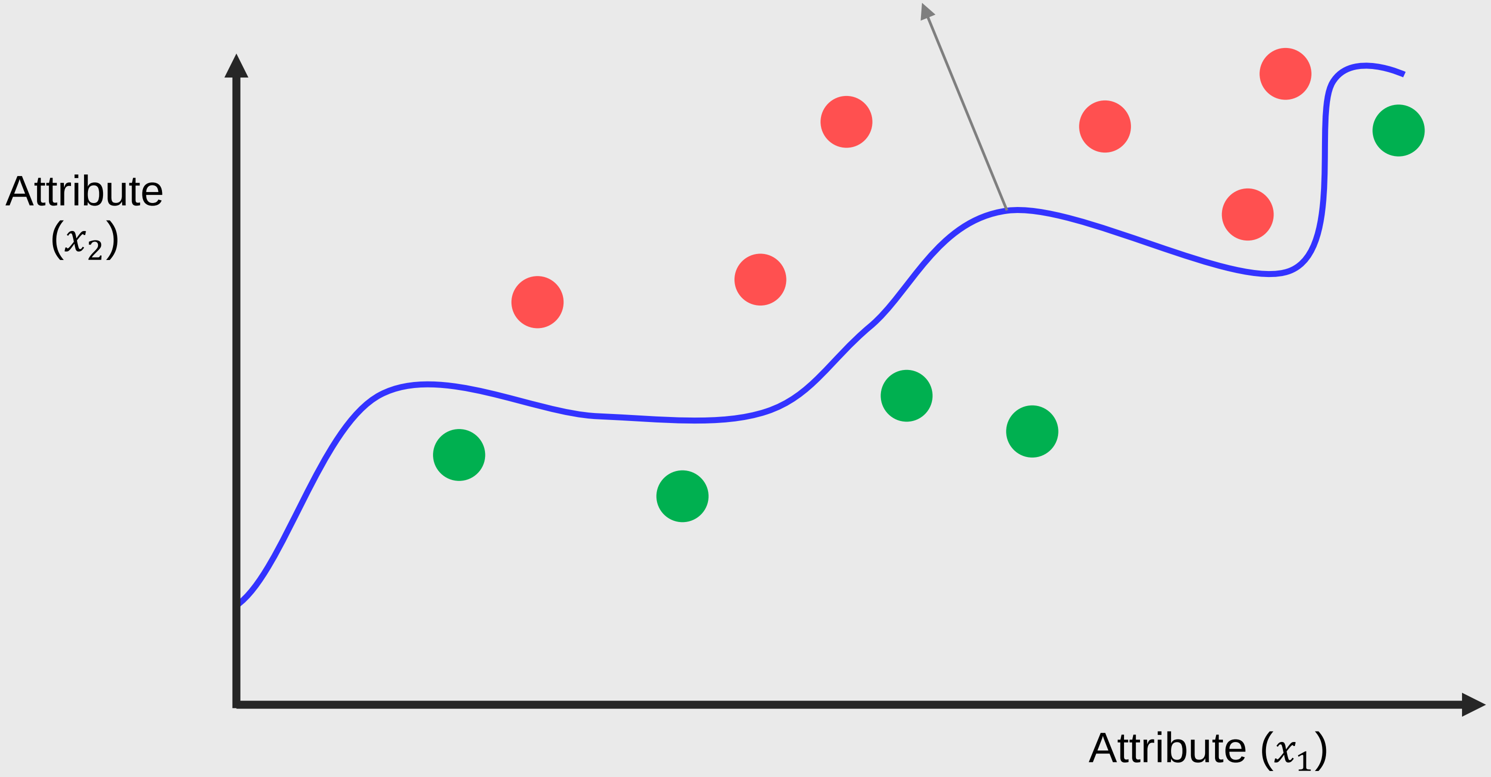


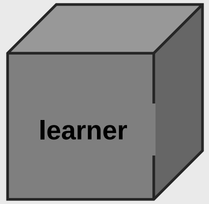
- ✓ Best Fit
- ✓ Find the line (parameters of a line equation) that minimize the error (misclassification) rate

$$\mathbf{e} = \frac{1}{n}\sum_{i=1}^{n} \hat{y}_i \neq y_i$$

Classification: Non-Linear function

$$\hat{y} = f(x) = w_0 + w_1 x + w_2 x^2 + \cdots + w_m x^m$$





Find values of the weight: $w_1, w_2, \ldots w_m$

Loss function: Misclassification rate, $\boldsymbol{E}$

$$\boldsymbol{E} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i \neq y_i)$$

$\hat{y}_i$ - predicted output
y_i - target output
n - number of examples in training/test set

Loss function: Log loss

This part will be zero if $y_i = 0$

This part will be zero if $y_i = 1$

$$-\mathbf{log}\,\mathbf{P}(y_i \mid \hat{y}_i) = -\left(\boxed{(\boldsymbol{y_i})\mathbf{log}(\hat{y}_i)} + \boxed{(\mathbf{1} - \boldsymbol{y_i})\mathbf{log}(\mathbf{1} - \hat{y}_i)} \right)$$

$\hat{y}_i$ - predicted output
y_i - target output
n - number of examples in training/test set

Loss function: Log loss, $\boldsymbol{E}$

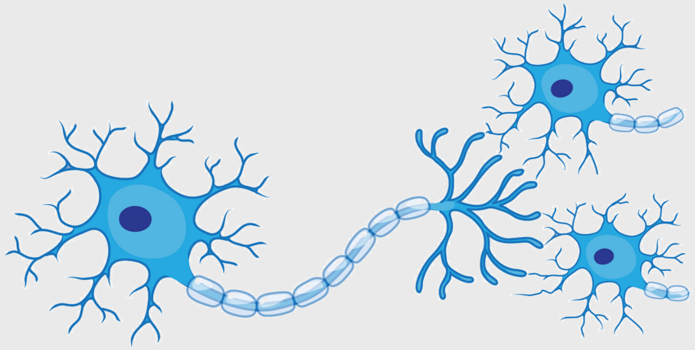
$$E = -\frac{1}{n}\sum_{i=1}^{n} \mathbf{log}\,\mathbf{P}(y_i \mid \widehat{y_i})$$

$\widehat{y_i}$ - predicted output
y_i - target output
n - number of examples in training/test set

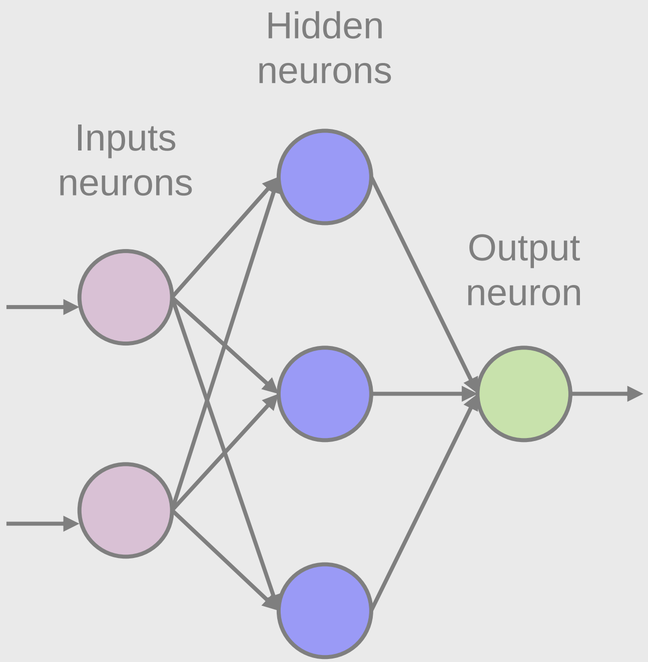
Part 3

Neural Networks

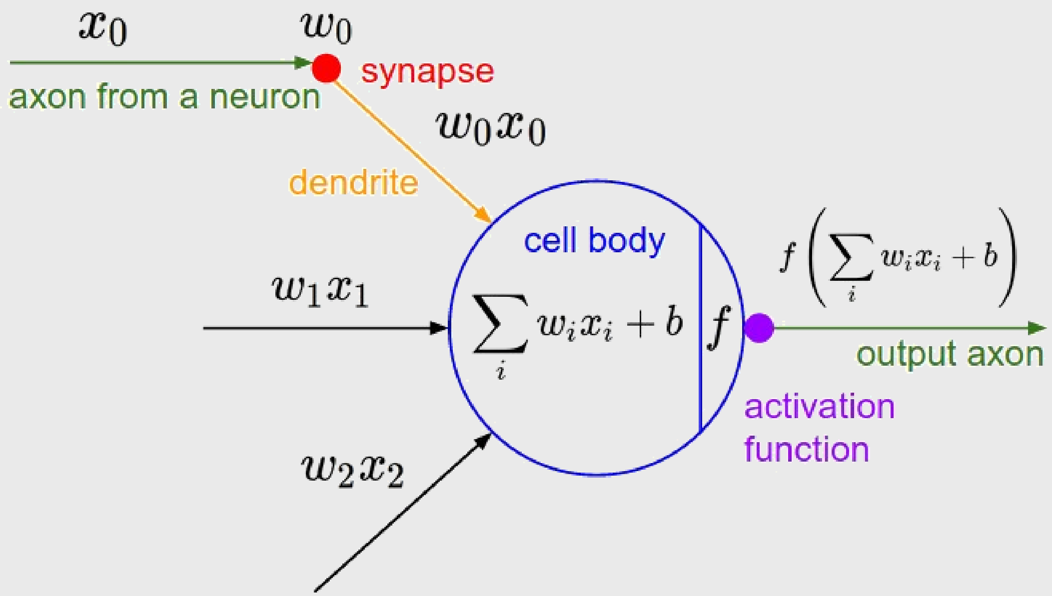
Learning Systems: Neural Networks



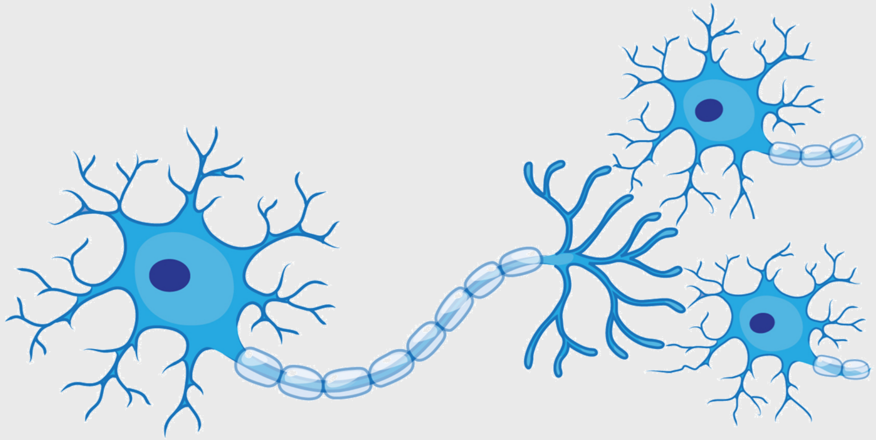
Biological networks of
neurons in human brains

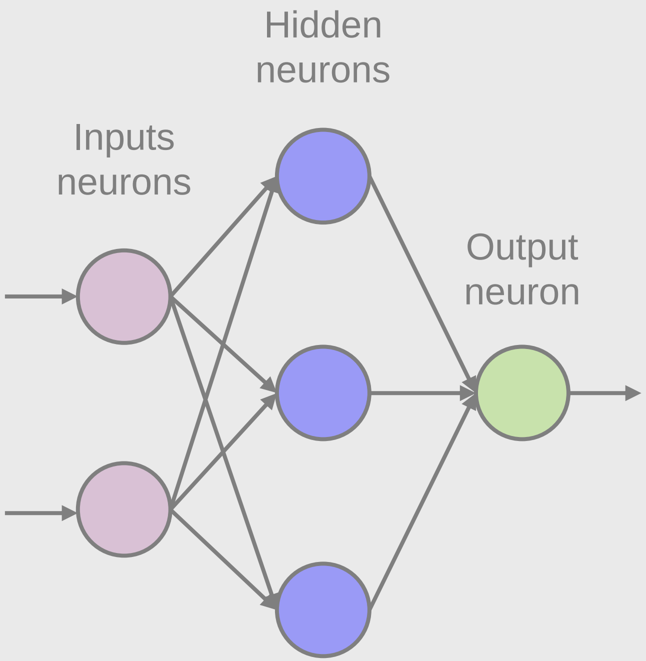


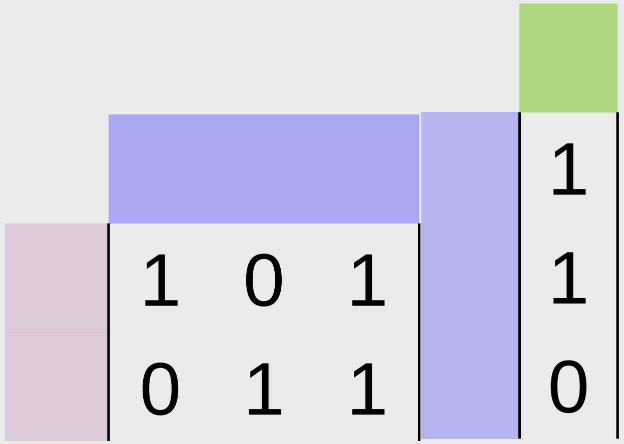
AI representation
of biological neural networks



Learning Systems: Neural Networks







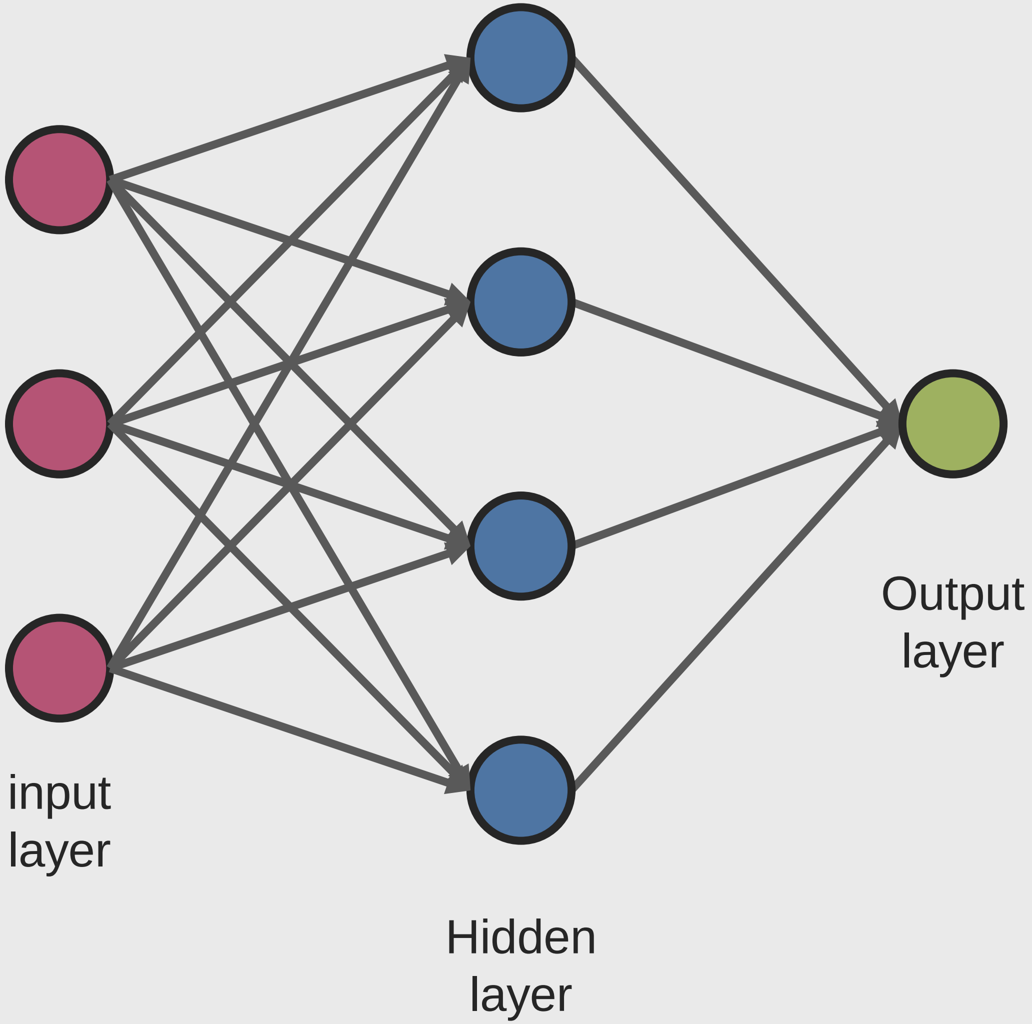
1 Biological networks of neurons in human brains

2 AI representation of biological neural networks

3 Mathematical representation of the neural networks

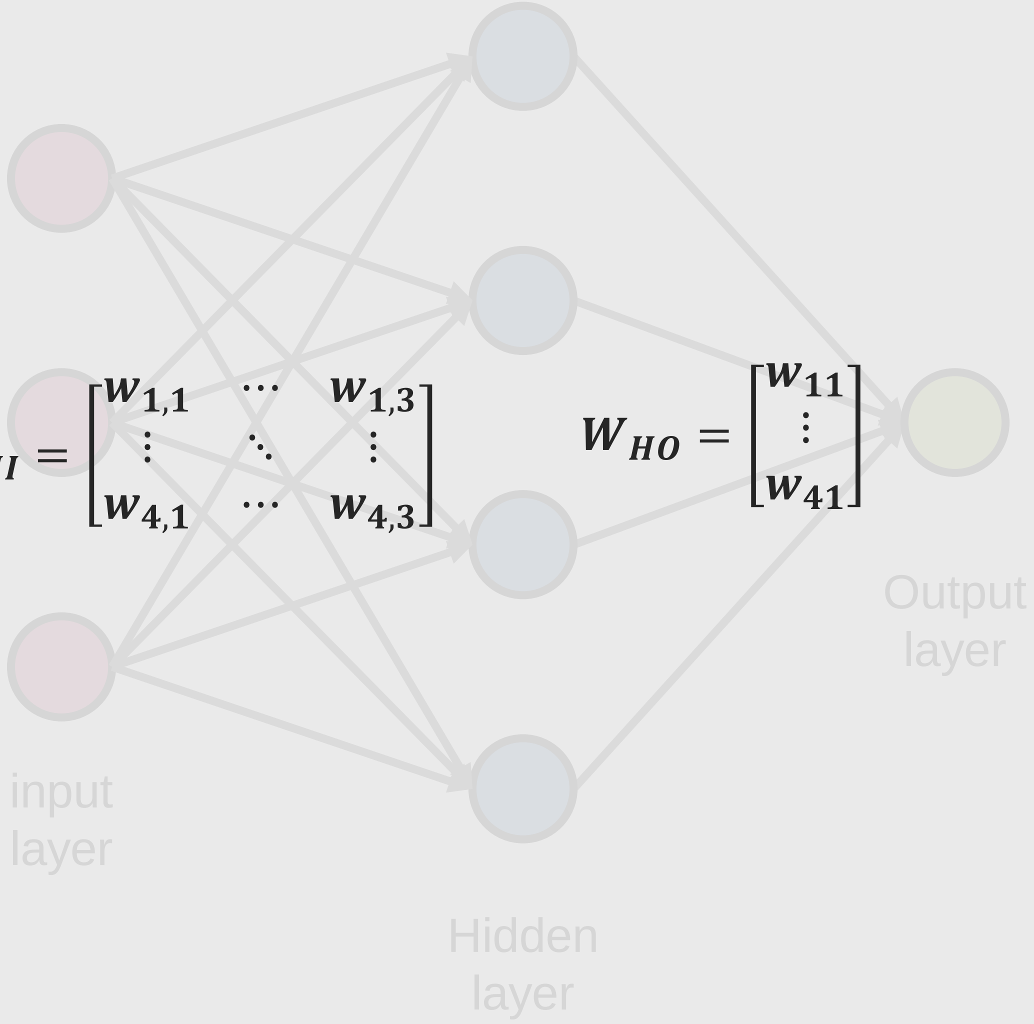
NEURAL NETWORK

Architecture



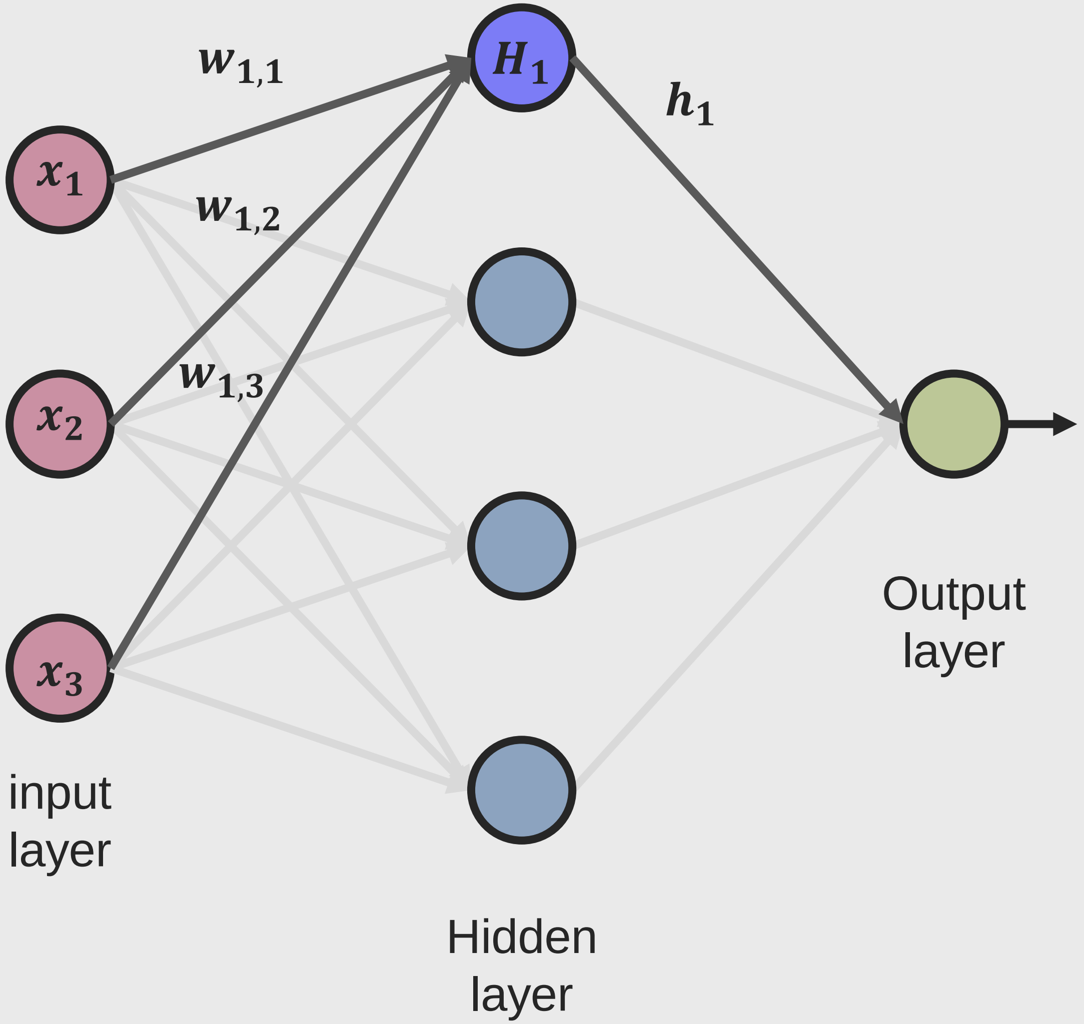
NEURAL NETWORK

Weights (parameters)



$$\boldsymbol{W_{HI}} = \begin{bmatrix} \boldsymbol{w_{1,1}} & \cdots & \boldsymbol{w_{1,3}} \\ \vdots & \ddots & \vdots \\ \boldsymbol{w_{4,1}} & \cdots & \boldsymbol{w_{4,3}} \end{bmatrix}$$

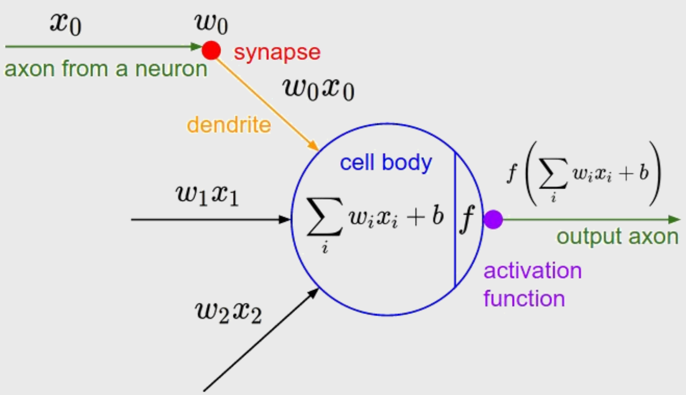
$$\boldsymbol{W_{HO}} = \begin{bmatrix} \boldsymbol{w_{11}} \\ \vdots \\ \boldsymbol{w_{41}} \end{bmatrix}$$



For n inputs, a hidden layer node's h_j output is expressed as:

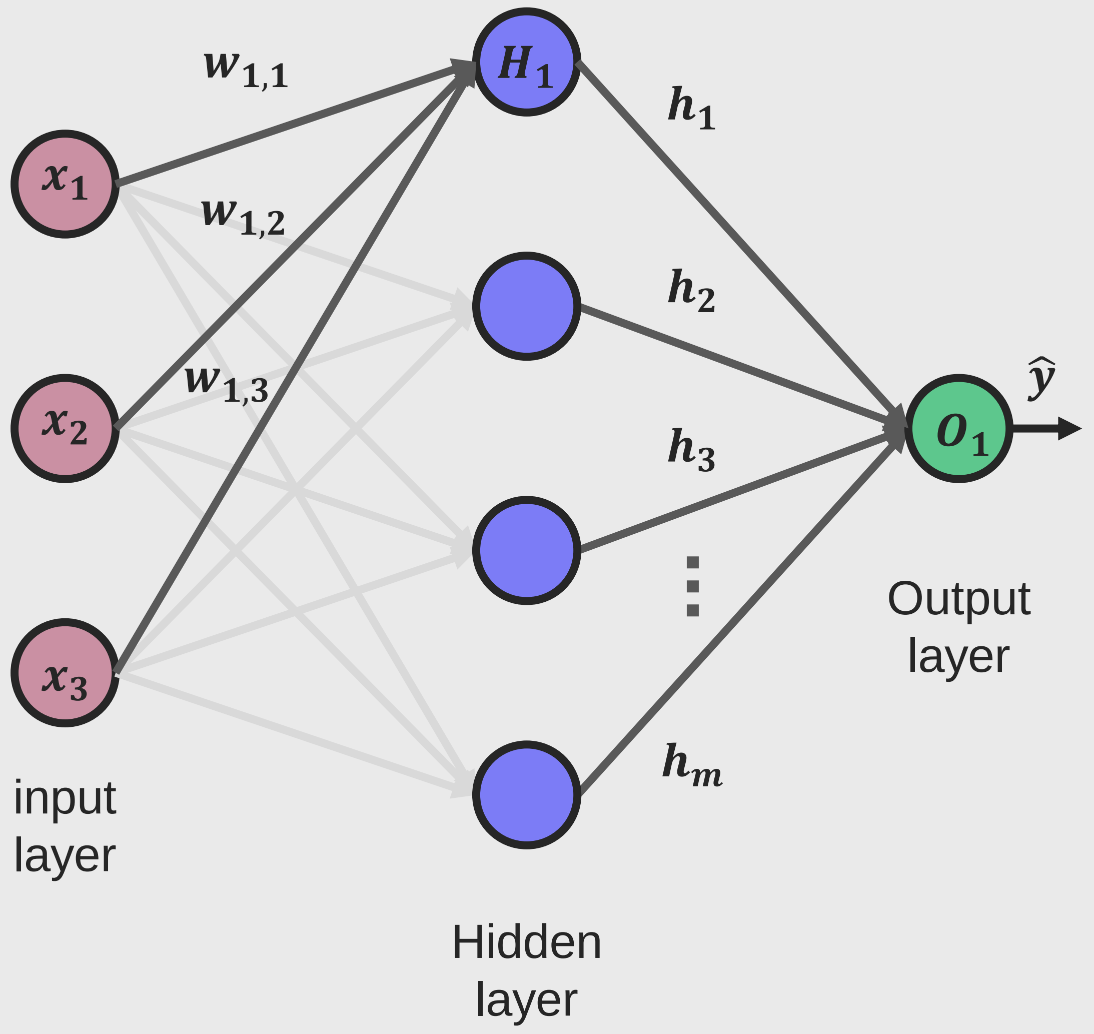
$$h_j = \varphi_h\left(\sum_{i=1}^{n} w_{ji}.\, x_i\right)$$

Where φ_h is an activation function:



NEURAL NETWORK

Computation: Hidden layer



For $\boldsymbol{n}$ inputs, a hidden layer node's $\boldsymbol{h_j}$ output is expressed as:

$$h_j = \varphi_h\left(\sum_{i=1}^{n} w_{ji} . x_i\right)$$

Where $\boldsymbol{\varphi_h}$ is an activation function:

For m hidden nodes and a output node, the output nodes output is expressed as:

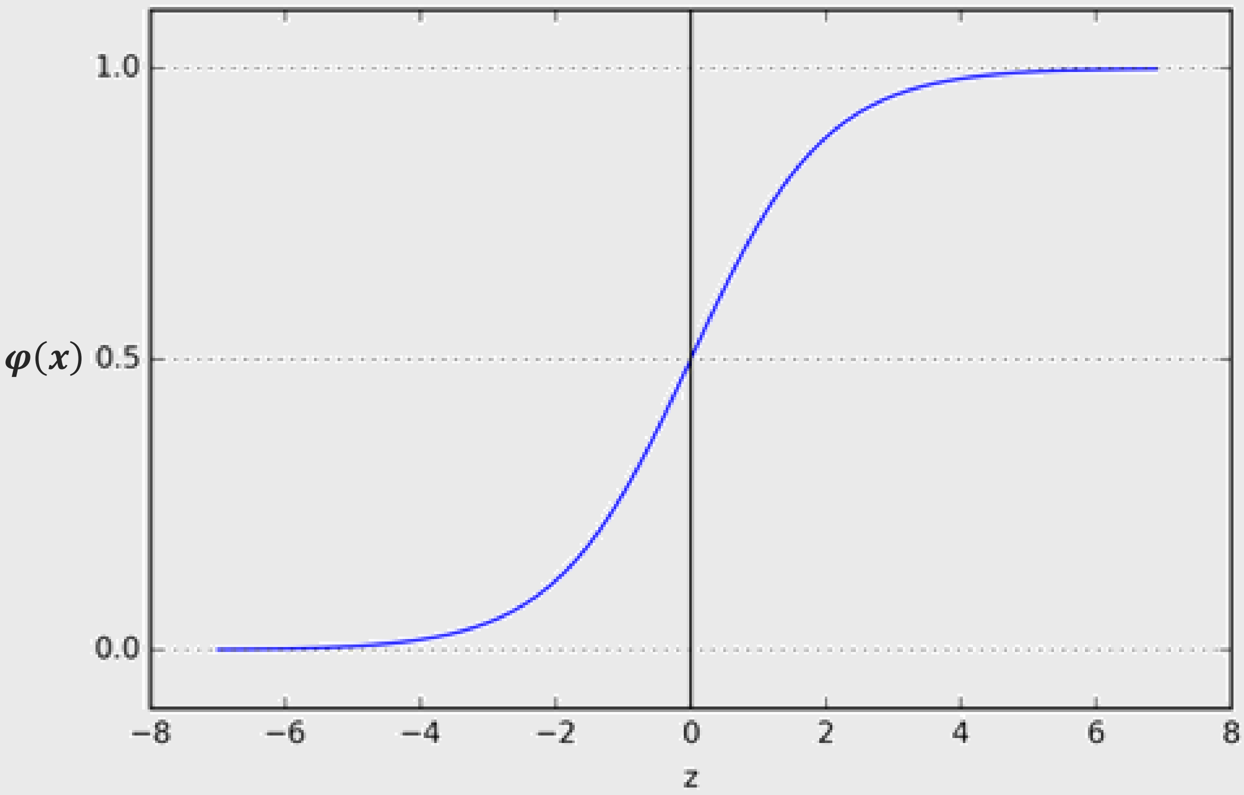
$$\hat{y} = \varphi_O\left(\sum_{j=1}^{m} w_{jk} . h_j\right)$$

NEURAL NETWORK

Computation: Output layer

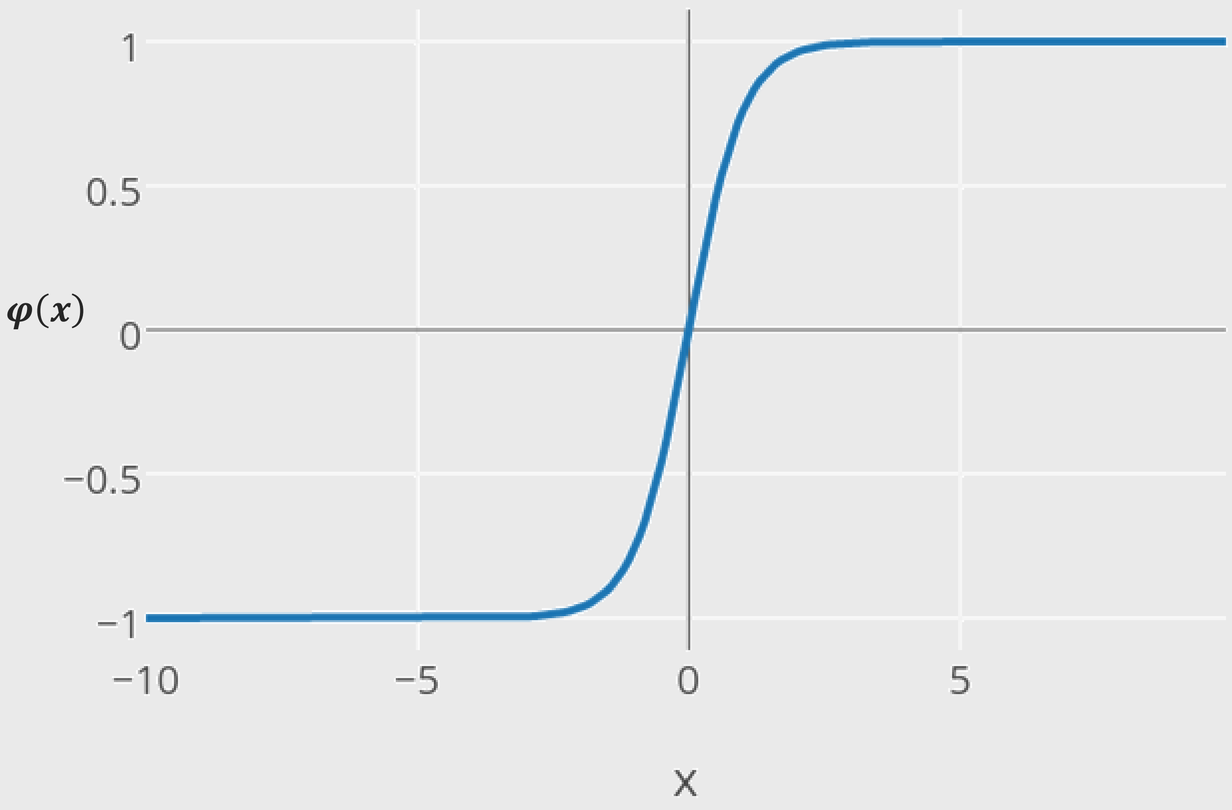
Sigmoid activation

$$\varphi(x) = \frac{1}{1 + e^{-x}}$$



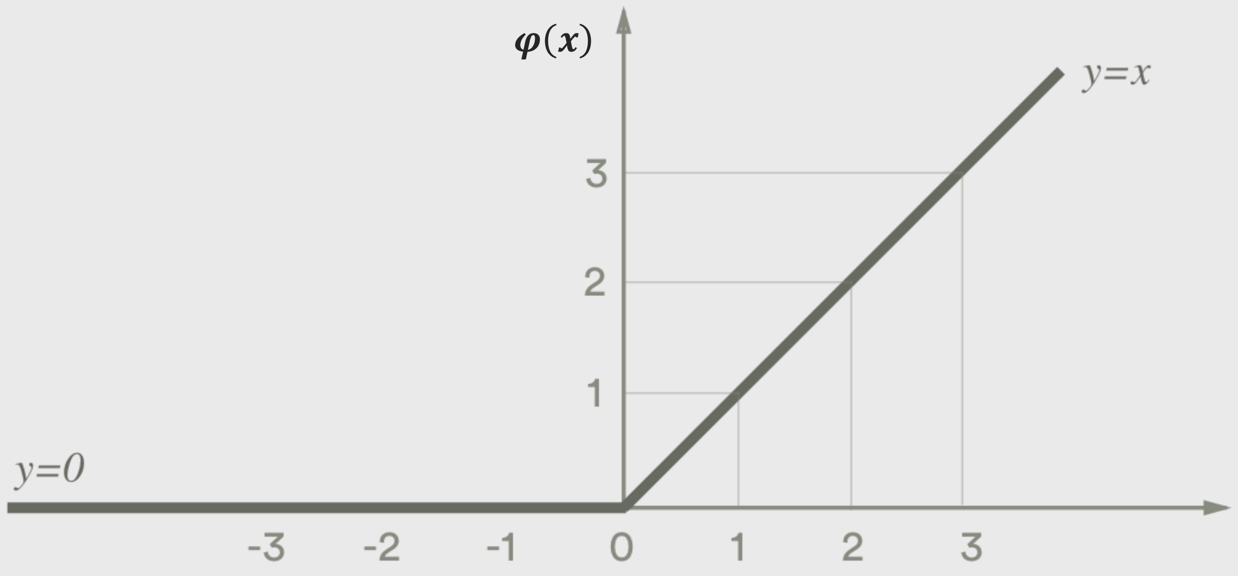
Tangent hyperbolic activation

$$\varphi(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$



Rectified Linear Unit (ReLU)

$$\varphi(x) = max\ (0, x)$$

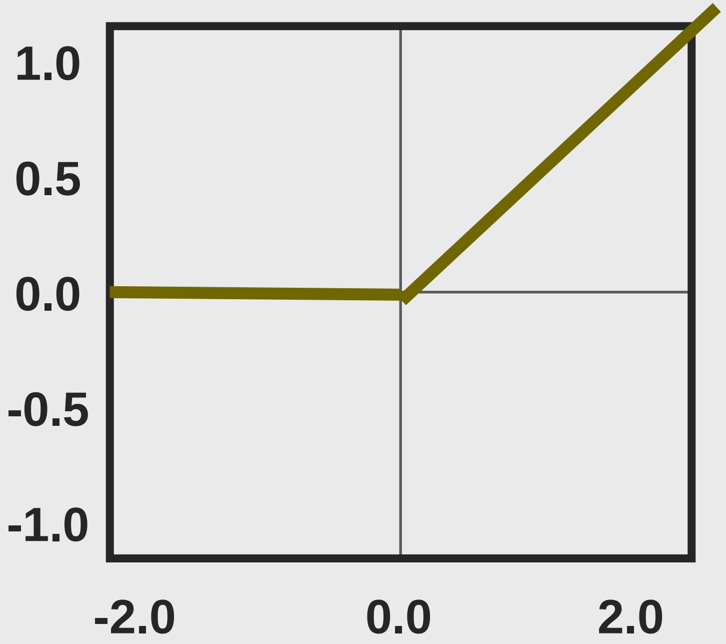


Source: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

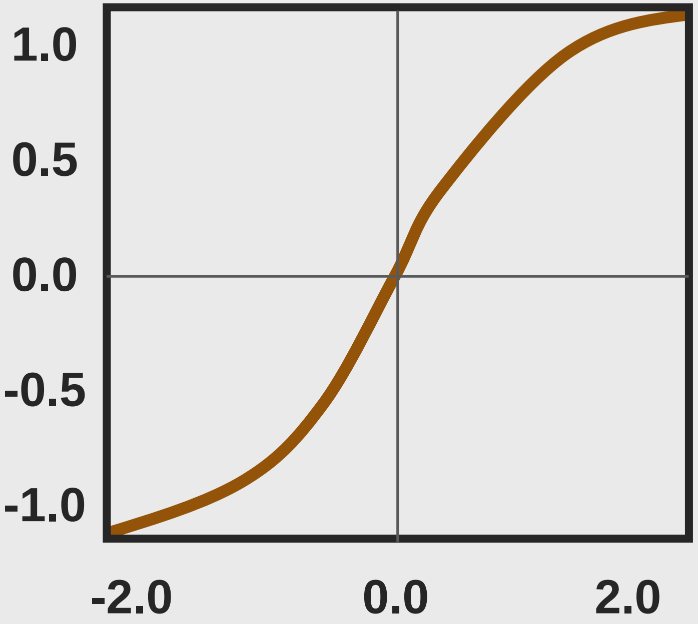
NEURAL NETWORK

Activation functions

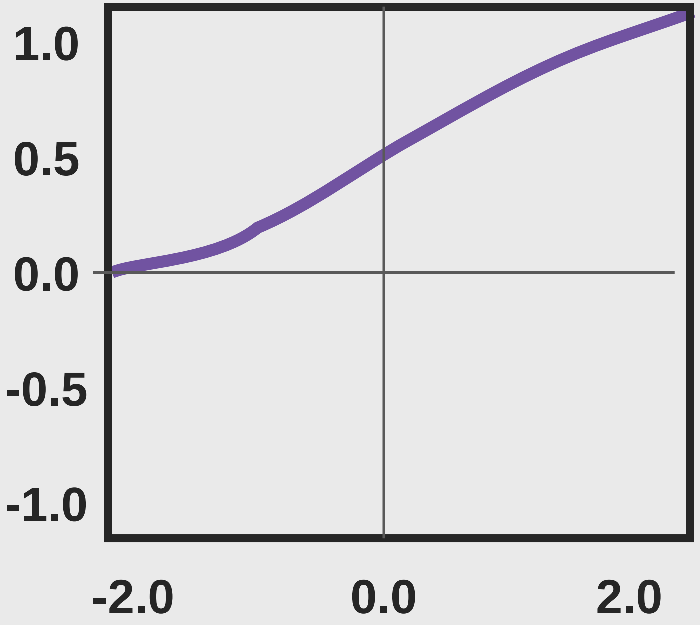
ReLU



Tanh

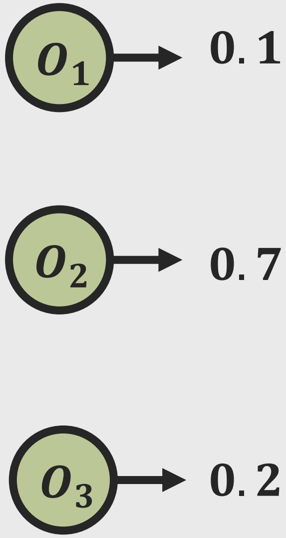


Sigmoid



SoftMax Activation

$$\boldsymbol{\varphi}(\boldsymbol{x_i}) = \frac{\boldsymbol{e^{x_i}}}{\sum_{\boldsymbol{i}}^{\boldsymbol{k}} \boldsymbol{e^{x_j}}} \text{ for } \mathrm{k} \text{ units}$$



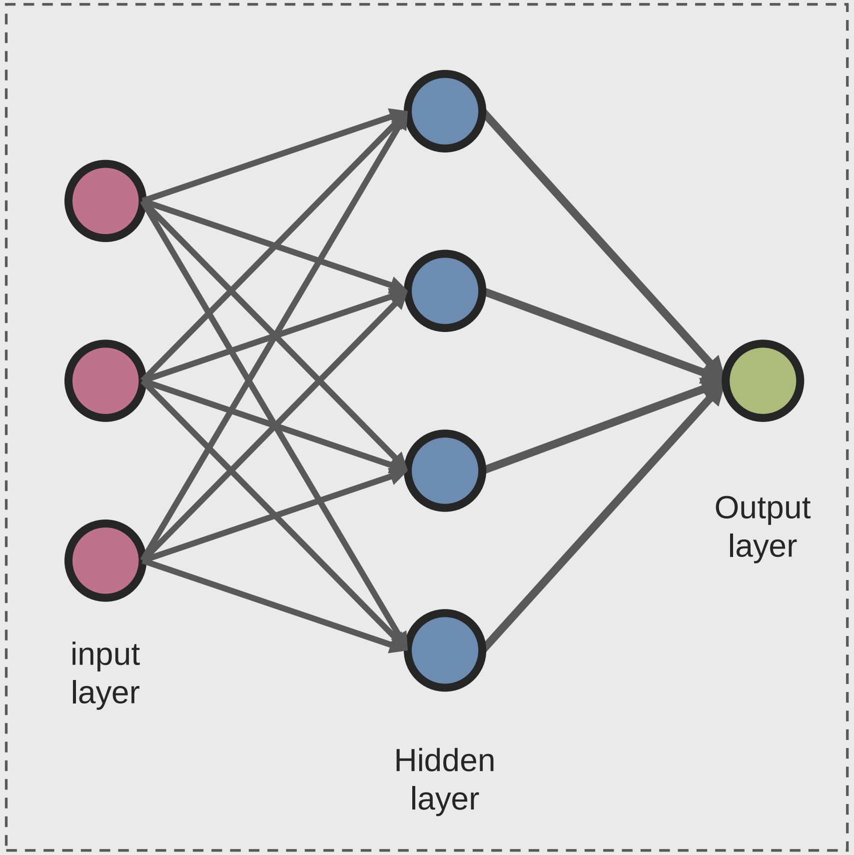
PROBABILITIES DISTRIBUTION OF ALL LABELS

NEURAL NETWORK

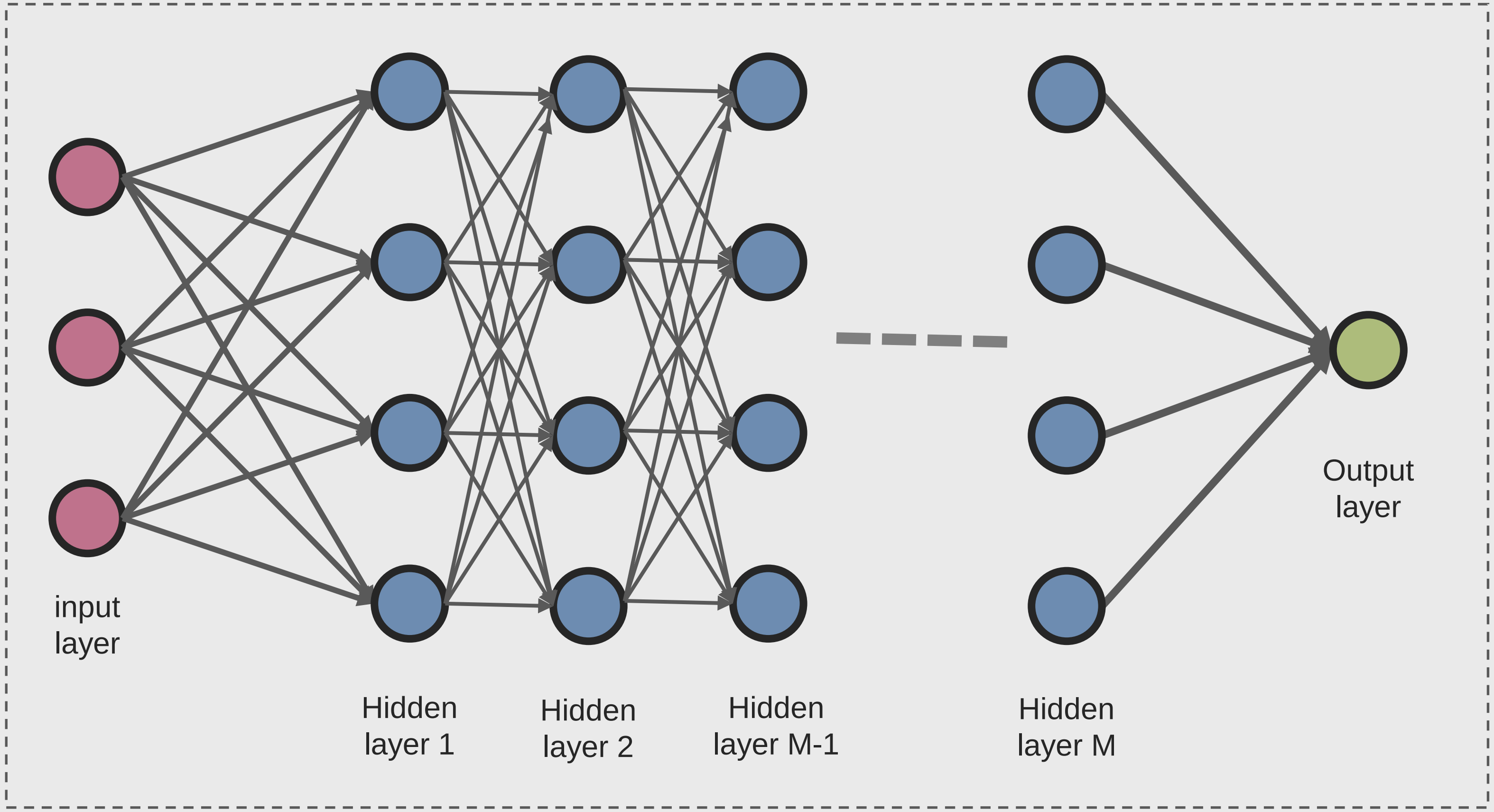
Activation function

NEURAL NETWORK: Architecture

A regular neural network architecture



A deep neural network architecture



SHALLOW LEARNING

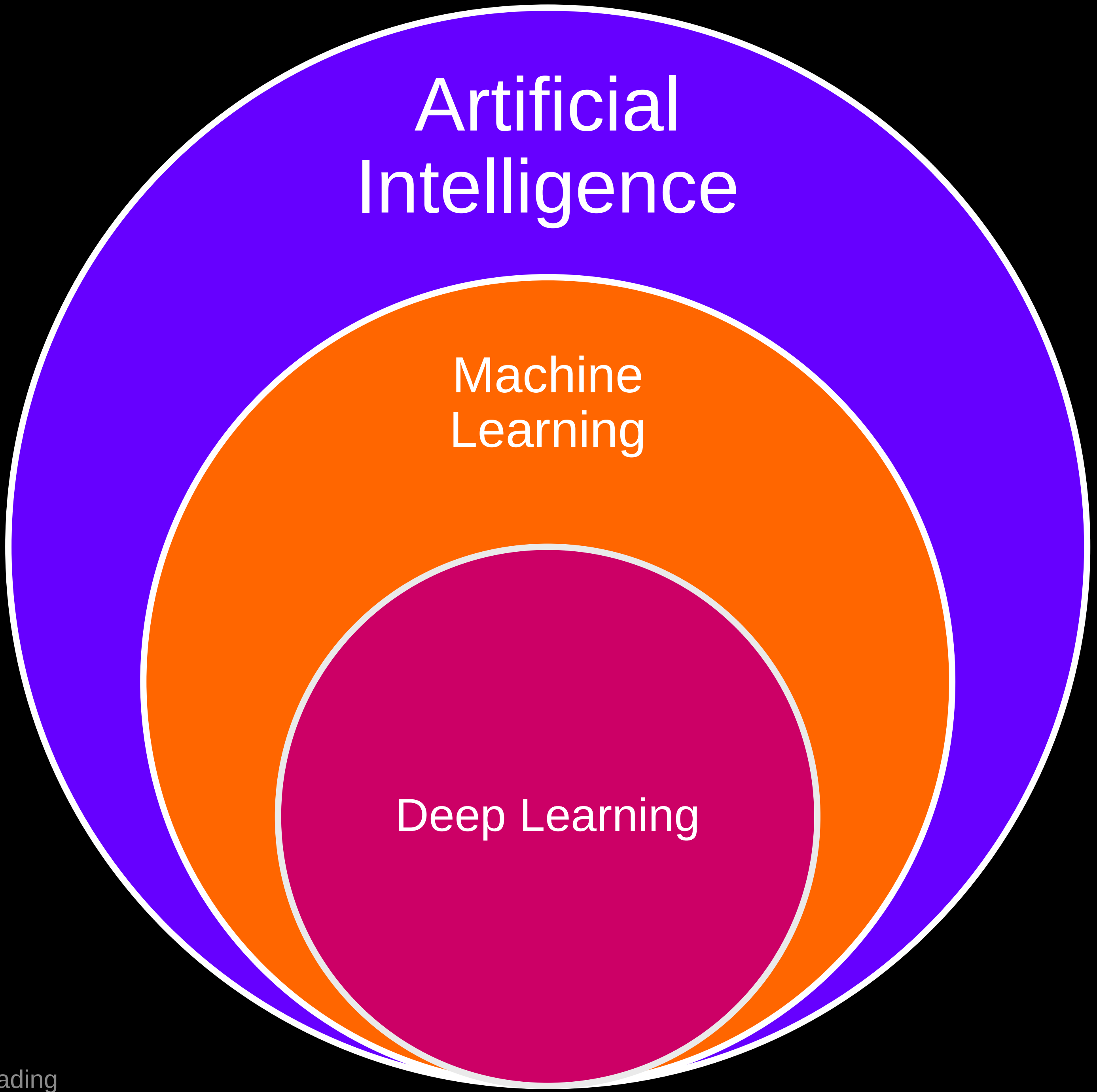
DEEP LEARNING

Deep Learning

Deep learning is an artificial intelligence function that imitates the workings of the human brain in processing data and creating patterns for use in decision making.

Deep learning is a **subset of machine learning in artificial intelligence** that has networks capable of learning supervised/unsupervised from data that is structured/unstructured or labelled/unlabelled.

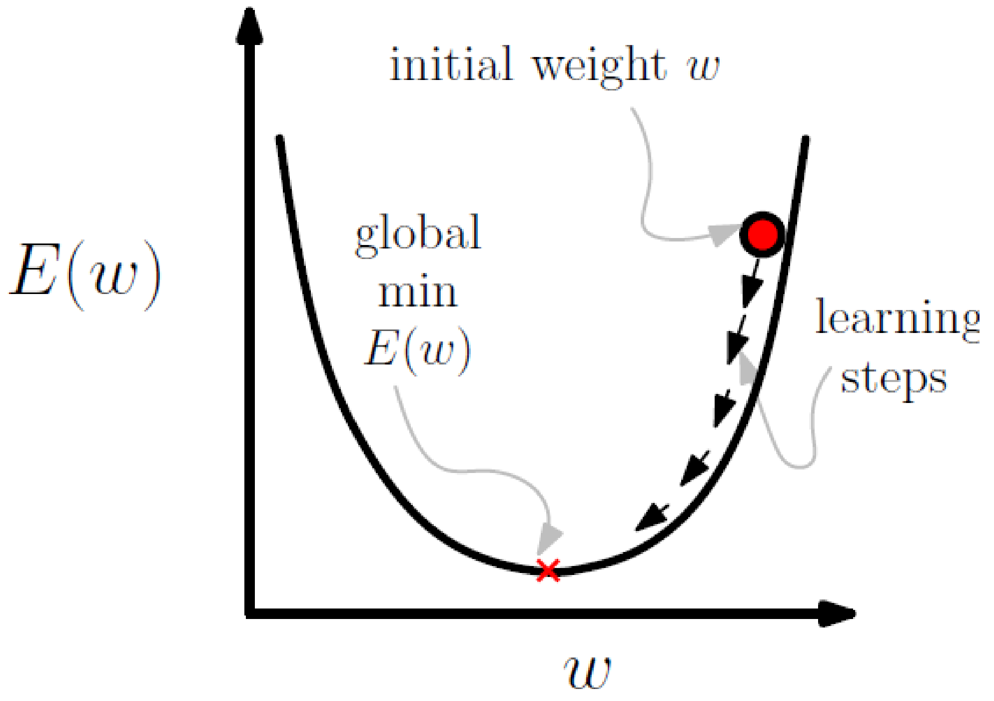
Source: https://www.investopedia.com/terms/d/deep-learning.asp


Artificial
Intelligence
Machine
Learning
Deep Learning

(DEEP) NEURAL NETWORK

Optimisation

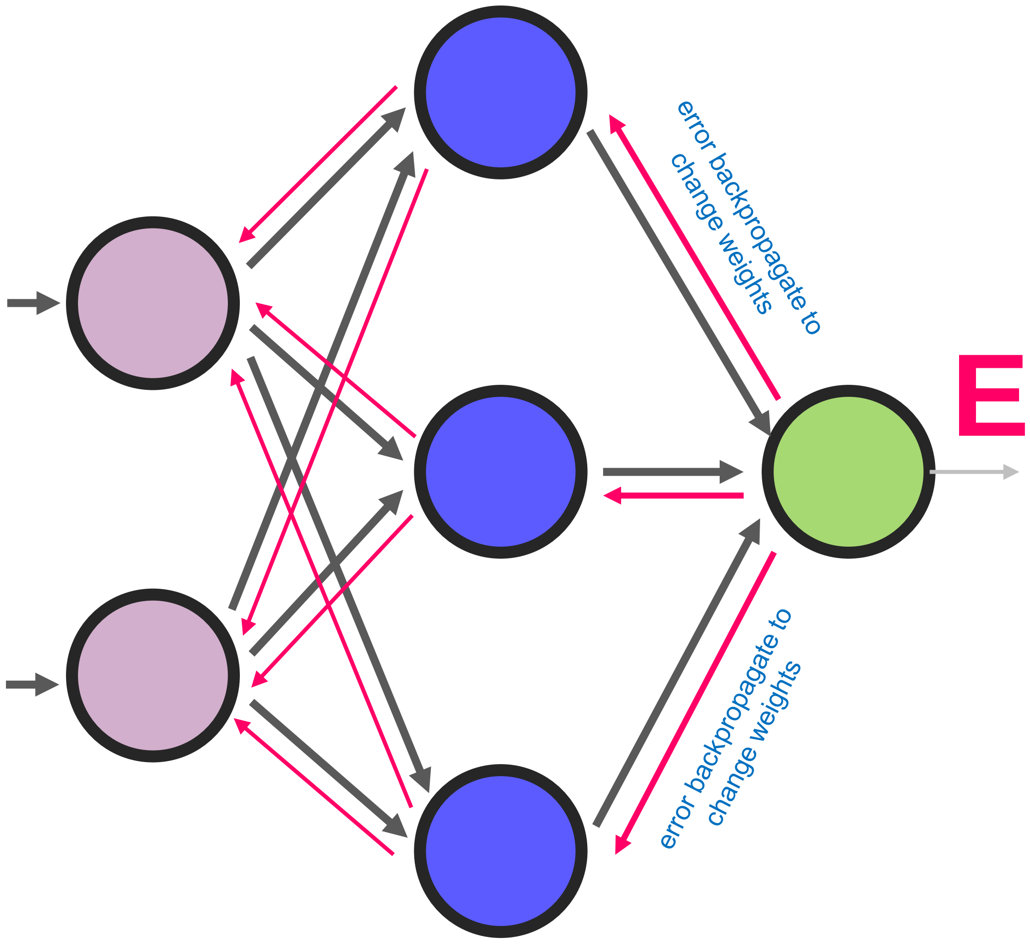
- Stochastic gradient descent
- Mini-batch gradient descent
- Batch gradient descent
- Backpropagation



Details are in Lecture 5 Slides

BACKPROPAGATION

Algorithm for Updating Learning Systems



E = Output - Target

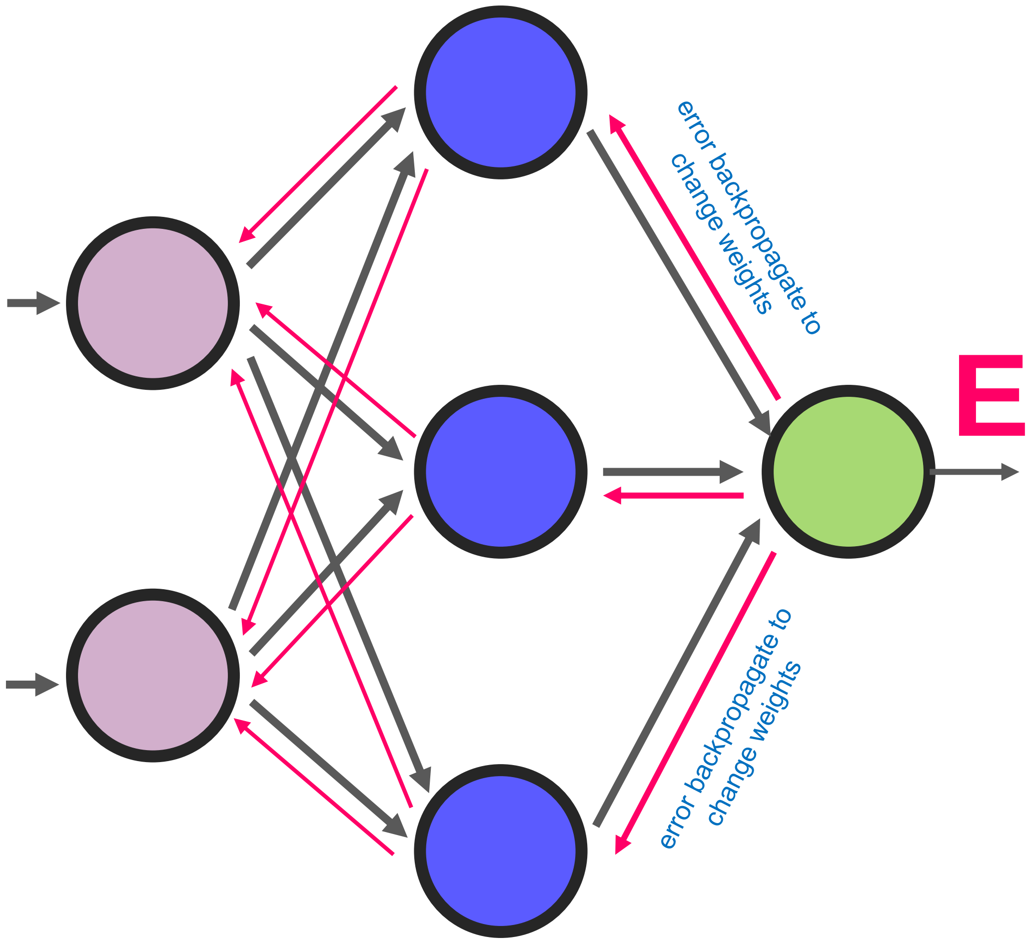
Learning Systems Update

$$W_{NEW} \leftarrow W_{OLD} + W_{CHANGE}$$

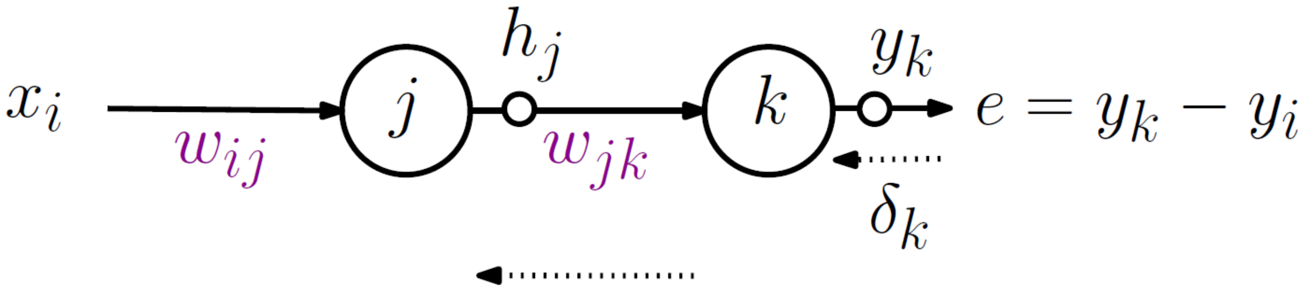
BACKPROPAGATION

Algorithm for Updating Learning Systems

Details are in Lecture 5 Slides



E = Output - Target

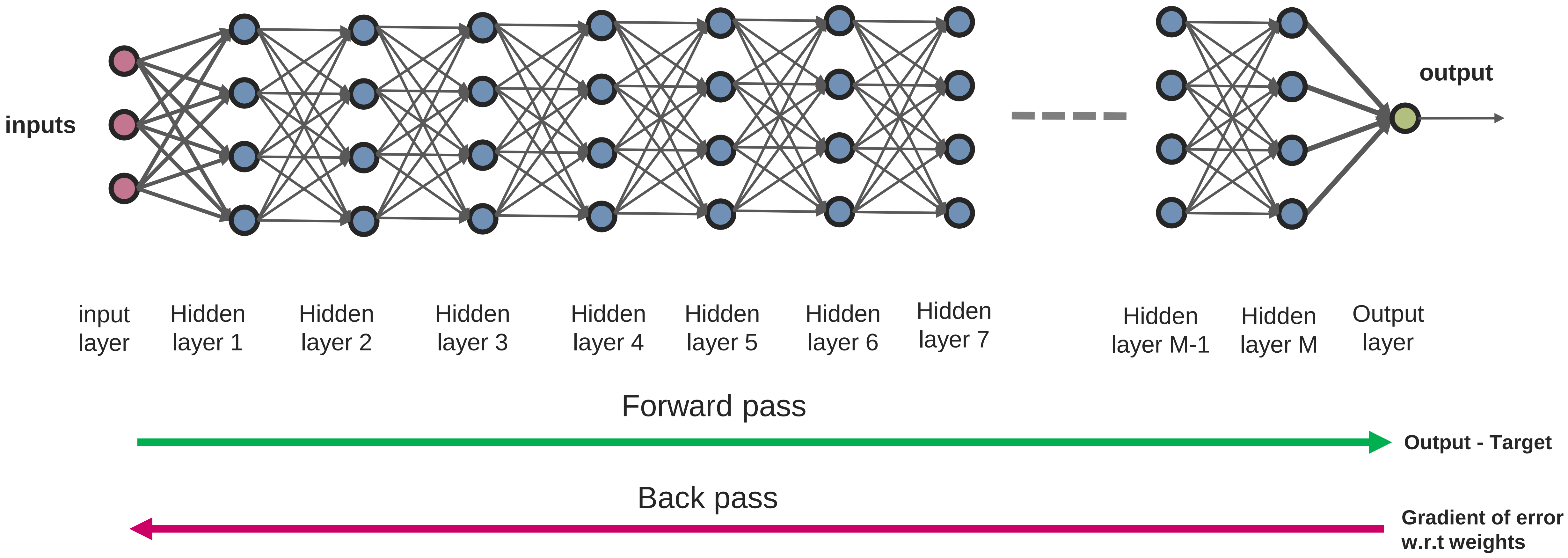


$$\Delta w_{jk} = \delta_j h_j$$

$$\Delta w_{ij} = \delta_i x_i$$

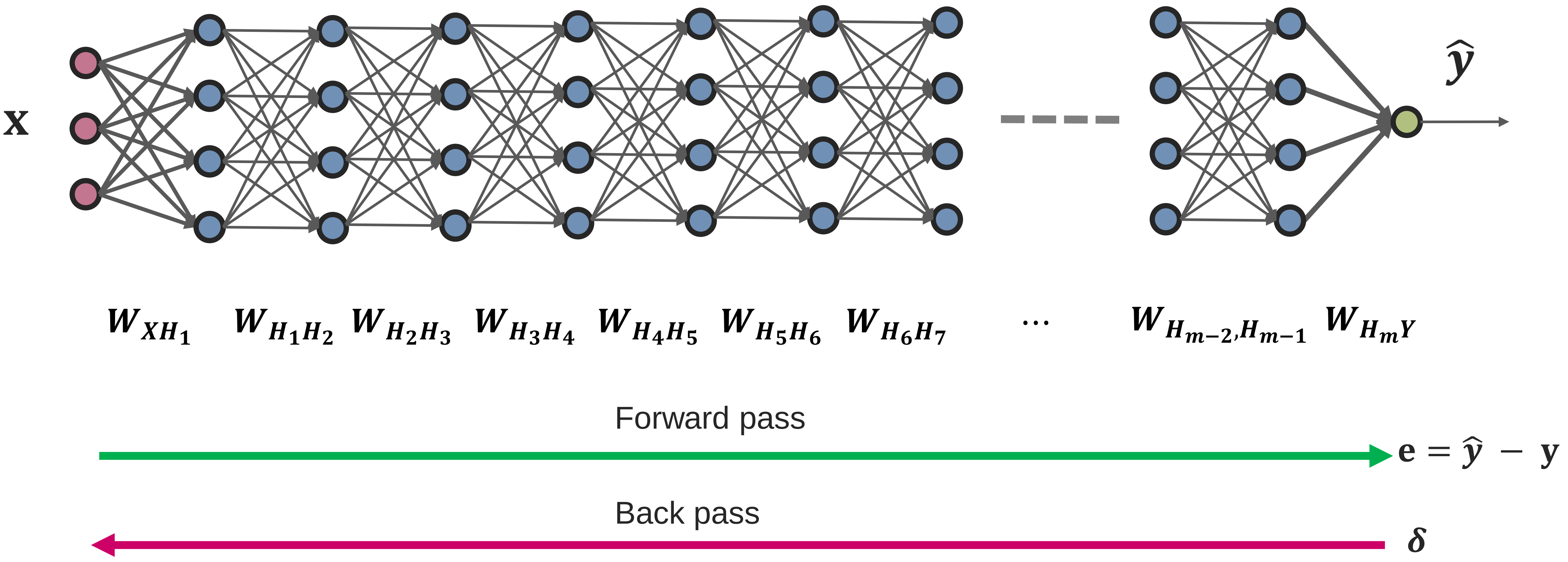
BACKPROPAGATION

Deep Learning

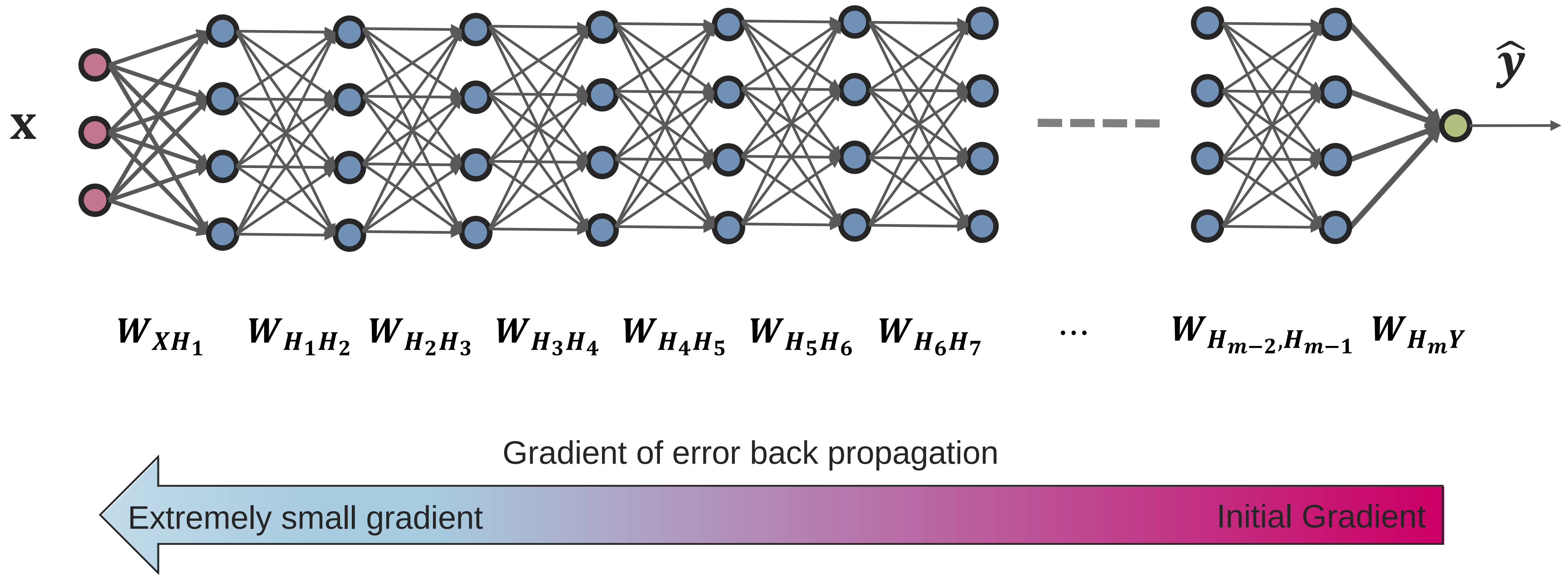


BACKPROPAGATION

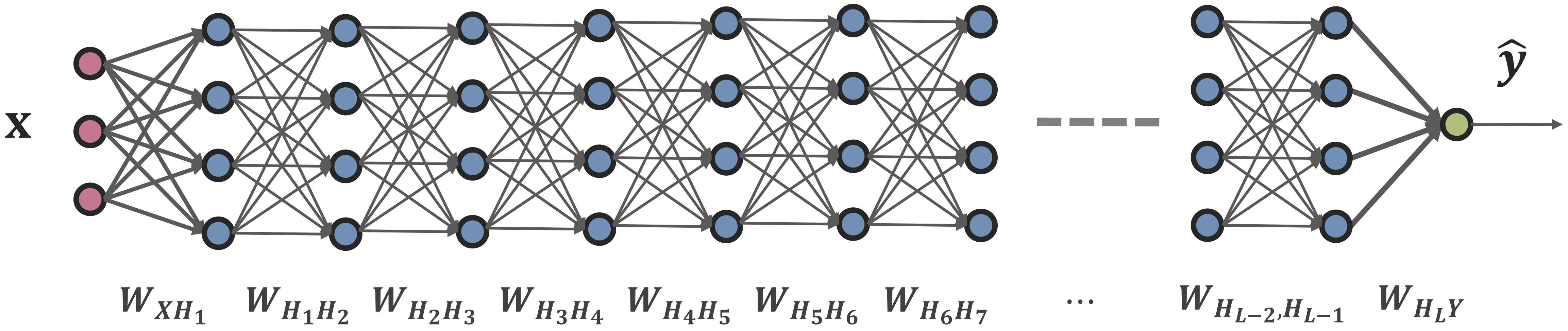
Deep Learning



Vanishing Gradient



Vanishing Gradient



Forward pass: $\hat{y} = \varphi_L(W_L\varphi_{L-1}(W_{L-1}\cdots\varphi_3(W_3\varphi_2(W_2\varphi_1(W_1\mathbf{x})))\cdots))$

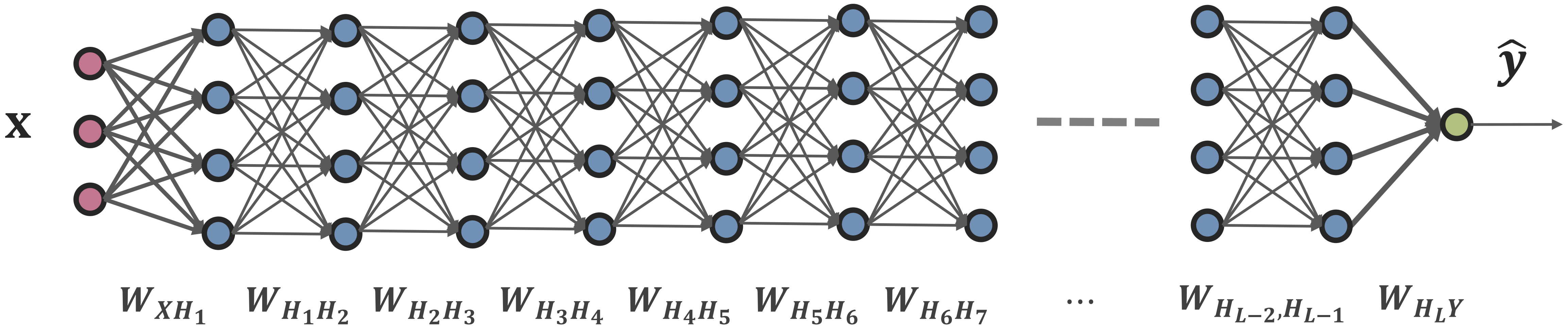
$\mathbf{e} = \hat{y} - \mathbf{y}$

$$\hat{y} = W_L \begin{bmatrix} 0.5 & \cdots & 0.0 \\ \vdots & \ddots & \vdots \\ 0.0 & \cdots & 0.5 \end{bmatrix}^{L-1}$$

$w = 0.5^{L-1}$ for a large L this will be **extremely small**. That is, weight w is a an exponentially **decreasing** function of L

This is caused by **sigmoid function** because its derivative lies between 0.0 and 0.25

Vanishing Gradient



Forward pass: $\hat{y} = \varphi_L(W_L\varphi_{L-1}(W_{L-1}\cdots\varphi_3(W_3\varphi_2(W_2\varphi_1(W_1\mathbf{x})))\cdots))$

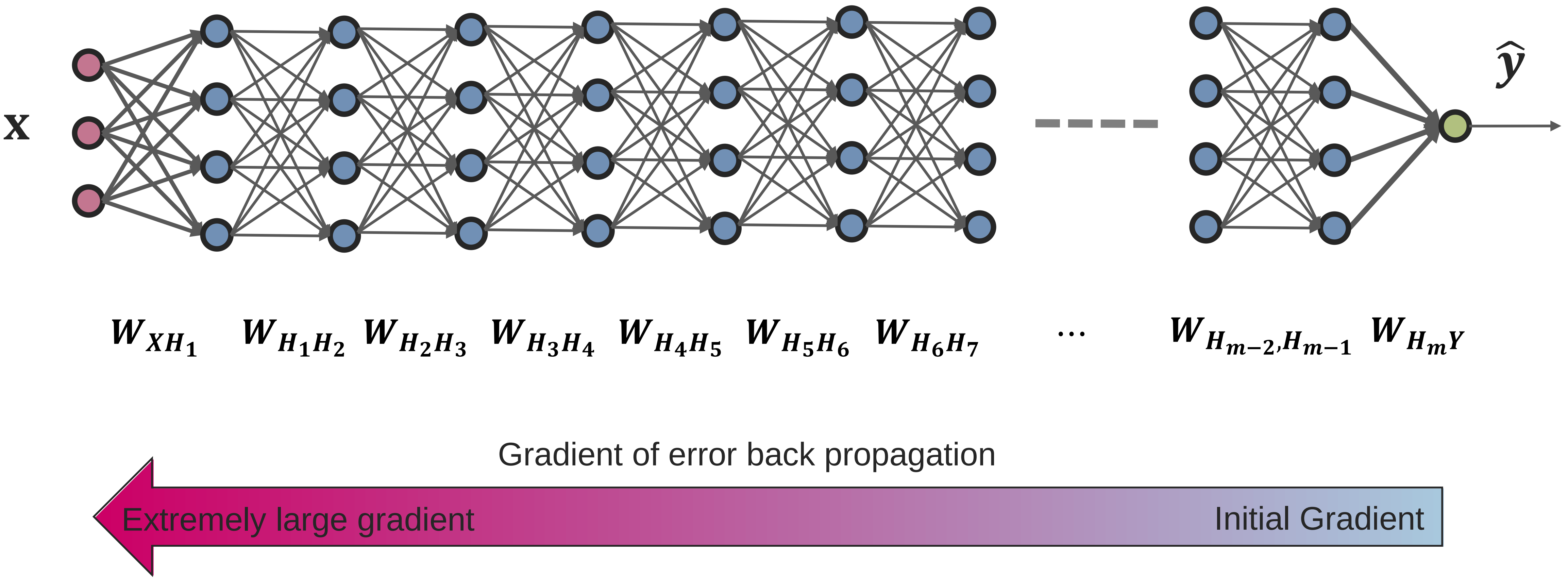
$\mathbf{e} = \hat{y} - \mathbf{y}$

$$\hat{y} = W_L \begin{bmatrix} 0.5 & \cdots & 0.0 \\ \vdots & \ddots & \vdots \\ 0.0 & \cdots & 0.5 \end{bmatrix}^{L-1}$$

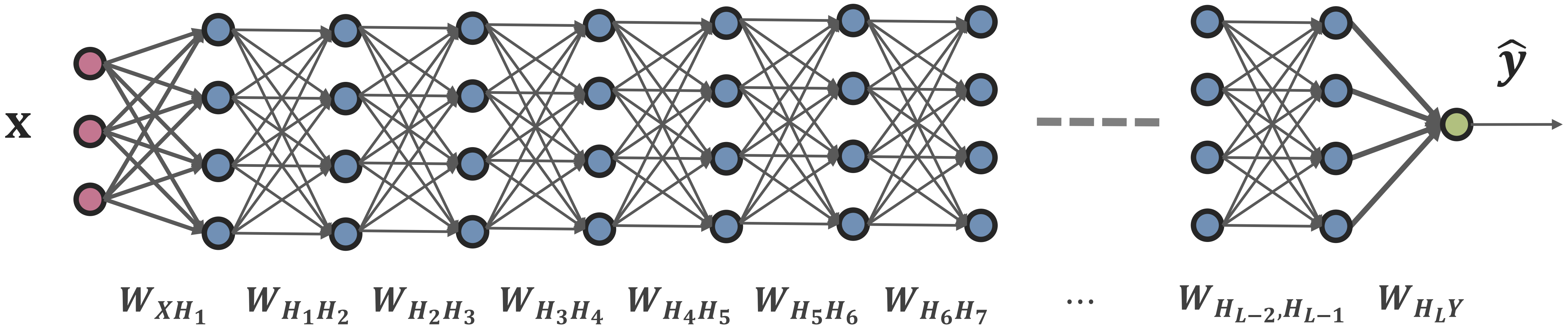
$w = 0.5^{L-1}$ for a large L this will be extremely larger. That is, weight w is a an exponentially **decreasing** function of L

Solution:
Use of **ReLU** function
$\varphi_1(x) = max(0, x)$

Exploding Gradient



Exploding Gradient



Forward pass: $\hat{\mathbf{y}} = \varphi_L(W_L\varphi_{L-1}(W_{L-1}\cdots\varphi_3(W_3\varphi_2(W_2\varphi_1(W_1\mathbf{x})))\cdots))$

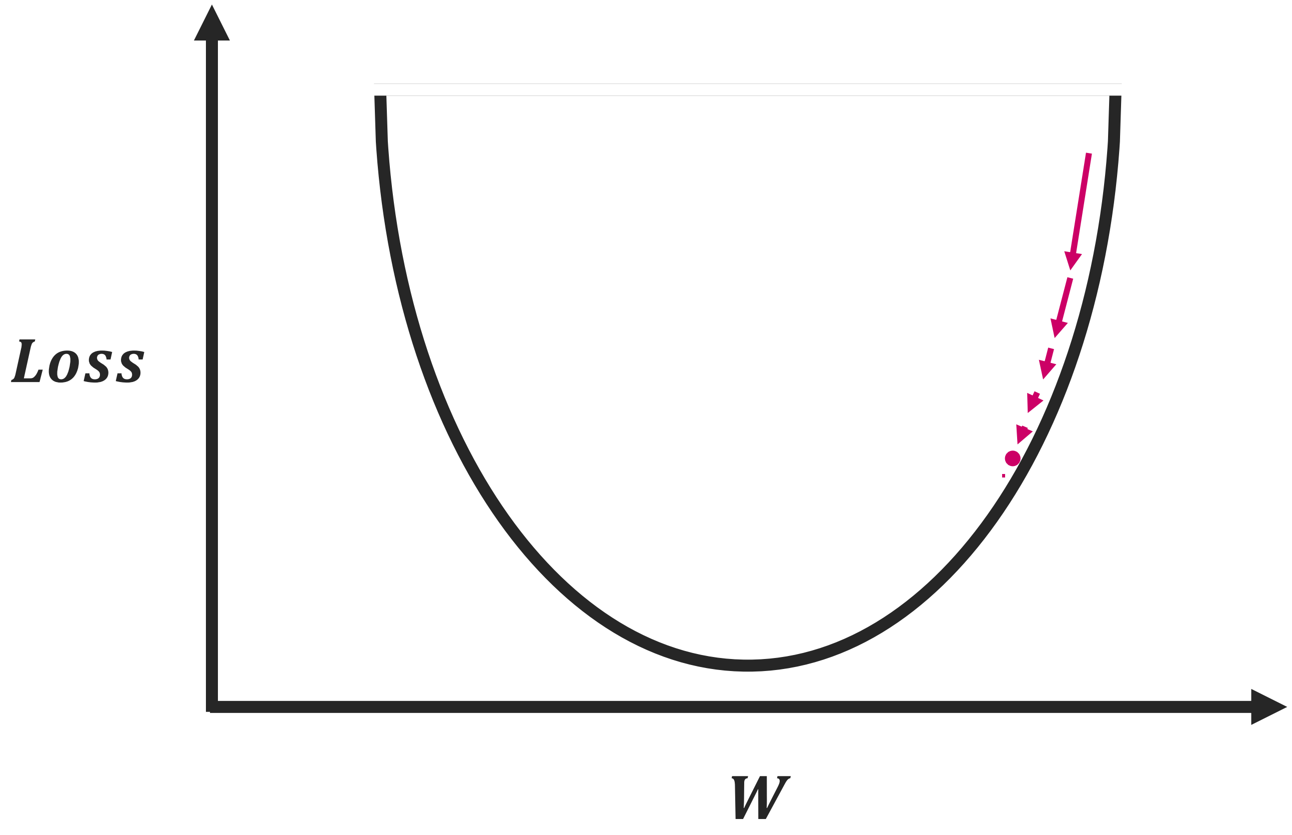
$\mathbf{e} = \hat{\mathbf{y}} - \mathbf{y}$

$$\hat{\mathbf{y}} = W_L \begin{bmatrix} 1.5 & \cdots & 0.0 \\ \vdots & \ddots & \vdots \\ 0.0 & \cdots & 1.5 \end{bmatrix}^{L-1}$$

$w = 1.5^{L-1}$ for a large L this will be **extremely larger**. That is, weight w is a an exponentially **increase** function of L

This is caused by **initialization of weights with large values.**

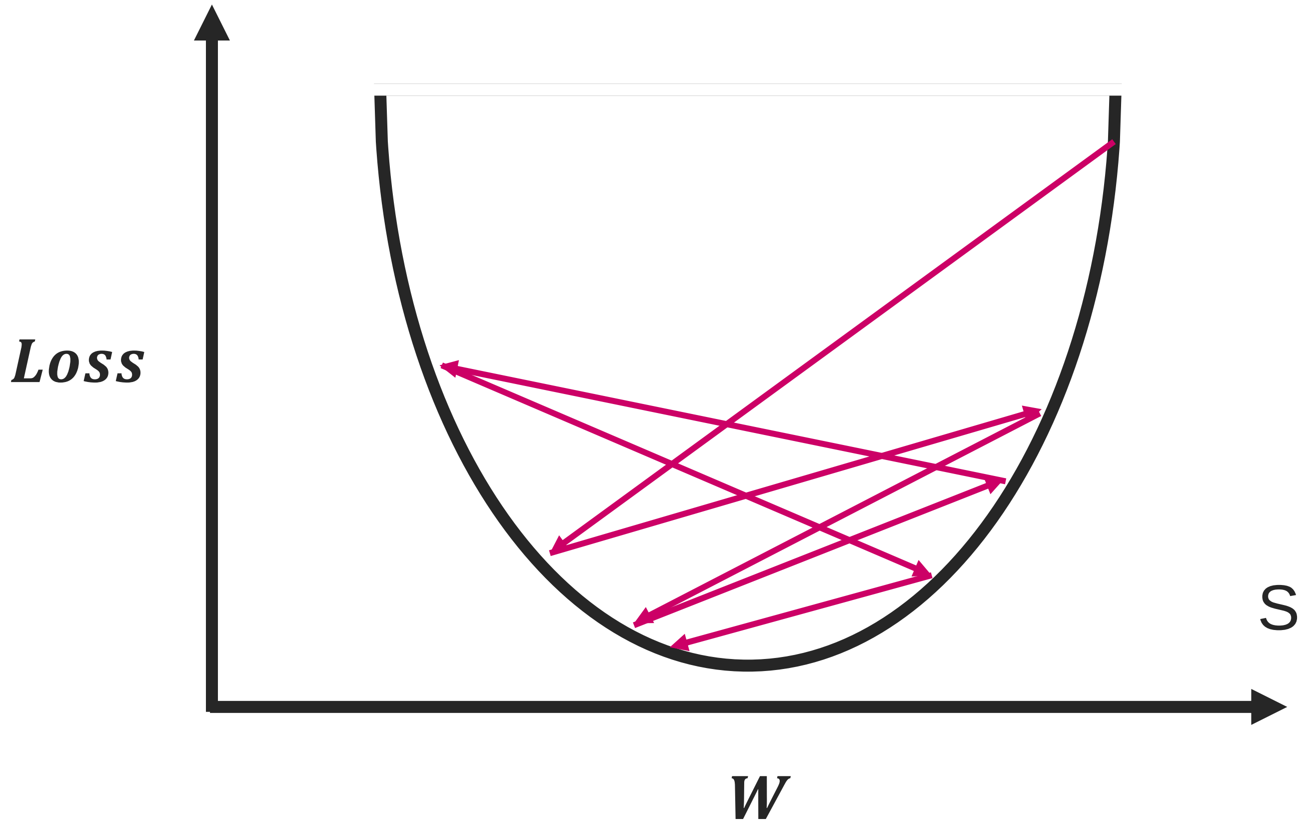
Vanishing Gradient



gradient become very small

Convergence virtually stops because weights do not change any more

Exploding Gradient

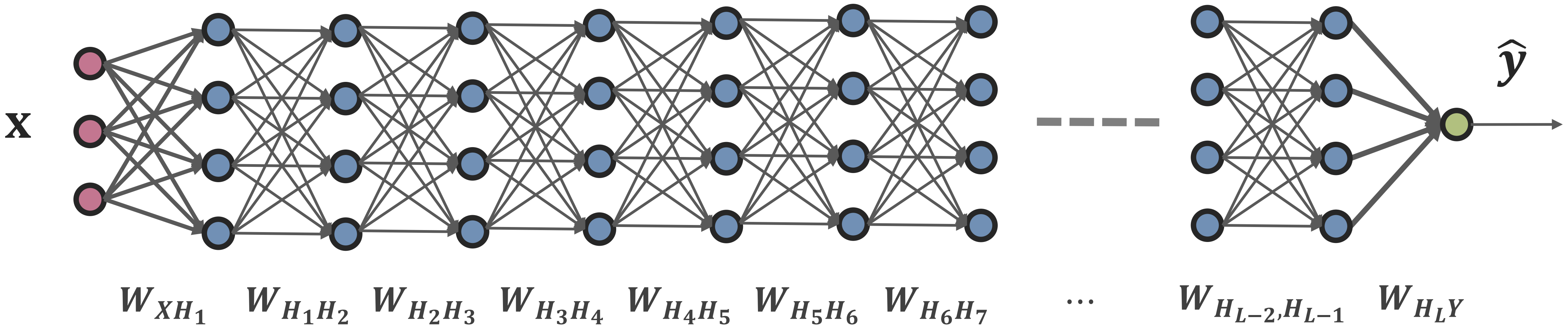


Highly fluctuating gradient descent

Weight abruptly changes.

Skips Global minima

Exploding Gradient



Forward pass: $\hat{\mathbf{y}} = \varphi_L(W_L\varphi_{L-1}(W_{L-1}\cdots\varphi_3(W_3\varphi_2(W_2\varphi_1(W_1\mathbf{x})))\cdots))$

$\mathbf{e} = \hat{\mathbf{y}} - \mathbf{y}$

$$\hat{\mathbf{y}} = W_L \begin{bmatrix} 1.5 & \cdots & 0.0 \\ \vdots & \ddots & \vdots \\ 0.0 & \cdots & 1.5 \end{bmatrix}^{L-1}$$

$w = 1.5^{L-1}$ for a large L this will be extremely larger. That is, weight w is a an exponentially **increase** function of L

Solution:
Gradient clipping and/or better weight initialization

Part 4

Convolutional Neural Network

Data

Image: Gary scale

Data: 2D

1:13 PM

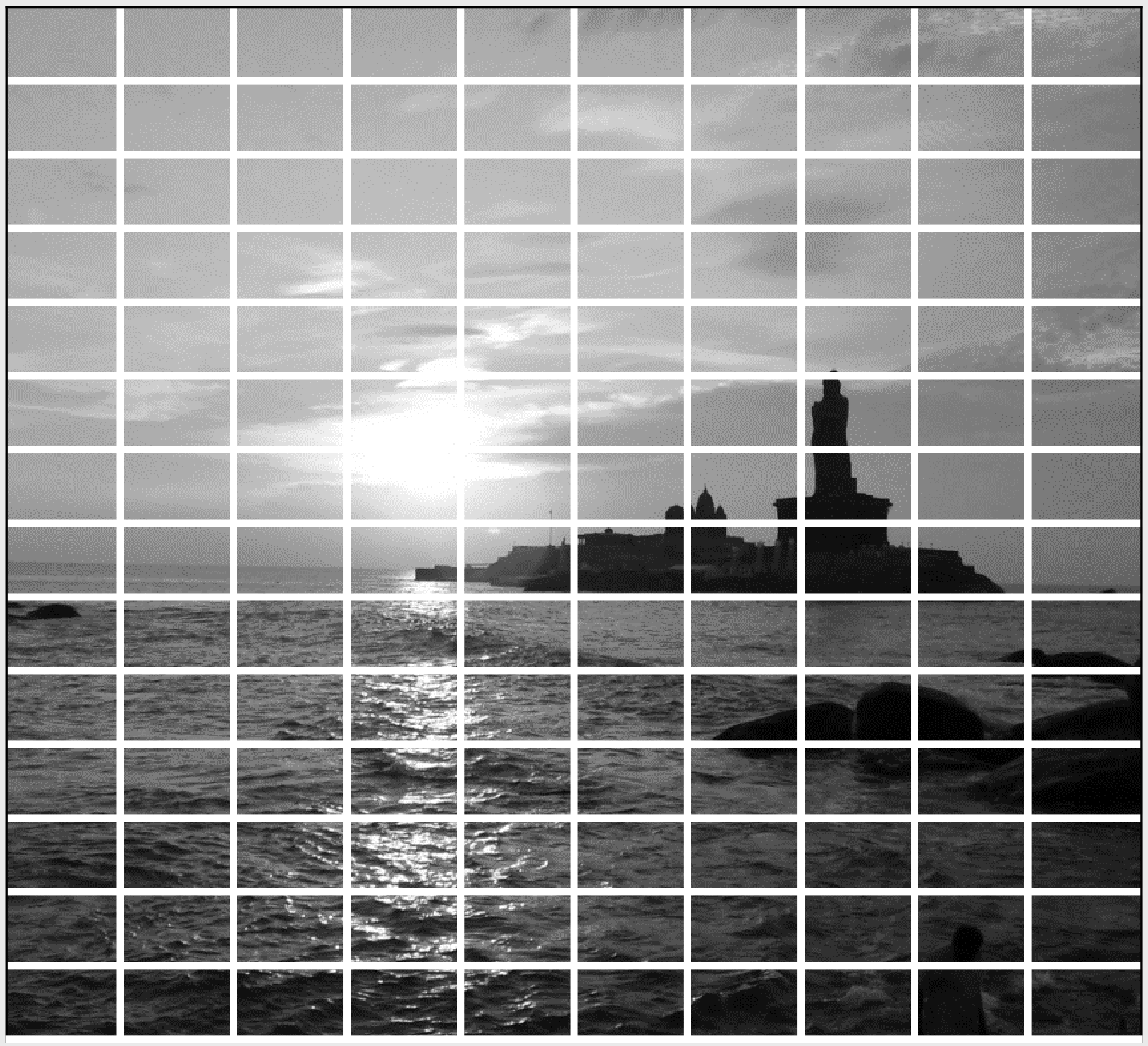
Image: Gary scale

$$I = \begin{bmatrix} \boldsymbol{p}_{11} & \cdots & \boldsymbol{p}_{1,W} \\ \vdots & \ddots & \vdots \\ \boldsymbol{p}_{H,1} & \cdots & \boldsymbol{p}_{H}, \boldsymbol{p}_{W} \end{bmatrix}$$

For $Hight = 256, Width = 256$

$$I = \begin{bmatrix} p_{11} & \cdots & p_{1,256} \\ \vdots & \ddots & \vdots \\ p_{256,1} & \cdots & p_{256}, p_{256} \end{bmatrix}$$

$Hight\ (H)$



$Width\ (W)$

Data: 2D

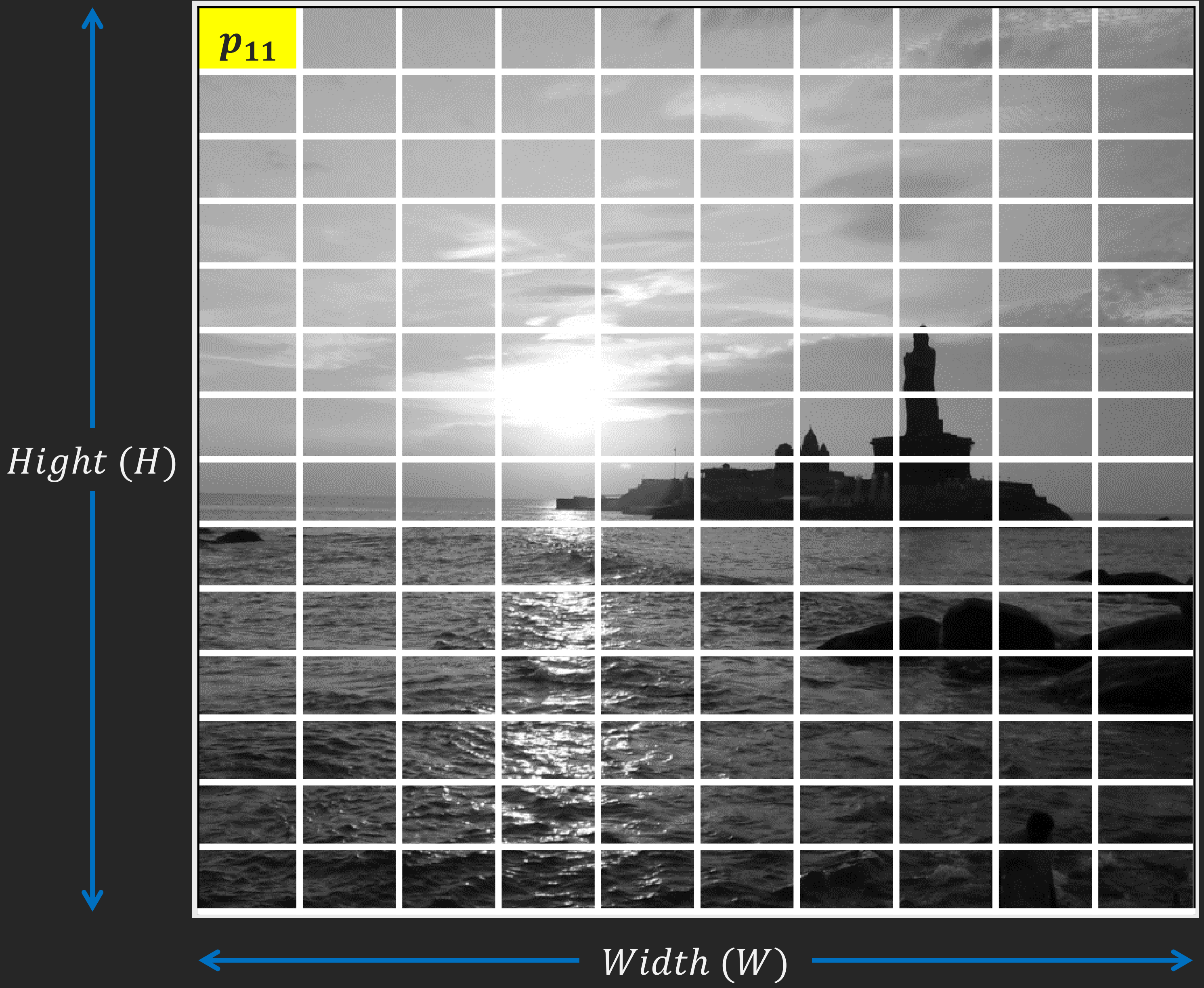
1:13 PM

Image: Gary scale

$\boldsymbol{p_{ij}} \in \{0,1,2., \dots, 256\}$

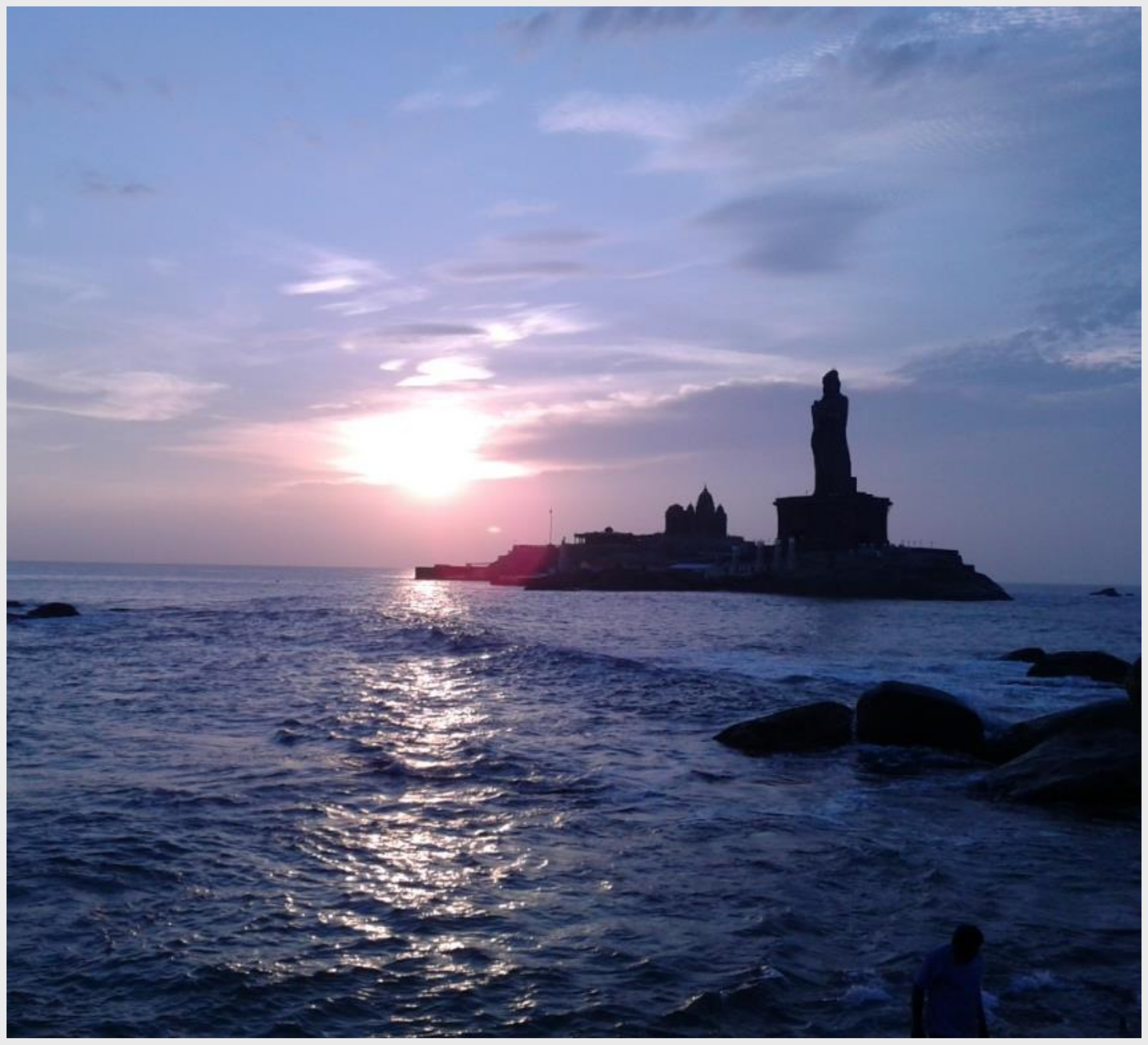
For $Hight = 256, Width = 256$

$$I = \begin{bmatrix} p_{11} & \cdots & p_{1,256} \\ \vdots & \ddots & \vdots \\ p_{256,1} & \cdots & p_{256}, p_{256} \end{bmatrix}$$



Data

Image: Colour



Data: 3D

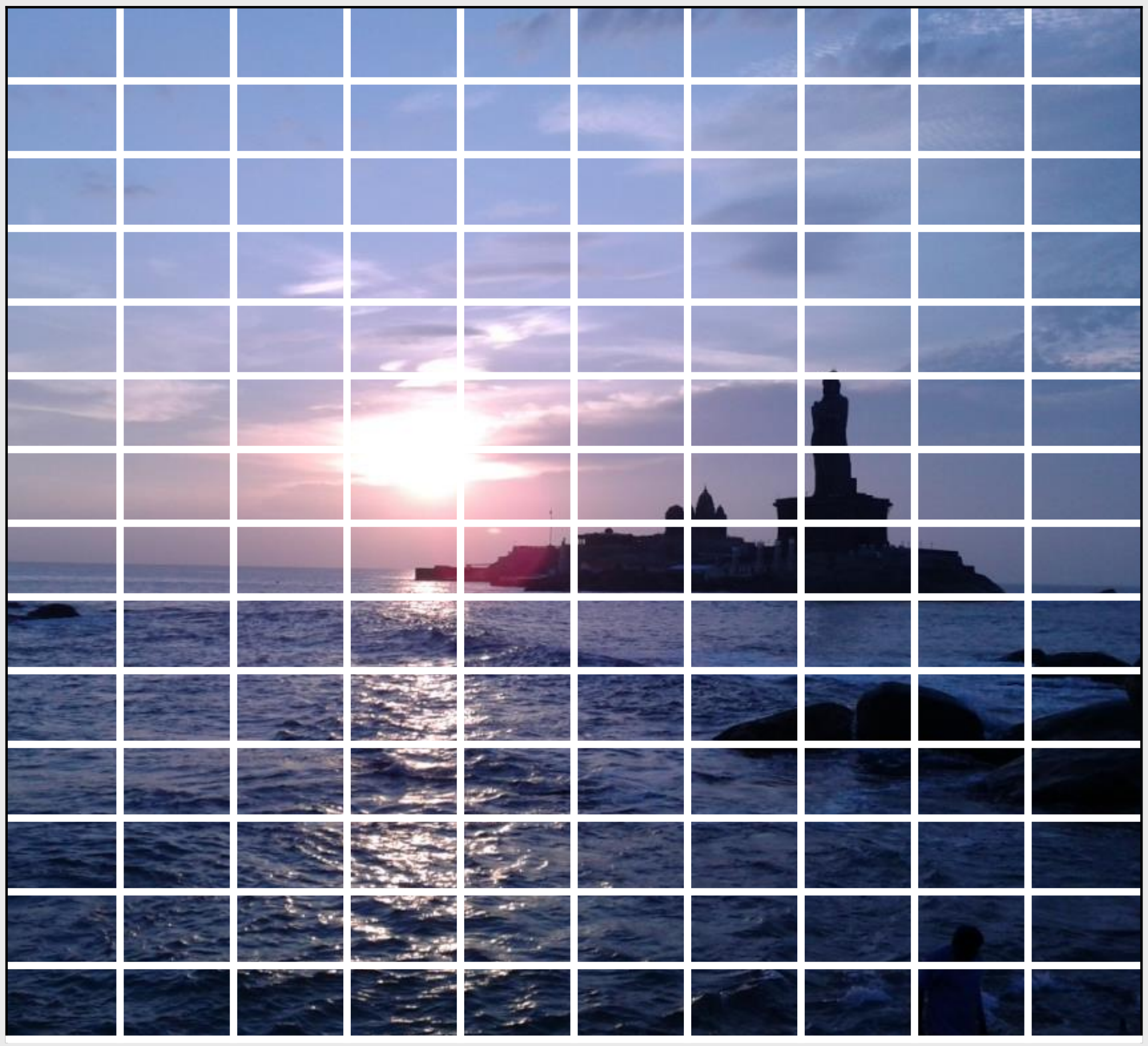
Image: Colour

$$I_{RED} = \begin{bmatrix} p_{11} & \cdots & p_{1,W} \\ \vdots & \ddots & \vdots \\ p_{H,1} & \cdots & p_{H}, p_{W} \end{bmatrix}$$

$$I_{Green} = \begin{bmatrix} p_{11} & \cdots & p_{1,W} \\ \vdots & \ddots & \vdots \\ p_{H,1} & \cdots & p_{H}, p_{W} \end{bmatrix}$$

$$I_{Blue} = \begin{bmatrix} p_{11} & \cdots & p_{1,W} \\ \vdots & \ddots & \vdots \\ p_{H,1} & \cdots & p_{H}, p_{W} \end{bmatrix}$$

$Hight\ (H)$



$Width\ (W)$

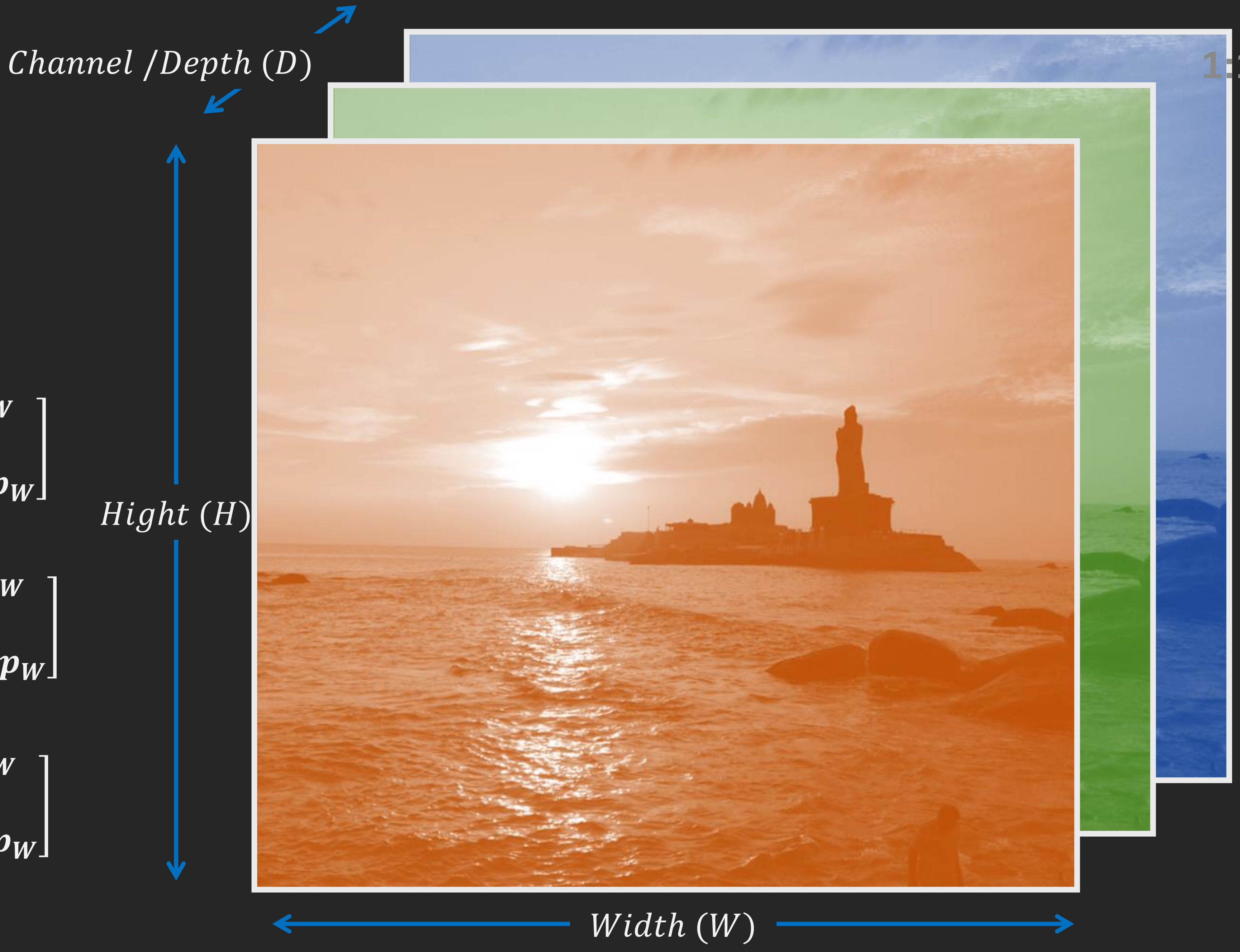
Data

Image: Colour

$$I_{RED} = \begin{bmatrix} p_{11} & \cdots & p_{1,W} \\ \vdots & \ddots & \vdots \\ p_{H,1} & \cdots & p_{H}, p_{W} \end{bmatrix}$$

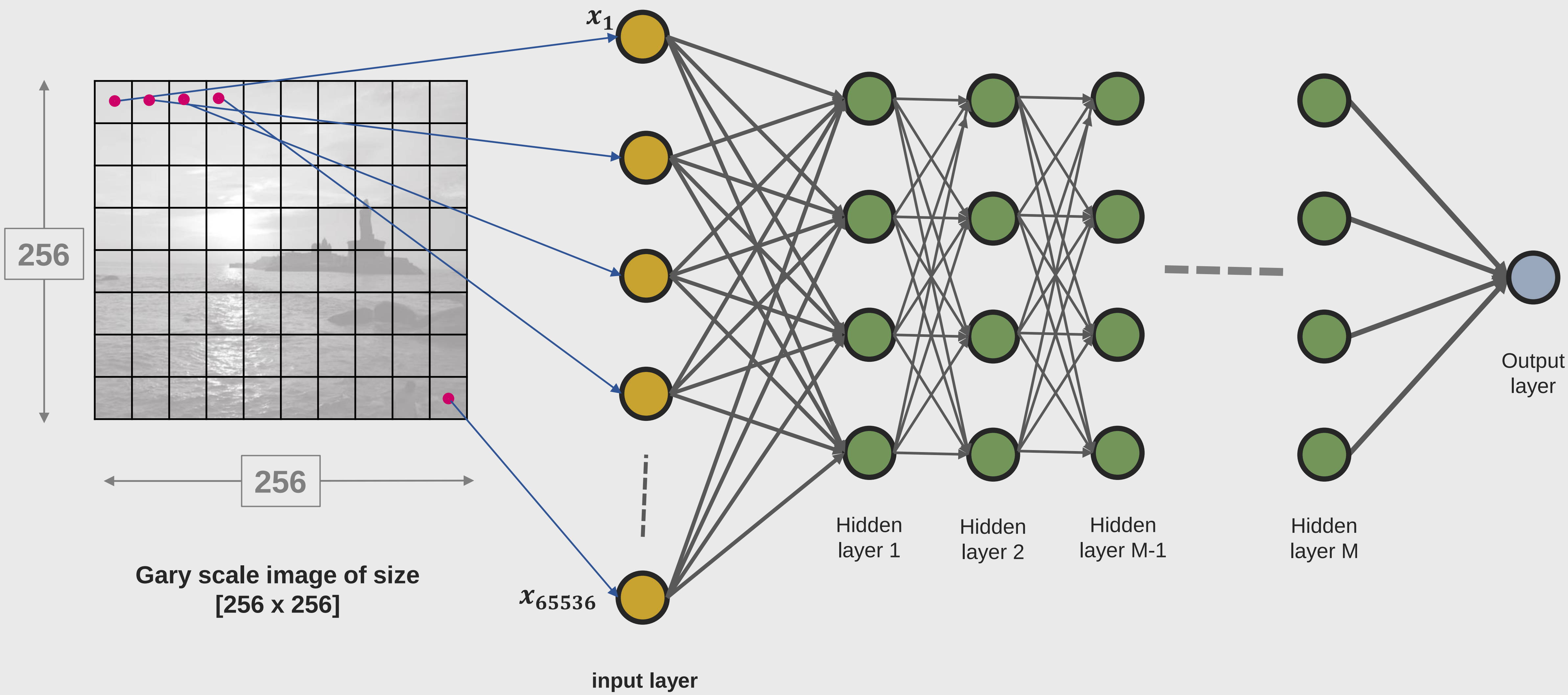
$$I_{Green} = \begin{bmatrix} p_{11} & \cdots & p_{1,W} \\ \vdots & \ddots & \vdots \\ p_{H,1} & \cdots & p_{H}, p_{W} \end{bmatrix}$$

$$I_{Blue} = \begin{bmatrix} p_{11} & \cdots & p_{1,W} \\ \vdots & \ddots & \vdots \\ p_{H,1} & \cdots & p_{H}, p_{W} \end{bmatrix}$$



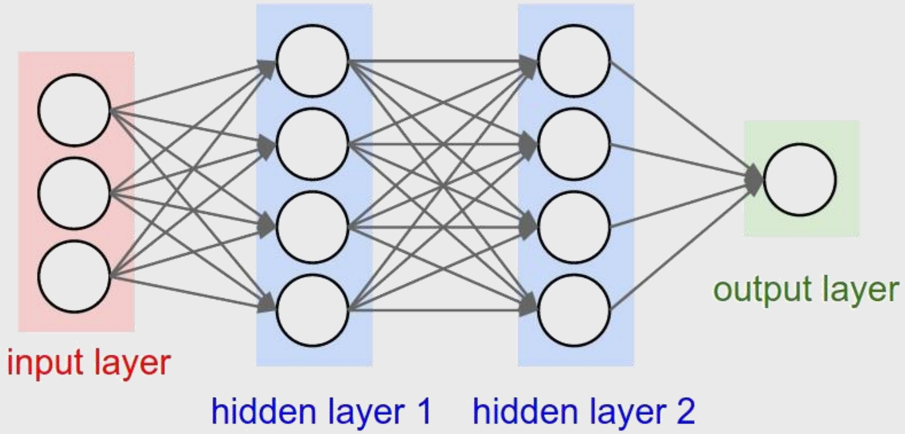
1:13 PM

Deep Learning

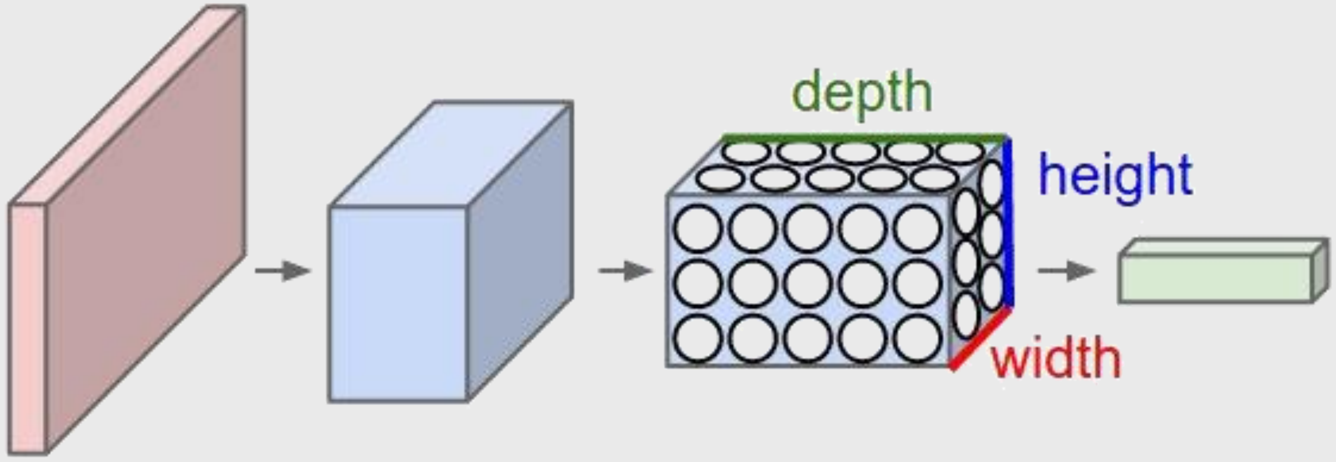


Gary scale image of size [256 x 256]

Convolutional Neural Network (CNN)



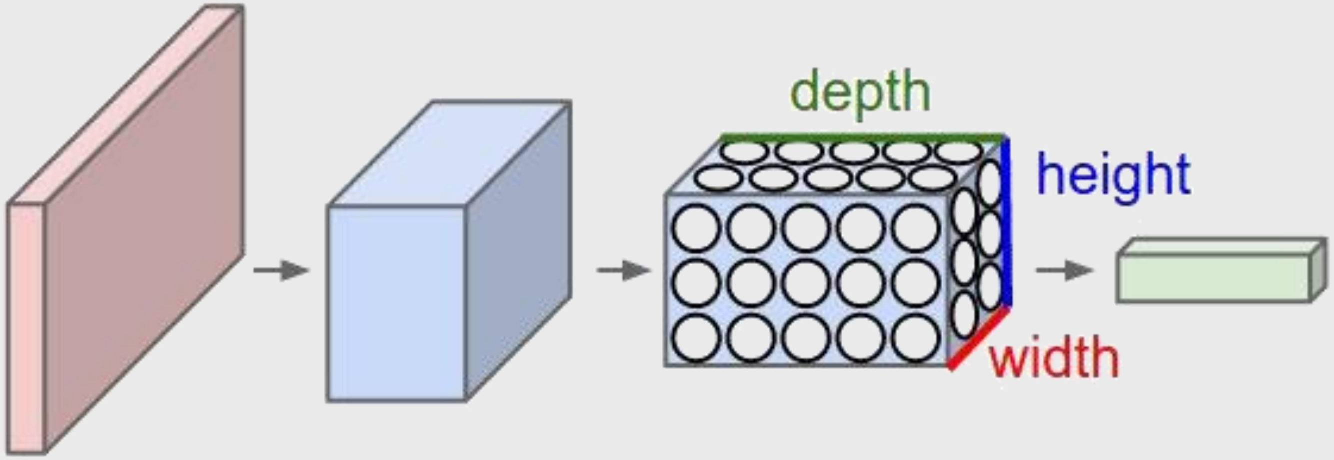
A regular Neural Network



Convolutional Neural Network

A very good source: http://cs231n.github.io/convolutional-networks/

Convolutional Neural Network (ConvNet)



A **ConvNet** arranges its neurons in three dimensions (**width, height, depth**).

Every layer of a **ConvNet** transforms the 3D input volume to a 3D output volume of neuron activations.

In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels)

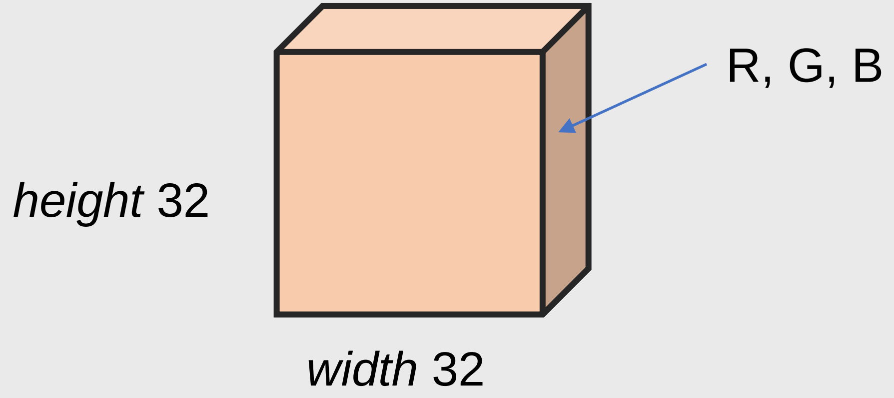
A very good source: http://cs231n.github.io/convolutional-networks/

ConvNet/ CNN

Architecture: A Simple ConvNet / CNN

[**INPUT** - CONV - RELU - POOL - FC]

INPUT [32x32x3] will hold the raw pixel values of the image.
Image *width* 32, *height* 32, and with *three* colour channels R,G,B.



A very good source: http://cs231n.github.io/convolutional-networks/

ConvNet/ CNN

Architecture: A Simple ConvNet / CNN

[INPUT - **CONV** - RELU - POOL - FC]

CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume.

This may result in volume such as [32x32x12] if we decided to use 12 filters

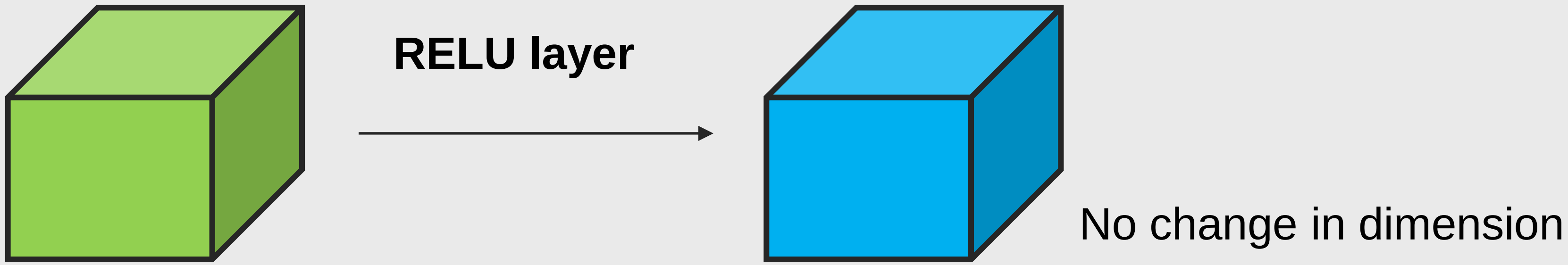
A very good source: http://cs231n.github.io/convolutional-networks/

ConvNet/ CNN

Architecture: A Simple ConvNet / CNN

[INPUT - CONV - **RELU** - POOL - FC]

RELU layer will apply an elementwise activation function, such as the $\boldsymbol{max(0, x)}$ thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).



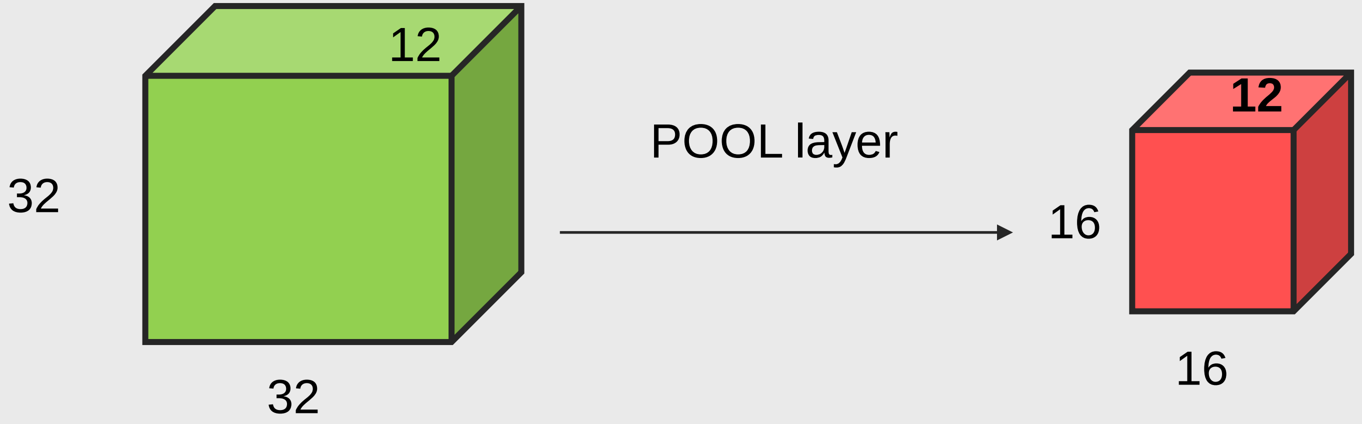
A very good source: http://cs231n.github.io/convolutional-networks/

ConvNet/ CNN

Architecture: A Simple ConvNet / CNN

[INPUT - CONV - RELU - POOL - FC]

POOL layer will perform a **down sampling** operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12]



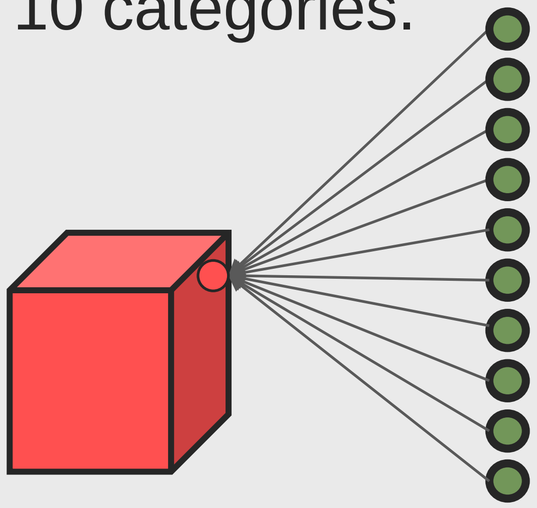
A very good source: http://cs231n.github.io/convolutional-networks/

ConvNet/ CNN

Architecture: A Simple ConvNet / CNN

[INPUT - CONV - RELU - POOL - **FC**]

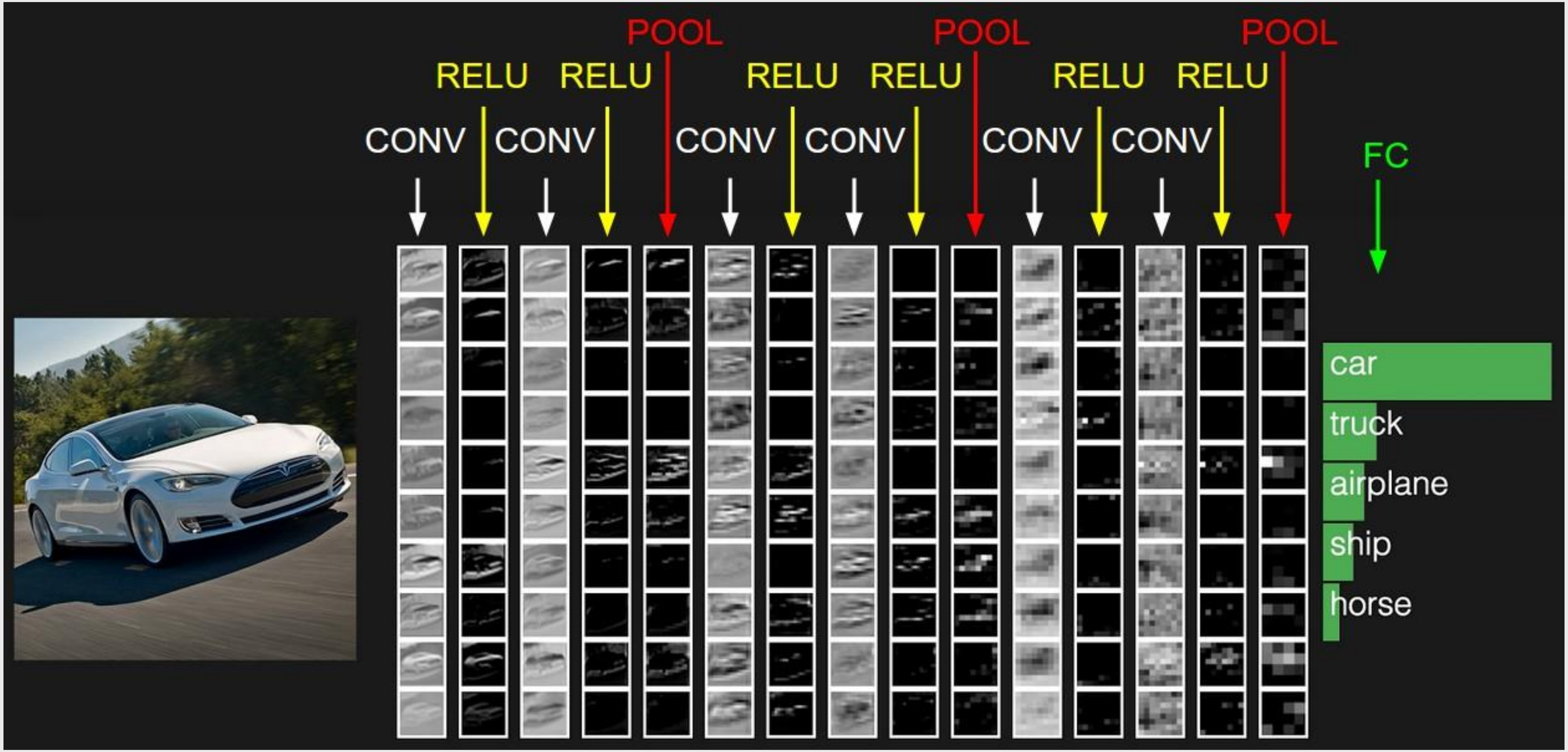
FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 neurone correspond to a class score, such as among the 10 categories.



A very good source: http://cs231n.github.io/convolutional-networks/

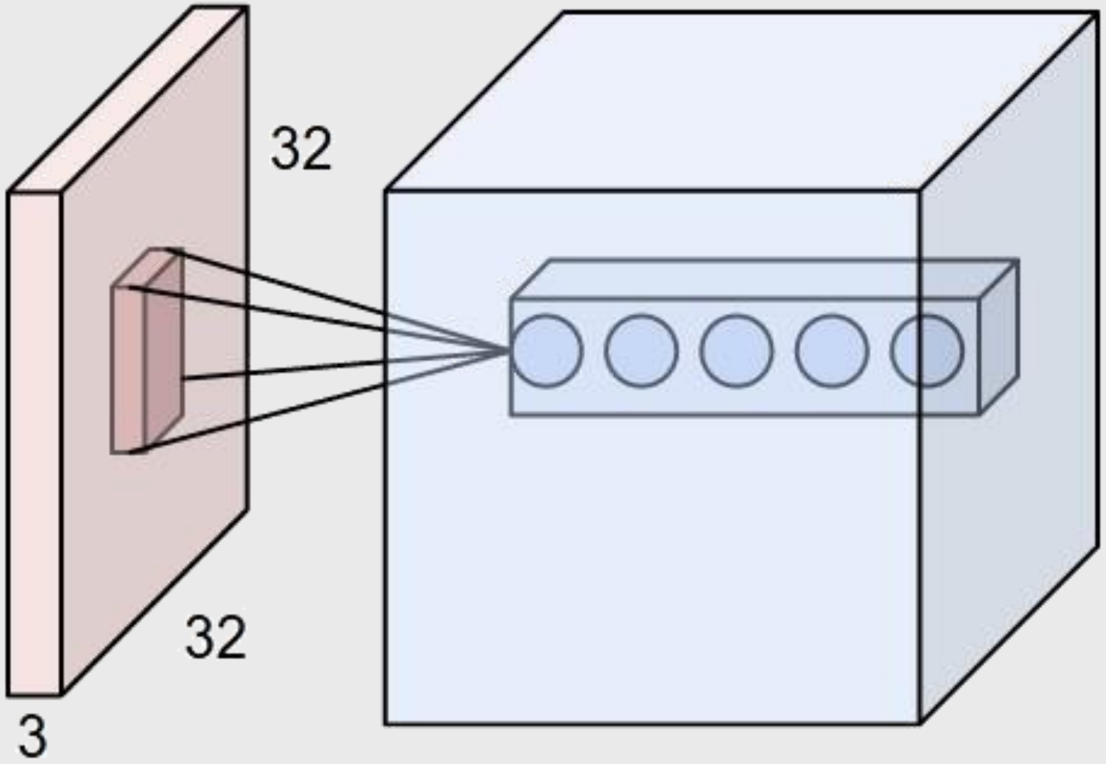
ConvNet/ CNN: A Simple Example

Live demo http://cs231n.stanford.edu/



ConvNet

Convolution Layer



An input volume in red (e.g. a 32x32x3), and an example volume of neurons in the first Convolutional layer.

A very good source: http://cs231n.github.io/convolutional-networks/

ConvNet

Convolution Layer

- CONV layer’s parameters consist of a set of **learnable filters**.
- Every filter is small spatially (along width and height), but extends through the full depth of the input volume.
- A typical filter on a first layer of a ConvNet might have size **5x5x3** (i.e. 5 pixels width and height, and 3 because images have depth 3, the colour channels)

A very good source: http://cs231n.github.io/convolutional-networks/

ConvNet
Convolution Layer

- Forward pass: we slide (**convolve**) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position.

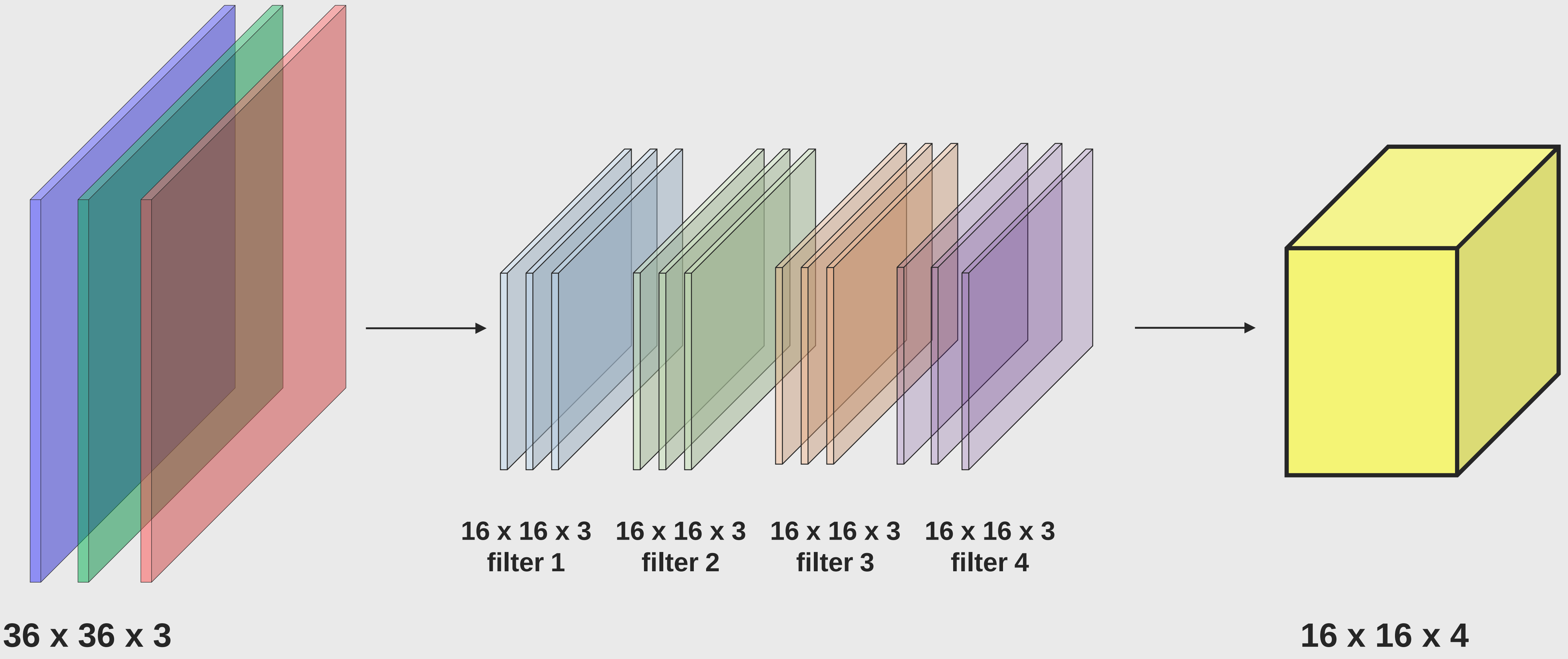
- When we slide the **filter** over the width and height of the input volume we will produce a **2-dimensional activation map** that gives the responses of that filter at every spatial position

- We can have a set of filters (e.g., 12)

A very good source: http://cs231n.github.io/convolutional-networks/

ConvNet

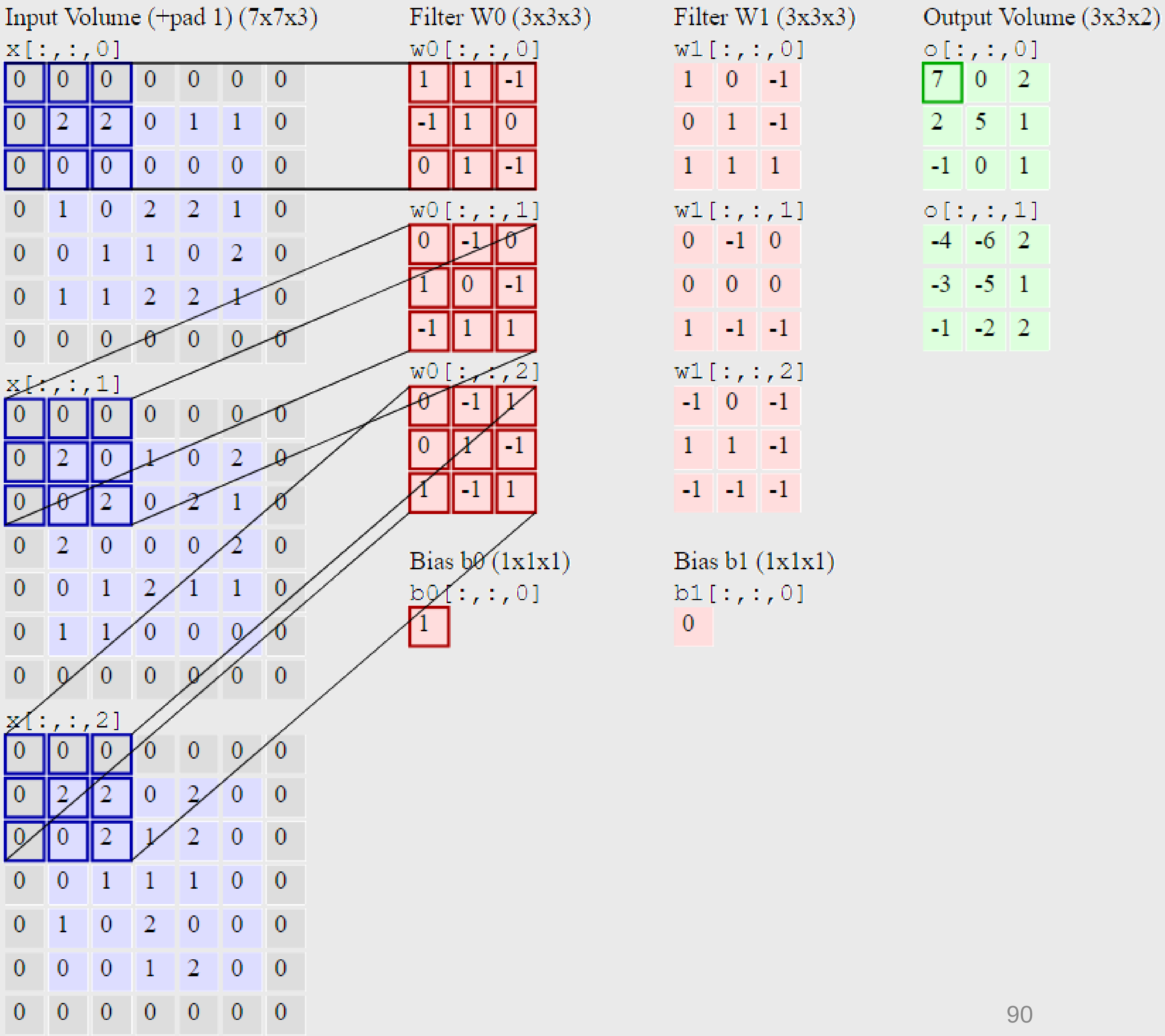
Convolution Layer



ConvNet
Convolution Layer

Requires four hyperparameters:

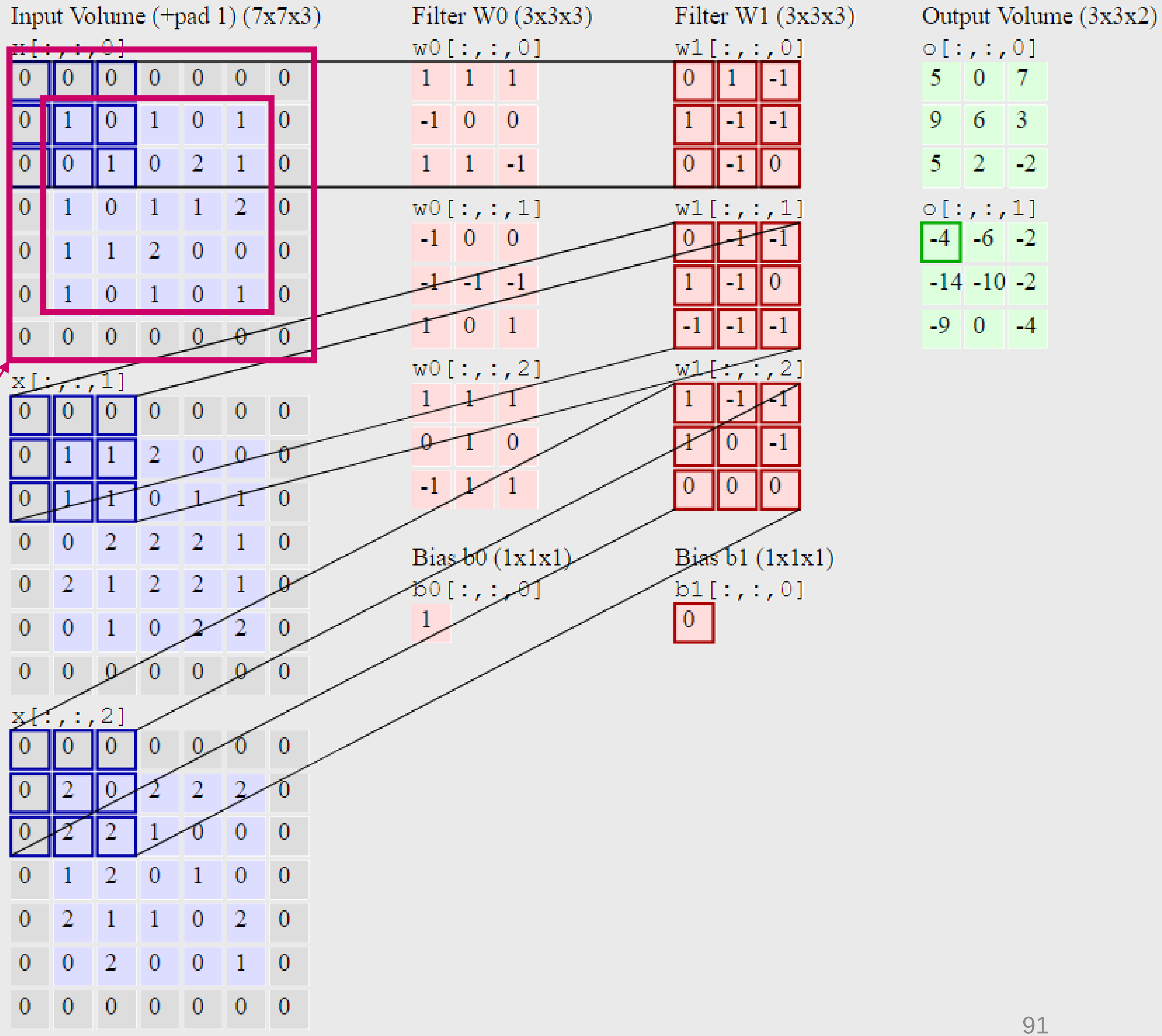
- Number of filters **K**,
- their spatial extent **F**,
- the stride **S**,
- the amount of zero padding **P**.



Source: http://cs231n.github.io/convolutional-networks/

ConvNet
Convolution Layer

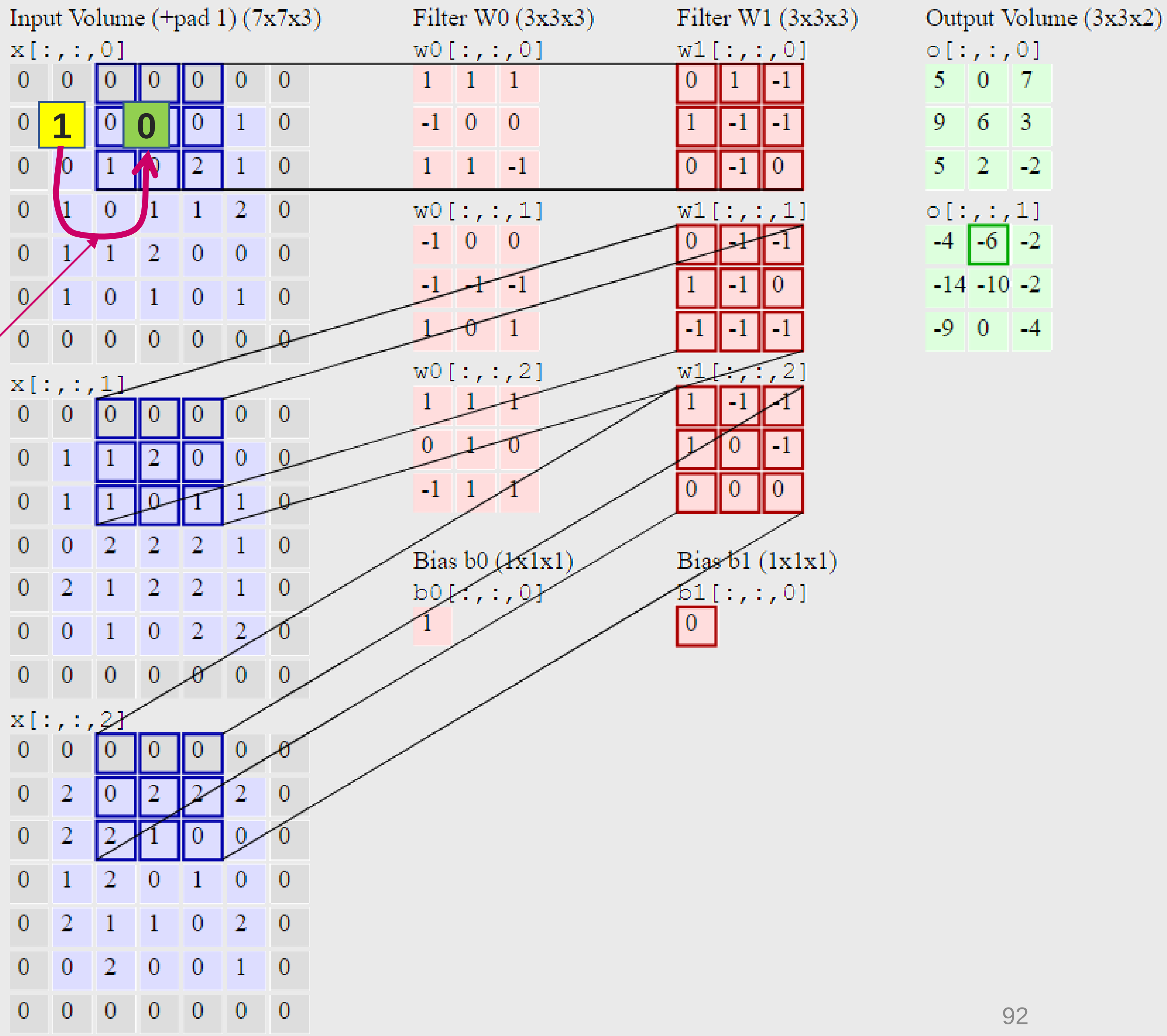
Requires four hyperparameters:
- Number of filters **K = 2**,
- their spatial extent **F = 3**,
- the stride **S = 2**,
- the amount of zero padding **P = 1**.



Source: http://cs231n.github.io/convolutional-networks/

ConvNet

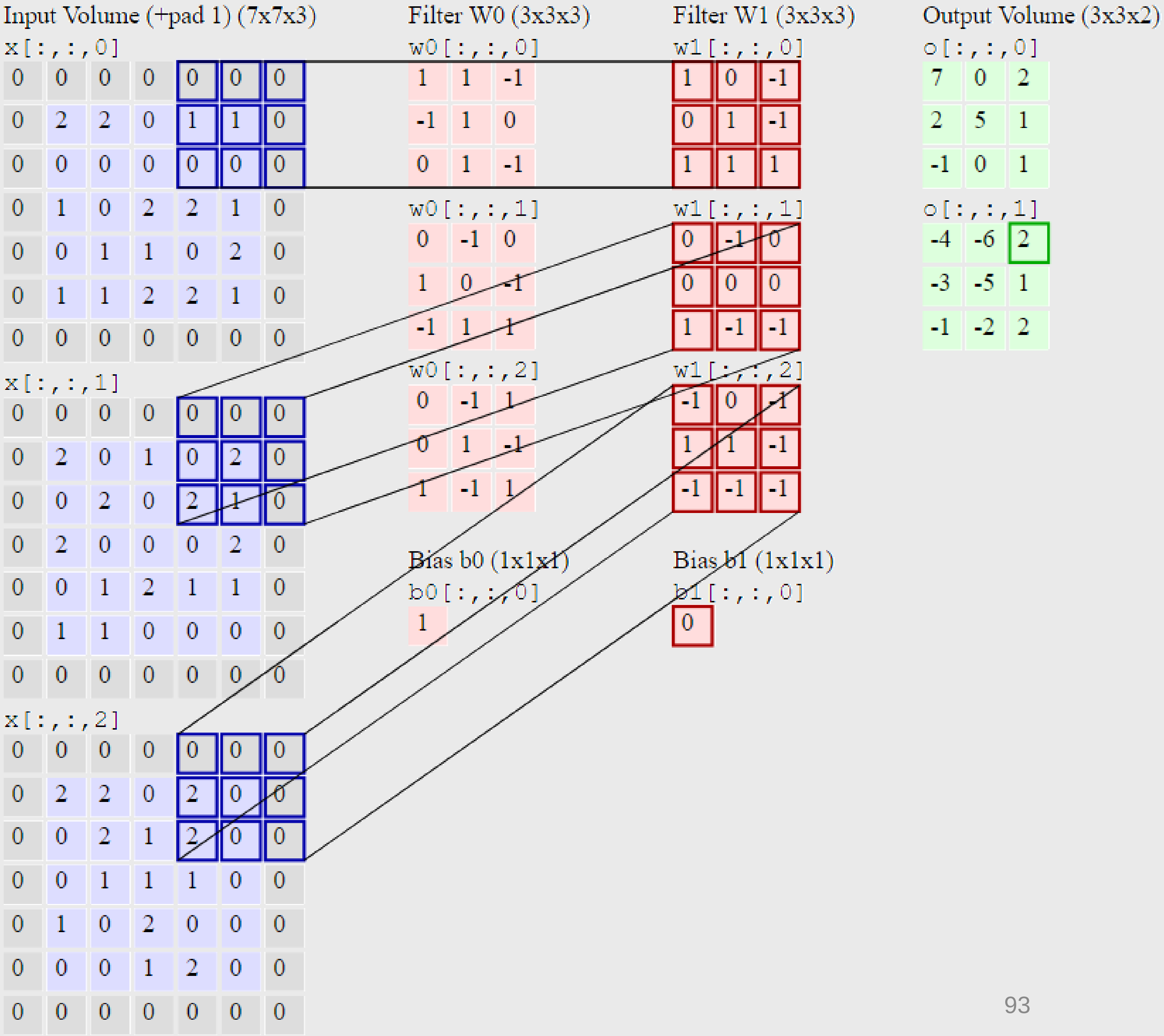
Convolution Layer



Stride (central cell jump) = 2

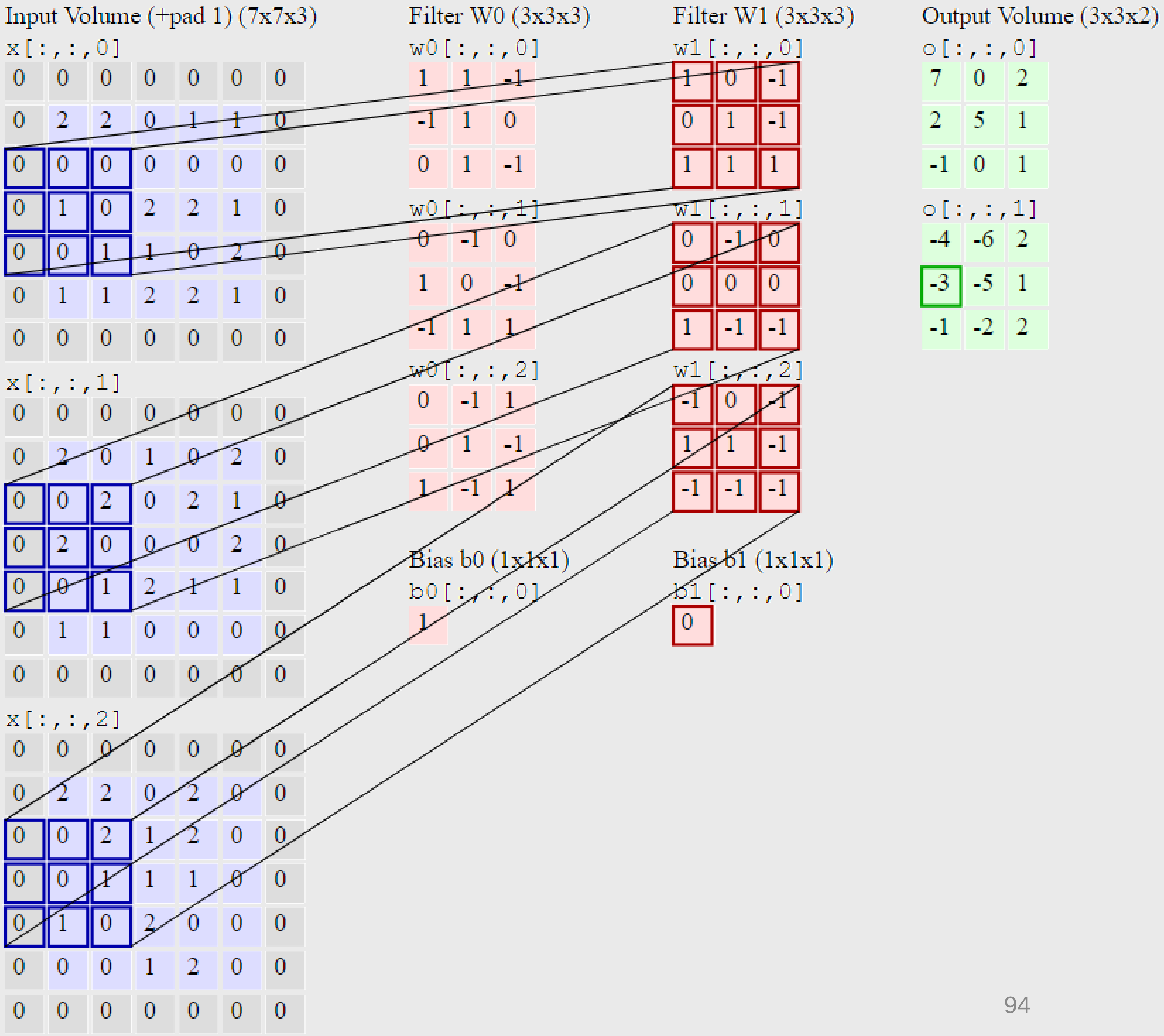
Source: http://cs231n.github.io/convolutional-networks/

ConvNet
Convolution Layer



Source: http://cs231n.github.io/convolutional-networks/

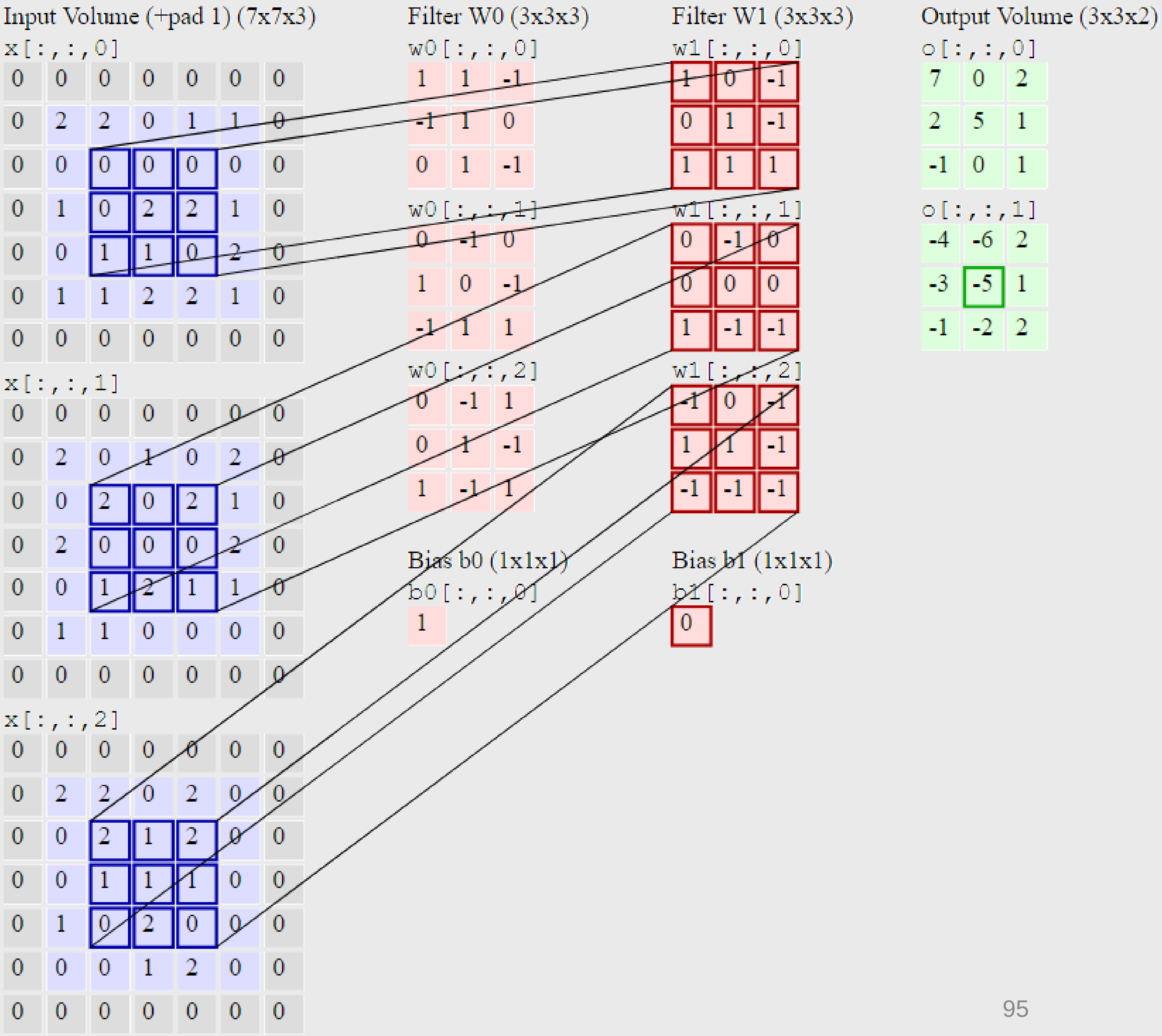
ConvNet
Convolution Layer



Source: http://cs231n.github.io/convolutional-networks/

ConvNet

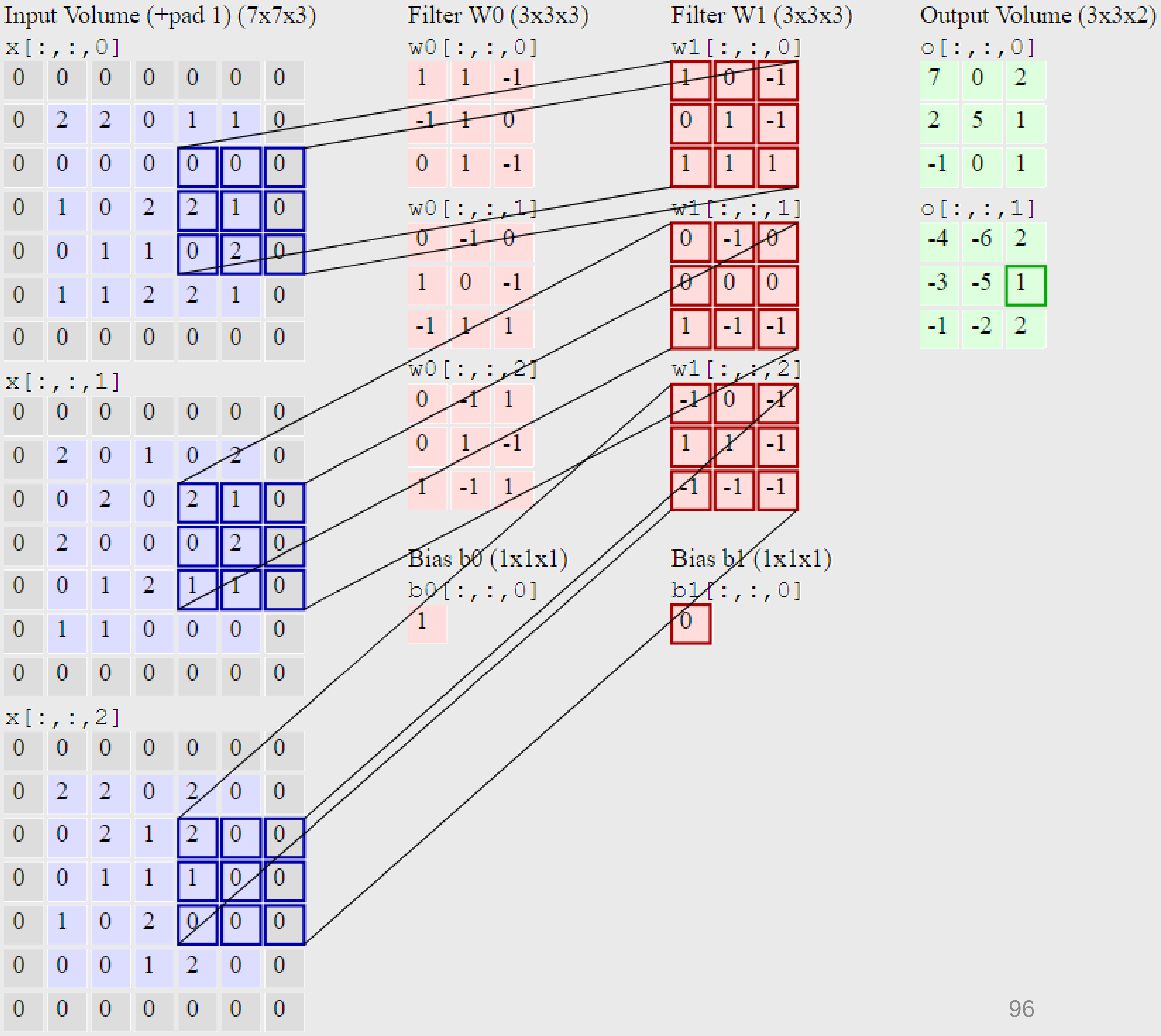
Convolution Layer



Source: http://cs231n.github.io/convolutional-networks/

ConvNet

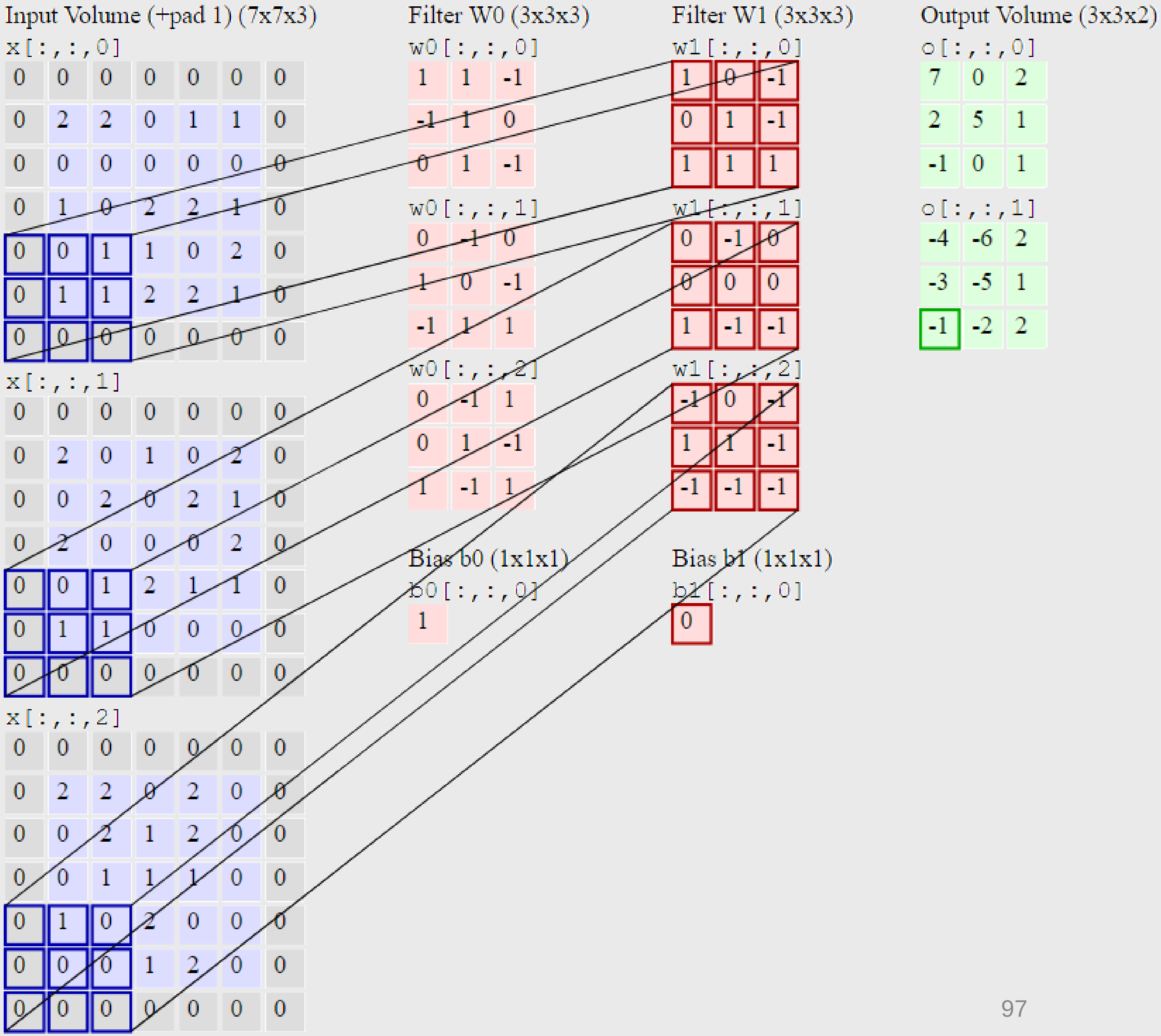
Convolution Layer



Source: http://cs231n.github.io/convolutional-networks/

ConvNet

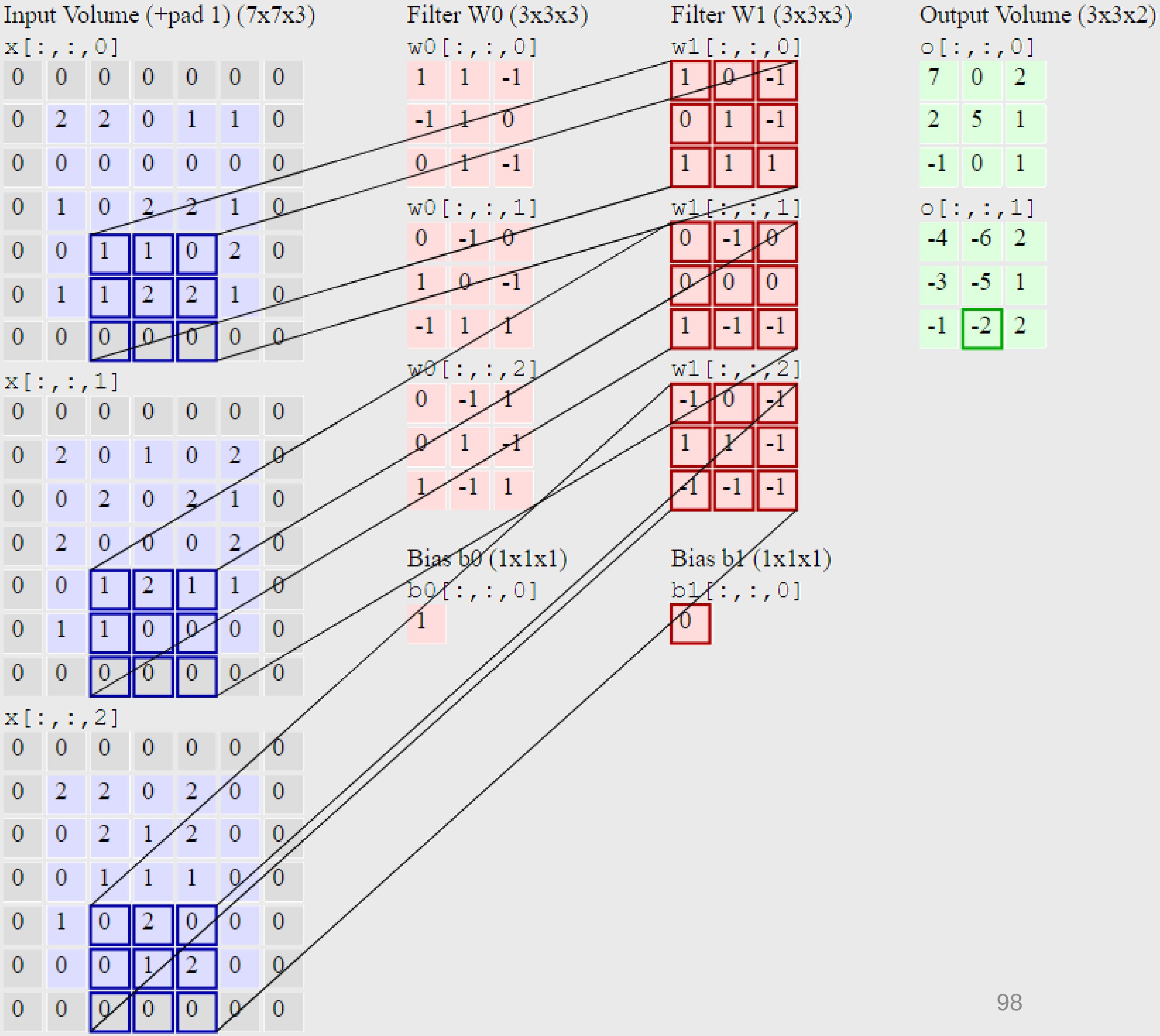
Convolution Layer



Source: http://cs231n.github.io/convolutional-networks/

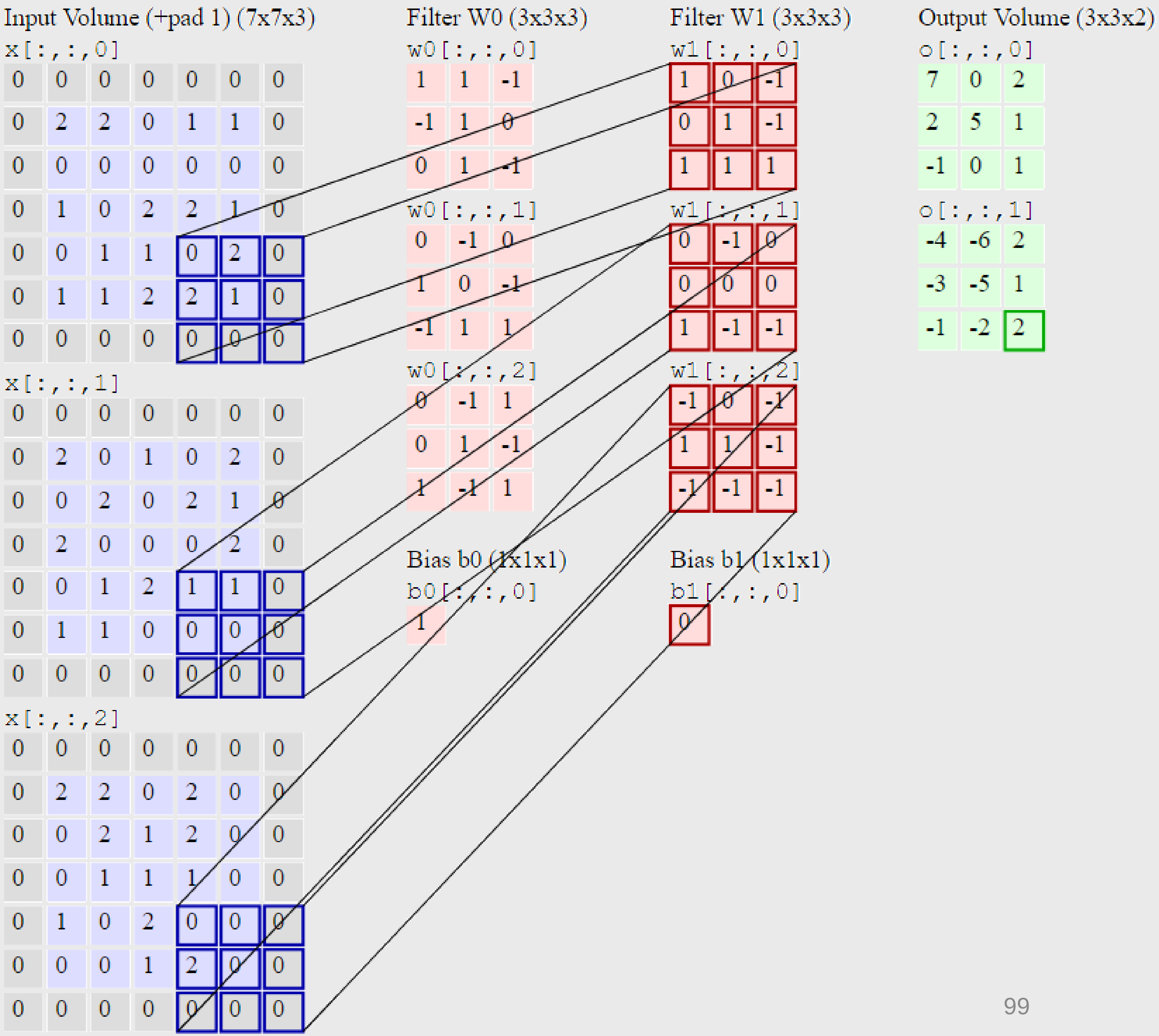
ConvNet

Convolution Layer



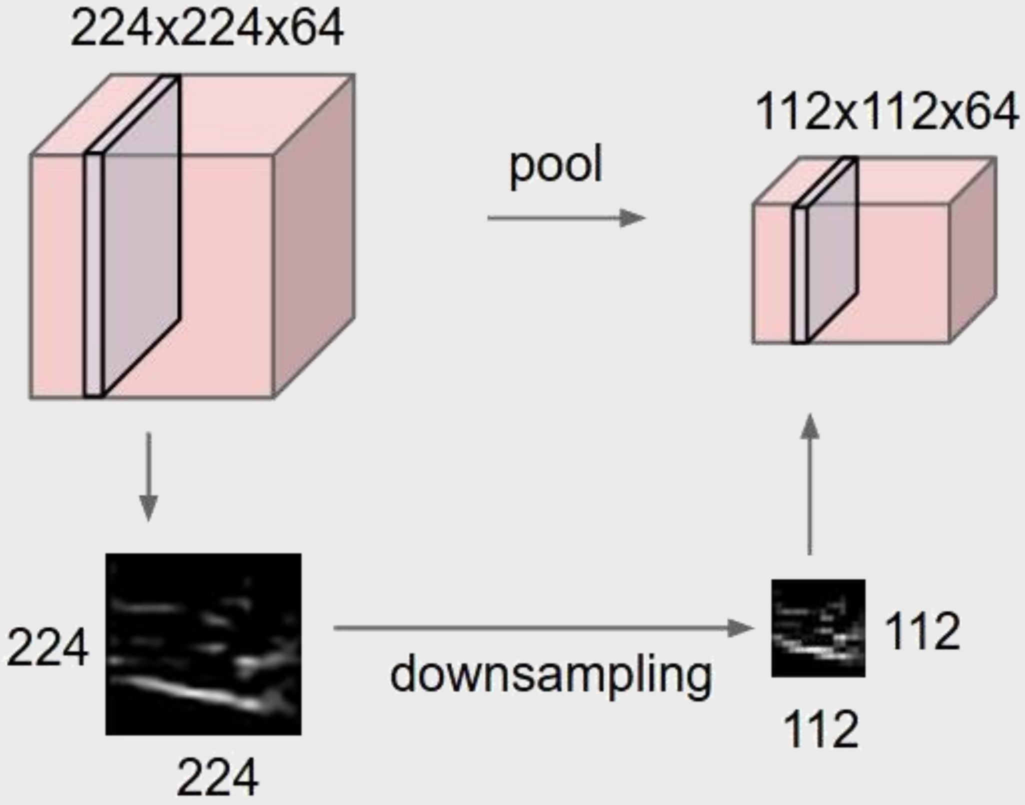
Source: http://cs231n.github.io/convolutional-networks/

ConvNet
Convolution Layer



ConvNet

Pooling Layer

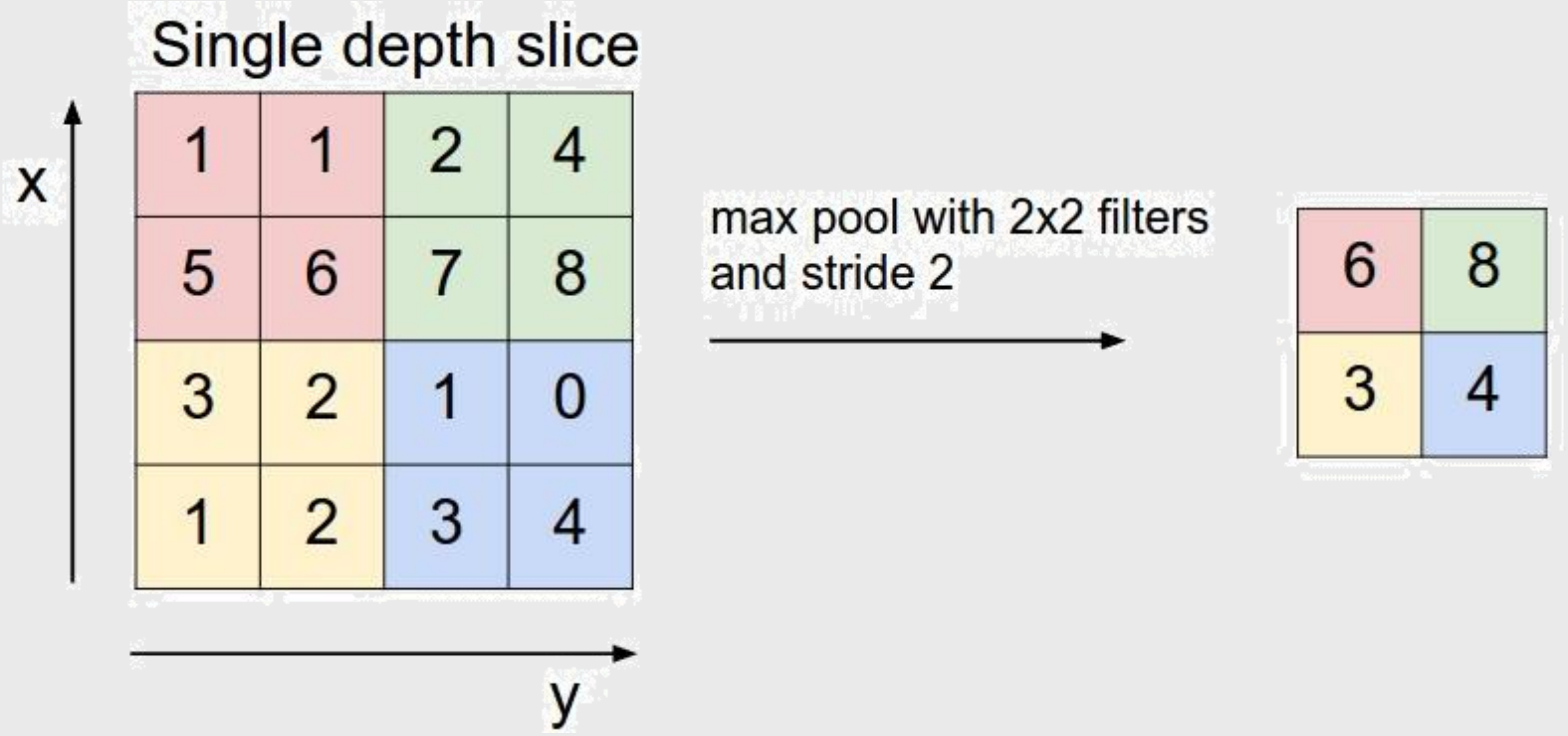


Pooling layer **down samples** the volume spatially, independently in each depth slice of the input volume

A very good source: http://cs231n.github.io/convolutional-networks/

ConvNet

Pooling Layer

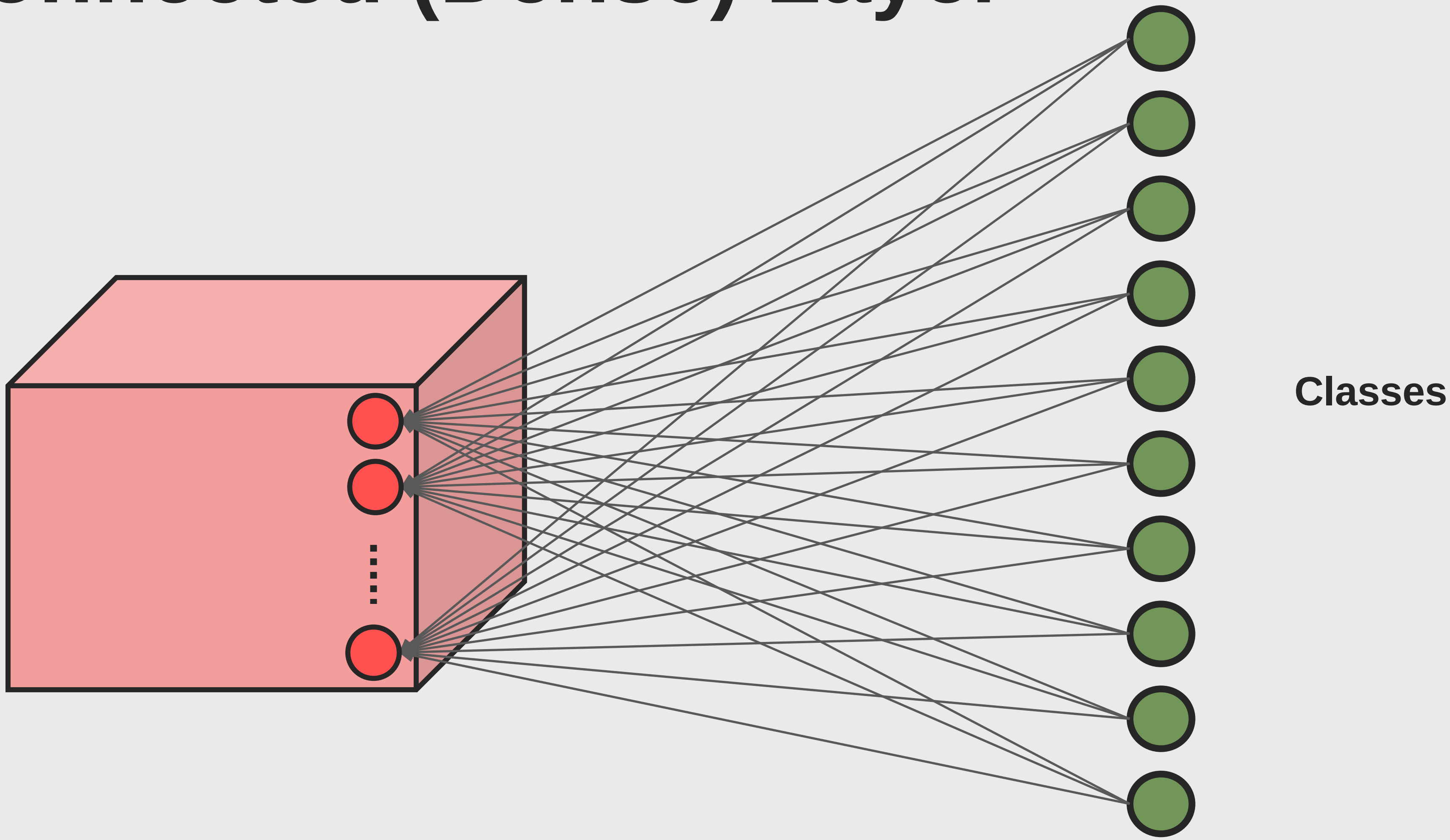


Max Pooling layer

A very good source: http://cs231n.github.io/convolutional-networks/

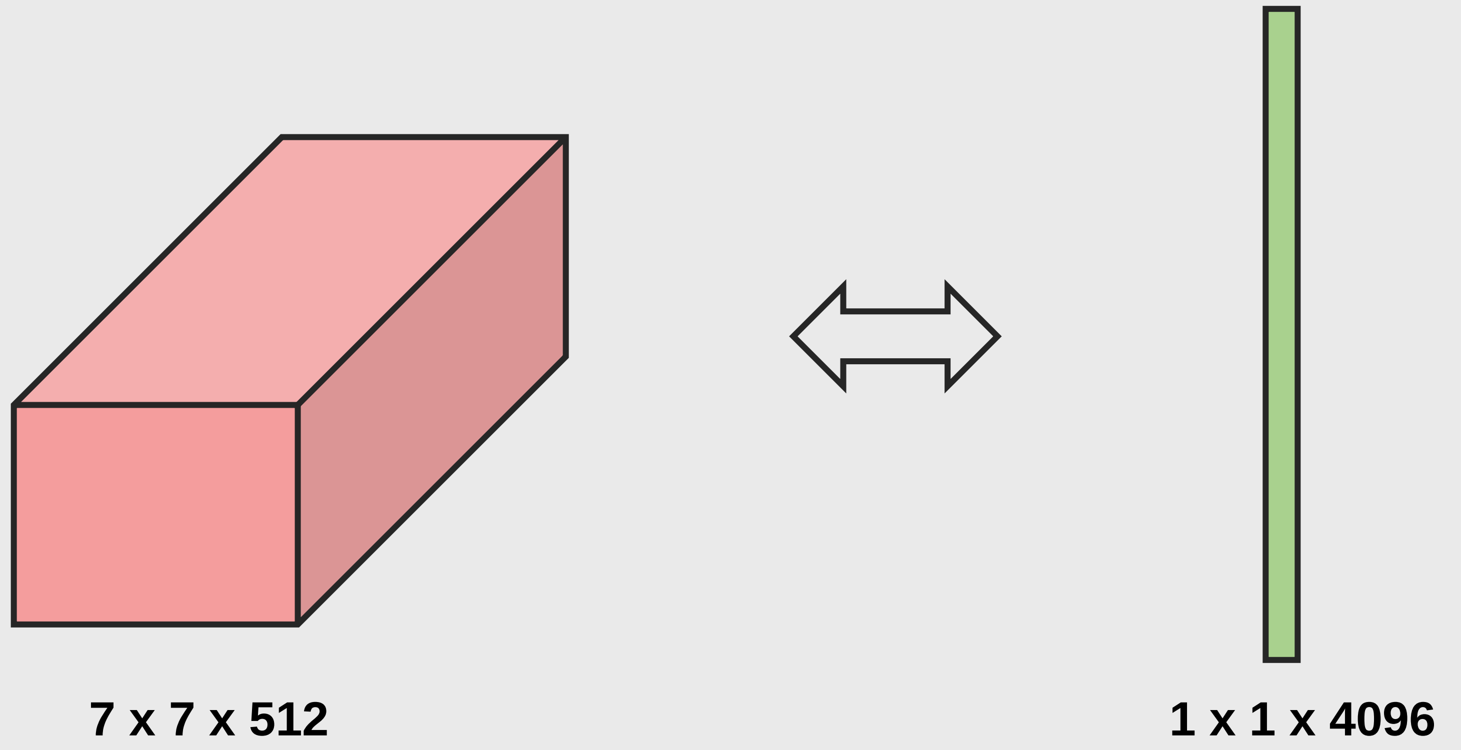
ConvNet

Fully Connected (Dense) Layer



ConvNet

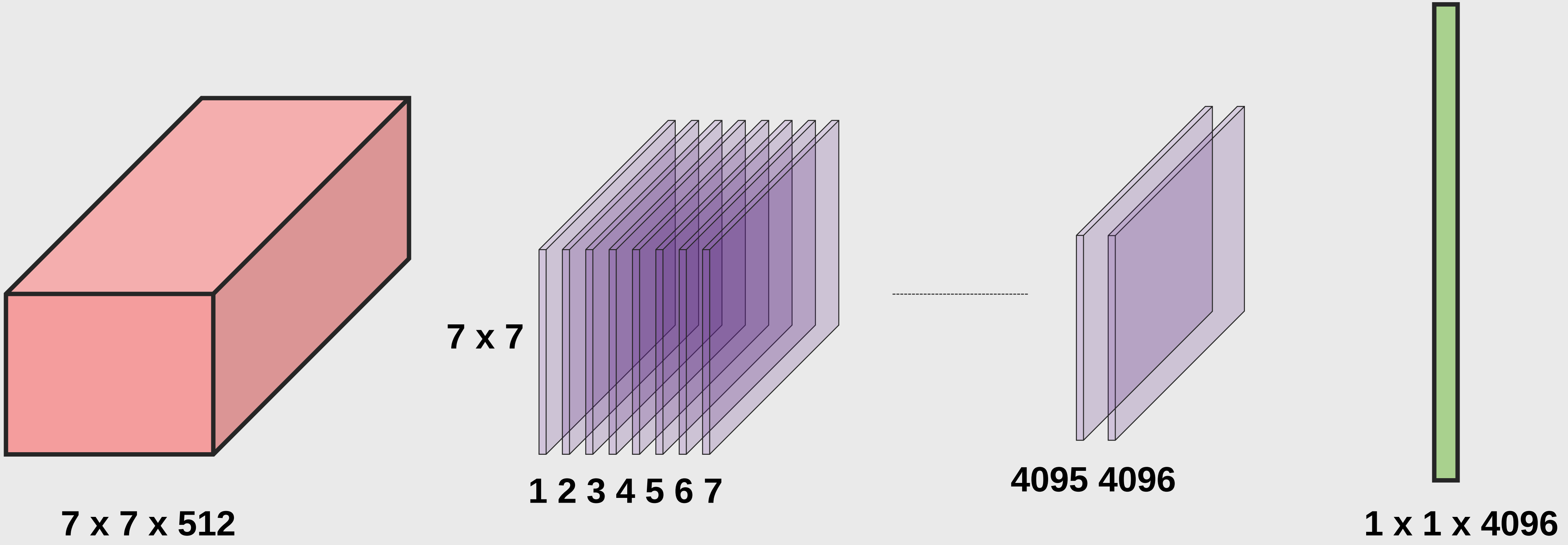
Dense Layer ⟺ Convolution Layer



A very good source: http://cs231n.github.io/convolutional-networks/

ConvNet

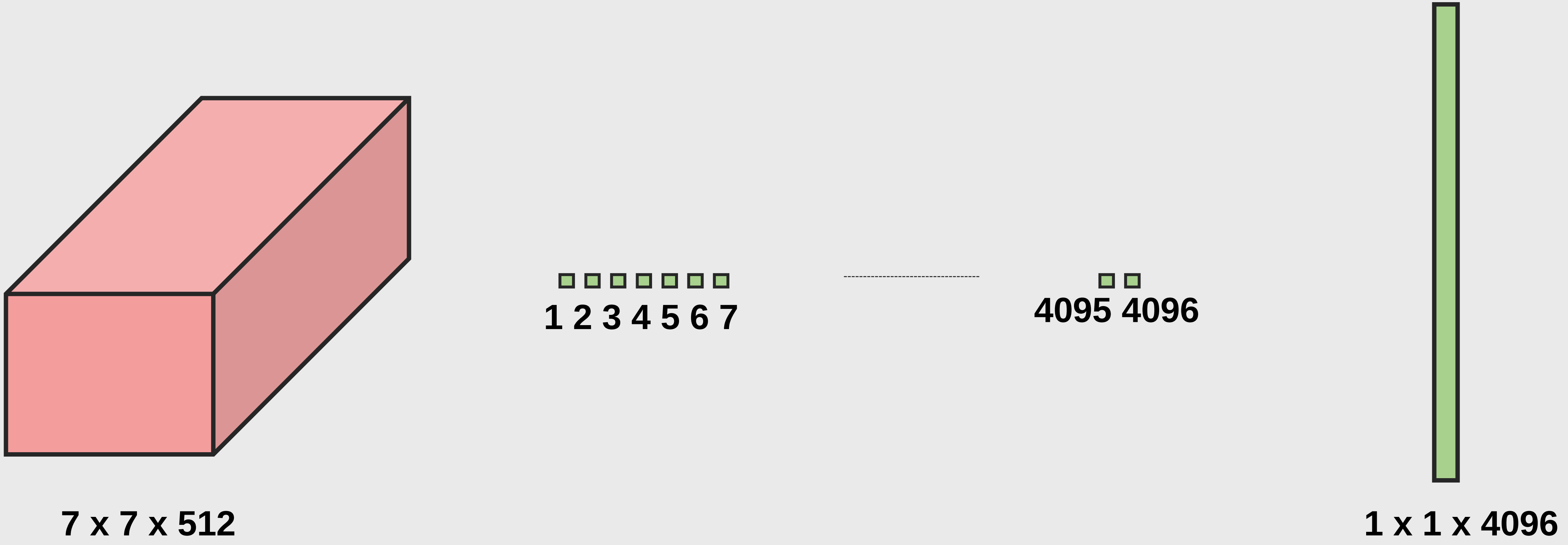
Dense Layer ⟺ Convolution Layer



A very good source: http://cs231n.github.io/convolutional-networks/

ConvNet

Dense Layer ⟺ Convolution Layer

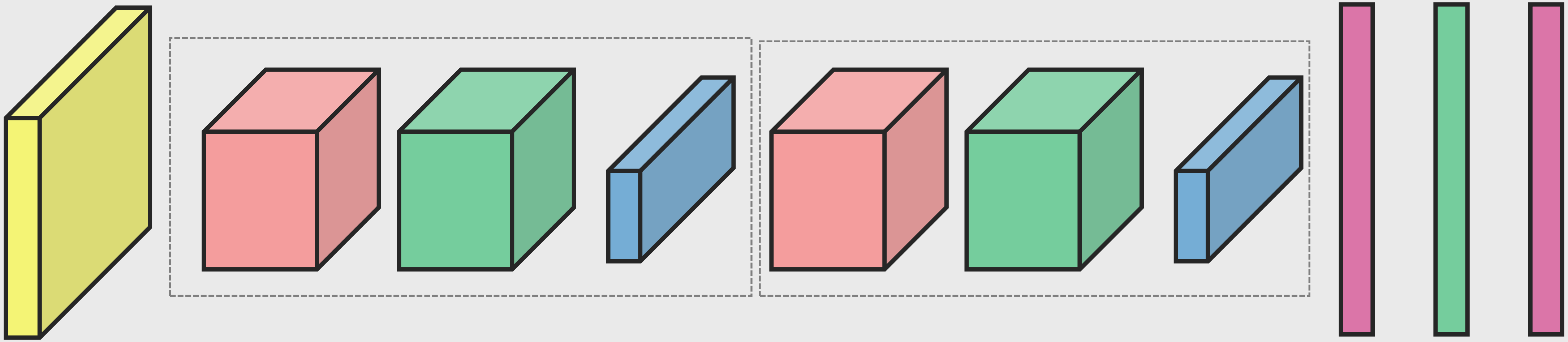


A very good source: http://cs231n.github.io/convolutional-networks/

ConvNet

ConvNet Architecture

$$\mathbf{INPUT} \longrightarrow [\mathbf{CONV} \longrightarrow \mathbf{RELU} \; -> \; \mathbf{POOL}] * \mathbf{2} \longrightarrow \mathbf{FC} \longrightarrow \mathbf{RELU} \longrightarrow \mathbf{FC}$$



A very good source: http://cs231n.github.io/convolutional-networks/

Deep Learning, Yoshua Bengio, Ian Goodfellow, Aaron Courville, MIT Press