

# Build your first AI game!

powered by:

**IEEE AUTH SB**

Παρασκευή 19/4 : 16:00- 19:00

Αμφιθέατρο 1 ΚΕ.Δ.Ε.Α

Κυριακή 21/4 : 12:30 - 15:30

Αμφιθέατρο 3 ΚΕ.Δ.Ε.Α

Workshop

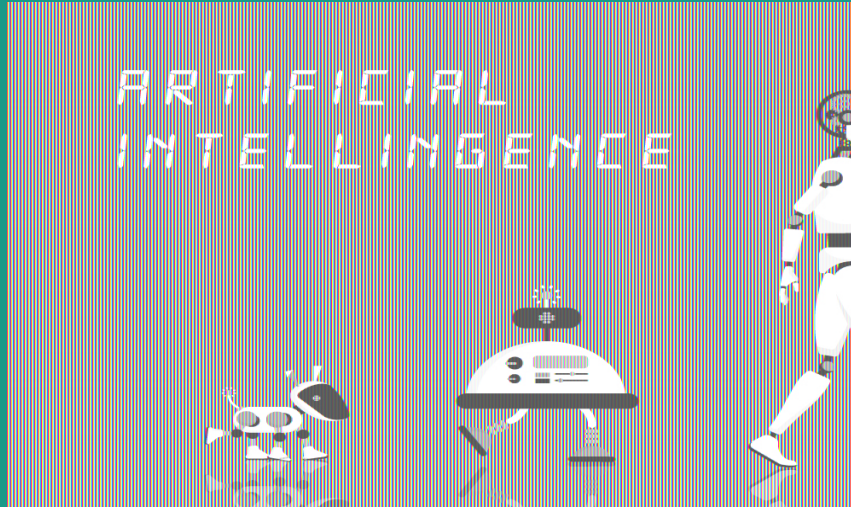


ARISTOTLE UNIVERSITY  
OF THESSALONIKI  
**IEEE**  
STUDENT BRANCH

συνμμ11

---

# Artificial Intelligence





# Artificial Intelligence

- The science engineering of making intelligent machines, especially intelligent computer programs.

- John McCarthy, father of AI

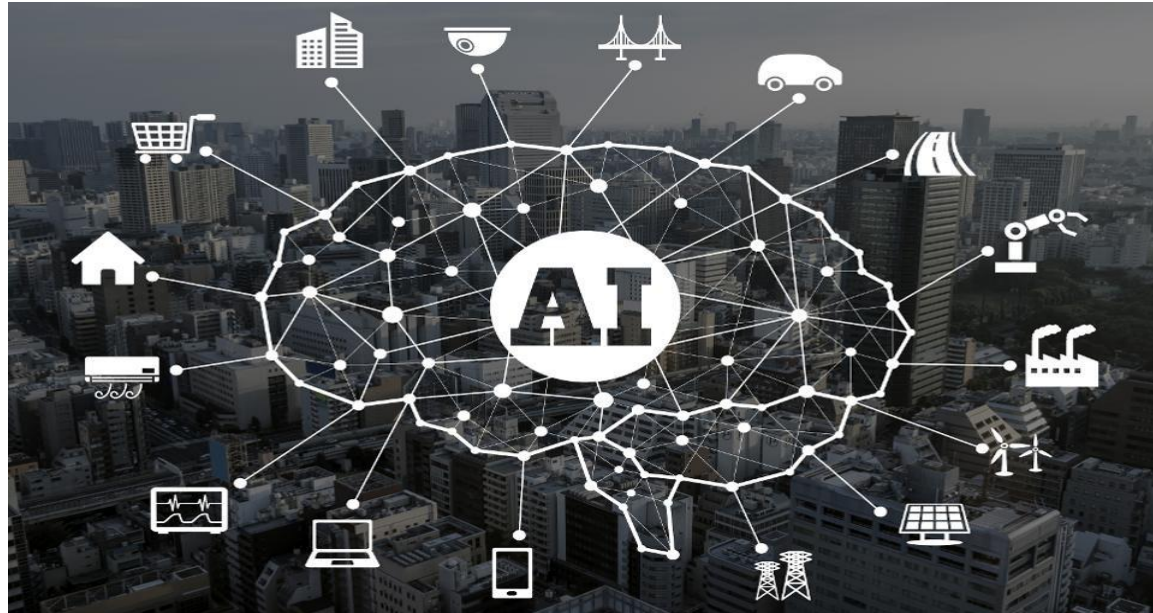
- The development of AI started with the intention of creating similar intelligence in machines that we find and regard high in humans.



# Artificial Intelligence - Human Process

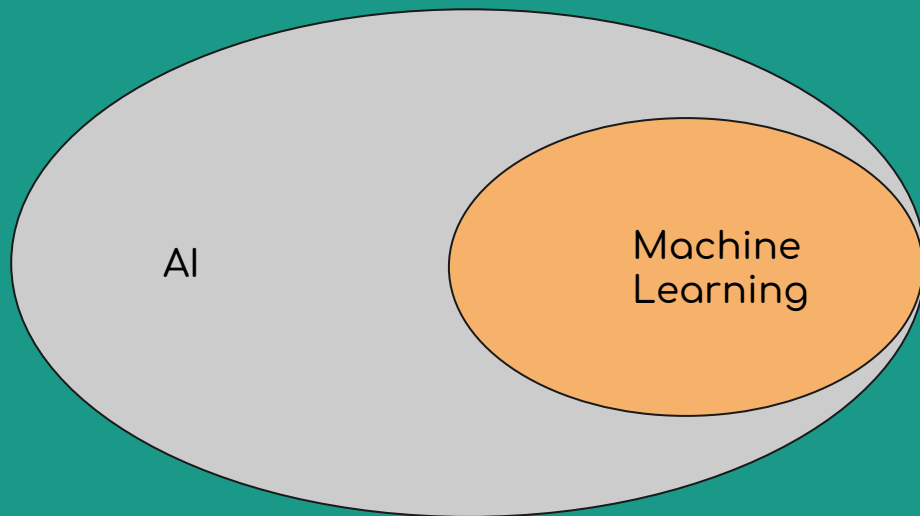
- Learns from experience.
- Uses the learning to reason.
- Recognizes images.
- Solves complex problems.
- Understands language and its nuances.
- Creates perspectives.

# AI Applications in real life projects



---

# Machine Learning





# Machine Learning

- The ability to learn without being explicitly programmed.
- The main elements are statistical analysis and predictive analysis used to spot patterns and find hidden insights based on observed data from previous computations without being programmed on where to look.



# ML Categories

- Supervised Learning.
- Unsupervised Learning.
- Reinforcement Learning.





# Supervised Learning

- The group of algorithms that require dataset which consists of example input-output pairs. Each pair consists of data sample used to make prediction and expected outcome called label.
- Word “supervised” comes from a fact that labels need to be assigned to data by the human supervisor.
- Then they are used for making predictions on unknown data, that was not a part of training dataset.



# Supervised Learning

- **Classification**

Process of assigning category to input data sample. Example usages: predicting whether a person is ill or not, detecting fraudulent transactions, face classifier.

- **Regression**

Process of predicting a continuous, numerical value for input data sample. Example usages: assessing the house price, forecasting grocery store food demand, temperature forecasting.



# Unsupervised Learning

- Group of algorithms that try to draw inferences from non-labeled data (without reference to known or labeled outcomes).
- There are no correct answers.
- Models based on this type of algorithms can be used for discovering unknown data patterns and data structure itself.



# Unsupervised Learning

- **Clustering**

Process of dividing and grouping similar data samples together. Example usages: segmentation of supermarkets or customers, data visualisation.

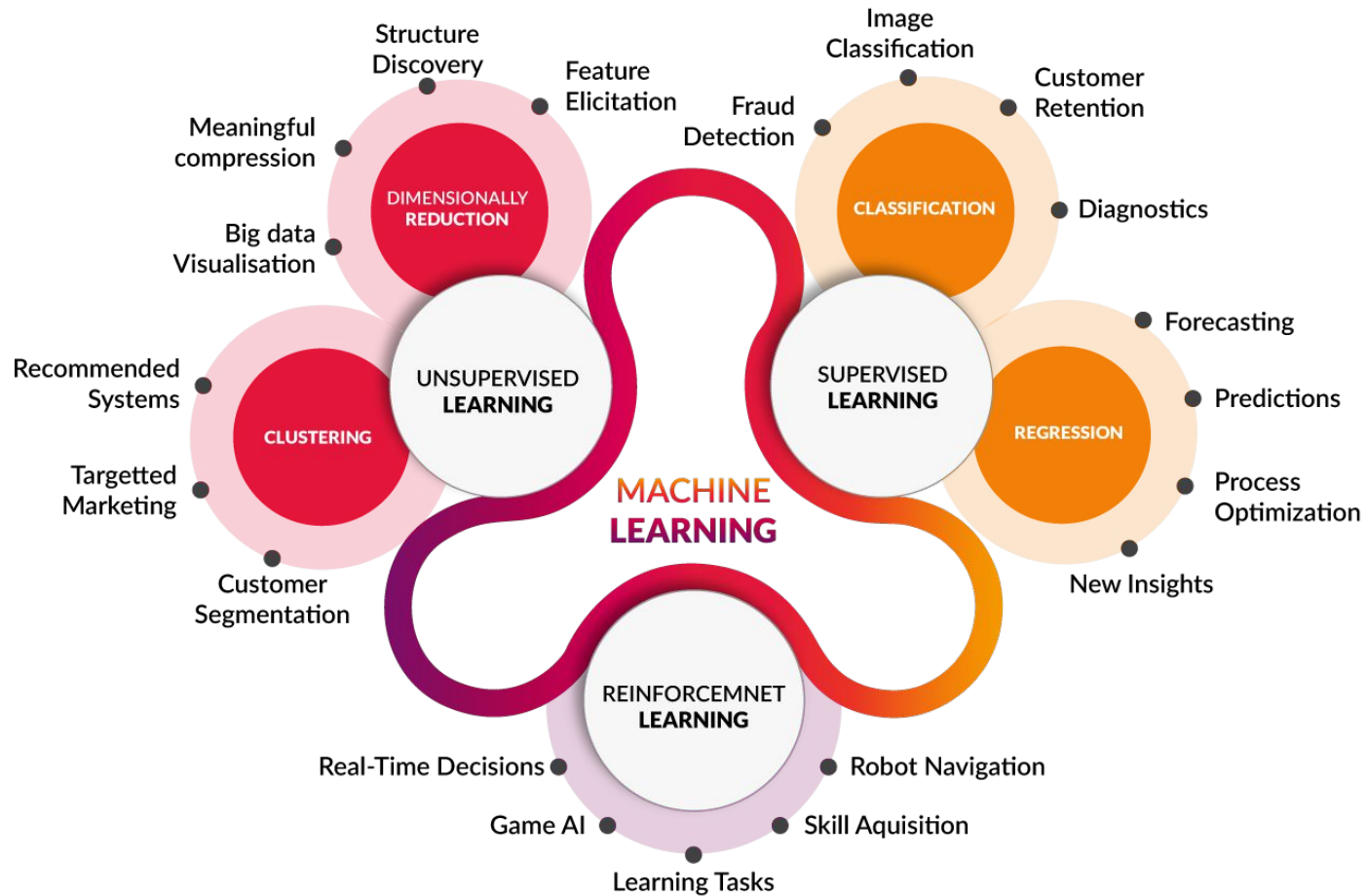
- **Dimensionality Reduction**

A process of compressing features into so-called principal values which conveys similar information concisely. Example usages: speeding up other Machine Learning algorithms by reducing numbers of calculations, finding a group of most reliable features in data.

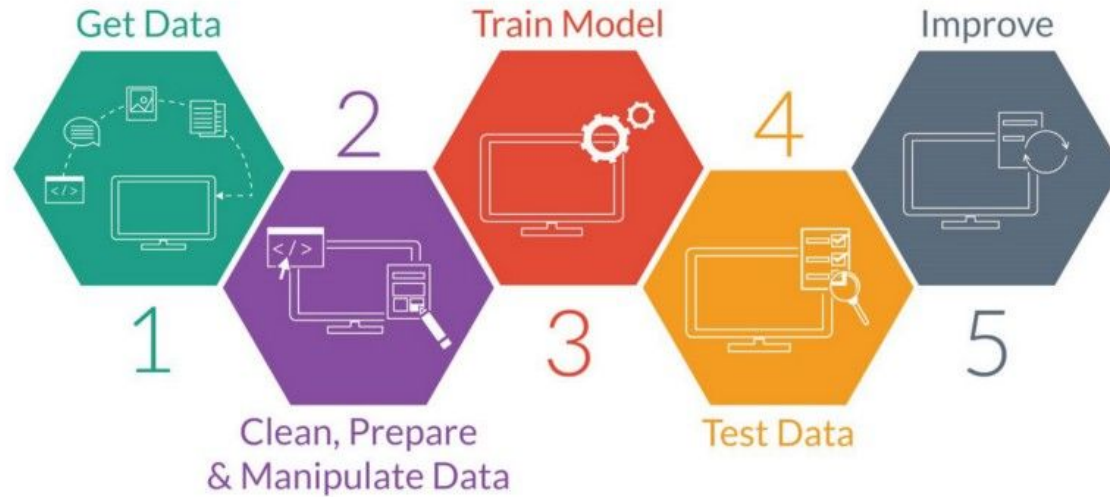


# Reinforcement Learning

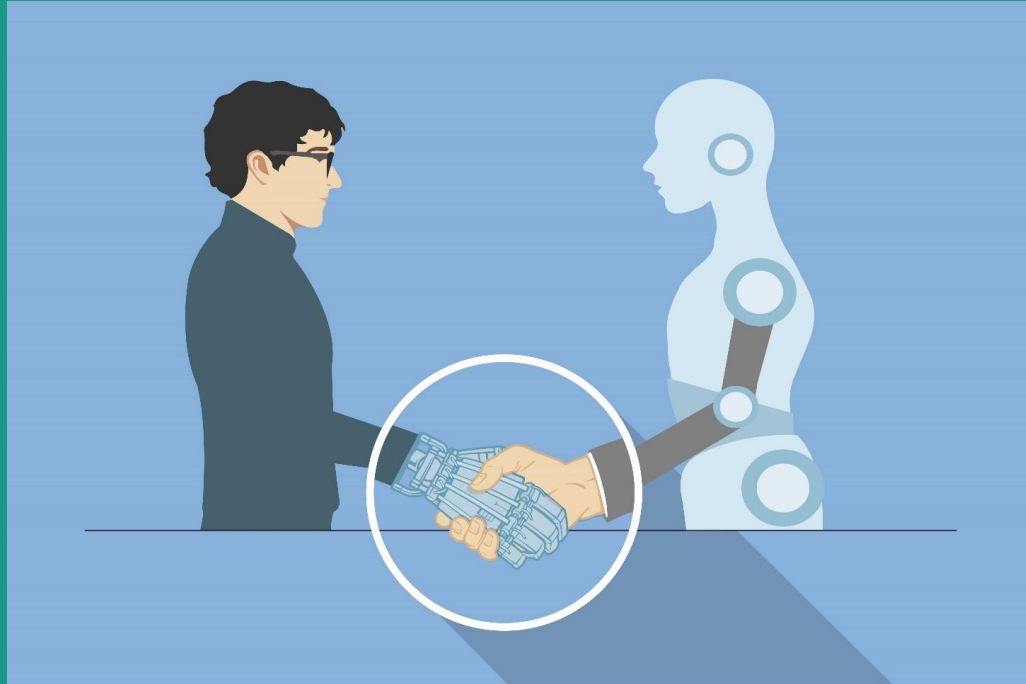
- Produces so-called agents.
- The agent role is to receive information from the environment and react to it by performing an action. The information is fed to an agent in form of numerical data, called state, which is stored and then used for choosing right action.
- As a result, an agent receives a reward that can be either positive or negative. The reward is a feedback that can be used by an agent to update its parameters.
- Trial and error training process.



# The Process



# Thank you!





# Let's play

---

---

# Mountain Car





# Απαιτούμενα

- Python
- Numpy (βιβλιοθήκη)
- Gym (βιβλιοθήκη)
- Matplotlib (βιβλιοθήκη)

# Βασική σύνταξη σε Python

## Μεταβλητές

- `counter = 100` # integer
- `miles = 1000.0` # float
- `name = "John"` # string
- `status = True` # Boolean

#Δεν χρειάζεται να δηλώσουμε τον τύπο

```
a, b, c = 1, 2, 3      >> a=1; b=2; c=3
# ανάθεση πολλών τιμών
```

## Λίστες

- `squares = [1, 4, 9, 16, 25]` # Λίστα
- `empty = []` # Κενή λίστα
- `['a', 'b', 'c'], [1, 2, 3]` #εμφωλευμένη λίστα
- `squares.append(121)` # Μέθοδος που προσθέτει ένα στοιχείο στο τέλος της λίστας
- `len(squares)` # Συνάρτηση που επιστρέφει το μήκος της λίστας

# For, While, If...

```
for x in range(6):
```

```
    print(x)
```

```
>>>0 1 2 3 4 5
```

# range(6) επιστρέφει μια  
λίστα από το 0 έως το 5

# η μεταβλητή x παίρνει  
διαδοχικά αυτές τις τιμές

```
while (count < 9):  
    print( 'The count is:' + count)  
    count = count + 1
```

```
>>>The count is: 0  
    The count is: 1  
    The count is: 2  
    The count is: 3
```

# οι εντολές στην while εκτελούνται  
όσο είναι αληθής η συνθήκη στην  
παρένθεση

```
x = 30  
if x < 0:                                # false  
    x = 0  
    print('changed to zero')  
elif x == 0:                             # false  
    print('Zero')  
elif x == 1:                             # false  
    print('Single')  
else:                                     # true  
    print('More')
```

```
>> More
```

**ΠΡΟΣΟΧΗ!! ΣΤΗΝ ΣΤΟΙΧΙΣΗ ΤΟΥ ΚΩΔΙΚΑ**



# Input/Output

```
x = input('Enter your name:')  
print('Hello, ' + x)  
>>>Hello, person
```

```
x = 3  
print('The number is {}'.format(x))  
>>>The number is 3
```

\*\*\* Ο φάκελος που θα σας κάνουμε share περιέχει ppt αρχείο με αναλυτική περιγραφή των βασικών εντολών και μεθόδων για σωστή συγγραφή σε python

---

# Μια εισαγωγή στη Numpy

Είναι μια βιβλιοθήκη για Python η οποία υποστηρίζει εισαγωγή και διαχείριση **πινάκων**, καθώς και ένα εύρος από συναρτήσεις για πράξεις μεταξύ αυτών.

Εμείς θα χρησιμοποιήσουμε **συναρτήσεις** που φέρνουν τα δεδομένα μας σε επιθυμητή μορφή ώστε να τα επεξεργαστούμε πιο εύκολα.



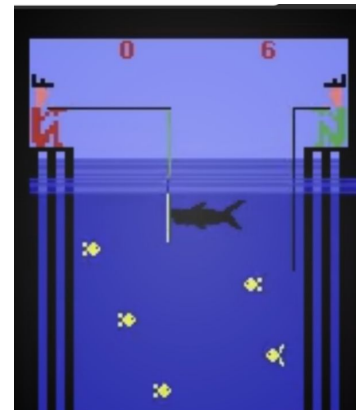
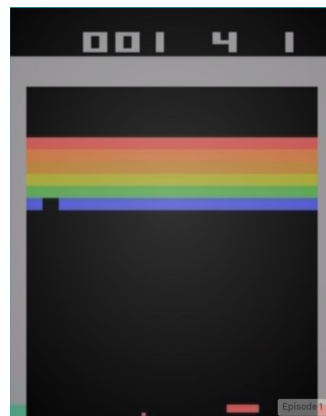
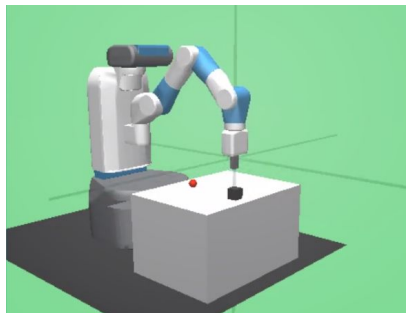
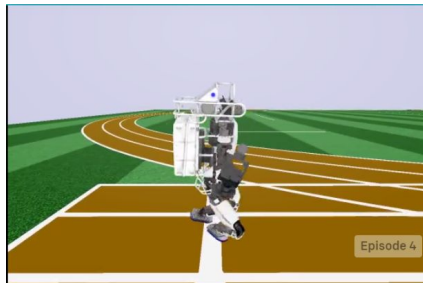
# Χρήσιμες Εντολές Numpy

- `import numpy as np` Εισαγωγή της βιβλιοθήκης στο πρόγραμμά μας
- `np.array(my_list)` Δημιουργία πίνακα από λίστα
- `np.round(array, n).astype(int)` Στρογγυλοποίηση πίνακα στο n-οστό δεκαδικό ψηφίο και αποθήκευση ως ακέραιο
- `np.random.uniform(low, high, size)` Δημιουργία τυχαίου πίνακα με διαστάσεις `size=(a, b, ...)` τα στοιχεία του οποίου ακολουθούν ομοιόμορφη κατανομή στο διάστημα `[low, high)`
- `np.random.randint(low, high)` Δημιουργία τυχαίου **ακέραιου** αριθμού στο διάστημα `[low, high)`
- `np.random.random()` Δημιουργία τυχαίου αριθμού στο διάστημα `[0, 1)`
- `np.argmax(array)` Εύρεση δείκτη του μεγαλύτερου στοιχείου του πίνακα
- `np.mean(array)` Εύρεση της αριθμητικής μέσης τιμής του πίνακα





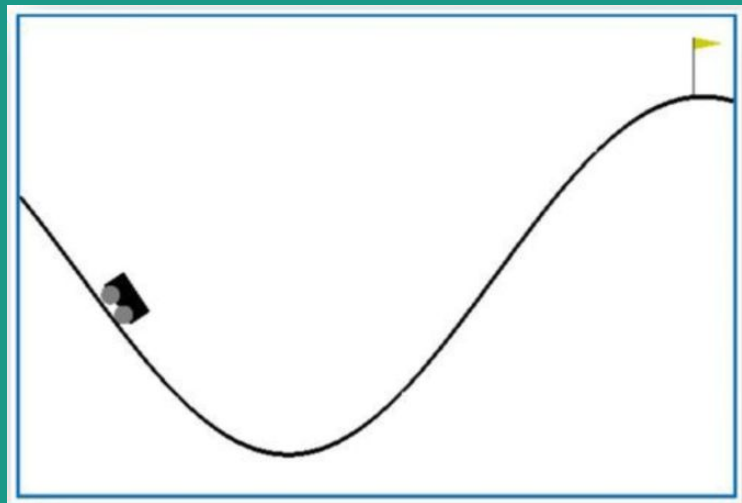
# OpenAI Gym



---

# Περιβάλλον Mountain Car

- `env = gym.make('MountainCar-v0')`
- `env.reset()`
- `env.render()`
- `env.close()`





## Μεταβλητές Κατάστασης

Index	Observation	Min	Max
0	position	-0.6	1.2
1	velocity	-0.07	0.07

`env.observation_space.high`

`>> [1.2, 0.07]`

`env.observation_space.low`

`>> [-0.6, -0.07]`

`state_array = [position, speed]`

π. χ. `[-0.50837305, 0.00089253]`



## Μεταβλητή Δράσης

Num	Action
0	Push left
1	No push
2	Push right

### **env.step(action)**

Επιστρέφει τις τιμές (με τη συγκεκριμένη σειρά):

```
state_array, reward, done, info =  
env.step(action)
```

### **env.action\_space.n**

Επιστρέφει τον αριθμό των actions

Παρατήρηση: η `env.reset()` επιστρέφει κι αυτή το `state_array` (αλλά μόνο αυτό)  
Δηλαδή γράφουμε: `state_array = env.reset()`

---

# Q-Learning

- Reinforcement learning: μαθαίνει καλές τακτικές ελέγχου με ανάλυση των state, action, rewards μέσω των simulation.
- Q-Learning: Αλγόριθμος του reinforcement learning, που καθορίζει το action του agent σε κάθε state με βάση τα reward που προκύπτουν.
  - Agent: Το αντικείμενο που πράττει (πχ ρομπोट)
  - State: Η κατάσταση του agent (πχ θέση, ταχύτητα)
  - Action: Πιθανές κινήσεις (πχ δεξιά, αριστερά)
  - Reward: Μέτρο αξιολόγησης του κάθε action

# Q-Table

- Πίνακας που υπολογίζει τη μέγιστη αναμενόμενη τιμή των rewards για το action σε κάθε state
- Στήλες: actions
- Γραμμές: states
- Οι τιμές αρχικοποιούνται τυχαία και στη συνέχεια υπολογίζονται μέσω του αλγορίθμου Q-Learning:
  - $Q(s, a) = \text{reward} + \text{discount} * \max(Q(s', a'))$

	1	2	3	4	5	6	7	8	9	10
Tiles	Hole						Beer			

	Action	
State	Left	Right
0	0	0
1	-100	65.61
2	59.049	72.9
3	65.61	81
4	72.9	90
5	81	100
6	0	0
7	100	81
8	90	72.9
9	81	0



# Q-Value

$$Q'(s1, s2, a) = (1-w)*Q(s1, s2, a) + w*(r+d*Q(s1', s2', \operatorname{argmax}_{a'} Q(s1', s2', a')))$$

- w: learning rate
- r: reward
- d: discount rate  $\in (0,1)$
- argmax: δείκτης a της μέγιστης q-value για δεδομένη κατάσταση

!!! Για τις τερματικές καταστάσεις:  $Q(s1, s2, a) = r$   
(στο Mountain Car τερματική κατάσταση έχουμε όταν  $s1 \geq 0.5$ )




# Epsilon greedy strategy

Σε οποιαδήποτε κατάσταση διάλεξε:

- Την δράση που μεγιστοποιεί την  $Q(s_1, s_2, a)$  (την “άπληστη” ενέργεια)
  - Με πιθανότητα  $P(!\text{random}) = 1 - \text{epsilon}$
- Μια τυχαία δράση
  - Με πιθανότητα  $P(\text{random}) = \text{epsilon}$

$\text{epsilon} \in (0, 1)$





# Αλγόριθμος Q-Learning

Q-Learning( $s, a, d, w$ )

## Inputs

$s, a, d, w$

## Local

real array  $Q[s, a]$

previous state  $s$

previous action  $a$

initialize  $Q[s, a]$  arbitrarily

observe current state  $s$

## repeat

Select and carry out an action  $a$

Observe reward  $r$  and action  $a$

$Q'(s, a) = (1 - w) * Q(s, a) + w * (r + d * Q(s, \arg\max_{a'} Q(s, a')))$

$s \leftarrow s'$  **until** termination

# Let's code!

Python ✓

Numpy ✓

Gym ✓

Matplotlib ✓

---



## Step 0

Ζητούμενα:

- Import `numpy as np`, `gym`, `matplotlib.pyplot as plt`
- Δημιουργήστε το περιβάλλον `'MountainCar-v0'`



# Step 1

Ζητούμενα:

- Ορίστε τις μεταβλητές: ( **learning** , **discount** , **epsilon** , **min\_eps** , **episodes** ) με τις αντίστοιχες τιμές (0.2, 0.9, 0.8, 0, 5000)
- Βρείτε τον αριθμό των διακριτών καταστάσεων του περιβάλλοντος , **num\_states**

Tips:

- Για την διακριτοποίηση των state , βρείτε πρώτα το εύρος των τιμών και έπειτα πολλαπλασιάστε και στρογγυλοποιήστε κατάλληλα ώστε να βγει μια ακέραια τιμη. πχ 10-20



## Step 2

Ζητούμενα:

- Αρχικοποιήστε (κενές) δύο λίστες που θα αποθηκεύουν τα rewards (`reward_list`, `ave_reward_list`)
- Αρχικοποιήστε το Q-Table, `Q`, με τυχαίες τιμές από -1 έως 1 και για διαστάσεις ορίστε των αριθμό των διακριτών state και του action



## Step 3

Ζητούμενα:

- Αρχικοποιήστε τις μεταβλητές `done`, `state`, `reward`, `tot_reward`
- Διακριτοποιήστε το αρχικό state -> `state_adj` ώστε να πάρει μια από τις τιμές που βρήκατε στο step 1

Tips:

- Για την διακριτοποίηση των `state_adj`, κάντε το ίδιο με το step 1 αλλά αντί για το μέγιστο state τοποθετήστε το τωρινό.



## Step 4

### Ζητούμενα:

Τρέξτε το παιχνίδι για ένα επεισόδιο δημιουργώντας ένα loop που θα τερματίζει όταν λήγει και το παιχνίδι. Συγκεκριμένα:

- Κάντε render το περιβάλλον
- Καθορίστε το επόμενο **action** με βάση το epsilon greedy strategy
- Πάρτε την επόμενη κατάσταση και το reward
- Διακριτοποιήστε και στρογγυλοποιήστε το επόμενο state , όπως στο step 3



## Step 4

- Ορίστε τη συνθήκη που αποφασίζει πότε κερδίζει το παιχνίδι
  - Όταν κερδίζει ανανεώνει την τιμή του Q-Table με τη τιμή του reward
  - Όταν δεν κερδίζει ορίζει την τιμή του Q-Table με τον τύπο της Q-function
- Ανανεώστε το state και την μεταβλητή που αποθηκεύει τα συνολικά rewards
- Κλείστε το περιβάλλον μετά το loop

Προσοχή στην στοίχιση !!





## Step 5

Ζητούμενα:

- Υπολογίστε την μείωση του epsilon ,(reduction), ώστε να γίνεται min\_eps στα 1000 επεισοδια
- Δημιουργήστε μια for που θα τρέχει για όσα επεισόδια ορίστηκαν στην αρχή του κώδικα και βάλτε μέσα των κώδικα από το STEP 3 και το STEP 4
- Εκτός της while αλλά μέσα στη for μειώστε το epsilon με κατάλληλη συνθήκη
- Προσθέστε τα rewards σε λίστα και υπολογίστε τον μέσο όρο ανα 100 επεισόδια

Tips:

- Render κάθε 200 επεισόδια και κλείσιμο περιβάλλοντος μετα την for



## Step 6

Ζητούμενα:

- Κάντε plot τα rewards
  - Άξονας x: επεισόδια
  - Άξονας y: μέσο reward

Tips:

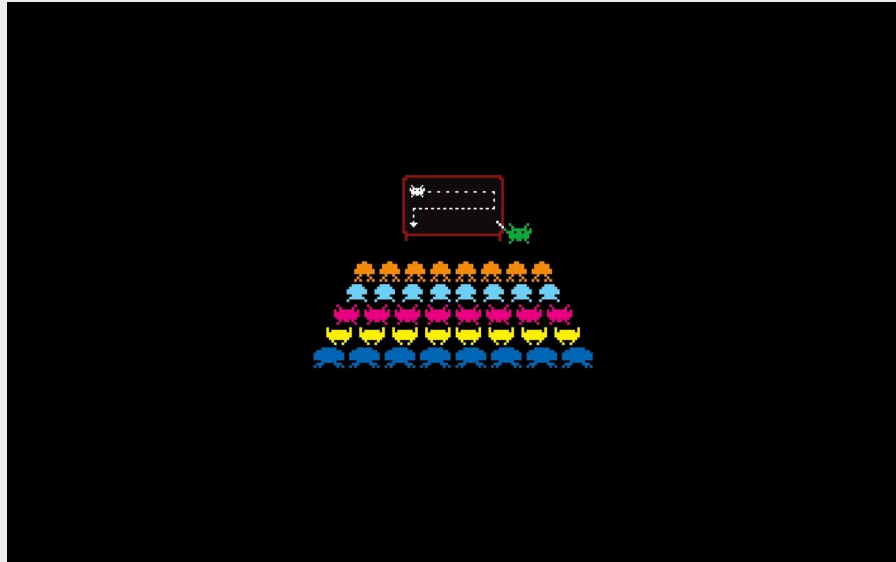
- Χρήσιμες εντολές: `plt.plot(x,y)`, `plt.xlabel()`, `plt.ylabel()`, `plt.title()`, `plt.show()`
- `x,y` είναι σημεία , δηλαδή arrays που περιέχουν τα αντίστοιχα x και y
- `numpy.arange([start, stop, step])` επιστρέφει τιμές,σε array, με ίση απόσταση μεταξύ τους

---

**Thank you!**



# Space Invaders



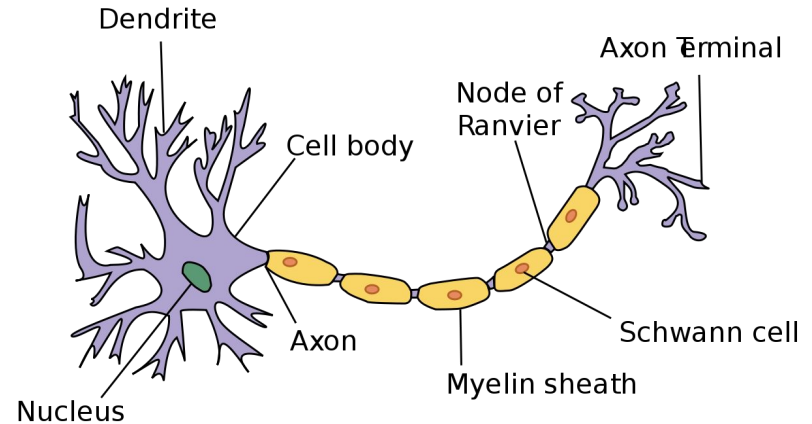
---

# Neural Networks

# From biology...

A neuron consists of:

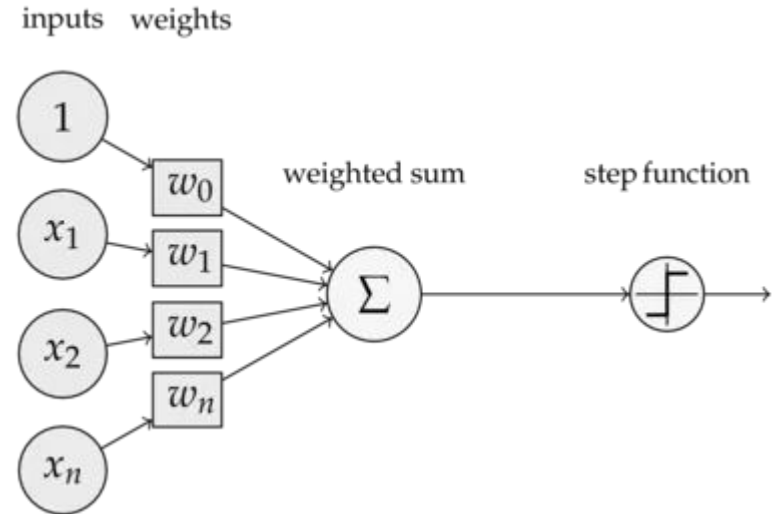
- **Dendrites:** input from other neurons
- **Body:** Makes decisions
- **Axon:** Connects body with the terminals
- **Axon terminals:** Connects axon with terminal nodes



## ... to Computer Science

Perceptron (simplest binary classifier)

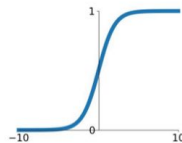
- **Input values  $x$ :** data & bias
- **Weights  $w$ :** To be calculated.
- **Activation Function**
- **Output value  $y$ :** in range  $[0,1]$ .



# Activation Functions

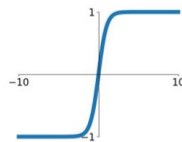
**Sigmoid**

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



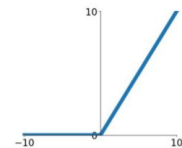
**tanh**

$$\tanh(x)$$



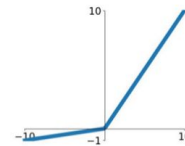
**ReLU**

$$\max(0, x)$$



**Leaky ReLU**

$$\max(0.1x, x)$$

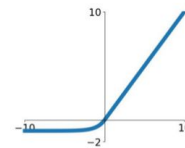


**Maxout**

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

**ELU**

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

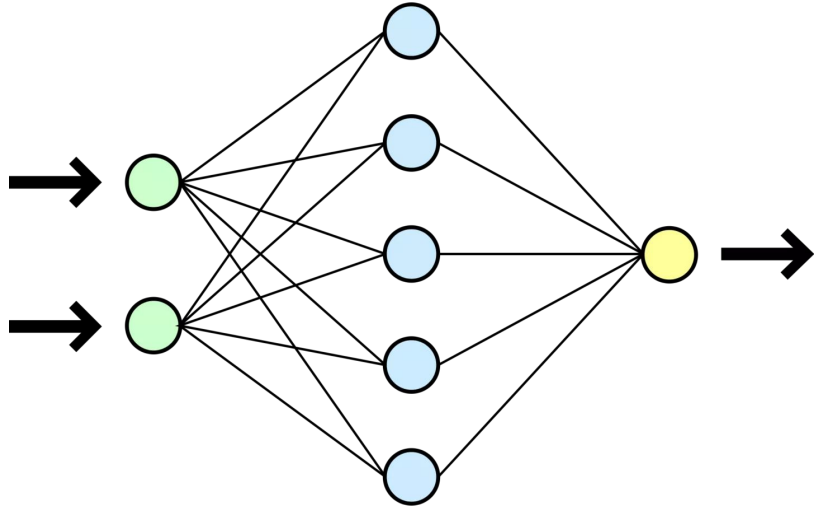




# A Simple Network

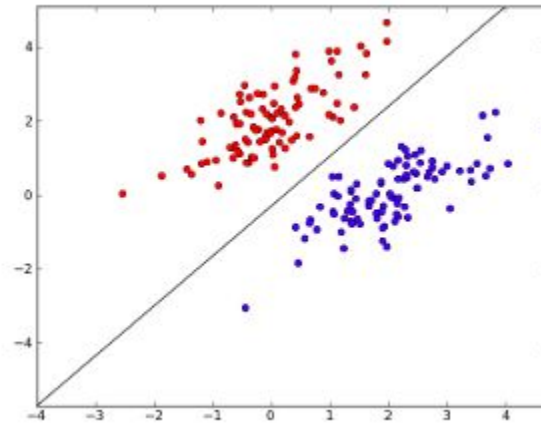
3 Layer Network

- Input layer
- Hidden(s) layer(s)
- Output layer





# Output



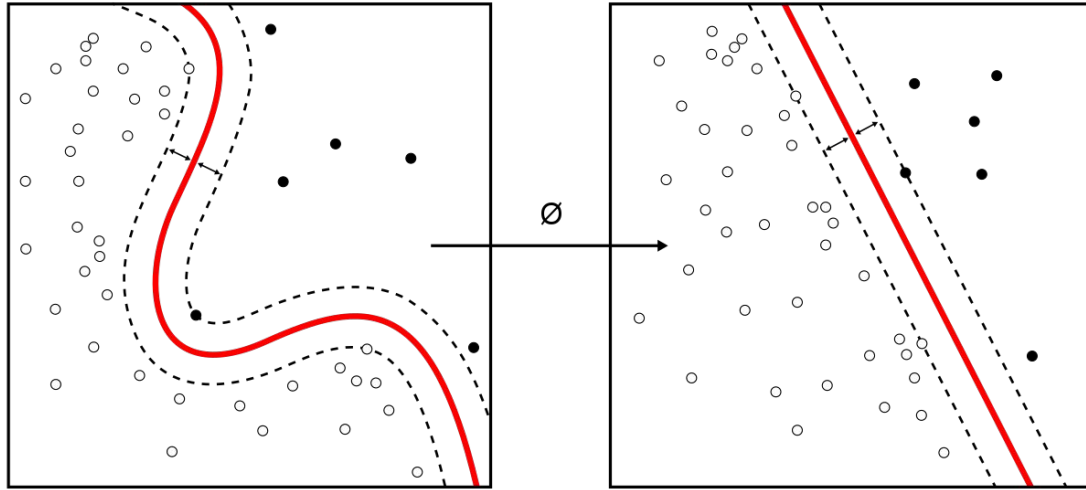


**But what happens in non linear cases?**

---

# Deep Learning

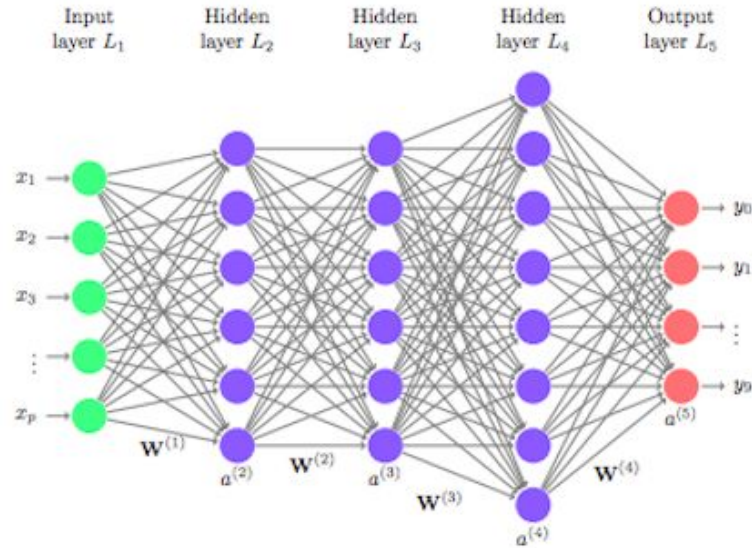
# Turns a non linear to a linear problem





# How?

# Add more hidden layers



---

Intro to



Keras





# Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation.

*Being able to go from idea to result with the least possible delay is key to doing good research.*

---

# Basic Methods



# Keras model

```
from keras.models import Sequential
model = Sequential()      # construct a linear stack of layers
model.add(Dense())        # stack (add) a layer
model.compile(optimizer, ...) # configure/create model
model.summary()           # prints a summary representation of model

model.fit(x_train, y_train, epochs)      # train model
model.predict(x_test)                    # predict values
model.evaluate(x_test, y_test)           # evaluate model
```



# Keras Layers

```
from keras.layers import Dense, Flatten, Dropout, Conv2D
```

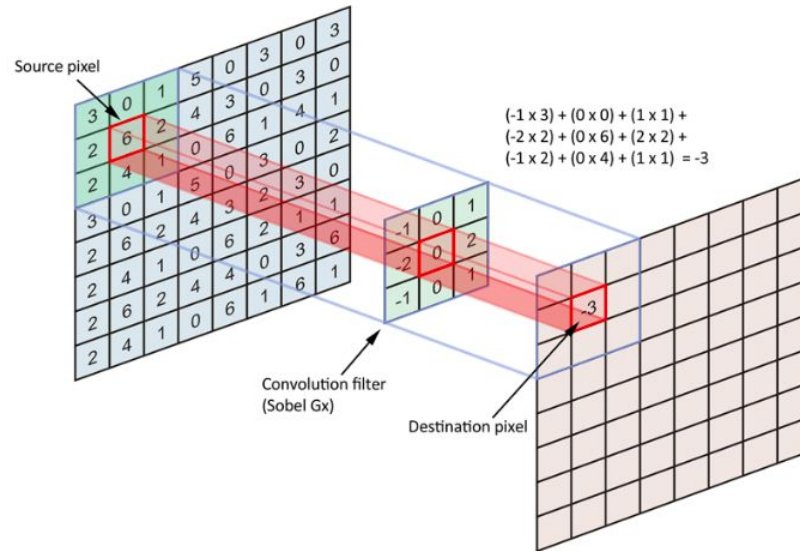
```
Dense(units, [activation], [input_shape])    # Normal layer
```

```
Flatten()                                     # Flattens the input
```

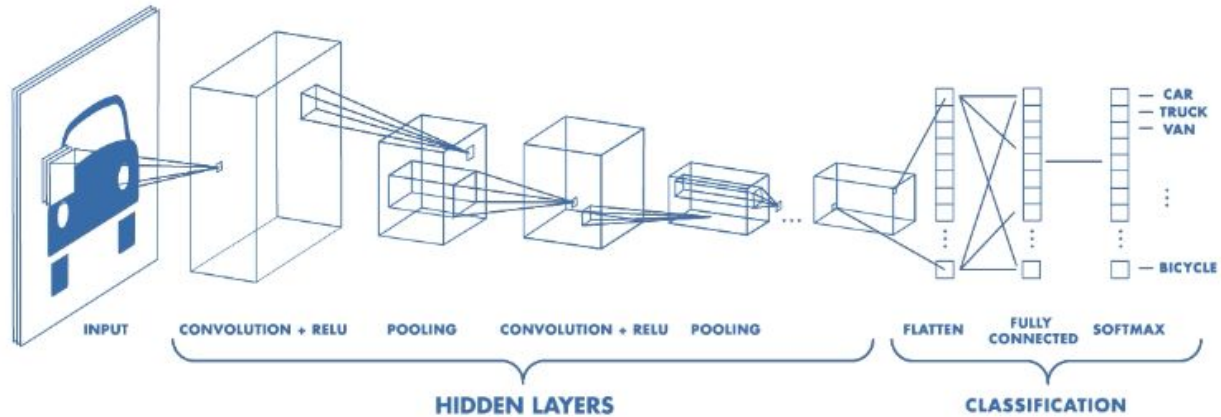
```
Dropout(rate)                                # Applies dropout to the input
```

```
Conv2D(filters, kernel_size, strides, data_format, activation)    # 2D Convolution layer
```

# Convolution Layer



# A Complete Network



# Let's write our model

---



## Step 1: Model

```
model = Sequential()  
model.add(Conv2D(32, (8, 8), strides=(4, 4), input_shape=(84, 84, 1), activation='relu'))  
model.add(Conv2D(64, (4, 4), strides=(2, 2), activation='relu'))  
model.add(Conv2D(64, (3, 3), activation='relu'))  
model.add(Flatten())  
model.add(Dense((512), activation='relu'))  
model.add(Dense(actions))  
model.compile(loss='mse', optimizer=Adam())  
model.summary()
```



---

# Preprocess

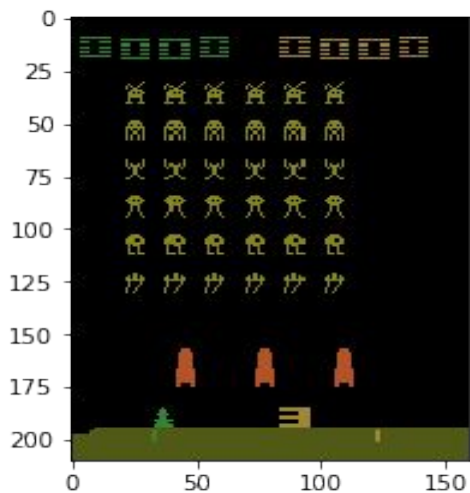


## Step 2: Preprocess

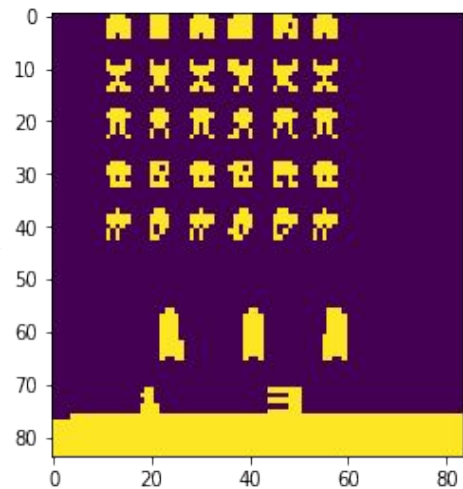
Import cv2

```
def preprocess(observation):  
    observation = cv2.cvtColor(cv2.resize(observation, (84, 110)), cv2.COLOR_BGR2GRAY)  
    observation = observation[26:110,:]  
    ret, observation = cv2.threshold(observation, 1, 255, cv2.THRESH_BINARY)  
    return np.reshape(observation, (1, 84, 84, 1))
```

# Preprocess



Preprocess



---

# Set the environment



## Step 3: Set the environment

```
env = gym.make('SpaceInvaders-v0')  
# Check game's specs  
actions = env.action_space.n  
print(actions)  
print(env.observation_space.shape)
```



## Step 4: Train model

Play the game many times

At each game:

- Play
- Predict best actions
- Collect data (state, action, reward, next\_state, done)
- Train model

# Questions?

---



# Useful Links

- Space invaders game: <https://keon.io/deep-q-learning/>
- Neural Networks:  
<https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/>
- Q learning: <http://uhaweb.hartford.edu/compsci/ccli/projects/QLearning.pdf>  
<http://mnemstudio.org/path-finding-q-learning-tutorial.htm>  
<https://medium.freecodecamp.org/an-introduction-to-q-learning-reinforcement-learning-14ac0b4493cc>
- Python: <https://docs.python.org/3/tutorial/>
- Mountain car game:  
<https://towardsdatascience.com/getting-started-with-reinforcement-learning-and-open-ai-gym-c289aca874f>





Thank you