Introduction to artificial intelligence

Summary

Artificial intelligence is experiencing a massive boom in interest and investment. The FAANG companies (Facebook, Amazon, Apple, Netflix, Google) along with others such as Microsoft, Uber, Airbnb, Snap, universities, militaries and so forth are spending billions of dollars on the field. Get a tase of what all the fuss is about with this introduction to the ideas of AI.

Unit information

MYP item	This unit
Key concept	tba
Related concepts	tba
Global context	Scientific and technical innovation
Statement of inquiry	tba
Assessment objectives	A (inquiring & analysing), D (evaluating)

Lesson overviews

12 lessons as follows:

1. Install tools, understand basic ideas of AI
2. The CartPole problem - with a random agent
3. The CartPole problem - with simple logic
4. Intro to ML theory
5. The CartPole problem - with a neural network
6. The Mountain Car problem - converting continuous to discrete actions
7. Simple game - complete demo
8. Simple game - training
9. Simple game - tweak and retrain
10. Analysis

Website

The website for this unit is [https://pbaumgarten.com/myp-design/game-making/]

0. Prequisites

Python installed

This assumes you have a recent version of Python installed, typically at least version 3.6

If you don't have it, go to https://wwww.python.org and download it.

When running the installer, make sure you turn on the option to "Add Python to PATH"

I have a video walk through of the process for installing Python and VS Code at

- https://pbaumgarten.com/python/install/, or
- https://youtu.be/-R6HFLp7tTs

Libraries required

Once you have Python installed, open the command prompt and run the following

```
pip install Pillow ImageToolsMadeEasy
```

If you get a permissions error with the above, try it again with the `--user` switch as follows

```
pip install --user gym numpy
```

Basic Python knowledge

This guide assumes a basic familiarity with Python. I have written a quick recap designed for a one hour lesson should you need it. It is available at:

- https://pbaumgarten.com/python/recap/

If you need a more detailed introduction to Python I have a set of detailed tutorials on my website. Each lesson contains detailed notes, videos and practice exercises. Each lesson is roughly an hour in length with 9 lessons in "the basics" (though only the first 5 are required for this tutorial).

- https://pbaumgarten.com/python/

1. Basics of AI

Lets start with a video...

- How Machines Learn by CGP Grey (9min) - https://www.youtube.com/watch?v=R9OHn5ZF4Uo

What is artificial intelligence?

Artificial intelligence (AI) is an area of computer science that involves the creation of machines that work and react like humans.

Some of the human activities that are commonly being programmed using artificial intelligence include:

- Computer vision: ability of computers to identify objects, scenes, and activities in images
- Natural Language Processing: ability of computers to understand meaning from text or generating text that is readable
- Speech Processing: automatically transcribing human speech or generating speech from the corresponding text

For instance,

- Google Maps uses AI to update addresses and street names based on street view pictures.
- Facebook uses AI to analyze text posted by users and better suggest ads they may need or be interested in. It also uses AI to analyse photos and identify faces.
- Amazon & Netflix uses AI in its recommendation engine.

AI is rapidly moving from the laboratory towards business and consumer application for 3 big reasons:

- Big Data: Live, interactive, automatically generated, and often self-correcting data that fuels real-time decisions and real-time responses.
- Cheaper Computing: Cloud computing, massively parallel processing and new CPUs are powering AI techniques that simply weren't practical before.
- Better Algorithms: AI techniques have existed for decades but there is a surge in innovation and performance with the rapid growth in computational infrastructure, data and sensors.

What is machine learning? How is it different to artificial intelligence?

Broadly speaking, you can consider machine learning to be a sub-set of the broader field of artificial intelligence. Where AI is simply about creating intelligent machines and programs, ML refers to systems that can learn from experience.

For example, say you provide machine learning program with lots of images of skin conditions along with the what do those conditions mean. The algorithm can mine this image data and help to analyze skin condition in the future. The algorithm examines images and identifies patterns that exist between these images that have similar conditions.

When the algorithm is given a new skin image (whose condition is unknown) in the future, it will compare the pattern that is present in the current image with the pattern it learned from analyzing all the past pictures and be able to able to predict what kind of skin condition it is.

If there are new skin conditions or if the existing patterns of the skin conditions changes, the algorithm will not predict those conditions correctly. One has to feed in all the new data to the algorithm for it to learn to predict based on new skin conditions.

On the other hand, Artificial Intelligence learns by acquiring knowledge and learning how to apply it. The aim of artificial intelligence is to increase the chances of success and to find the optimal solution. AI is the study to train the computers to attempt to do things which at present human can do better. AI tends to be used in situations where adapting to new scenarios are important.

In summary, machine learning uses the experience to look for the pattern it learned. AI uses the experience to acquire knowledge/skill and also how to apply that knowledge for new environments.

Types of learning

There are four commonly used types of learning algorithms:

- Supervised: Learn rules that map inputs to target outputs.
- Unsupervised: Learn to cluster and label similar inputs.
- Deep learning: Learn through a hierarchy of simple to complex concepts.
- Reinforced: Learn by continually interacting with an environment.

Supervised learning

Supervised learning happens when we feed the computer data and the outcome we want from that data. Then we let the computer find a correlation between the data and the outcome so that it may predict outcomes with new data without us providing the outcome.

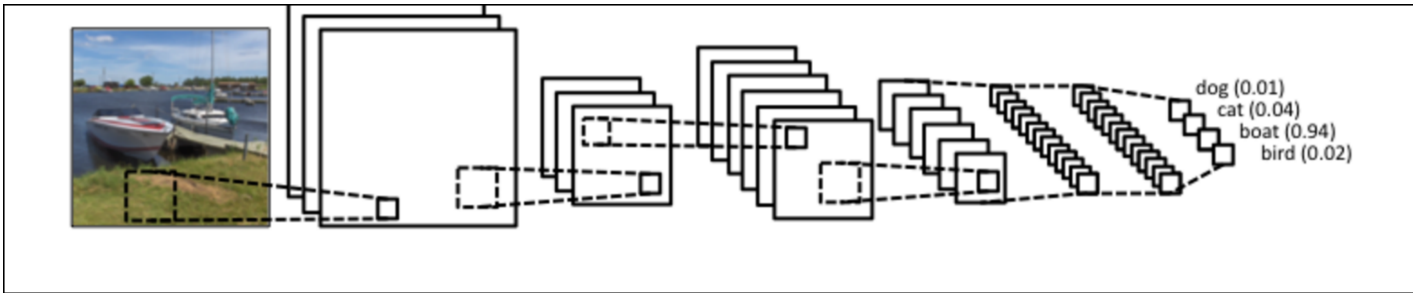
Supervised learning is the example illustrated in the CGP Grey video, "How machines learn". That is: Why feed the algorithm a stack of information, ask it to make a prediction based on the pattern that it could identify, we then tell it whether it made a correct or incorrect prediction.

The goal is to minimize the error between the model's predictions and the actual outcomes.

In a perfect world we would minimize errors on all possible inputs.

In reality, we usually don't have enough inputs with corresponding outcomes to teach the computer well enough.

Deep learning



Imagine we gave a computer millions of images of puppies and kittens. What characteristics about each might we pick to enable a computer to learn to identify future puppies and kittens? With images it's quite hard. This is where deep learning comes in.

- Deep neural networks are a subtype of supervised learning.
- Deep networks are multilayer networks on top of each other where each layer corresponds to a different level of abstraction.
- There are two main reasons behind the popularity and the high impact of deep neural networks in various fields such as computer vision and speech recognition:

    - The emergence of modern parallel computing architectures providing low-cost and fast computation for a large number of parameters of the deep networks.
    - The availability of vast amounts of images, video, speech and text on the Internet providing sufficient data for training these networks.
- Each layer in the deep architecture provides a nonlinear information processing.
- The output is a parameterized function of the inputs and the output of each layer is the input for the higher layer.

Watch this excellent introduction to neural networks with deep learning. "But what is a Neural Network? | Deep learning, chapter 1" by 3blue1brown.

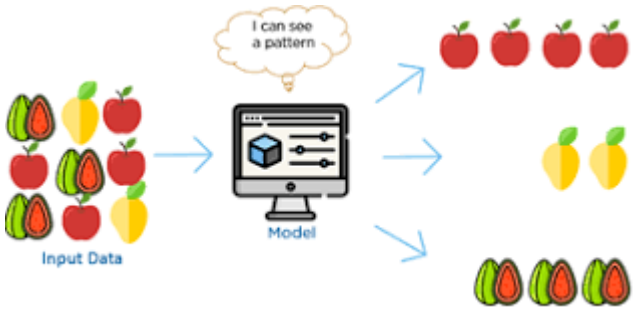
- https://www.youtube.com/watch?v=aircAruvnKk

Key terms to make notes on from the video:

- Activation value
- Hidden layer
- Neuron
- Weights
- Sigmoid
- What does Learning mean in this context?

Warning: The video will start talking about vectors and matrices from the mathematical field of linear algebra. I'm not too worried about you understanding the math at this stage as you won't have covered that in math yet.

Unsupervised learning



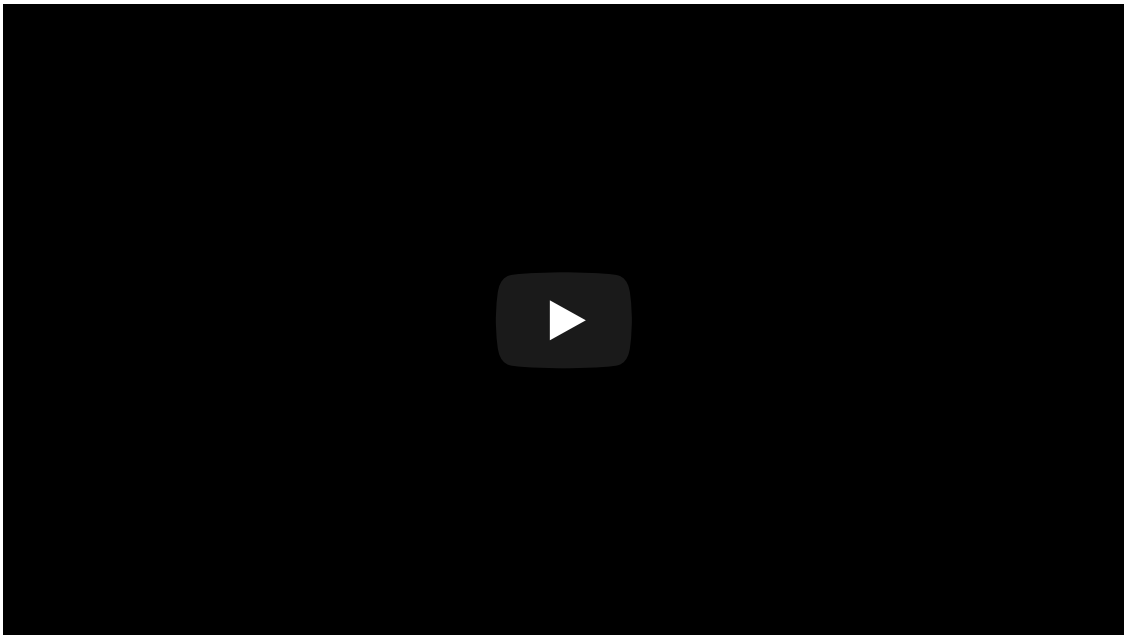
In unsupervised learning we give the computer unlabeled data (Input representations without their corresponding outcomes). We just present the data and let the computer algorithm make it's best attempt at sorting and categorising it.

The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

These are called unsupervised learning because unlike supervised learning above there is no correct answers and there is no teacher. Algorithms are left to their own devises to discover and present the interesting structure in the data.

Reinforcement learning

This video provides an excellent demostration of reinforcement learning in real life.



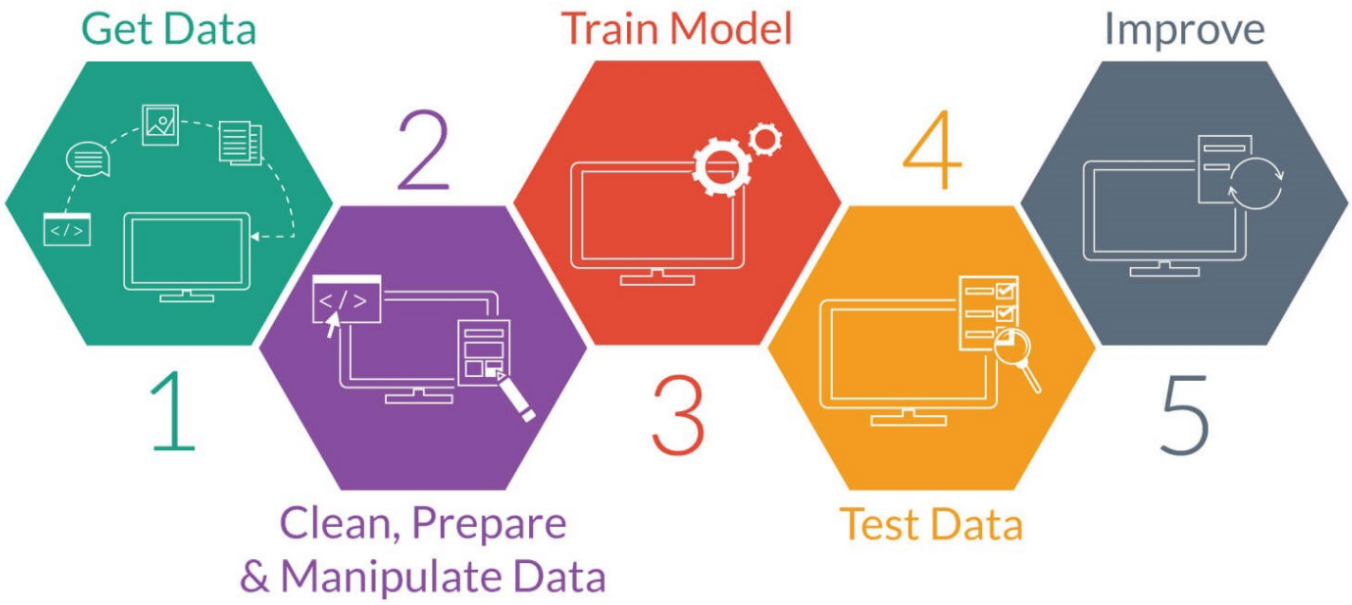
- https://youtu.be/6lp-LPc3LGI

Reinforced Learning: The machine is exposed to an environment where it gets trained by trial and error method, here it is trained to make a much specific decision. The machine learns from past experience and tries to capture the best possible knowledge to make accurate decisions based on the feedback received.

Practical applications include playing computer games and self driving cars.

The workflow

It is typically stated there are 5 steps to the process of building an AI.



- **Get the data**.
- **Clean, prepare and manipulate the data.**. - Pre-processing and cleaning data are important tasks that must be applied before using data to train an ML model Why? To avoid "garbage in, garbage out". While the "fun part" might be training the model, most of the time and energy tends to be on preparing your data.
- **Train the model** - Identify the inputs and outputs you want from the model, structure your layers, and feed the model your data.
- **Test the data** - In supervised learning some known data is typically held back for testing the accuracy of predictions. This is usually about 20 to 30% of your dataset.
- **Improve the data and model** - Analyse the results. Change the training settings to determine which inputs and algorithms might give you the best results.

Recognize that perfect performance is rarely, if ever, possible. Performance objectives should be defined based on the end goal for the model that you are building. Understanding the error tolerance is a critical step in identifying the risks in every model prediction.

Challenges

Overfitting:

We can find a hypothesis that predicts perfectly the training data but does not generalize well to new data. The function ”memorizes” the data points, but is wild everywhere else. Typical overfitting means that error on the training data is very low, but error on new instances is high.

Underfitting:

Typically, underfitting means that error on the training data is very high.

More information

Podcasts

- https://tech2025.com/podcast-show/

Credits

- Parinaz Sobhani for Canada Learning Code - https://github.com/ladieslearningcode/teenslc-intro-to-ai
- Key differences between Artificial Intelligence and Machine Learning - https://towardsdatascience.com/key-differences-between-artificial-intelligence-and-machine-learning-fe637cd0deca
- Machine learning types and alogrithms - https://towardsdatascience.com/machine-learning-types-and-algorithms-d8b79545a6ec
- Supervised and Unsupervised Machine Learning Algorithms by Jason Brownlee on March 16, 2016 - https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/
- https://towardsdatascience.com/reinforcement-learning-with-python-8ef0242a2fa2

2. The CartPole problem - with a random agent

Environment: MountainCarContinuous-v0

Documentation at https://github.com/openai/gym/wiki/CartPole-v0

Observation: Box 4

- 0 = Cart position -2.4 to 2.4
- 1 = Cart velocity -infinity to +infinity
- 2 = Pole angle -41.8 degrees to +41.8 degrees
- 3 = Pole velocity at top -infinity to +infinity Action: Discrete 1
- 0 = Push left, 1 = Push right

```
import gym
import random

class Agent():
    def __init__(self, env):
        self.action_size = env.action_space.n
        print(f"Number of actions available: {self.action_size}")

    def get_action(self, state):
        action = random.randint(0, self.action_size-1)
        return action

# Load the game environment
env_name = "CartPole-v1"
env = gym.make(env_name)
# Create our AI player agent (see class code above)
agent = Agent(env)
# Reset the game environment
state = env.reset()
# Play the "game" 200 times
for timestep in range(200):
    # Get a random action from the list available
    action = agent.get_action(state)
    # Apply that action to the environment
    print(f"for round {timestep}, taking action {action}")
    state, reward, done, info = env.step(action)
    env.render()
```

3. The CartPole problem - with simple logic

state documentation at
https://github.com/openai/gym/blob/master/gym/envs/classic_control/cartpole.py

```
import gym
import random

class Agent():
    def __init__(self, env):
        self.action_size = env.action_space.n
        print(f"Number of actions available: {self.action_size}")

    def get_action(self, state):
        pole_angle = state[2]
        if pole_angle < 0:
            return 0
        else:
            return 1

# Load the game environment
env_name = "CartPole-v1"
env = gym.make(env_name)
# Create our AI player agent (see class code above)
agent = Agent(env)
# Reset the game environment
state = env.reset()
# Play the "game" 200 times
for timestep in range(200):
    # Get a random action from the list available
    action = agent.get_action(state)
    # Apply that action to the environment
    print(f"for round {timestep}, taking action {action}")
    state, reward, done, info = env.step(action)
    env.render()
```

4. The Mountain Car problem - Continuous action spaces

Mountain Car - Discrete

```
import gym
import random

class Agent():
    def __init__(self, env):
        self.action_size = env.action_space.n
        print(f"Number of actions available: {self.action_size}")

    def get_action(self, state):
        action = random.randint(0, self.action_size-1)
        return action

env_name = "MountainCar-v0"
env = gym.make(env_name)
agent = Agent(env)
state = env.reset()
for game_number in range(100):
    state = env.reset()
    done = False
    move = 0
    while not done:
        # Get a random action from the list available
        action = agent.get_action(state)
        print(f"game {game_number:3} move {move:3}: taking action {action}")
        # Apply that action to the environment
        next_state, reward, done, info = env.step(action)
        state = next_state
        env.render()
        move += 1
```

Mountain Car - Continuous

Environment: MountainCarContinuous-v0

Documentation at https://github.com/openai/gym/wiki/MountainCarContinuous-v0

Observation: Box 2

- 0 = Car position from -1.2 to 0.6
- 1 = Car velocity from -0.07 to 0.07 Action: Box 1
- Negative push left, positive push right

```
import gym
import random
import numpy as np

class Agent():
    def __init__(self, env):
        print("Contineous action space")
        self.low = float(env.action_space.low)
        self.high = float(env.action_space.high)
        self.action_shape = env.action_space.shape
        print(f"Actions range available: from {self.low} to {self.high}")
```

```
    def get_action(self, state):
        action = [random.uniform(self.low, self.high)]
        return action

env_name = "MountainCarContinuous-v0"
env = gym.make(env_name)
agent = Agent(env)
for game_number in range(100):
    state = env.reset()
    done = False
    move = 0
    while not done:
        # Get a random action from the list available
        action = agent.get_action(state)
        print(f"game {game_number:3} move {move:3}: taking action {action}")
        # Apply that action to the environment
        next_state, reward, done, info = env.step(action)
        state = next_state
        env.render()
        move += 1
```

5. Q Learning

Q-Learning is one of the most commonly used algorithms for reinforcement learning. We will use Q-learning to teach our AI how to navigate through simple worlds using the Gym environments.

Let's start with a terminology primer:

- Agent: The name given to our computer algorithm simulating a human playing the "game". Think Agent Smith from the Matrix... it is the AI-bot.
- State: The state is a description of condition of the world we are observing. It is a collection of data that provides that description. In more complex uses, it could be providing a full pixel-by-pixel image of whats on the screen to the algorithm. In simplier cases it could be a handful of variables that sum up the key information about the "world" at any moment in time.
- Action: What action will our algorithm take? Action is usually based on the environment, different environments lead to different actions based on the agent. Set of valid actions for an agent are recorded in a space called an action space. These are usually finite in number.
- Environment: This where the agent lives and interacts. The "game world".
- Reward: The reward function is a measure of the benefit the agent received as a result of taking an action. If we win the game, we'll earn a great reward. If we lose the game, we'll obtain a negative reward. Other actions may obtain smaller positive or negative rewards along the way. It plays a vital role in reinforcement learning. The algorithm will seek to maximise reward.
- Policies: Policy is a rule used by an agent for choosing the next action, these are also called as agents brains.

The most commonly used algorithm for reinforcement learning is known as q-learning. As you start researching into Q-learning, you will likely encounter a formula that looks like this...

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

I don't want you to stress over the math, so ignore the formula for now. Instead I will endeavour to explain by way of a walk through what the process is that's occurring.

```
TO DO
```

Discuss:

- Role of q table - a list of every possible state and every possible action, and a historical record of the reward earned by taking that action when in that state.
- Updating the q table (with a walk through)
- Mapping a continuous space into discrete set of possiblities for the q table
- Role of epsilon - especially at the start. exploration is key.

6. Mountain Car - with Q Learning

```
import gym
import random
import numpy as np
from pprint import pprint

class Agent():
    def __init__(self, env):
        print(f"[agent init] env.action_space = {env.action_space}, env.action_space =
{env.observation_space}")
        # adjust the rate of learning
        self.epsilon = 1.0 # likelihood of choosing exploration vs learnt pathways
        self.epsilon_reduction_rate = 0.95
        self.epsilon_floor = 0.05
        self.discount_rate = 0.97
        self.learning_rate = 0.03
        # Set the actions we can select
        self.action_space = [-1.0, 1.0]
        # Set the observations we can make
        self.position_space = [-1.2, -1.1, -1.0, -0.9, -0.8, -0.7, -0.6, -0.5, -0.4, -0.3,
-0.2, -0.1, 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]
        self.velocity_space = [-0.07, -0.06, -0.05, -0.04, -0.03, -0.02, -0.01, 0.0, 0.01,
0.02, 0.03, 0.04, 0.05, 0.06, 0.07]
        # Build a list containing all possible states
        states = []
        for position in self.position_space:
            for velocity in self.velocity_space:
                states.append((position, velocity))
        # Build our q-table
        self.q = {}
        for state in states:
            for action in self.action_space:
                self.q[state, action] = 0 # initialise value for this state and action

    @staticmethod
    def find_closest(dataset, target):
        closest_index = 0
        closest_distance = 10e10 # a really big number, ie: 10^10
        for i in range(len(dataset)):
            this_distance = abs(dataset[i] - target)
            if this_distance < closest_distance:
                closest_index = i
                closest_distance = this_distance
        return dataset[closest_index]

    def get_state(self, observation):
        # Get an observation (continuous) and return a discrete state
        position, velocity = observation
        return (self.find_closest(self.position_space, position),
self.find_closest(self.velocity_space, velocity))

    def max_action(self, state):
        # Search our q-table for all entries that match the current state.
        # Then, look through those results to find the action that obtained the highest
reward.
        max_index = 0
        max_value = -10e10 # a really negative number, ie: -10^10
        for i in range(len(self.action_space)):
            action_val = self.action_space[i]
            q_val = self.q[state, action_val]
            if q_val > max_value:
                max_index = i
                max_value = q_val
        return self.action_space[max_index]

    def get_action(self, state):
```

```
        r = random.random()
        # if our random number is less than epsilon, take a random action
        if r < self.epsilon:
            action = random.choice(self.action_space)
        # otherwise, take our learned action
        else:
            action = self.max_action(state)
        return action

    def train(self, experience):
        prev_state, action, new_state, reward, done = experience
        # find the new action
        new_action = self.max_action(new_state)
        # create shortcut variable names so the final line is comprehendible
        discount = self.discount_rate
        learn = self.learning_rate
        q_prev = self.q[prev_state, action]
        q_new = self.q[new_state, new_action]
        # update q table using "q formula"
        self.q[prev_state, action] = q_prev + learn * ( reward + discount * q_new - q_prev )

    def reduce_greediness(self):
        # Reduce epsilon each time
        self.epsilon = self.epsilon * self.epsilon_reduction_rate
        if self.epsilon < self.epsilon_floor:
            self.epsilon = self.epsilon_floor

env_name = "MountainCarContinuous-v0"
env = gym.make(env_name)
agent = Agent(env)
for game_number in range(101):
    # reset game
    done = False
    score = 0
    move_number = 0
    observation = env.reset()
    # get our starting observation state
    state = agent.get_state((observation))
    while not done:
        # Get the action to perform for this state
        action = agent.get_action(state)
        # Apply that action to the environment
        new_observation, reward, done, info = env.step([action])
        # convert the infinite observation into our 20x20 possible states
        new_state = agent.get_state((new_observation))
        # display game frame on screen (remove this line to speed things up)
        env.render()
        # Prepare for the next move
        agent.train((state, action, new_state, reward, done))
        state = new_state
        move_number += 1
        score += reward
    # Print diagnostic information
    print(f"game_number {game_number:3}: moves made {move_number:3}, reward earned
{round(score)}")
    agent.reduce_greediness()
print("all done!")
```

7. Moon lander

8. Analysis

References

Lesson 2, 3 (Pole) derived from

- "Getting Started With OpenAI Gym" by TheComputerScientist, 2018 - https://youtu.be/8MC3y7ASoPs
- "Play Any OpenAI Gym Environment with a Single Agent" by TheComputerScientist, 2018 - https://www.youtube.com/watch?v=nvhWfk7R0RM

Lesson 5, 6 (Mountain cart) derived from

- "Q learning with just numpy | solving the Mountain car | Tutorial" by Machine Learning with Phil, 2019 - https://www.youtube.com/watch?v=rBzOyjywtPw
- "Youtube-Code-Repository/ReinforcementLearning/mountaincar.py" by philtabor, 2019 - https://github.com/philtabor/Youtube-Code-Repository/blob/master/ReinforcementLearning/mountaincar.py

Lesson 7+ (Tic Tac Toe) derived from

- "Gym TicTacToe is a light Tic-Tac-Toe environment for OpenAI Gym" by ClementRomac, 2019 - https://github.com/ClementRomac/gym-tictactoe