Artificial Intelligence II

Lesson 6- Learning Methods

First Code Academy

Today's Plan

Teach Back	00 - 5 min
Supervised Learning	10 - 15 min
Reinforcement Learning	15 - 20 min
Quiz	20-25 min
Break	25 - 28 min
Project - Game Training	28 - 55 min
Lesson Recap	55 - 60 min

Teach Back

___________ allows us to easily find how variables relate to each other

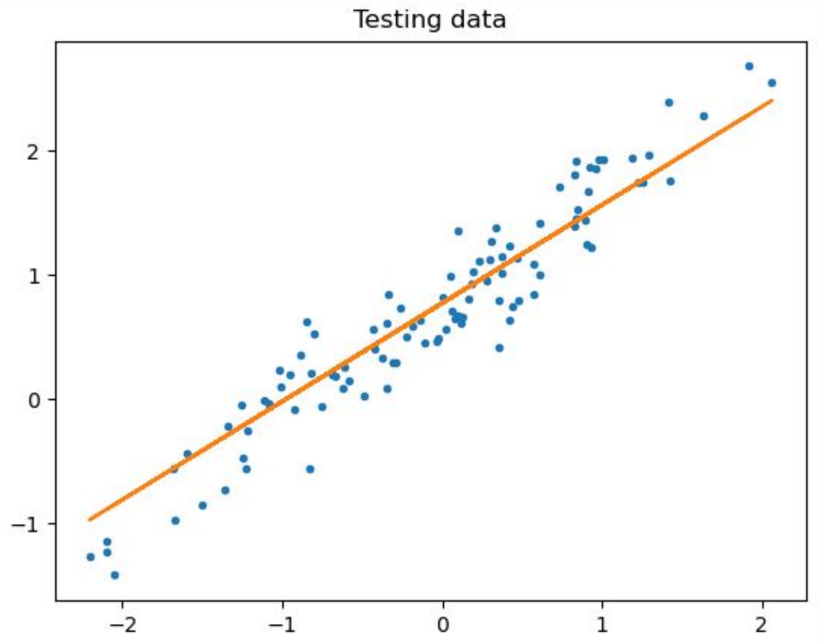
The algorithm we use to improve our model is called gradient _______

What did we learn last time?

Regression

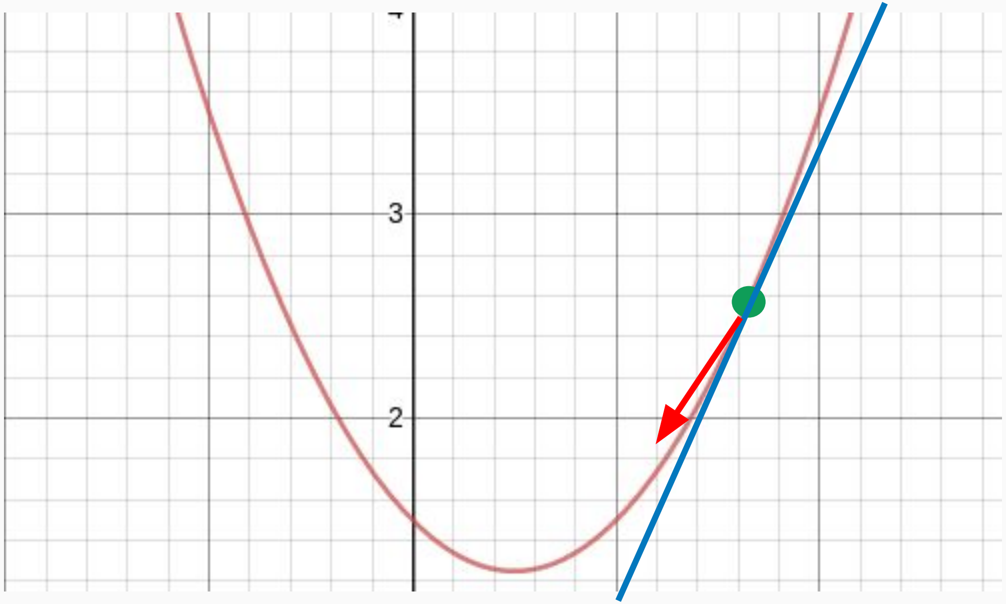
Given many (x, y) points, find the relation between the two variables

We need the **slope** and **intercept** in this case



Gradient Descent

Gradient Descent finds the optimal parameters by moving in the **descending** direction



Key Terms

Supervised Learning

Reinforcement Learning

Supervised Learning

Supervised Learning

Supervised Learning means that we know the intended answer during training.

We have a **reference** answer during training.

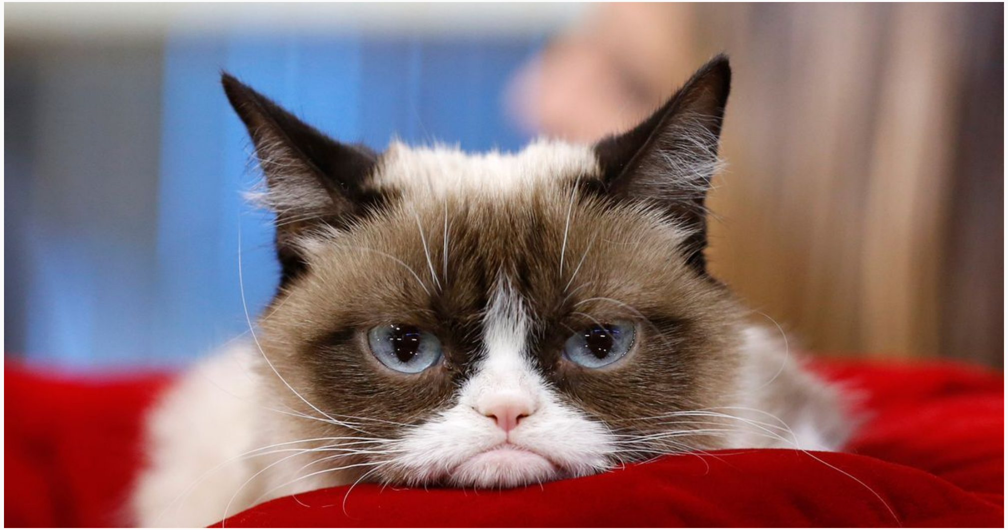
Supervised Learning

Suppose we want to see if a picture contains a cat.

The training data would say if there is a cat or not.

The data is “**labeled**”

[cat]



Supervised Learning

In our regression project, the y in (x, y) was our label.

We saw how accurate our model was

Labeled Training Data

(x, y)

Our model's data

(x, y)

The **error** is their difference

Labeling Data

However, labeling data can be a time-consuming and expensive process

A lot of it is done by hand!

Amazon Mechanical Turk

Marketplace where companies can put up jobs such as labeling

Workers compete for these jobs.

Reinforcement Learning

Reinforcement Learning

Instead of using labeled data, we think in terms of **rewards** and **punishments**

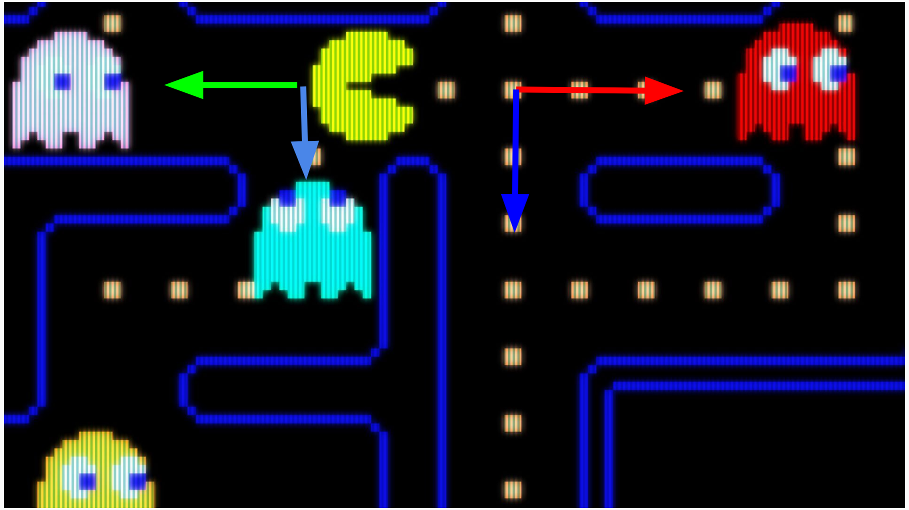
Reinforcement Learning

Components in reinforcement learning:

1. **State**: Current environment conditions.
2. **Action**: A possible move.
3. **Reward**: The value in taking a certain action.

Reinforcement Learning

We want to take the actions with the highest reward



Which way should Pacman go?

Each **action** gives us a different value

Exploration vs. Exploitation

Our model wants to **explore** new strategies but also **exploit** the reward

An optimal strategy might involve short-term losses

Q-Learning

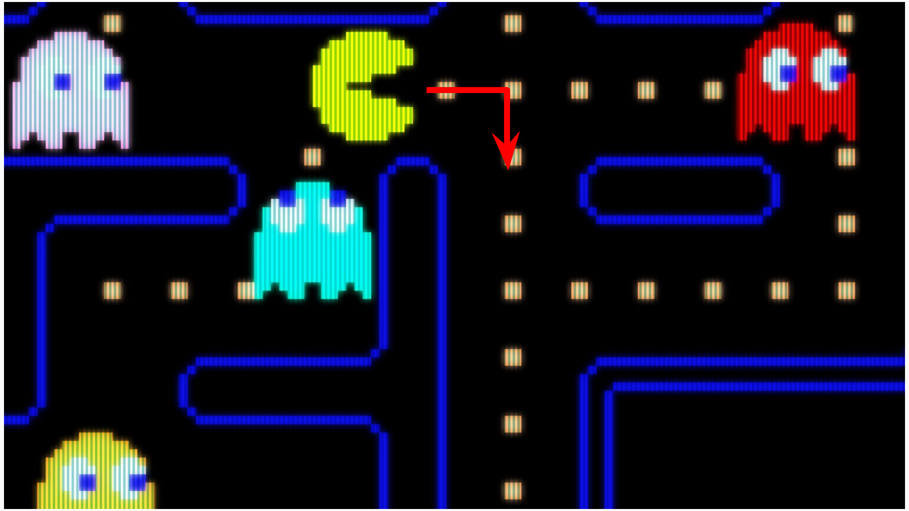
In Q-Learning, we have a function that gives us the "quality" of taking an action at a state

$$Q(s, a)$$

If at state s we take action a, what is the value of **all the following actions**?

Q-Learning

If we take this path, we would have more moves left in total, so higher value!

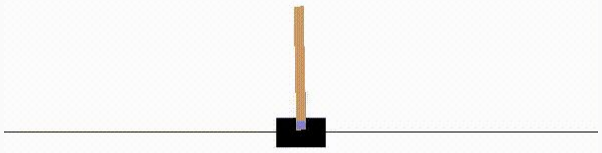


Quiz: bit.ly/FCA_AI_Quiz

Project : Game Training

Objective

We want to create an agent that learns to play a simple game using Q-Learning!



Our agent has to learn to balance a pole on a moving cart

OpenAI Gym

Gym is a Python library that allows us to learn reinforcement learning in games!

Install it on your machine by typing:

```
pip install gym
```

Get the starter file

http://bit.ly/FCA_AI2_Starter

Make an agent that chooses random action

Random action

`render()` shows the current frame on the screen

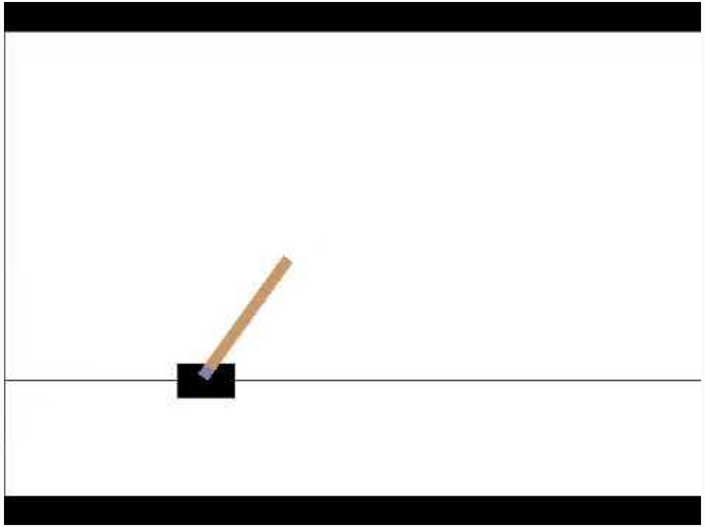
`step()` takes an action from set of possible actions.

Always add `env.close()` at the very end!

```
import gym

# Create the envrionment
env = gym.make('CartPole-v0')

# Reset the environment
env.reset()
for _ in range(1000):
    # show the current state on the screen
    env.render()

    # Take a random action
    env.step(env.action_space.sample())
env.close()
```

Random action

Our agent doesn't react to its environment and just moves randomly!



Let’s code a smart agent!

Creating agent

First we create the agent and environment

```python
# Create the environment
env = gym.make('CartPole-v0')
# Create our agent
agent = CartPole(env)
```

Outer Loop

Set variables for every iteration of the game.

```python
for ep in range(EPISODES):
    print(f'Episode: {ep}')

    # Get the initial state
    observation = env.reset()
    # Turn continuous state space into discrete
    state = agent.discretize(observation)

    # Get Learning and Exploration rates
    alpha = agent.get_alpha(ep)
    epsilon = agent.get_epsilon(ep)

    done = False
    i = 0
```

Inner Loop

Handle the main game-playing logic

```
        i = 0
        while not done:
            # Render the frame
            env.render()

            # Choose the action
            action = agent.get_action(state, epsilon)
```

Inner Loop

Take the step and update our Q-table

```
# Take the action in game
observation, reward, done, _ = env.step(action)
new_state = agent.discretize(observation)

# update our Q-Table
agent.update_q(state, action, reward, new_state, alpha)
state = new_state
i += 1
```

CartPole Agent Class

We then setup the CartPole agent class

```python
class CartPole():
    def __init__(self, env, buckets=(1, 2, 6, 12)):
        # Making the ranges into discrete ranges
        self.buckets = buckets
        self.env = env

        # Q table
        self.Q = np.zeros(self.buckets + (self.env.action_space.n,))

```

Get Action

Update the `get_action()` method.

```
def get_action(self, state, epsilon):
    return self.env.action_space.sample() if (np.random.random() <= epsilon ) else np.argmax(self.Q[state])
```

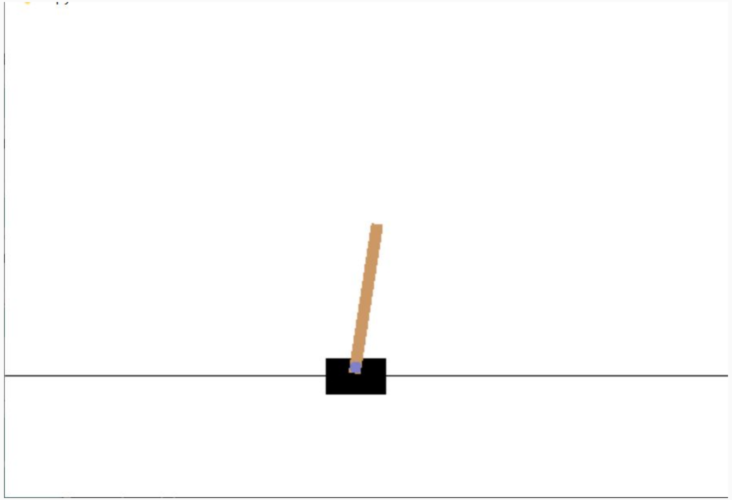
Update Q-Table

After each state transition, we update our table

```
    def update_q(self, state, action, reward, new_state, alpha):
        self.Q[state][action] += alpha * (reward + GAMMA * np.max(self.Q[new_state]) - self.Q[state][action])

```

Try it out!

Our agent learns slowly, so it will take a while for it to learn to play.



That’s it for today!

Key Terms

Supervised Learning

Reinforcement Learning

Artificial Intelligence II

Lesson 6 - Learning Methods

First Code
Academy