# Robotics Reading Group
## @ Instituto Superior Técnico

Session #4
29-11-2019

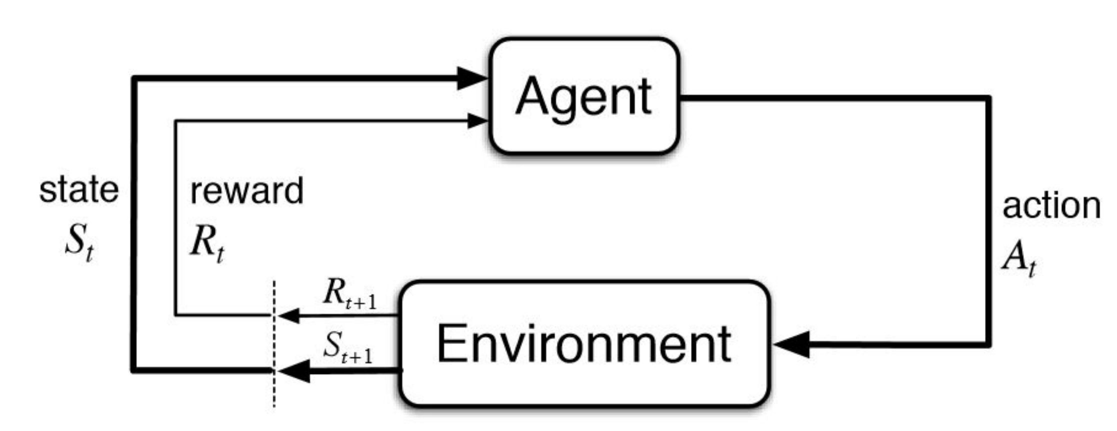
João Ribeiro

Google DeepMind

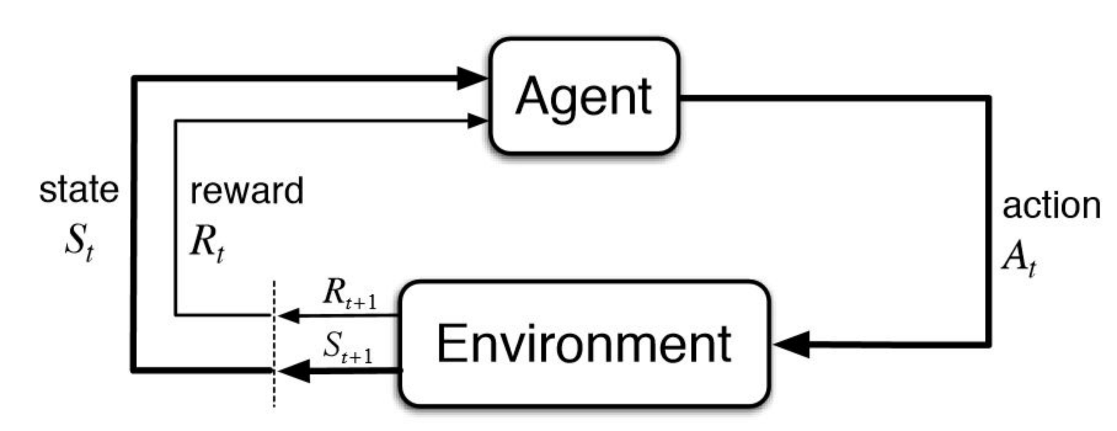
# DQN

## HUMAN-LEVEL CONTROL THROUGH **DEEP REINFORCEMENT LEARNING**

# Some background



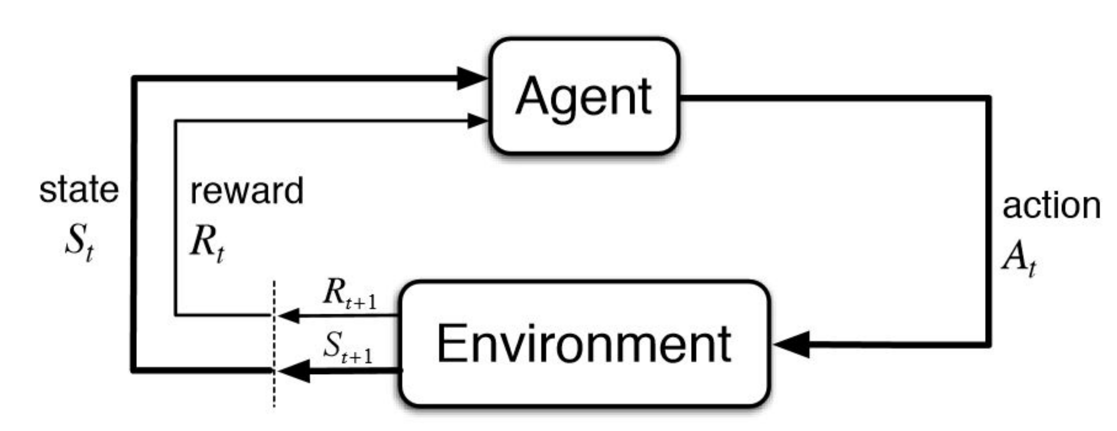
# Some background




## Action-Value Function Q(s,a)

- *Expectation of how much future discounted reward the agent will obtain by executing action **a** in state **s***

# Some background

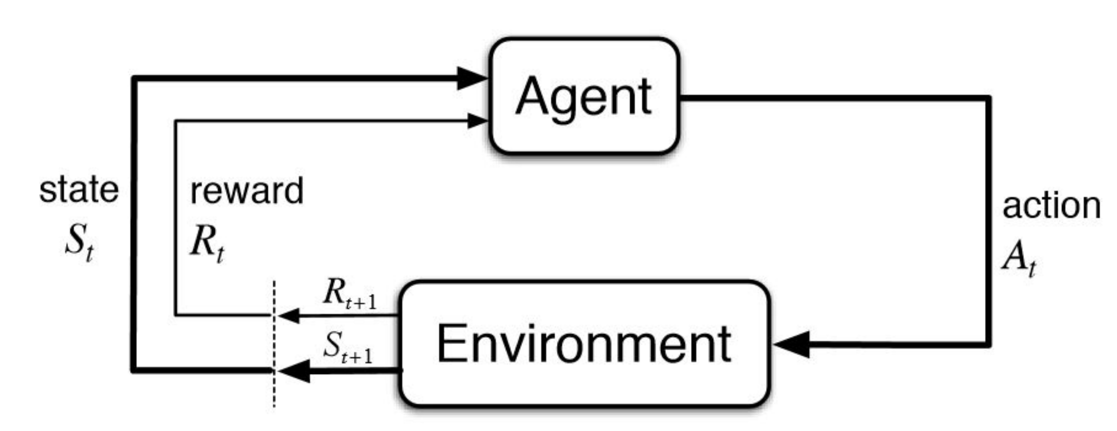



## Action-Value Function Q(s,a)

- *Expectation of how much future discounted reward the agent will obtain by executing action **a** in state **s***

(in other words)

# Some background




## Action-Value Function Q(s,a)

- *Expectation of how much future discounted reward the agent will obtain by executing action **a** in state **s***

## (in other words)

- Expectation of *"how good executing action a in state s is for the agent"*

# Learning Q(s,a) (trial-and-error)

# Learning Q(s,a) (trial-and-error)

In a given timestep:

# Learning Q(s,a) (trial-and-error)

In a given timestep:

1. Observe state s (described by some designed features)

# Learning Q(s,a) (trial-and-error)

In a given timestep:

1. Observe state s (described by some designed features)
2. Execute action a on the environment

# Learning Q(s,a) (trial-and-error)

In a given timestep:

1. Observe state s (described by some designed features)
2. Execute action a on the environment
3. Observe next state s' and reward r

# Learning Q(s,a) (trial-and-error)

In a given timestep:

1. Observe state s (described by some designed features)
2. Execute action a on the environment
3. Observe next state s' and reward r
4. Update Q(s,a) estimate iteratively:

# Learning Q(s,a) (trial-and-error)

In a given timestep:

1. Observe state s (described by some designed features)
2. Execute action a on the environment
3. Observe next state s' and reward r
4. Update Q(s,a) estimate iteratively:
    a. $Q(s,a) = (1-\alpha)\, Q(s,a) + \alpha[r + \gamma \max Q(s', a')]$

# Learning Q(s,a) (trial-and-error)

In a given timestep:

1. Observe state s (described by some designed features)
2. Execute action a on the environment
3. Observe next state s' and reward r
4. Update Q(s,a) estimate iteratively:
    a. $Q(s,a) = (1-\alpha)\, Q(s,a) + \alpha[r + \gamma \max Q(s', a')]$

*(known as Q-Learning)*

# We can do better!

# We can do better!

Instead of iteratively updating Q(s,a):

$$Q(s,a) = (1-\alpha)\, Q(s,a) + \alpha\, [r + \gamma \max Q(s', a')]$$

# We can do better!

Instead of iteratively updating Q(s,a):

$$Q(s,a) = (1-\alpha)\, Q(s,a) + \alpha\, [r + \gamma \max Q(s', a')]$$

Use function approximation (neural network)!

# We can do better!

Instead of iteratively updating Q(s,a):

$$Q(s,a) = (1-\alpha)\, Q(s,a) + \alpha\, [r + \gamma \max Q(s', a')]$$

Use function approximation (neural network)!

Predictions

$$\hat{y} = Q(s, a, \theta)$$

Targets

$$y = r + \gamma \max Q(s', a', \theta)$$

# We can do better!

Instead of iteratively updating Q(s,a):

$$Q(s,a) = (1-\alpha)\, Q(s,a) + \alpha\, [r + \gamma \max Q(s', a')]$$

Use function approximation (neural network)!

Predictions

$$y\_hat = Q(s, a, \theta)$$

Targets

$$y = r + \gamma \max Q(s', a', \theta)$$

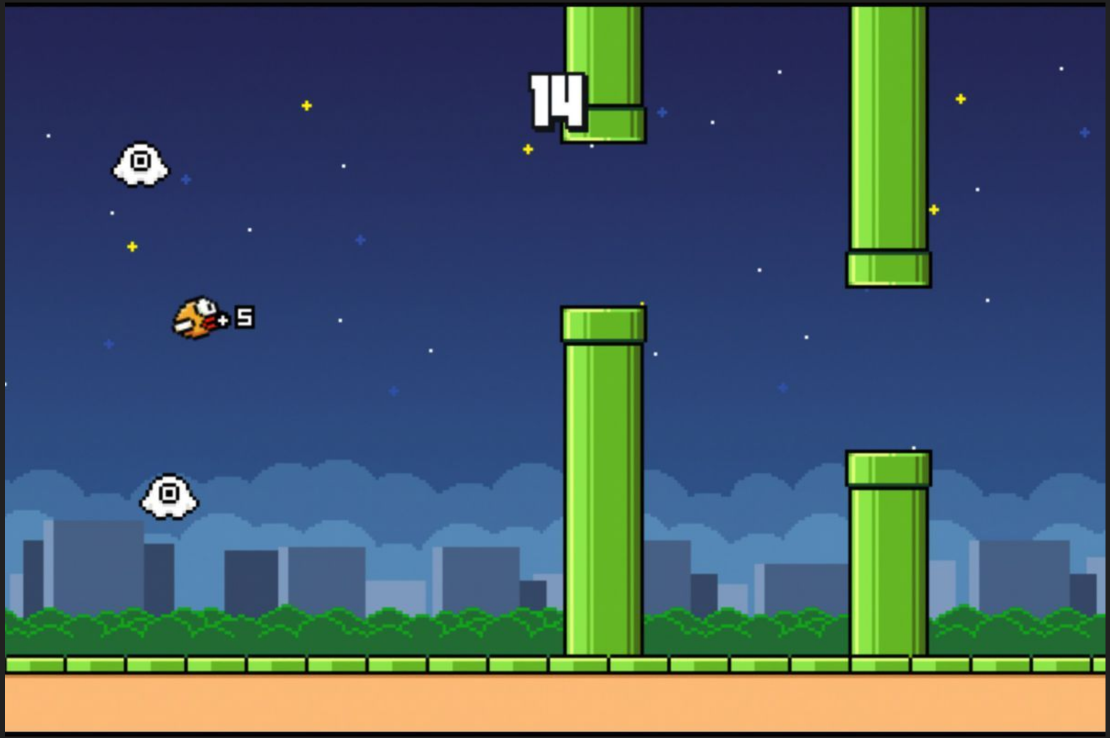
Loss (Mean Squared Error):

1/N sum (y-yhat)^2

# Three Problems

# 1 - How to design good feature representations?

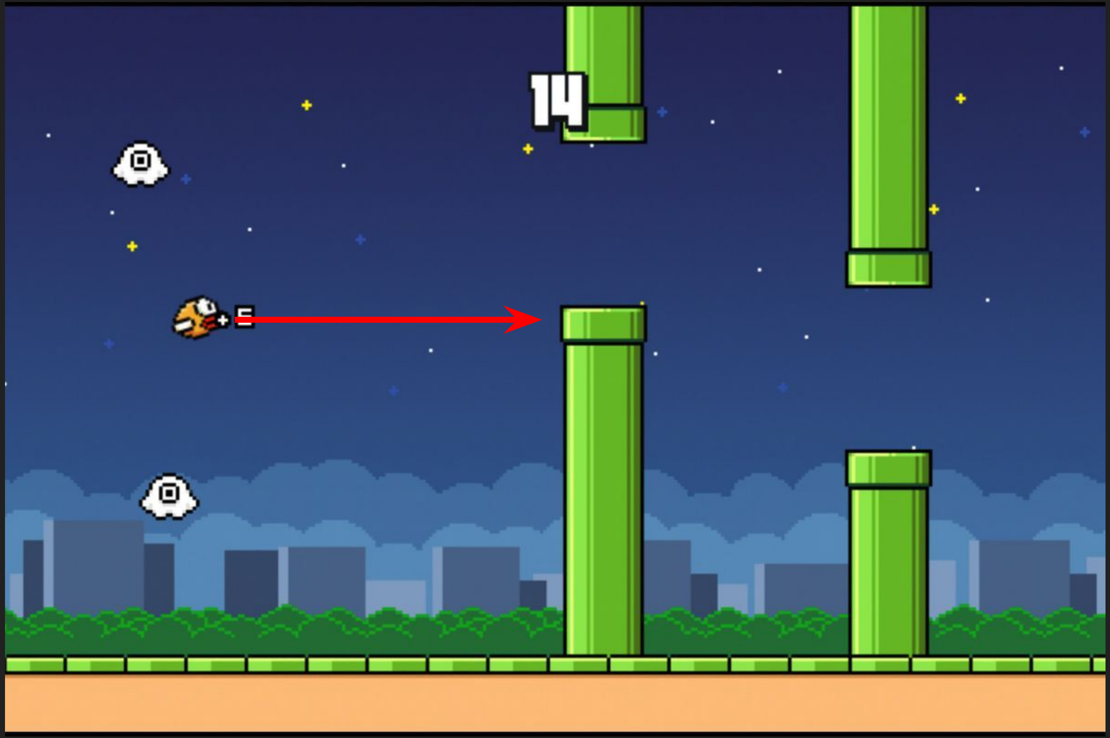
state = ?



# 1 - How to design good feature representations?

state = [
  *Distance to pipes?*

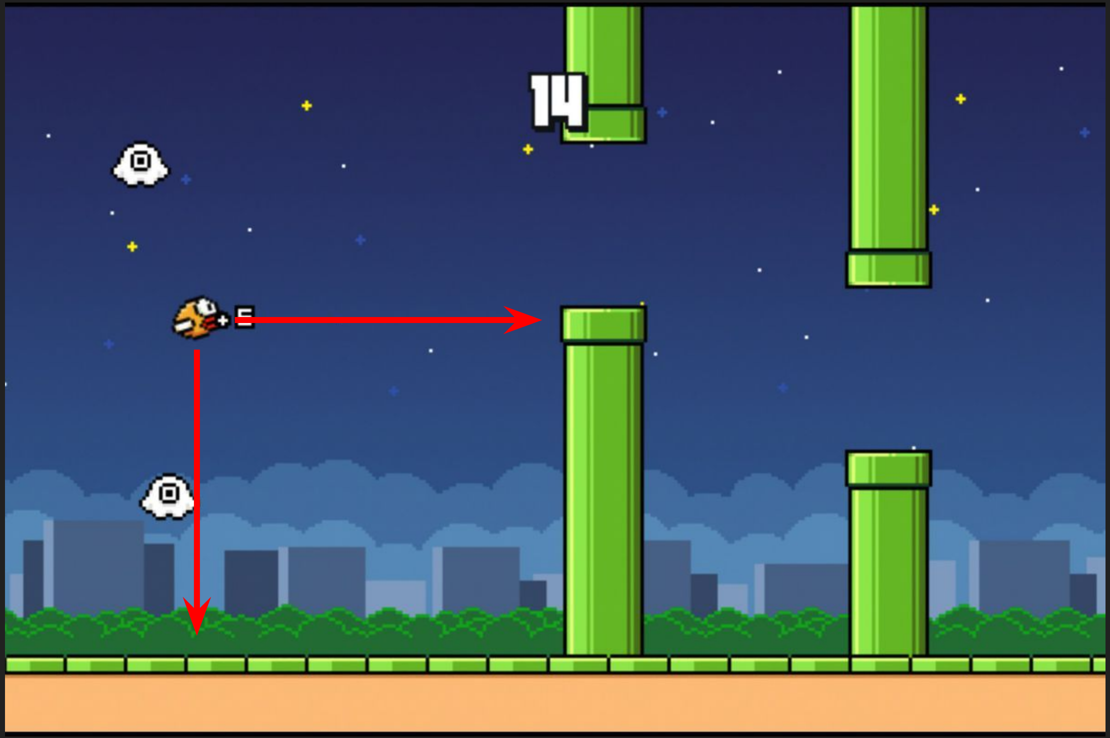
]



# 1 - How to design good feature representations?

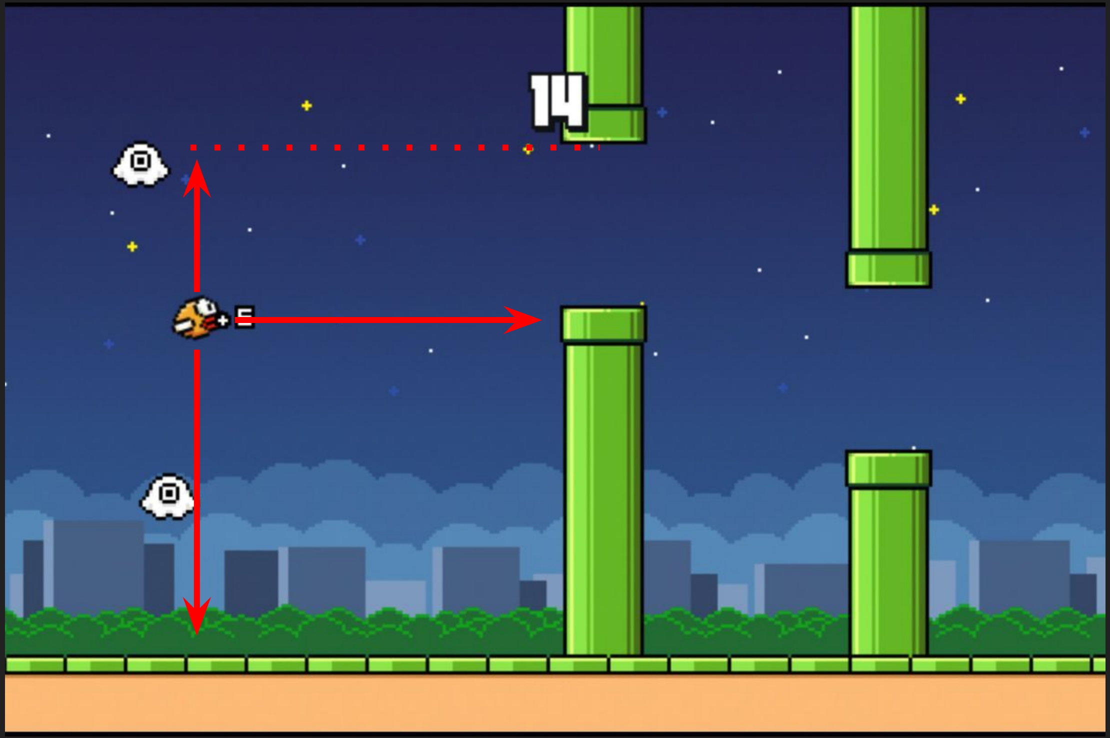
state = [
  *Distance to pipes?*
  *Distance to ground?*

]



# 1 - How to design good feature representations?

state = [
*Distance to pipes?*
*Distance to ground?*
*Distance to higher pipe?*
]



# 1 - How to design good feature representations?

state = [

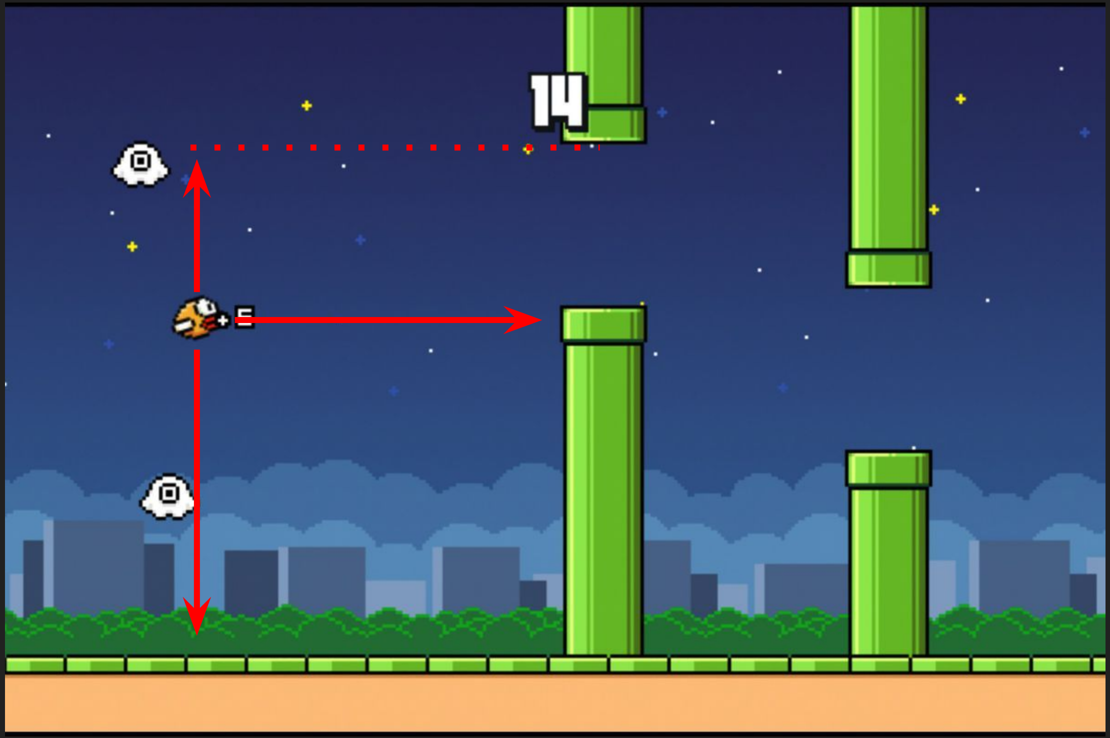
*Distance to pipes?*

*Distance to ground?*

*Distance to higher pipe?*

]

Bad!



# 1 - How to design good feature representations?

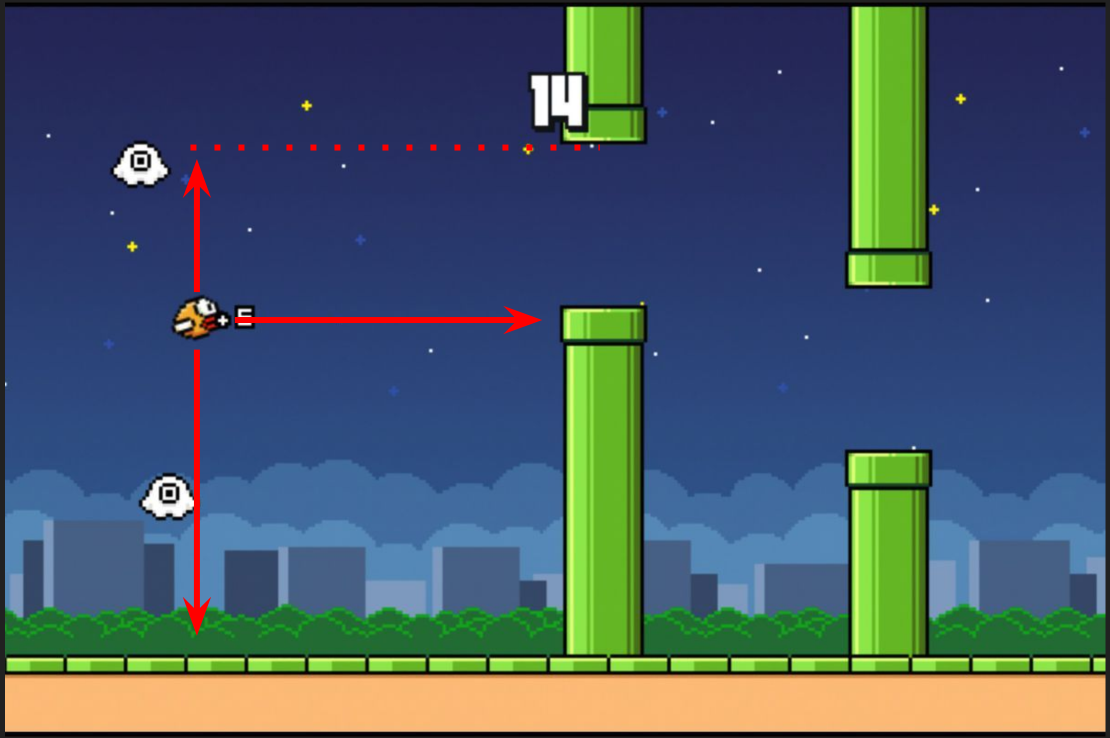
state = [

*Distance to pipes?*

*Distance to ground?*

*Distance to higher pipe?*

]

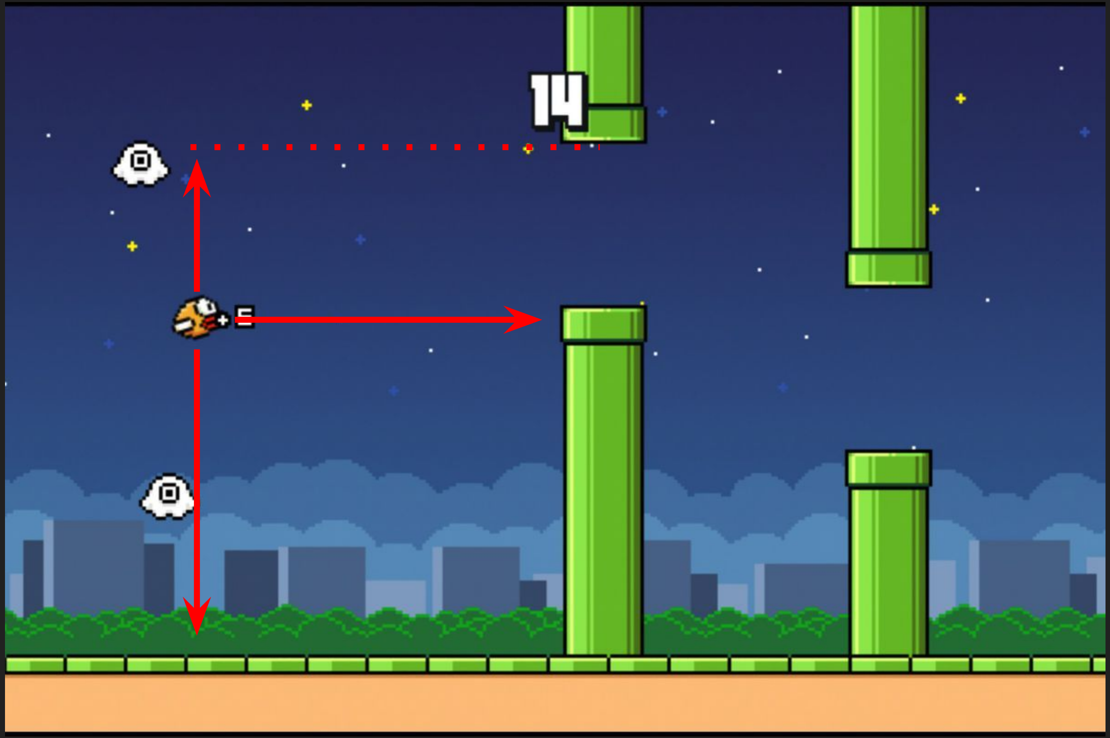



Bad!

1. Requires domain knowledge

# 1 - How to design good feature representations?

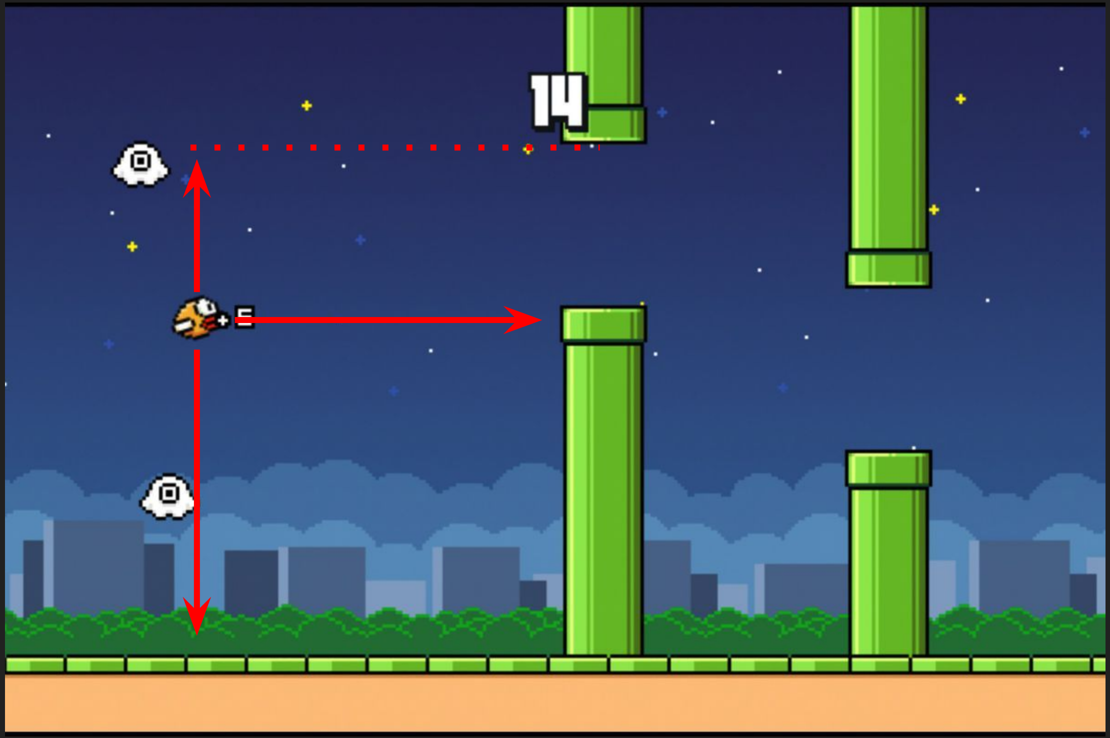
state = [
*Distance to pipes?*
*Distance to ground?*
*Distance to higher pipe?*
]




Bad!
1. Requires domain knowledge
2. Prone to human bias

# 1 - How to design good feature representations?

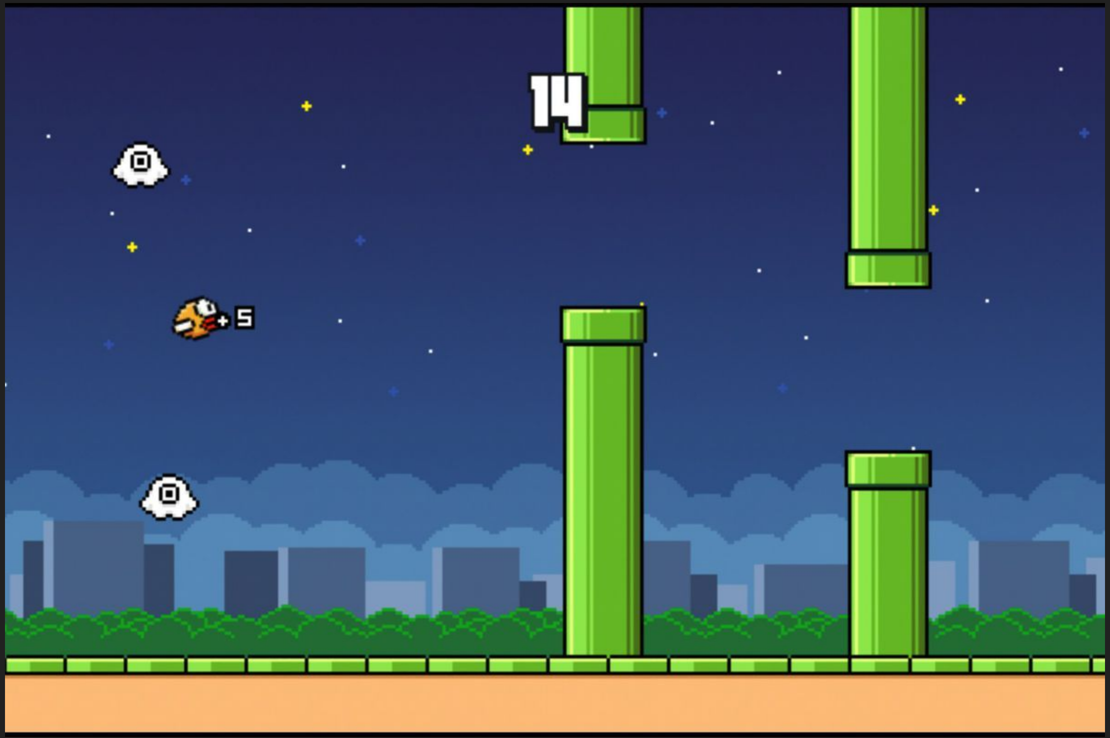
state = [
  *Distance to pipes?*
  *Distance to ground?*
  *Distance to higher pipe?*
]



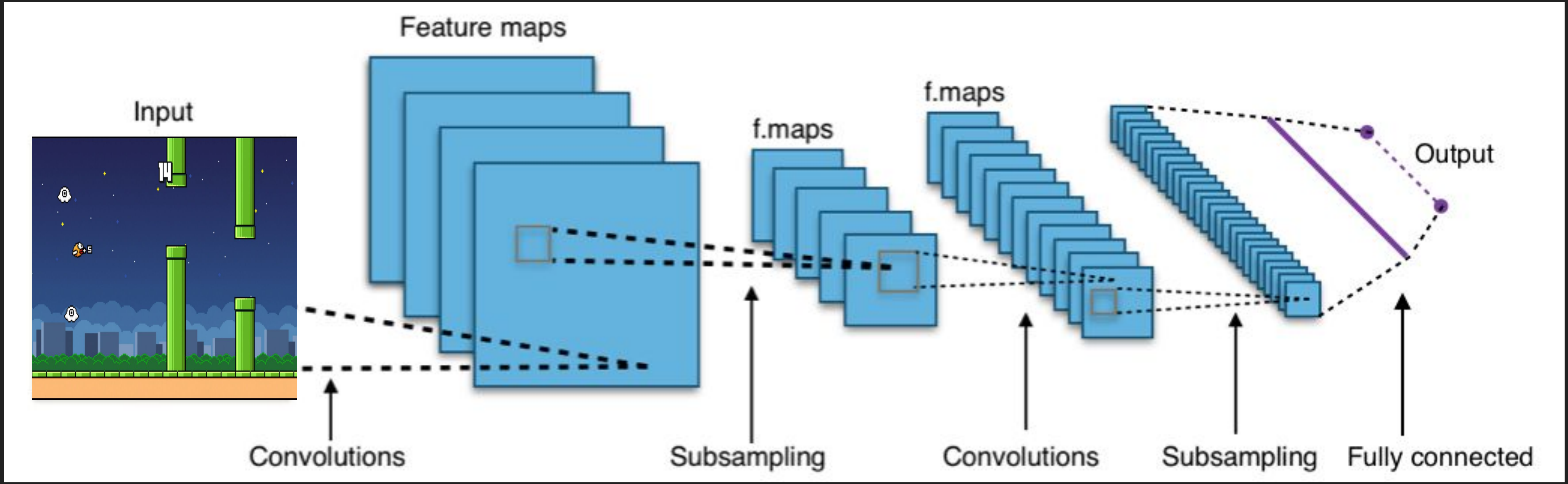

Bad!

1. Requires domain knowledge
2. Prone to human bias
3. Could limit learning!

# Solution?



# Deep Learning!



# 2 - Neural fitted-Q Iteration

# 2 - Neural fitted-Q Iteration

Neural Network Q(s, a; θ):

- iteratively trained using single datapoint (s, a, r, s')

# 2 - Neural fitted-Q Iteration

Neural Network Q(s, a; θ):

- iteratively trained using single datapoint (s, a, r, s')
- (Inefficient!)

# 2 - Neural fitted-Q Iteration

Neural Network Q(s, a; θ):

- iteratively trained using single datapoint (s, a, r, s')
- (Inefficient!)

Solution?

# 2 - Neural fitted-Q Iteration

Neural Network Q(s, a; θ):

- iteratively trained using single datapoint (s, a, r, s’)
- (Inefficient!)

Solution?

1. Collect (s, a, r, s’)

# 2 - Neural fitted-Q Iteration

Neural Network $Q(s, a; \theta)$:

- iteratively trained using single datapoint $(s, a, r, s')$
- (Inefficient!)

Solution?

1. Collect $(s, a, r, s')$
2. Store in dataset D

# 2 - Neural fitted-Q Iteration

Neural Network $Q(s, a; \theta)$:

- iteratively trained using single datapoint (s, a, r, s')
- (Inefficient!)

Solution?

1. Collect (s, a, r, s')
2. Store in dataset D
3. Randomly sample a batch of B datapoints for backpropagation!

# 2 - Neural fitted-Q Iteration

Neural Network Q(s, a; θ):

- iteratively trained using single datapoint (s, a, r, s’)
- (Inefficient!)

Solution?

1. Collect (s, a, r, s’)
2. Store in dataset D
3. Randomly sample a batch of B datapoints for backpropagation!

*(Known as Experience Replay!)*

# 3 - Target Neural fitted-Q

# 3 - Target Neural fitted-Q

Targets for loss computed using the same network

Predictions

$$\hat{y} = Q(s, a; \theta)$$

Targets

$$y = r + \gamma \max Q(s', a'; \theta)$$

# 3 - Target Neural fitted-Q

Targets for loss computed using the same network

Predictions

$$\hat{y} = Q(s, a; \theta)$$

Targets

$$y = r + \gamma \max Q(s', a'; \theta)$$

(Can heavily bias training!)

# 3 - Target Neural fitted-Q

Targets for loss computed using the same network

Predictions

$$\hat{y} = Q(s, a; \theta)$$

Targets

$$y = r + \gamma \max Q(s', a'; \theta)$$

(Can heavily bias training!)

Solution?

# 3 - Target Neural fitted-Q

Targets for loss computed using the same network

Predictions

$y\_hat = Q(s, a; \theta)$

Targets

$y = r + \gamma \max Q(s', a'; \theta)$

(Can heavily bias training!)

Solution - Two networks!

# 3 - Target Neural fitted-Q

Targets for loss computed using the same network

Predictions

$\hat{y} = Q(s, a; \theta)$

Targets

$y = r + \gamma \max Q(s', a'; \theta)$

(Can heavily bias training!)

Solution - Two networks!

One for predictions

$Q(s, a; \theta)$

# 3 - Target Neural fitted-Q

Targets for loss computed using the same network

Predictions

$\hat{y} = Q(s, a; \theta)$

Targets

$y = r + \gamma \max Q(s', a'; \theta)$

(Can heavily bias training!)

Solution - Two networks!

One for predictions

$Q(s, a; \theta)$

One for targets

$Q(s, a; \theta_{target})$

# Putting it all together

# Putting it all together

Automatic
feature extraction
|
CNN

# Putting it all together

Automatic
feature extraction
|
CNN

Experience
Replay
|
Dataset D

# Putting it all together

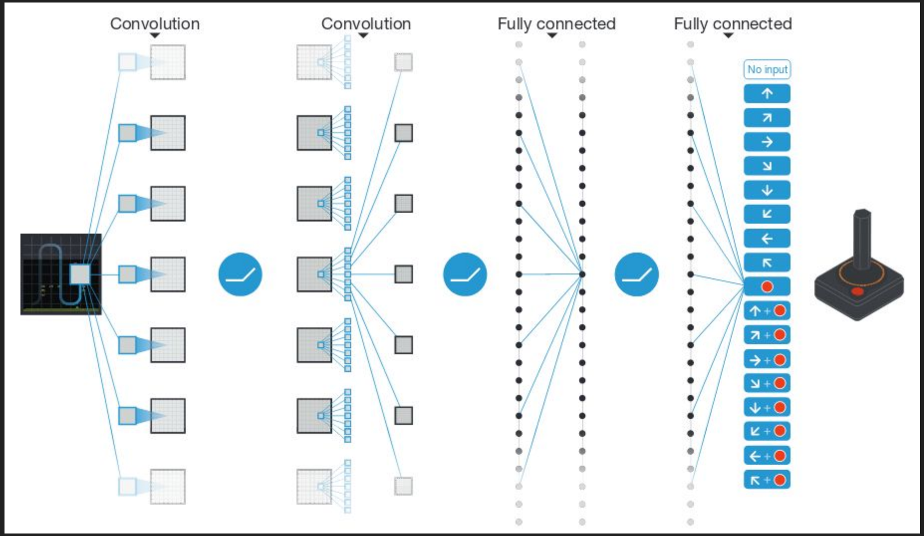
Automatic
feature extraction
|
CNN

Experience
Replay
|
Dataset D

Main & Target
Networks
|
$Q(s,a;\theta)$ $Q(s,a;\theta_{target})$

# The Deep Q-Network!

| Automatic feature extraction | Experience Replay | Main & Target Networks |
|---|---|---|
| \| | \| | \| |
| CNN | Dataset D | $Q(s,a;\theta)$ $Q(s,a;\theta_{target})$ |



# Does it work?

# Results

- 49 DQN's trained on 49 games from the Atari2600 Platform
    - *50M training **frames** (timesteps)*
    - *30 test **episodes** (mean ep. reward reported, N=30)*

# Results

- 49 DQN's trained on 49 games from the Atari2600 Platform
    - *50M training **frames** (timesteps)*
    - *30 test **episodes** (mean ep. reward reported, N=30)*

- Only high-level sensorial input
    - *Game screen (pixels)*
    - *Current score*

# Results

- 49 DQN’s trained on 49 games from the Atari2600 Platform
  - *50M training **frames** (timesteps)*
  - *30 test **episodes** (mean ep. reward reported, N=30)*

- Only high-level sensorial input
  - *Game screen (pixels)*
  - *Current score*

- 43/49 games

# Results

- 49 DQN’s trained on 49 games from the Atari2600 Platform
    - *50M training **frames** (timesteps)*
    - *30 test **episodes** (mean ep. reward reported, N=30)*

- Only high-level sensorial input
    - *Game screen (pixels)*
    - *Current score*

- 43/49 games - *Outperformed best algorithms for each individual game*

# Results

- 49 DQN's trained on 49 games from the Atari2600 Platform
  - *50M training **frames** (timesteps)*
  - *30 test **episodes** (mean ep. reward reported, N=30)*

- Only high-level sensorial input
  - *Game screen (pixels)*
  - *Current score*

- 43/49 games - *Outperformed best algorithms for each individual game*

- 29/49 games

# Results

- 49 DQN’s trained on 49 games from the Atari2600 Platform
  - *50M training **frames** (timesteps)*
  - *30 test **episodes** (mean ep. reward reported, N=30)*

- Only high-level sensorial input
  - *Game screen (pixels)*
  - *Current score*

- 43/49 games - *Outperformed best algorithms for each individual game*

- 29/49 games - *Human level or above*

# Results

- 49 DQN's trained on 49 games from the Atari2600 Platform
    - *50M training **frames** (timesteps)*
    - *30 test **episodes** (mean ep. reward reported, N=30)*

- Only high-level sensorial input
    - *Game screen (pixels)*
    - *Current score*

- 43/49 games - *Outperformed best algorithms for each individual game*

- 29/49 games - *Human level or above*
    - *15/29 games*

# Results

- 49 DQN’s trained on 49 games from the Atari2600 Platform
  - *50M training **frames** (timesteps)*
  - *30 test **episodes** (mean ep. reward reported, N=30)*

- Only high-level sensorial input
  - *Game screen (pixels)*
  - *Current score*

- 43/49 games - *Outperformed best algorithms for each individual game*

- 29/49 games - *Human level or above*
  - *15/29 games - Superhuman control!*


180 2 1

# Cool resources to check out:

- OpenAI Baselines - Tensorflow implementation of SOTA algorithms:
  - *https://github.com/openai/baselines*

- Stable Baselines - Fork from openai/baselines with refactored code:
  - *https://github.com/hill-a/stable-baselines*

- Tensorflow Agents - Full Deep RL library with good abstractions, written in tensorflow:
  - *https://github.com/tensorflow/agents*

- Deep Reinforcement Learning that Matters - Really cool paper on statistical significance and reproducibility of Deep RL work:
  - *https://arxiv.org/abs/1709.06560*

- Deep RL Hands on - Really practical book on DRL with code examples written in PyTorch:
  - https://github.com/aibooks/aibooks.github.io


Throttle: 0.00, Steering: 0.00
Use arrow keys to move, q or ESCAPE to exit.
space key: toggle recording – m: change mode
Mode:
MANUAL
Training Status:
X8 Speed