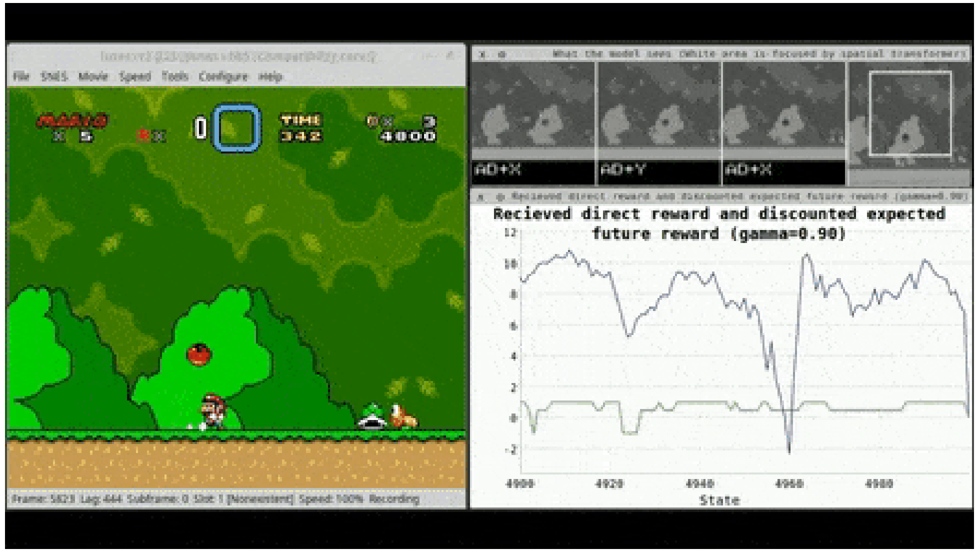
Deep Learning Workshop

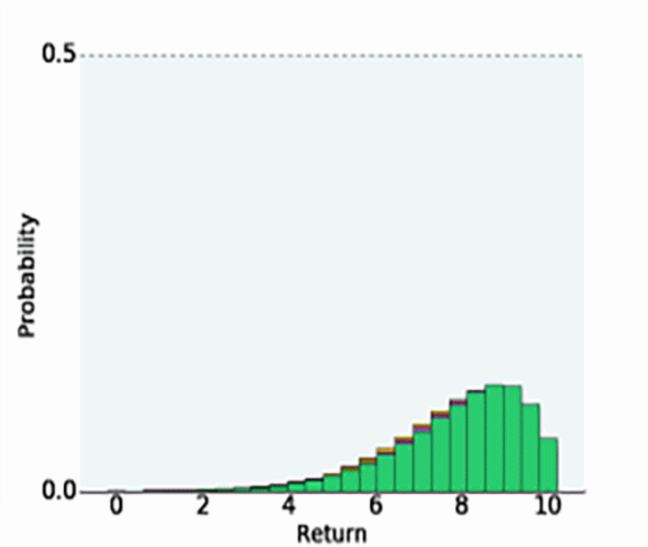
Reinforcement Learning

Instructor: Aaron Low

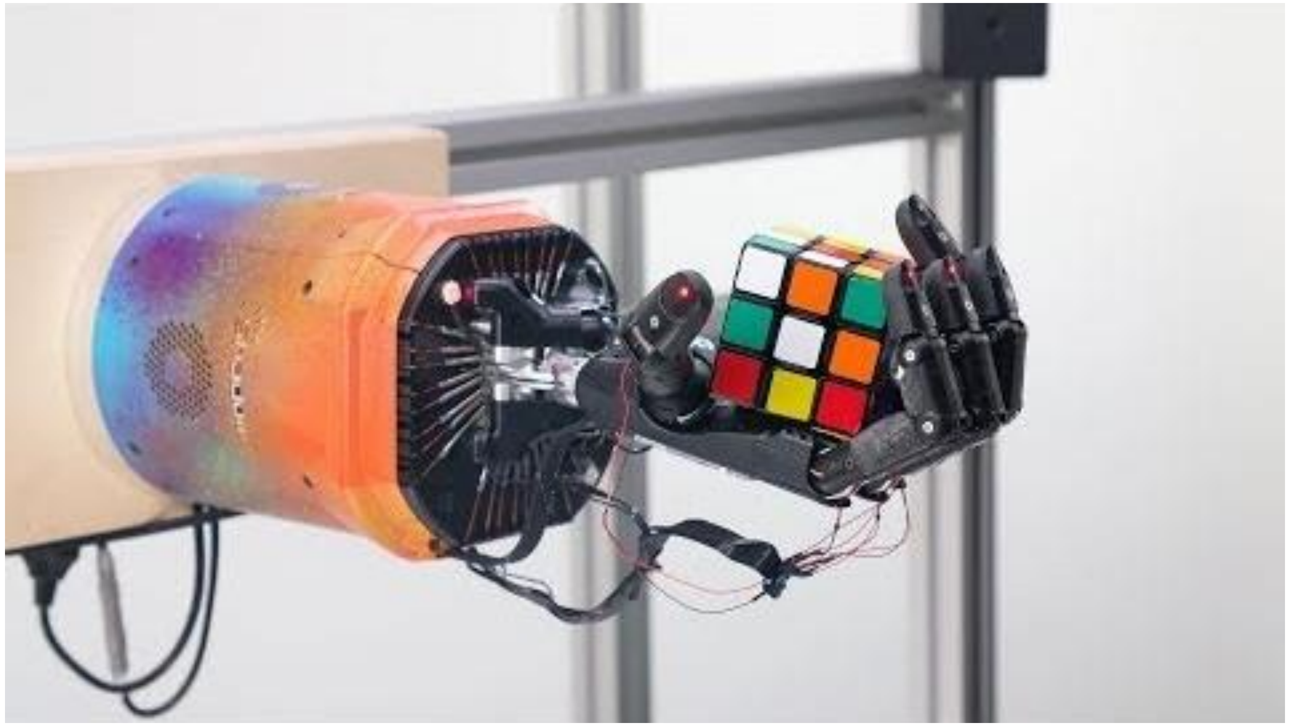
HELP University, Faculty of Computing and Digital Technology

Reinforcement Learning Example: Video Games





Reinforcement Learning Example: Robotics



https://openai.com/blog/solving-rubiks-cube/

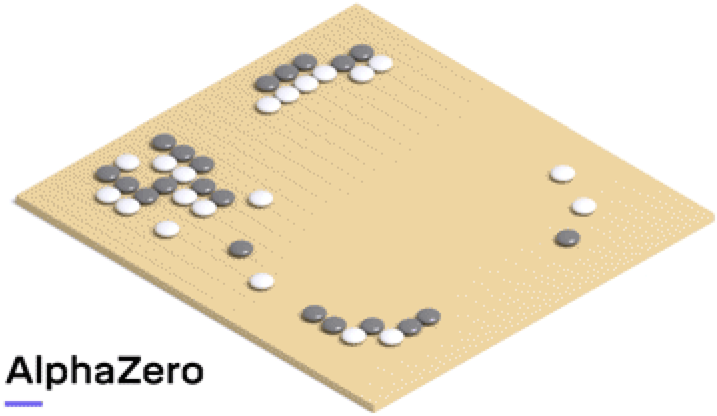
Reinforcement Learning Example: Robotics

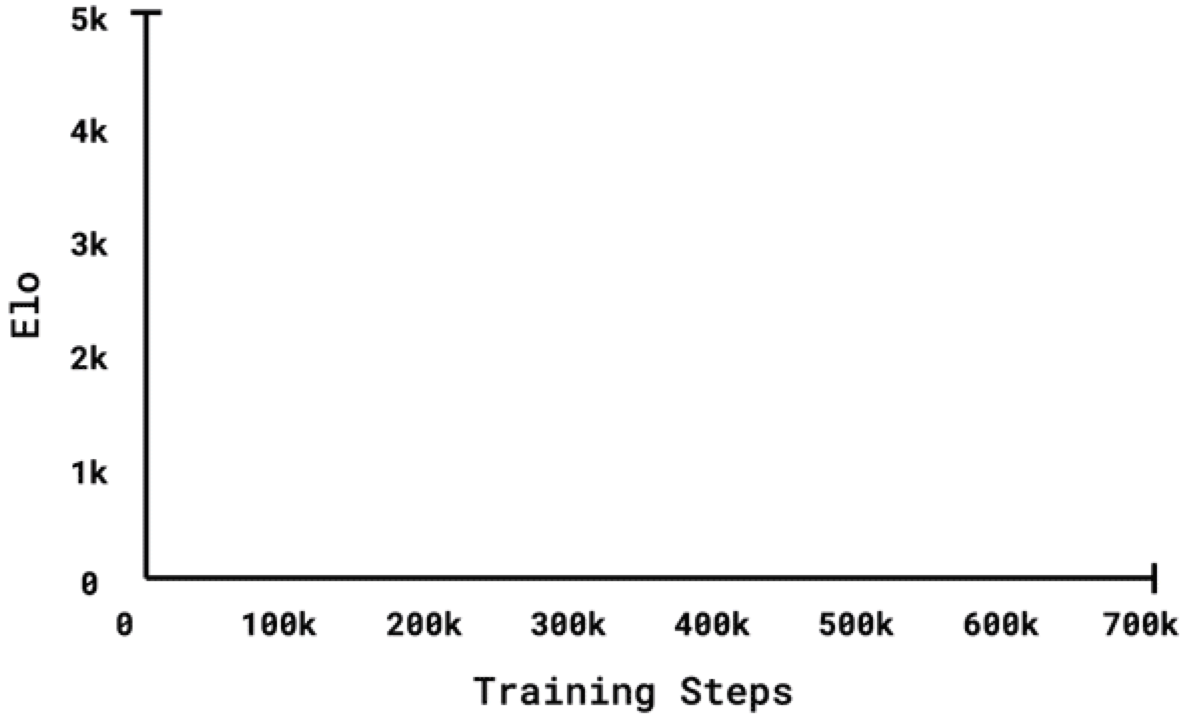
Not actually reinforcement learning

Reinforcement Learning Success in Games

DeepMind’s AlphaGo beats Lee Sedol

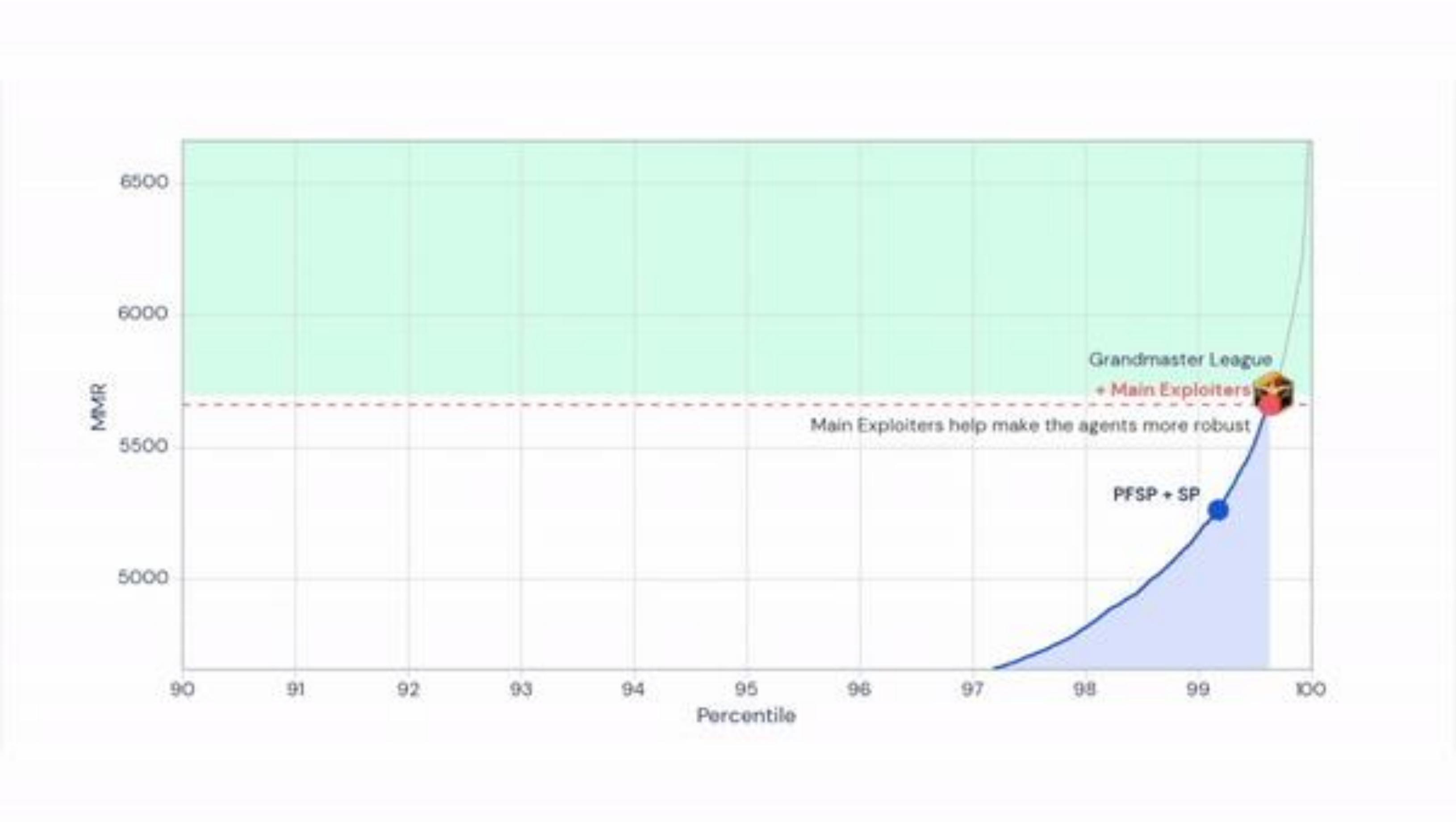
Reinforcement Learning Success in Games





DeepMind’s AlphaZero in Chess, Shogi, Go

Reinforcement Learning Success in Games



DeepMind's AlphaStar in Starcraft 2

Reinforcement Learning Success in Games

OpenAI's OpenAI Five in Dota 2

Reinforcement Learning Success in Games

Facebook and CMU’s Pluribus in Poker

Classes of Learning Problems

Supervised Learning

Data: (x, y)
x is data, y is label

Goal: Learn function to map
$$x \rightarrow y$$

Apple example:

This thing is an apple.

Unsupervised Learning

Data: x
x is data, no labels!

Goal: Learn underlying structure

Apple example:

This thing is like the other thing.

Reinforcement Learning

Data: state-action pairs

Goal: Maximize future rewards over many time steps

Apple example:

Eat this thing because it will keep you alive.

Reinforcement Learning

Agent and Environment

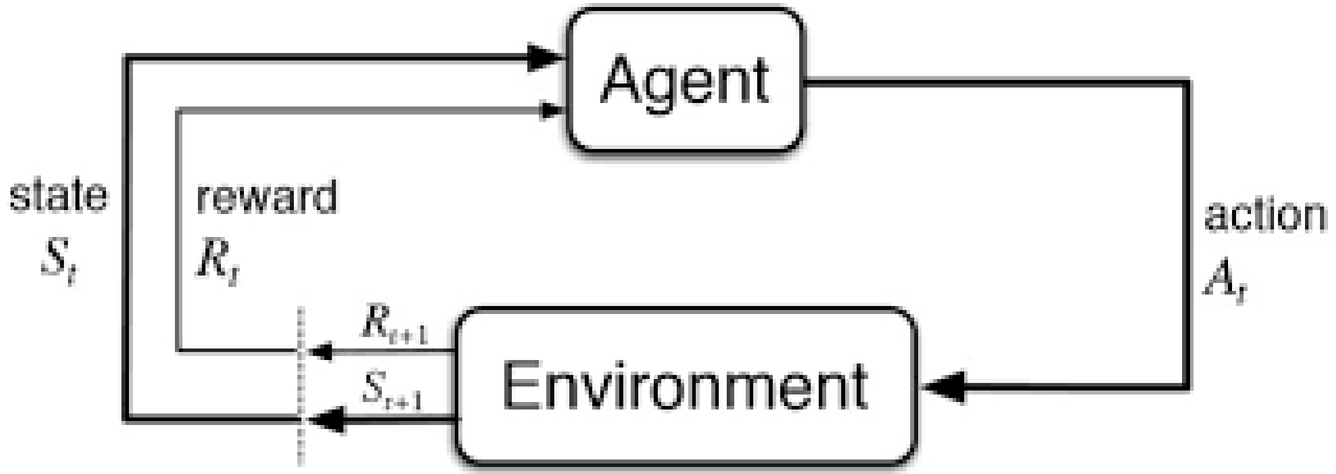
Agent: The intelligent unit that learns what actions to take

Environment: The surrounding world which provides feedback to the agent

Action: An action the agent takes in the environment e.g. moving

State: The configuration the agent is currently in

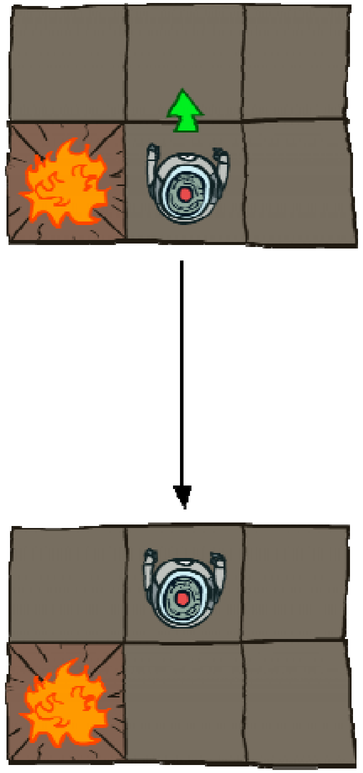
Reward: How much benefit taking an action from the current state provides



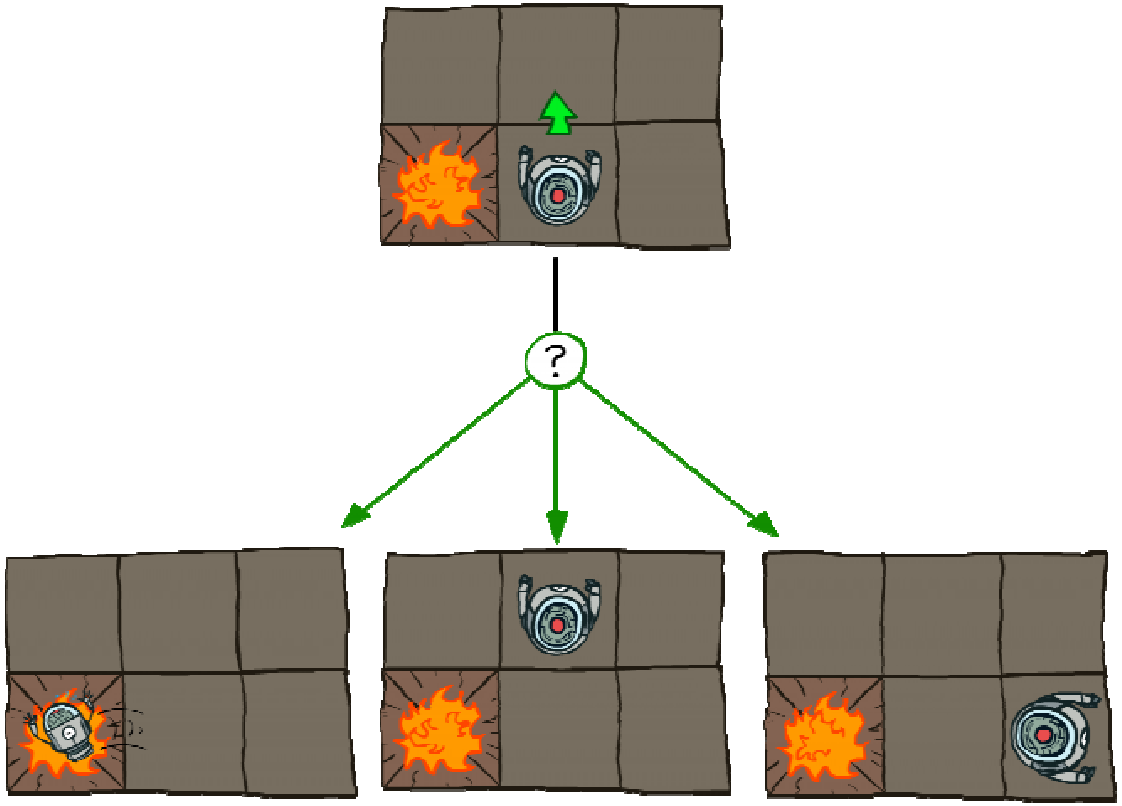
https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

Deterministic vs Stochastic

Deterministic Grid World



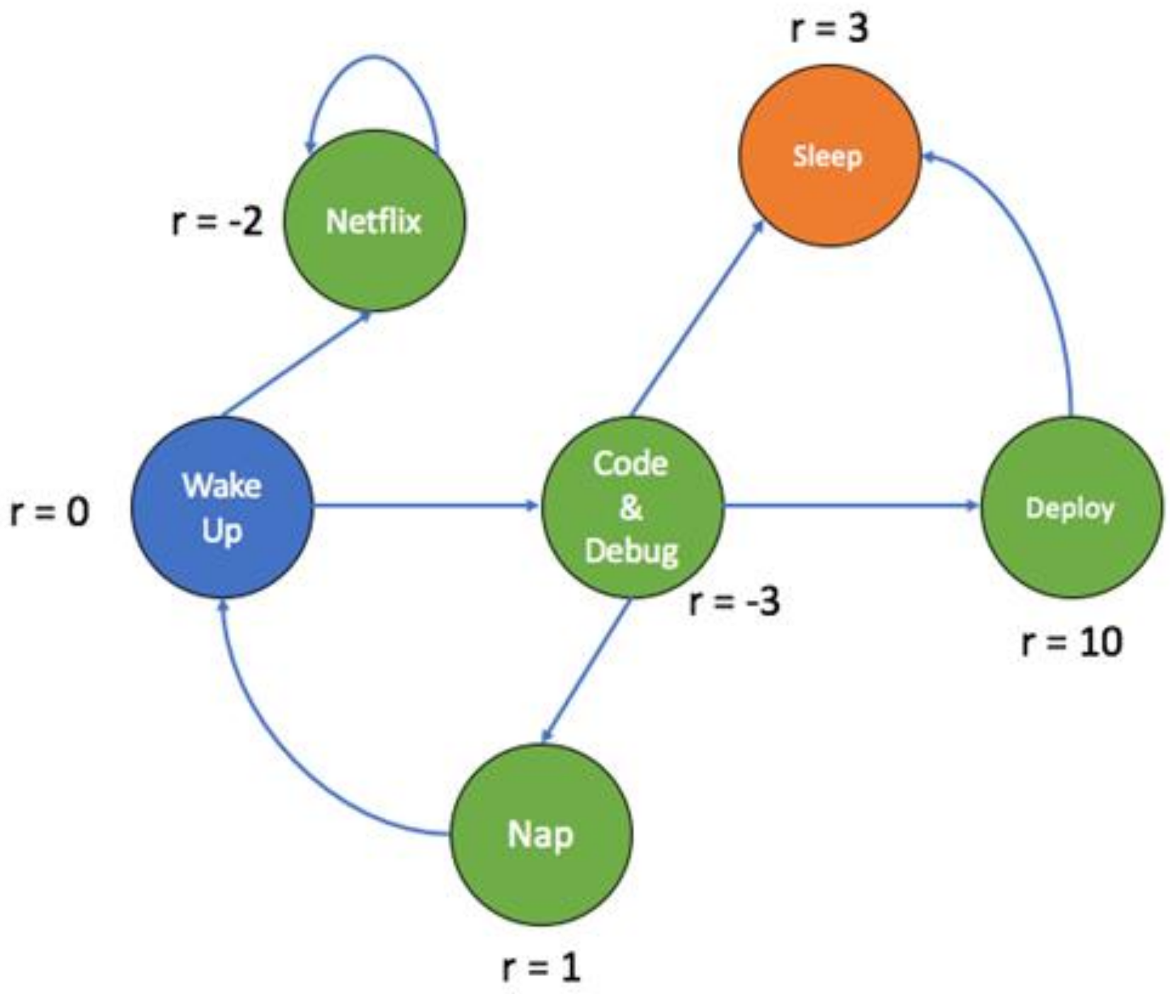
Stochastic Grid World



https://inst.eecs.berkeley.edu/~cs188/sp20/

Markov Decision Process

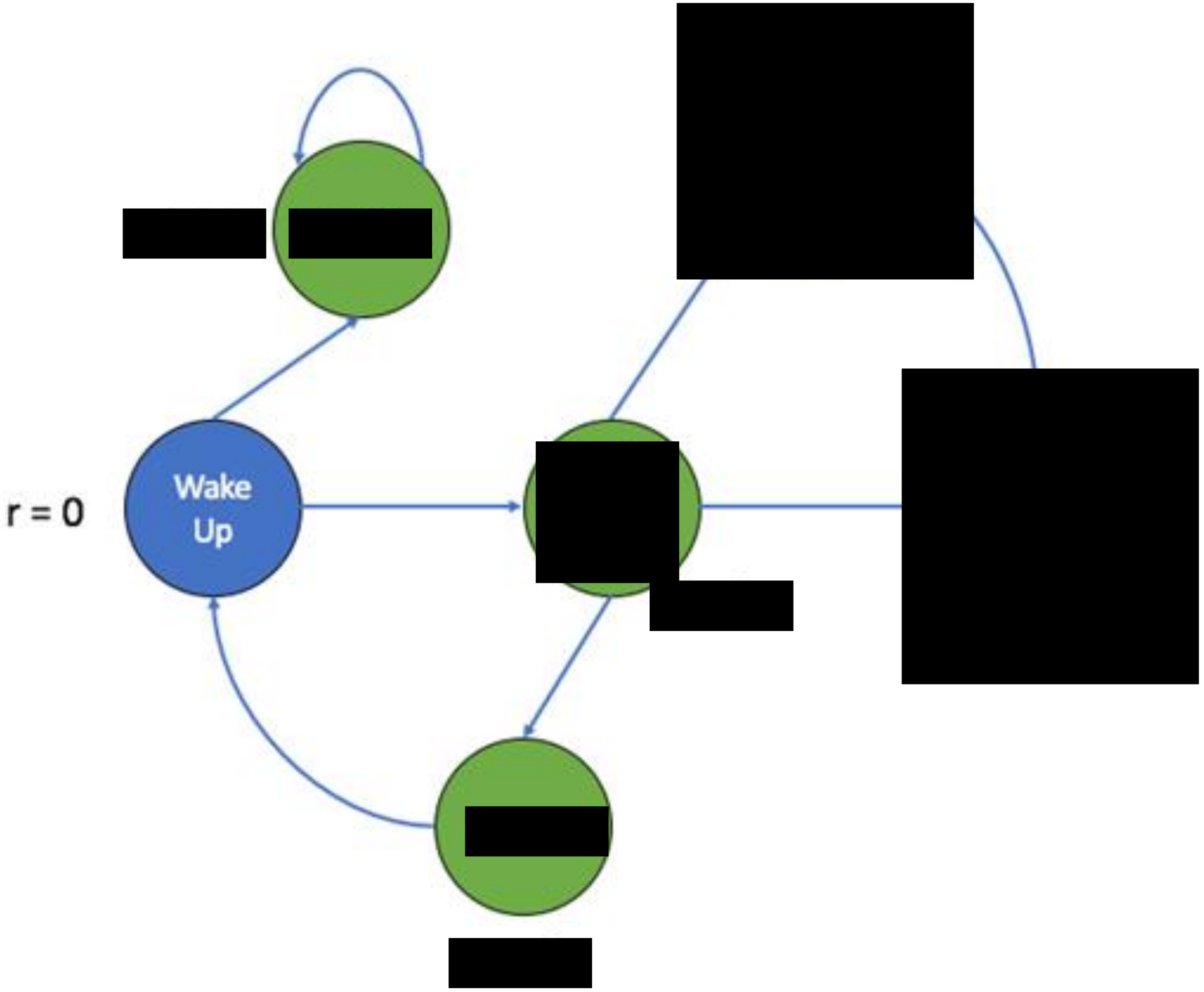
- **Transition function,** $T(s'|s,a)$
  - Probability that action, a from state, s will lead to next state, s’
  - Known as the `**model**`



https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da

Reinforcement Learning

- In reinforcement learning problems, we **don't** know the **Transitions** and the **Rewards**



https://towardsdatascience.com/introduction-to-reinforcement-learning-markov-decision-process-44c533ebf8da

Reinforcement Learning

- Learn to **maximize rewards**

Total **Reward:** $R_t = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots + \gamma^t r_t$

- **Q function:** $Q(s_t, a_t) = E[R_t | s_t, a_t]$
  - Expected total future **reward** an agent in state, s_t, receives by making action, a_t

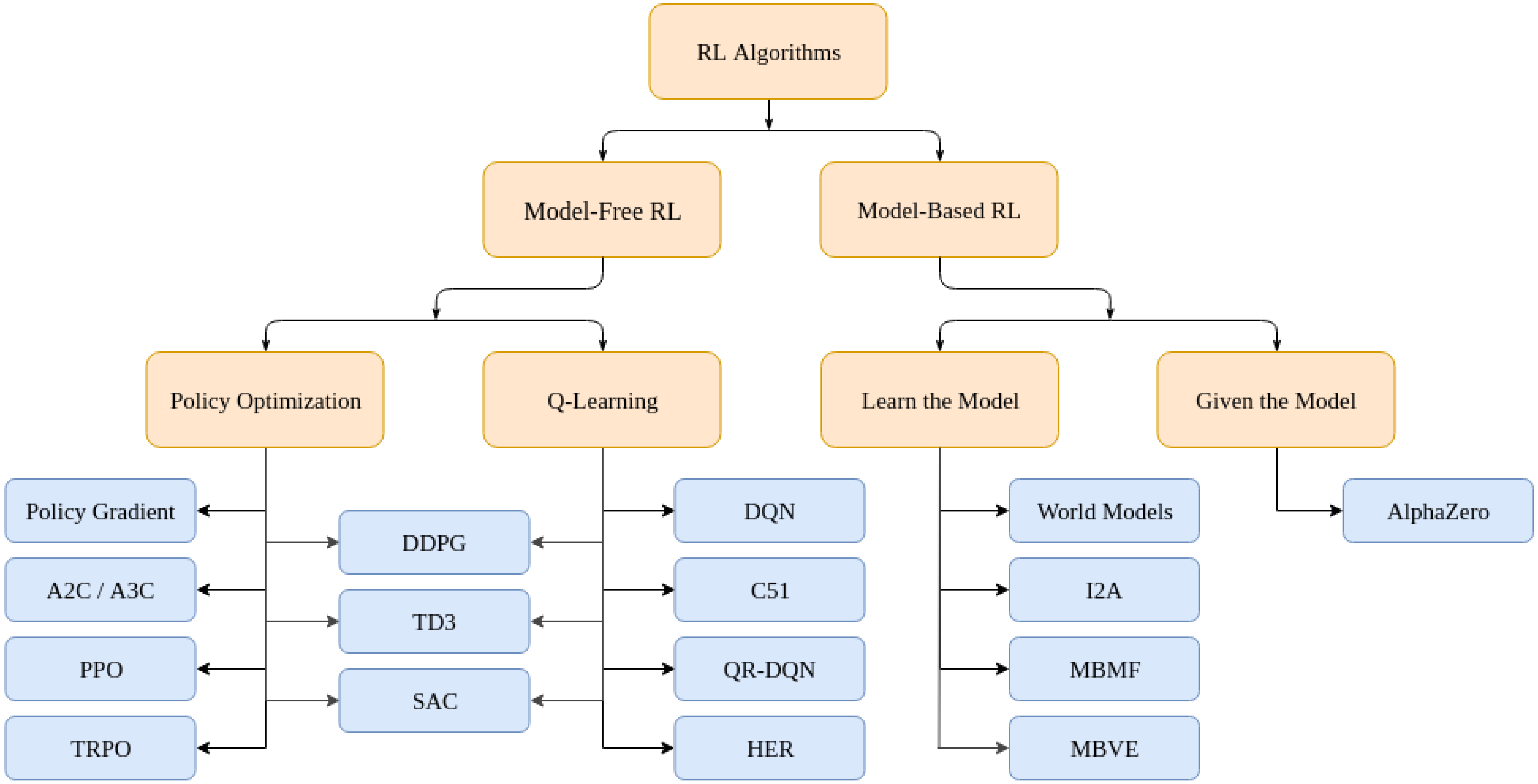
- **Policy:** $\pi(s_t) = argmax_a Q(s_t, a)$
  - The policy chooses the best action that maximizes future **rewards**

- **Discount factor:** γ, 0< γ<1
  - Discount future rewards
  - Immediate rewards are weighted more highly

Importance of Reward: Unintended Consequences

1: MIT Deep Learning Basics: Introduction and Overview Lex Fridman https://www.youtube.com/watch?v=O5xeyoRL95U

Reinforcement Learning Taxonomy



https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

Reinforcement Learning Taxonomy

- **Model-based**
  - Learn an approximate model based on experiences
  - Use approximate model to make decisions
  - "I have an idea of where I will be and what the reward will be if I take this action from this state"
- **Model-free**
  - Learn optimal policy or optimal Q-values directly for each action in each state from experience
  - "I know that since I'm in this state, taking this action will be the best"

- Can the agent make predictions about what the next state and reward will be before it takes each action?
  - If yes, then it is **Model-based**

Q-Learning

- Estimate $Q(s, a)$ that **maximizes future reward**
- Use any policy and keep updating (s, a) pairs

How to update Q value

Discount $0<\gamma<1$

Learning Rate $0< \alpha<1$

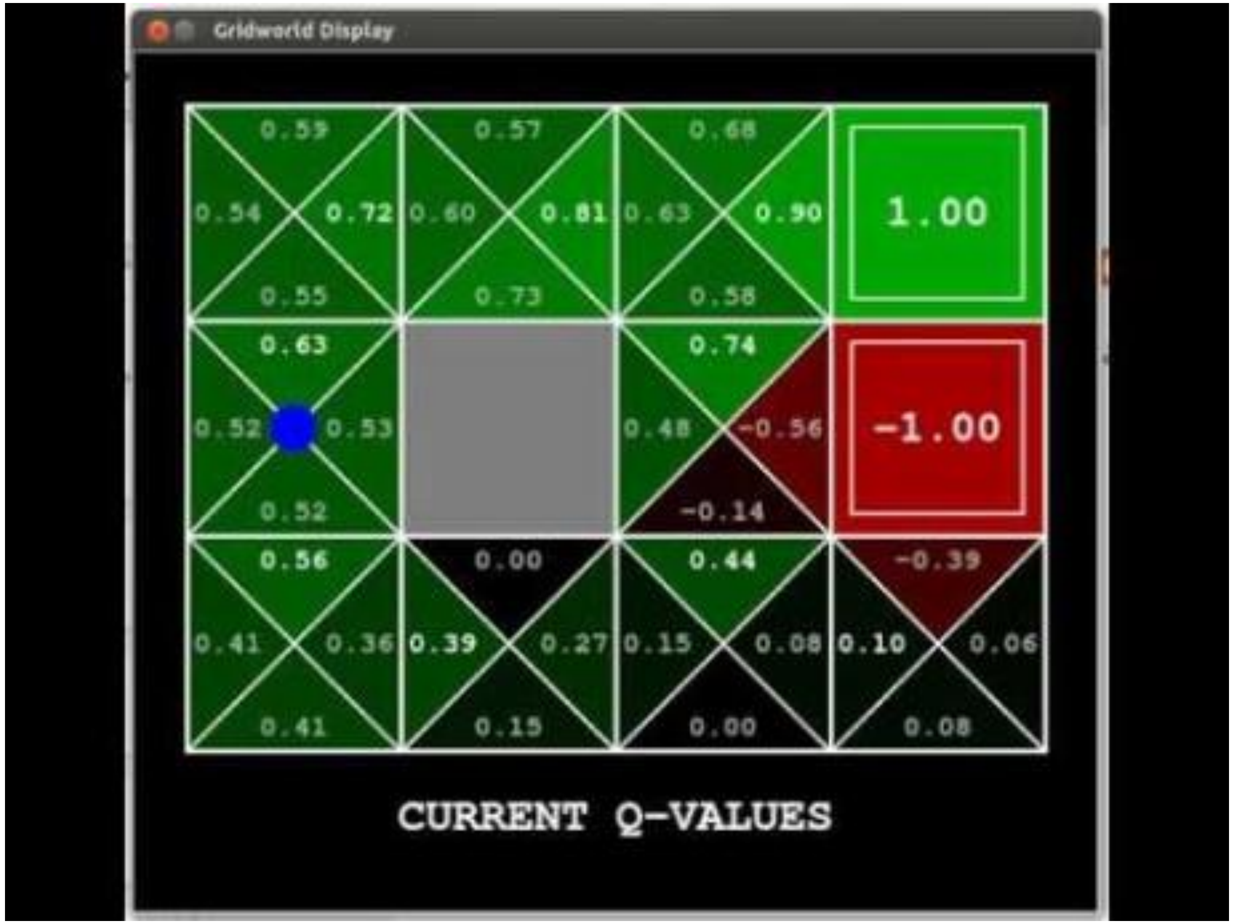
$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha(\boldsymbol{R_{t+1}} + \gamma max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t))$$

New Q value

Old Q value

Reward

Q-Learning: Value Iteration



Q-Learning: Value Iteration

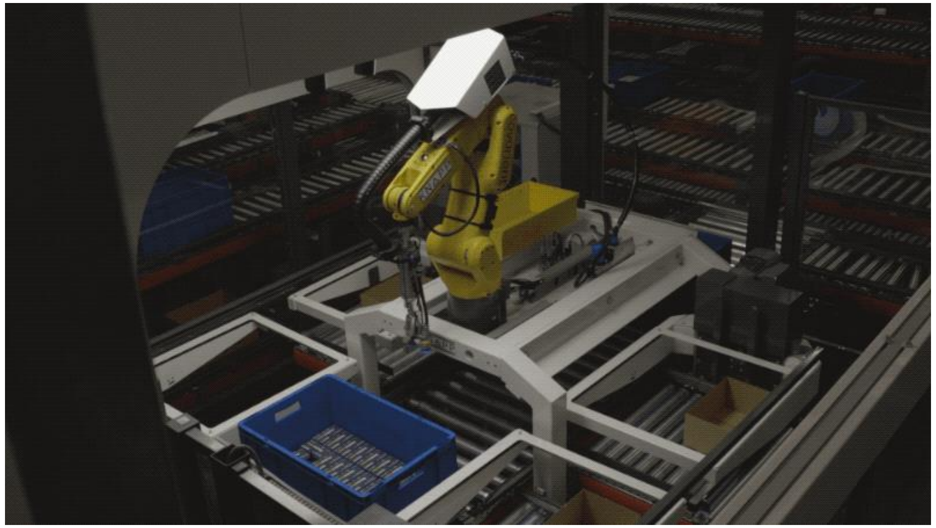
- Value Iteration has some **weaknesses**
  - Limited states/actions
  - Doesn't generalize to unseen states
- **Breakout** game
  - State: screen pixels
    - Image size: 84x84
    - 4 frames
    - Grayscale (possible levels)

  => $256^{84*84*4} = 10^{69970} >> 10^{82}$ atoms in the universe

Lex Fridman Introduction to Deep Reinforcement Learning https://www.youtube.com/watch?v=zR11FLZ-O9M&t=1405s

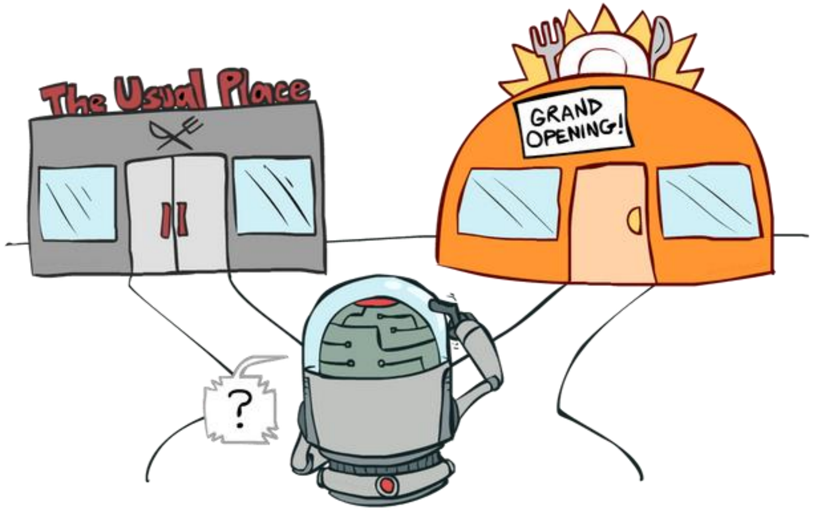
Reinforcement Learning in the Real World

- Training in the real world is not always feasible
- Large number of possible states (we cannot visit all of them and learn about them all)
- Running learning tasks to termination state is NOT always feasible
- Can use simulation to train first before deploying in real world
    - Limited to how well the simulation environment models the real world



Exploration vs Exploitation

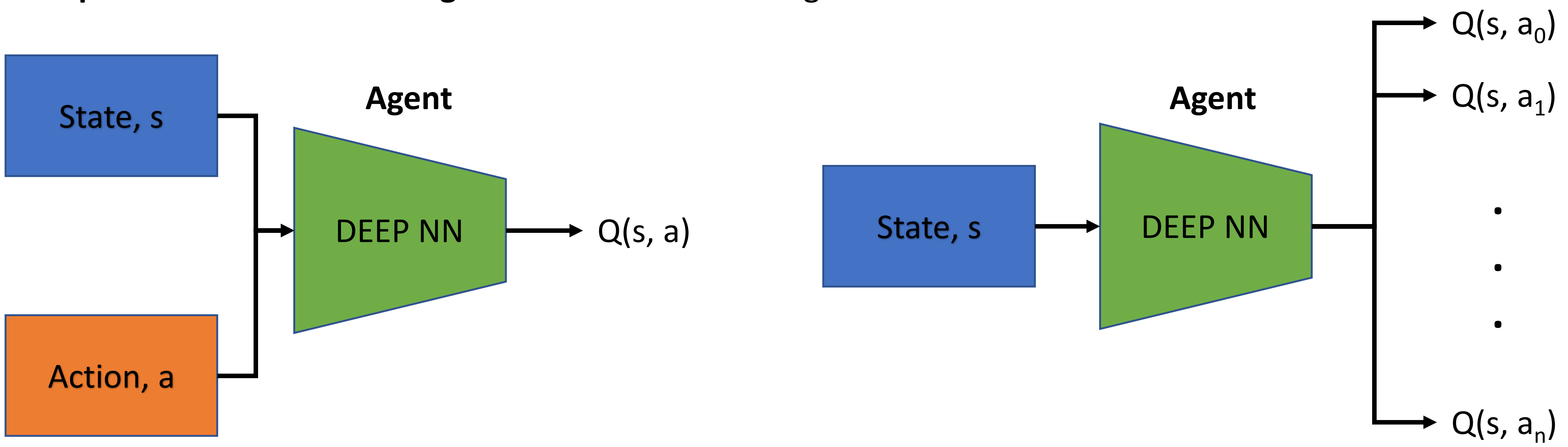
- **Exploitation**
  - Take deterministic best paths greedily
  - At the start (before training), this won't work well as the agent knows nothing about the environment
- **Exploration**
  - Explore areas we don't know anything about
  - Explore by taking random actions at a certain probability, *p*
  - Eventually stop exploring (lower *p* to 0)



https://inst.eecs.berkeley.edu/~cs188/sp20/

Deep Reinforcement Learning: Deep Q Network (DQN)

Deep Reinforcement Learning: Reinforcement Learning + Neural Networks

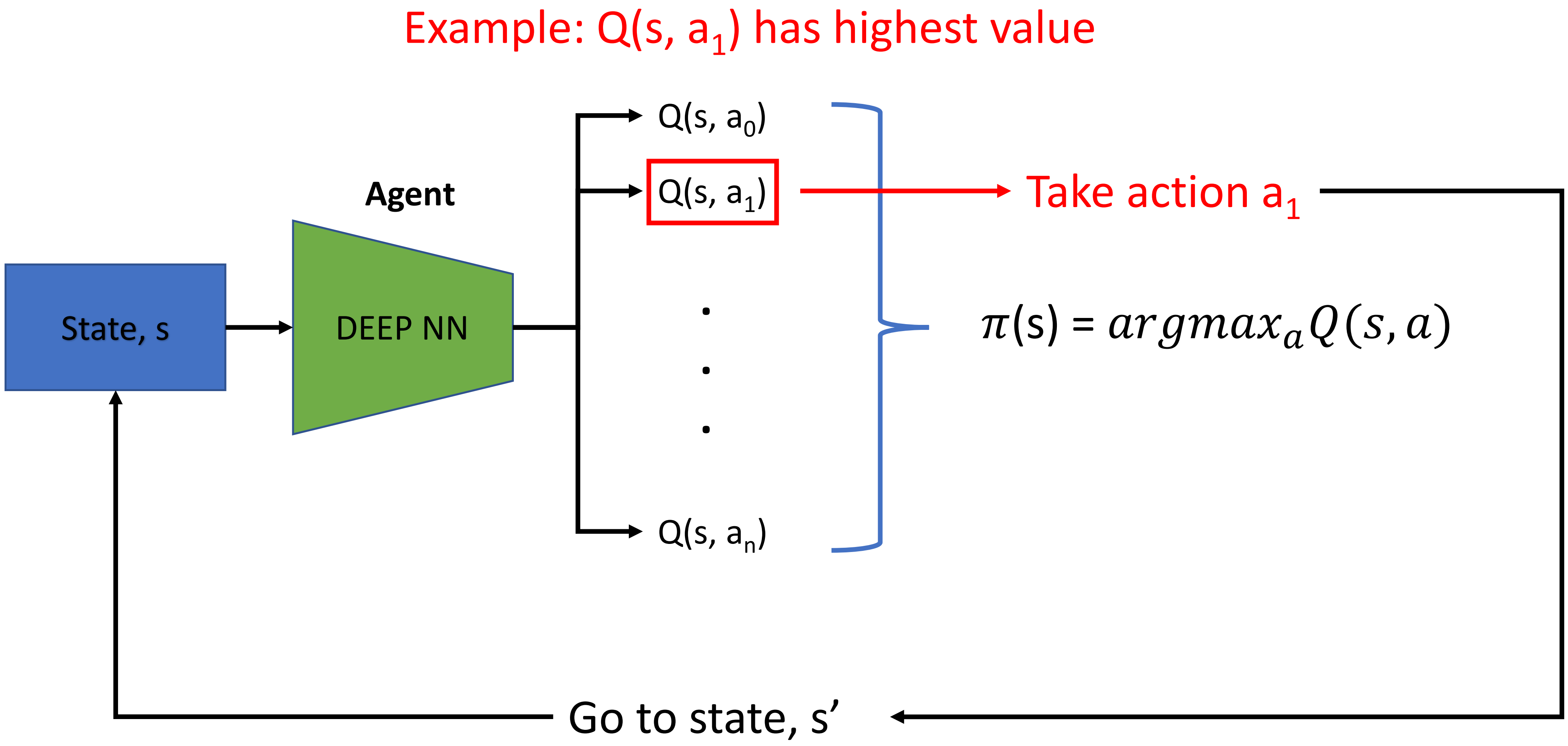


- Use NN to learn Q-function and then use to infer the optimal policy
- Obtain target by running agent in environment

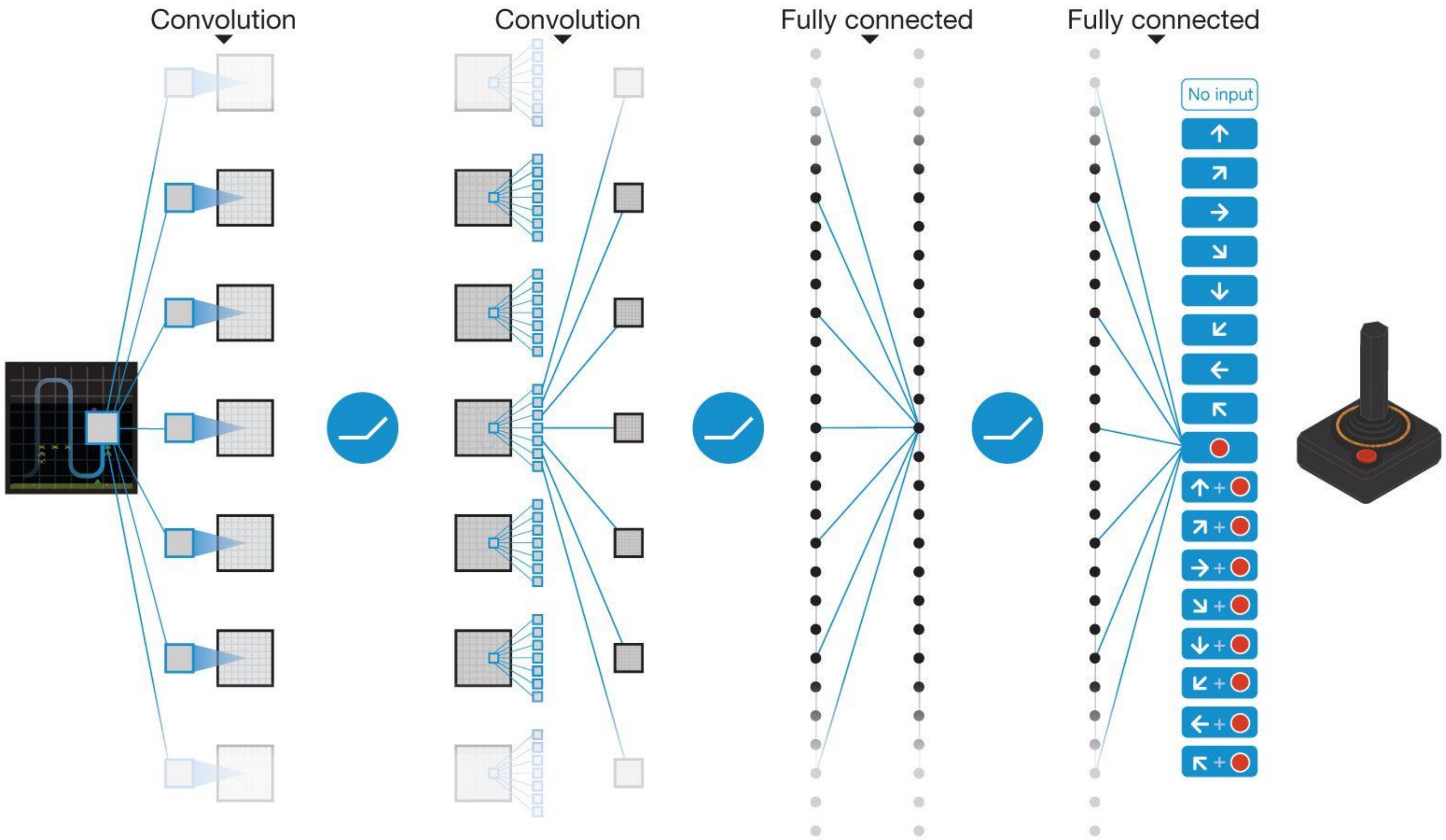
Loss Function: $L = E[(\underbrace{r + \gamma \max_{a'} Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{predicted}})^2]$

Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." arXiv preprint arXiv:1312.5602 (2013).

Deep Q Network (DQN)

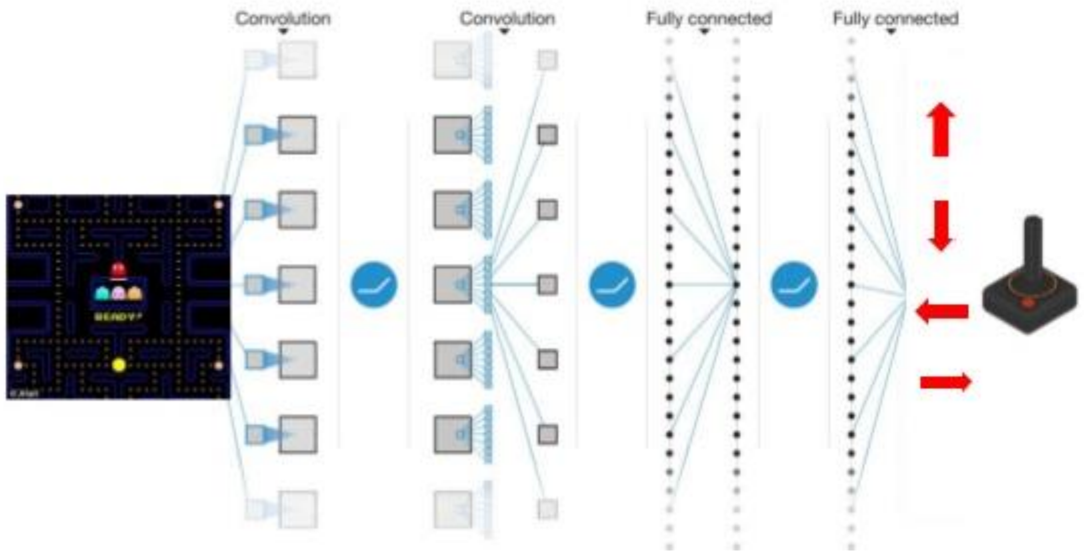


Deep Q Network (DQN)



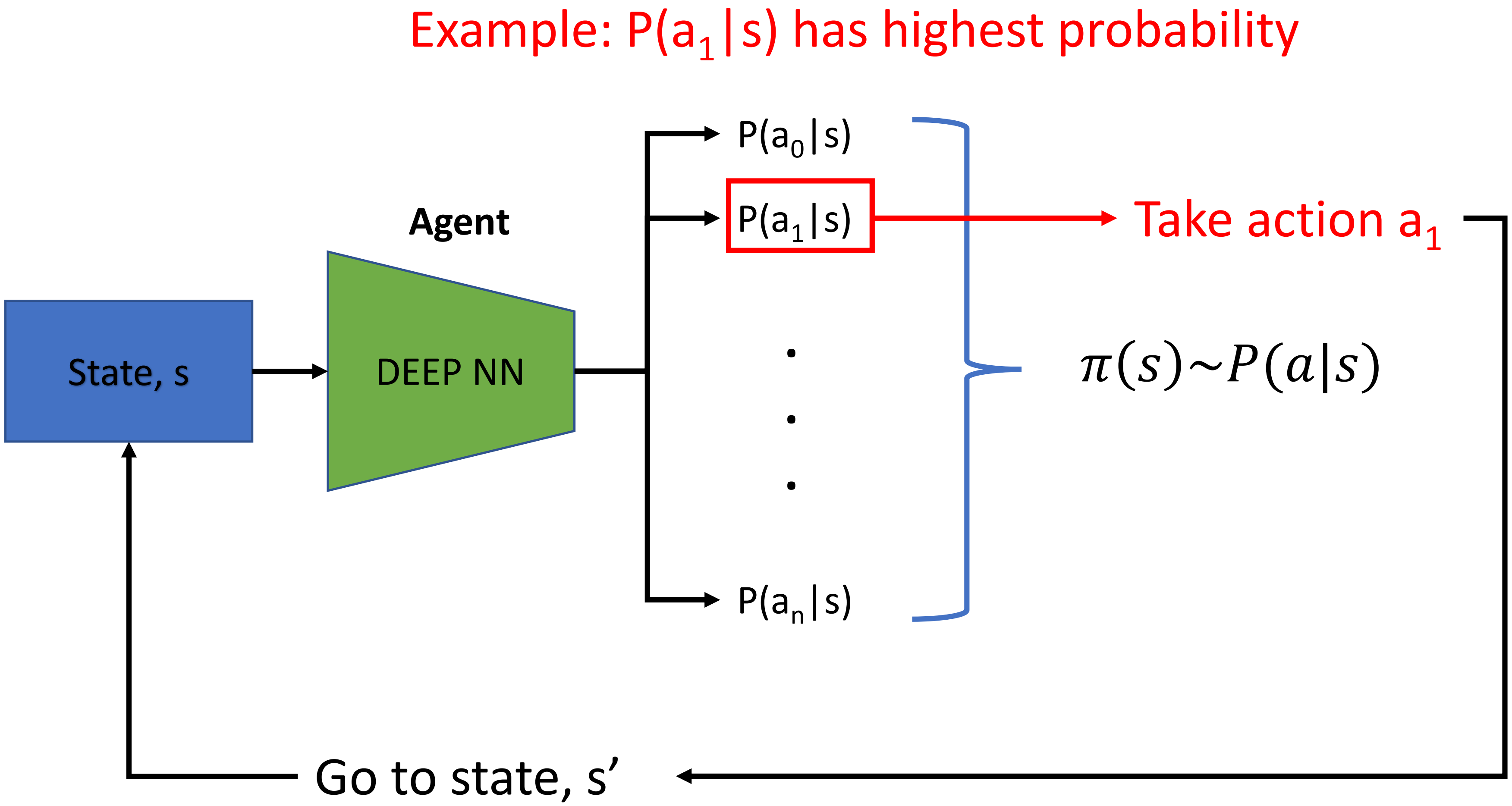
Deep Q Network (DQN)

- **Weaknesses**
  - Cannot handle **large** action space
  - Cannot handle **continuous** action space
  - Cannot learn **stochastic** policy (policy is deterministic)



Policy Gradient (PG)

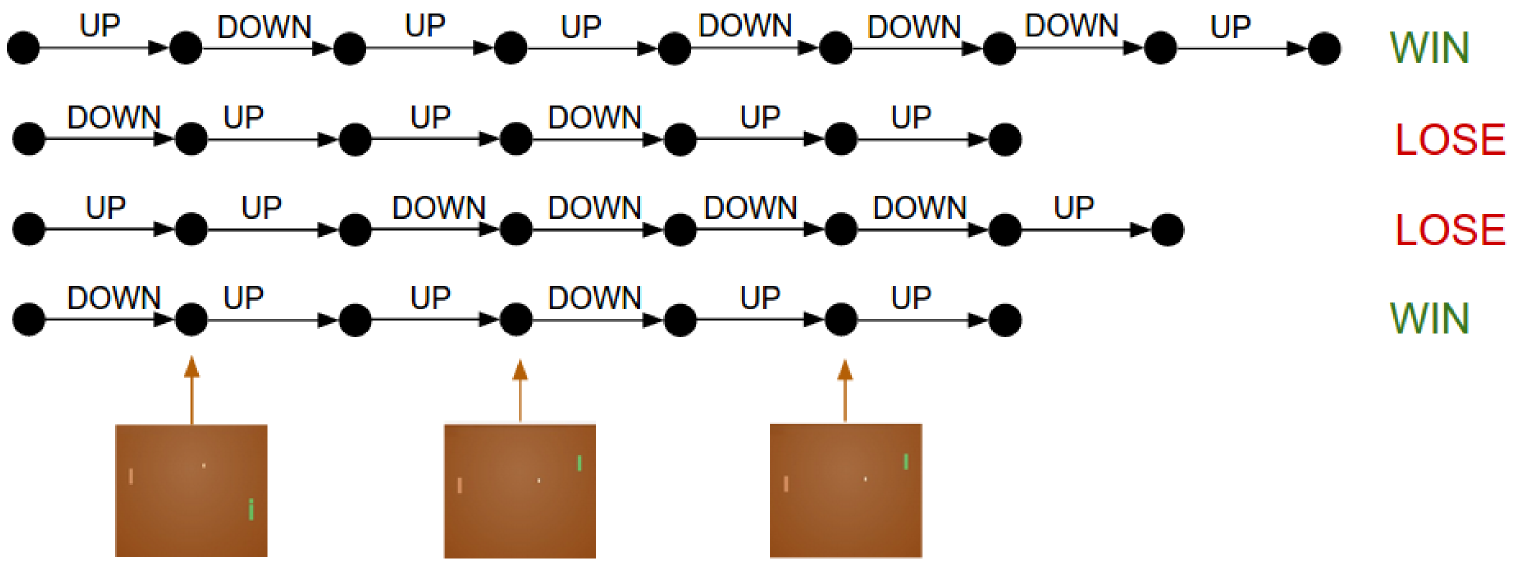
- On-policy (DQN is off-policy)
- Directly optimize the policy $\pi(s)$



Policy Gradient (PG): Training

- **Steps:**
  1. Initialize agent
  2. Run policy until termination
  3. Record all states, actions and rewards
  4. Decrease probability of actions that resulted in low reward
  5. Increase probability of actions that resulted in high reward

Example: Pong Game

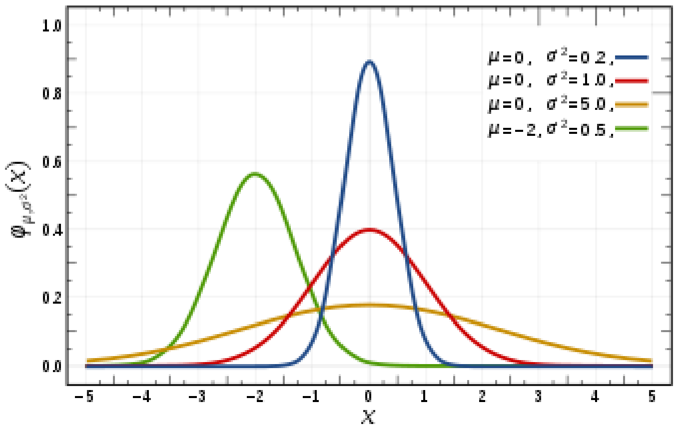


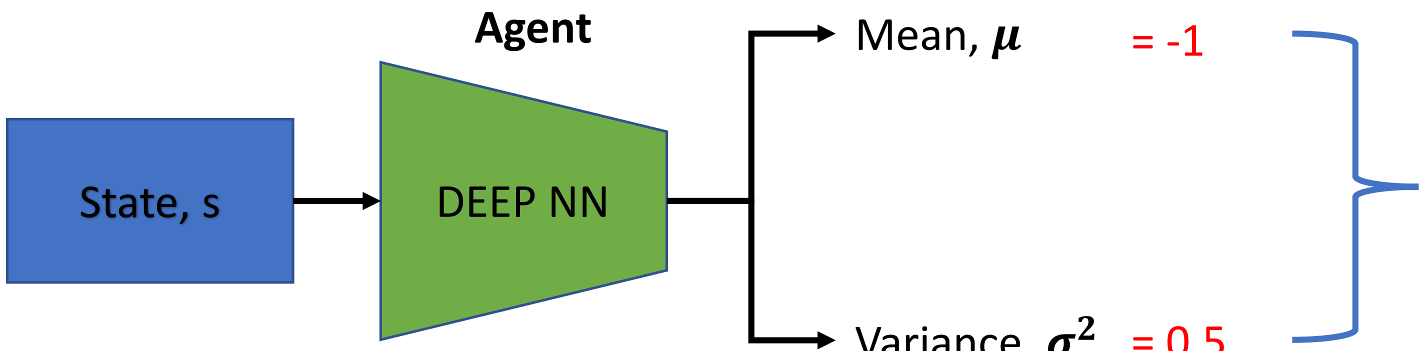
Policy Gradient (PG): Continuous Action State

- On-policy (DQN is off-policy)
- Directly optimize the policy $\pi(s)$

- **Example:** Assume action space follows Gaussian distribution
- Predict parameters of distribution only
- Sample from distribution based on predicted parameters

Normal distribution, $N(\mu, \sigma^2)$





$$\pi(s) \sim P(a|s) = N(\mu, \sigma^2)$$

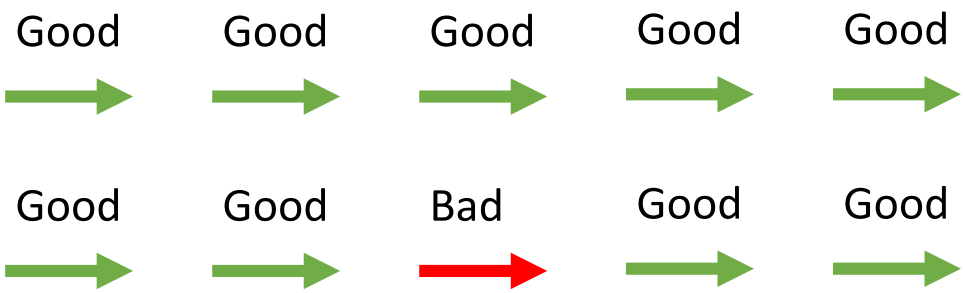
Take a sample from distribution = -0.8

Continuous: How fast left or right should I go?

Discrete: Left or Right?

Policy Gradient (PG): Continuous Action State

- **Weaknesses**
  - Needs more data
  - Less stable during training
  - Poor credit assignment to (s, a) pairs for delayed rewards
    - Calculating reward at the end means all actions will be averaged as good if total reward is high



Questions?