Reinforcement Learning

UNSUPERVISED LEARNING

Flavors of supervision

Supervised

All labels known

Unsupervised

No labels

Semi-supervised

Some labels known

Representation learning

Transfer learning

Self-supervised learning

Surrogate task from pseudo labels

Reinforcement learning

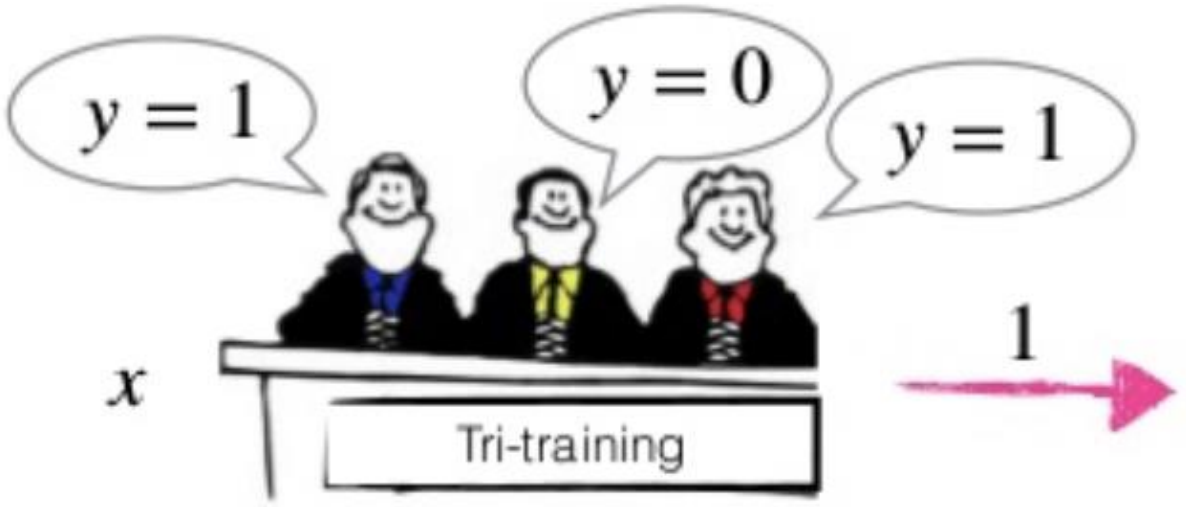
Agent gathers data, reward function known

Idea 2: Tri-training with disagreement

If all models agree, it might be an easy data point.

Not so useful

Only use data when two agree and the third disagree



https://www.slideshare.net/SebastianRuder/strong-baselines-for-neural-semisupervised-learning-under-domain-shift

Consistency training/teacher-student

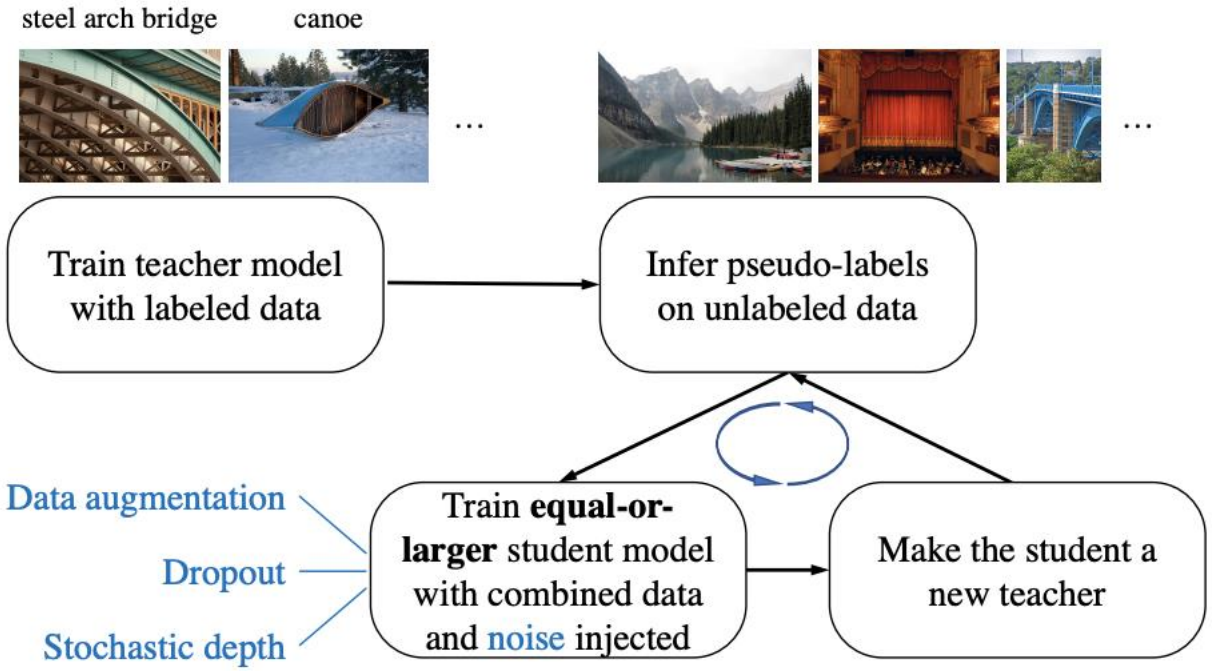
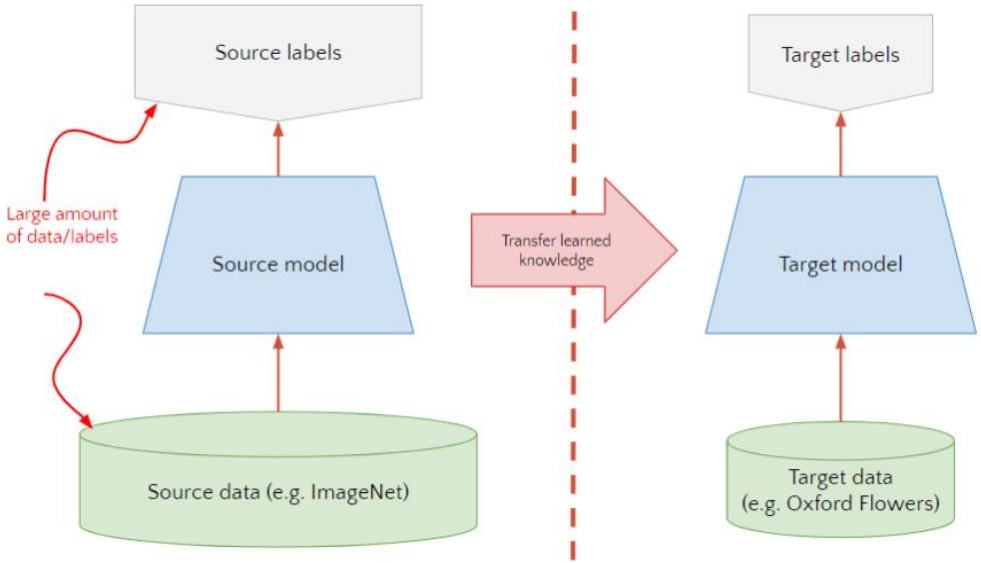


Figure 1: Illustration of the Noisy Student Training. (All shown images are from ImageNet.)

https://arxiv.org/pdf/1911.04252.pdf

Transfer learning (basics)

- We know networks captures good representations
- Can we use it for other tasks?
- Use trained networks to initialize a new network for a different task.
- Re-train the network using SGD on new data.



For CV tasks, we call the pre-trained network backbones

Self-supervised learning

Unsupervised learning trained using supervised learning techniques

Cleverly exploit property of the data to create pseudo labels

Mostly used for representation learning

Need small supervised data to map to useful task

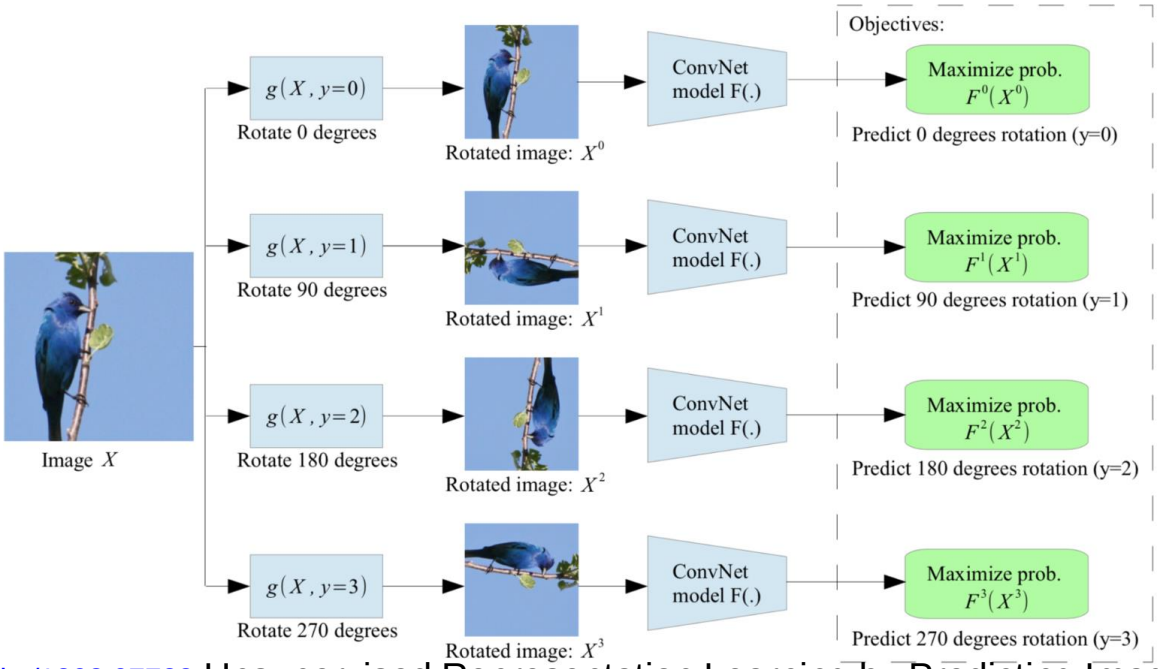
Examples

Text

Predict masked text - BERT

Images

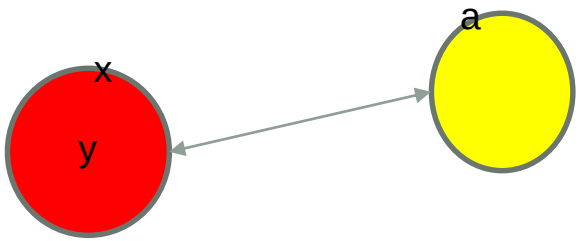
Predict missing patches, predict orientation, etc.

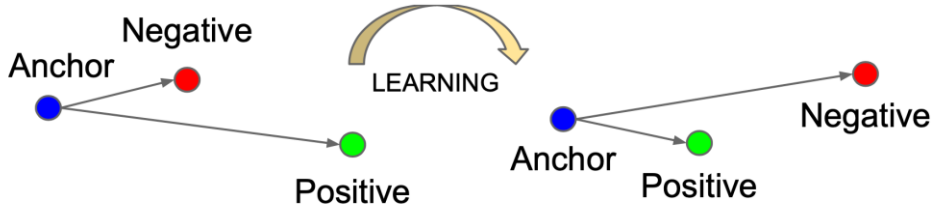


https://arxiv.org/abs/1803.07728 Unsupervised Representation Learning by Predicting Image Rotations

Contrastive training

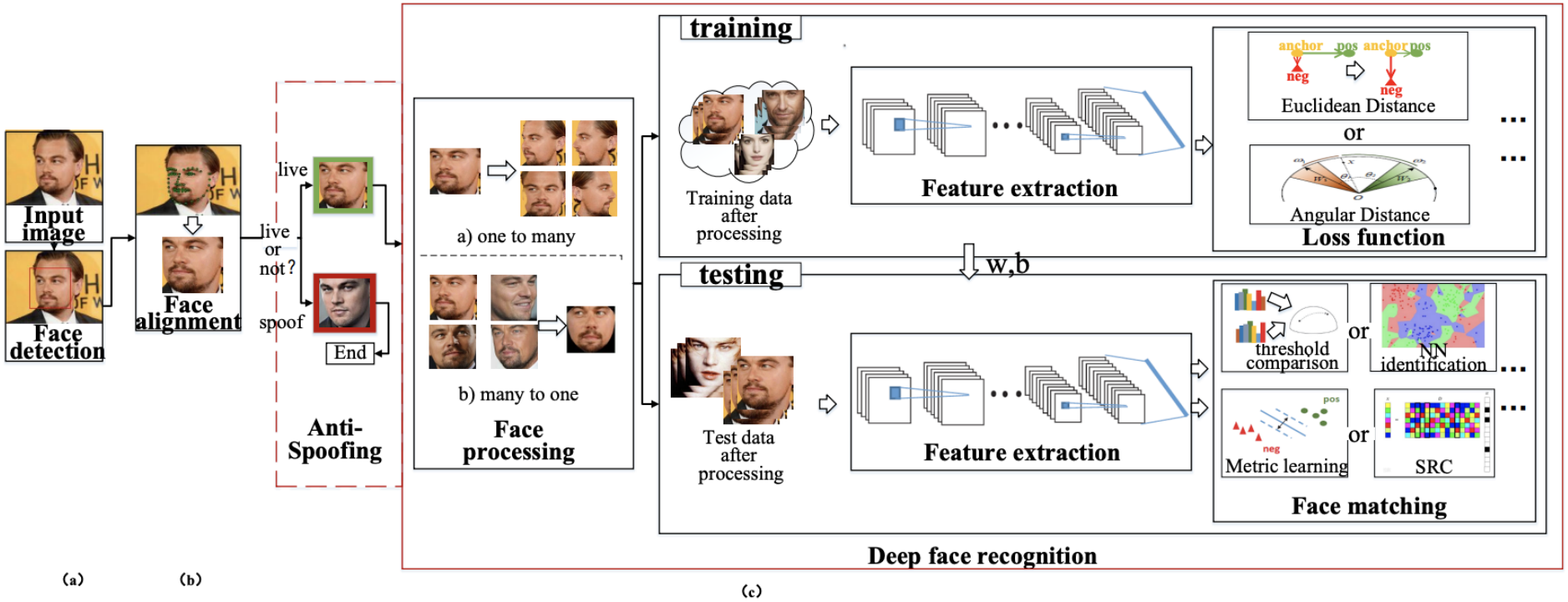
- Consistency training focus on pulling similar things together while ignoring noise
- Contrastive training focus on pushing different things away
- Contrastive loss are key in face verficiation task
- These two are often times used together, and the clear differentiation between the two are vague





https://arxiv.org/abs/1503.03832

Deep face verification



https://arxiv.org/pdf/1804.06655.pdf

Triplet loss

- One of the earliest deep learning contrastive loss
- Positive must be closer to anchor than some margin
- Uses Euclidean distance (features are normalized to unit norm)

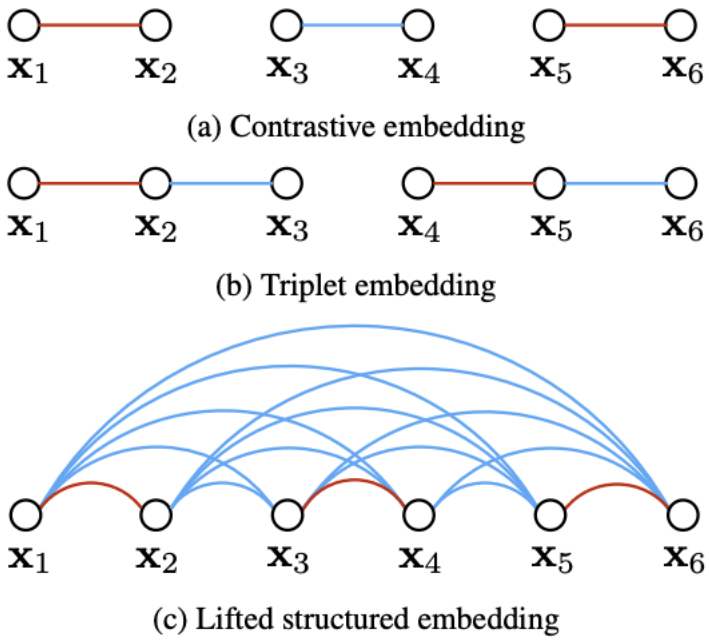
$$\sum_{i}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

Take positive only max(0,x)
Makes gradient not smooth

https://arxiv.org/abs/1503.03832

Dealing with minibatches

- Since we train in minibatches, most modern losses pair positive and negative samples within a minibatch for more efficient computation
  - Compute all pairwise distance within the minibatch



https://arxiv.org/pdf/1511.06452.pdf

NCE (Noise constrastive estimation) loss

- Maximize training data probability while reducing noise probability
- Learn in a constrastive way to reduce overhead for normalization (energy-based models)
  - Max LogP(data) – Log P(noise or negative samples)
  - Ex: used to train word embeddings such as W2V, too many classes in the softmax output

http://proceedings.mlr.press/v9/gutmann10a.html

InfoNCE

- Similar to NCE but just for categorical cross entropy (instead of binary cross entropy) https://arxiv.org/pdf/1807.03748.pdf
- Given a context vector c, the positive x should be selected rather than the negative x
- Effectively maximize mutual information between c and positive x

$$L_{InfoNCE} = -E[log\frac{f(x,c)}{\sum_{x'} f(x',c)}]$$

$$f(x,c) = exp(\mathbf{z}^T W c)$$

z is encoded x

- f( ) can be any function that describes similarity
- Can be extended to have multiple positive examples in a batch (soft nearest neighbor loss) https://arxiv.org/abs/1902.01889

Soft nearest neighbor loss

- Multiple positive and negative
- Adds temperature (either hyperparameter, or learned)

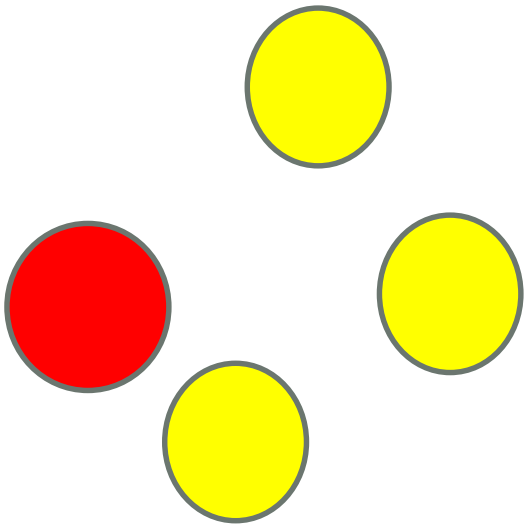
Definition. The *soft nearest neighbor loss* at temperature T, for a batch of b samples (x, y), is:

$$l_{sn}(x, y, T) = -\frac{1}{b} \sum_{i \in 1..b} \log \left(\frac{\sum_{\substack{j \in 1..b \\ j \neq i \\ y_i = y_j}} e^{-\frac{||x_i - x_j||^2}{T}}}{\sum_{\substack{k \in 1..b \\ k \neq i}} e^{-\frac{||x_i - x_k||^2}{T}}} \right) \quad (1)$$

https://arxiv.org/pdf/1902.01889.pdf

Key details to make this work

- Large batch
- Hard/semi-hard negative mining
- Augmentation on the anchor and postive (consistency training)
- Other improvement includes - adding classification loss (CE/softmax loss)



Softmax/angular-based loss

- Another popular construction of the loss is based on angular distance
- Consider a regular softmax/CE loss (only the correct class term, the wrong class is zerod out)

$$L = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

- Can be written as, where W is the weight associated with the class

$$L_i = -\log\left(\frac{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\cos(\theta_{y_i})}}{\sum_j e^{\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\cos(\theta_j)}}\right)$$

Margin in the softmax (L-softmax)

- To classify as class 1

$$\|\boldsymbol{W}_1\|\|\boldsymbol{x}\|\cos(\theta_1) > \|\boldsymbol{W}_2\|\|\boldsymbol{x}\|\cos(\theta_2)$$

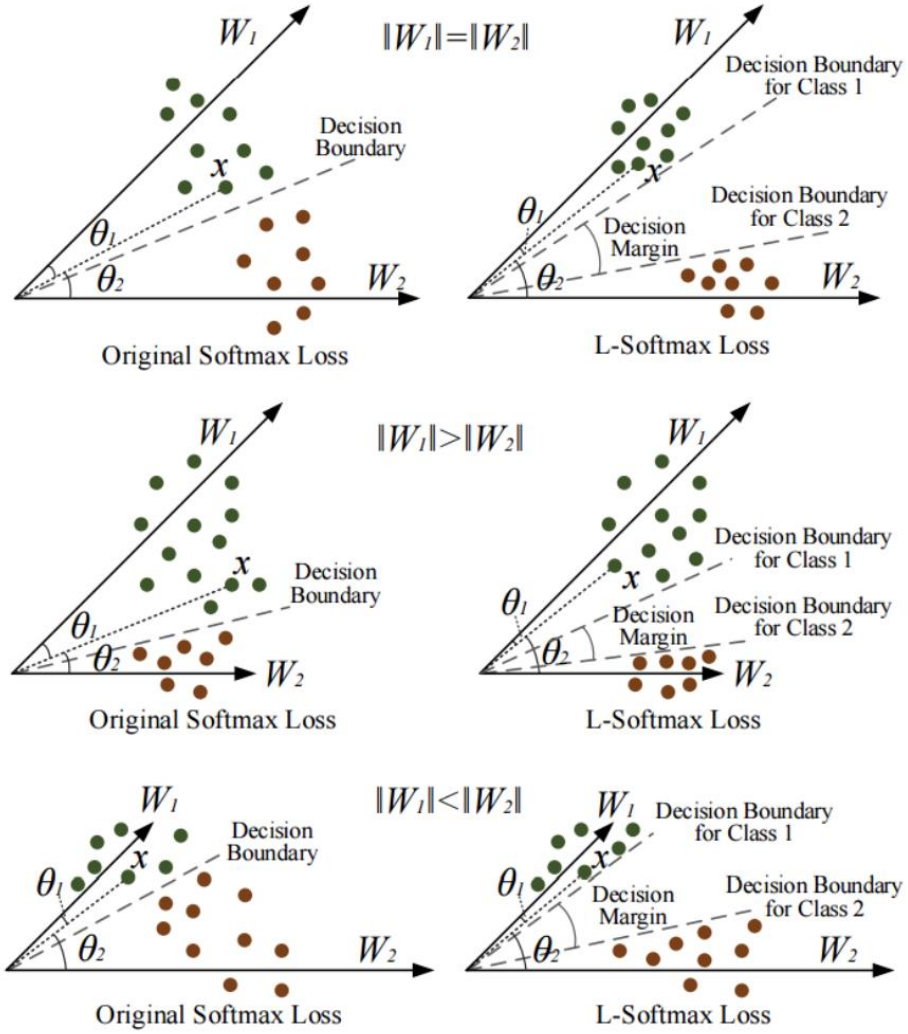
must be true

- We introduce an angular margin m

$$\|\boldsymbol{W}_1\|\|\boldsymbol{x}\|\cos(\theta_1) \geq \|\boldsymbol{W}_1\|\|\boldsymbol{x}\|\cos(m\theta_1) > \|\boldsymbol{W}_2\|\|\boldsymbol{x}\|\cos(\theta_2).$$

$$(0 \leq \theta_1 \leq \frac{\pi}{m})$$

Angular margin effect



https://wyliu.com/papers/LiuICML16_slides.pdf

Center loss

- Makes the data bunch up towards the centroid
- Unlike k-mean cluster, centroid is learned via gradient descent (for speed)
- Iterative update between centroid, feature, and classification

Algorithm 1. The discriminative feature learning algorithm

Input: Training data $\{\boldsymbol{x}_i\}$. Initialized parameters θ_C in convolution layers. Parameters W and $\{\boldsymbol{c}_j|j=1,2,...,n\}$ in loss layers, respectively. Hyperparameter λ, α and learning rate μ^t. The number of iteration $t \leftarrow 0$.

Output: The parameters θ_C.

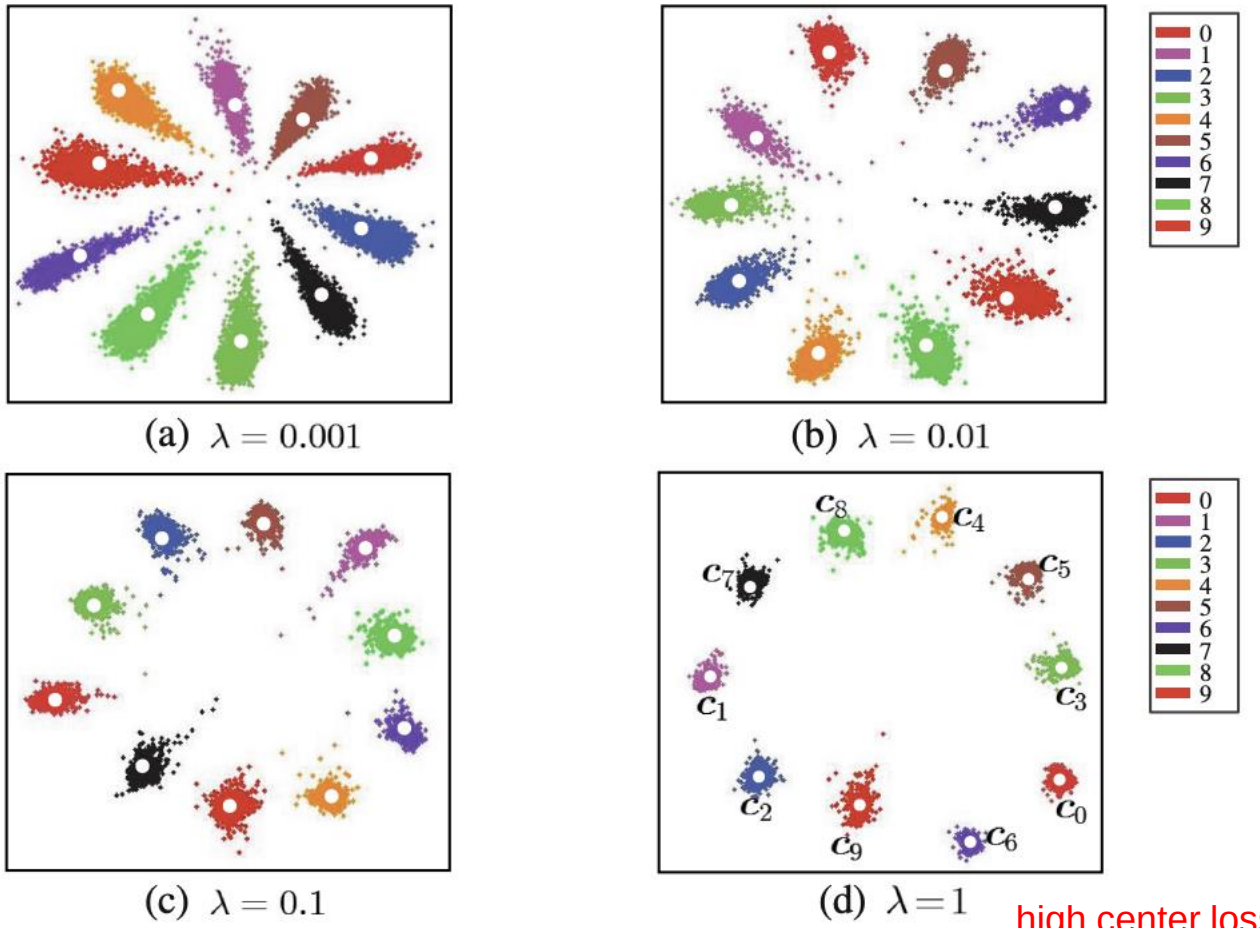
1: **while** not converge **do**
2: $t \leftarrow t+1$.
3: Compute the joint loss by $\mathcal{L}^t = \mathcal{L}_S^t + \mathcal{L}_C^t$.
4: Compute the backpropagation error $\frac{\partial \mathcal{L}^t}{\partial \boldsymbol{x}_i^t}$ for each i by $\frac{\partial \mathcal{L}^t}{\partial \boldsymbol{x}_i^t} = \frac{\partial \mathcal{L}_S^t}{\partial \boldsymbol{x}_i^t} + \lambda \cdot \frac{\partial \mathcal{L}_C^t}{\partial \boldsymbol{x}_i^t}$.
5: Update the parameters W by $W^{t+1} = W^t - \mu^t \cdot \frac{\partial \mathcal{L}^t}{\partial W^t} = W^t - \mu^t \cdot \frac{\partial \mathcal{L}_S^t}{\partial W^t}$.
6: Update the parameters $\boldsymbol{c}_j$ for each j by $\boldsymbol{c}_j^{t+1} = \boldsymbol{c}_j^t - \alpha \cdot \Delta \boldsymbol{c}_j^t$.
7: Update the parameters θ_C by $\theta_C^{t+1} = \theta_C^t - \mu^t \sum_i^m \frac{\partial \mathcal{L}^t}{\partial \boldsymbol{x}_i^t} \cdot \frac{\partial \boldsymbol{x}_i^t}{\partial \theta_C^t}$.
8: **end while**

$$\mathcal{L} = \mathcal{L}_S + \lambda \mathcal{L}_C$$

$$= -\sum_{i=1}^{m} \log \frac{e^{W_{y_i}^T \boldsymbol{x}_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T \boldsymbol{x}_i + b_j}} + \frac{\lambda}{2} \sum_{i=1}^{m} \|\boldsymbol{x}_i - \boldsymbol{c}_{y_i}\|_2^2$$

https://ydwen.github.io/papers/WenECCV16.pdf

low center loss



high center loss

Fig. 3. The distribution of deeply learned features under the joint supervision of softmax loss and center loss. The points with different colors denote features from different classes. Different λ lead to different deep feature distributions ($\alpha = 0.5$). The white dots ($\boldsymbol{c}_0$, $\boldsymbol{c}_1$,...,$\boldsymbol{c}_9$) denote 10 class centers of deep features. **Best viewed in color.** (Color figure online)

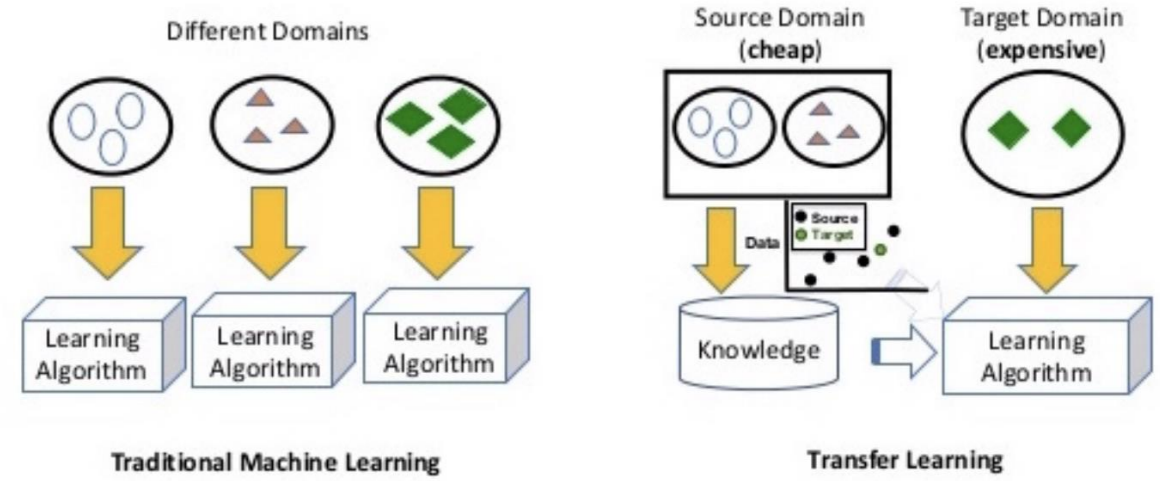
Other related tasks

Domain adaptation (domain-shift)

Classify sentiment on book reviews -> Classify sentiment on restaurant reviews

With labels in the target domain (supervised domain adaptation)

Without labels in the target domain (unsupervised domain adaptation)

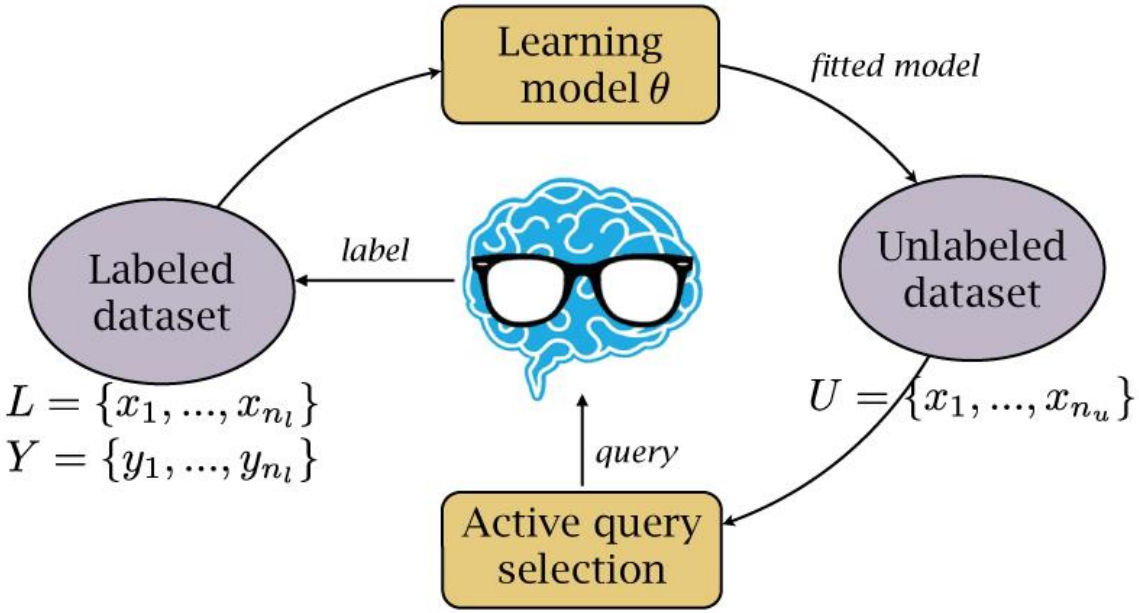


https://www.bnl.gov/nysds18/files/talks/session4/Qian-nysds18.pdf

Other related tasks

Active learning

Find interesting unlabeled data for additional labeling



https://cvlab.epfl.ch/research/research-machinelearning/page-123199-en-html/

Summary

Noise and augmentation

- Network noise and data augmentation

Pull positive and push negative

- Different variants

Further reading

- Contrastive loss

https://lilianweng.github.io/posts/2021-05-31-contrastive

- Self-supervise

https://lilianweng.github.io/posts/2019-11-10-self-supervised/

Reinforcement Learning

What is RL?

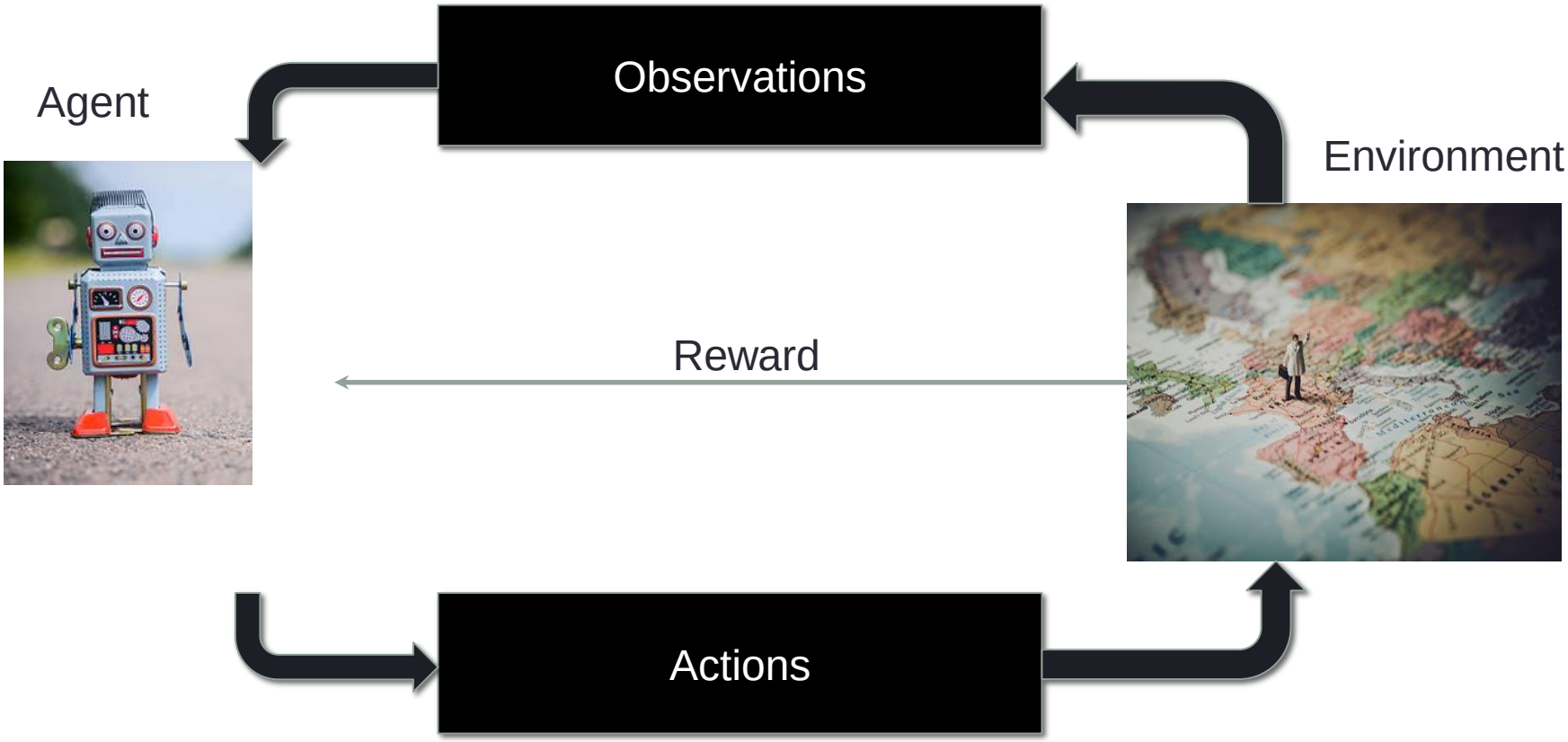
1) A problem
2) A community working on 1)
3) Methods produced by 2) which can be applicable to other problems

Benjamin Van Roy

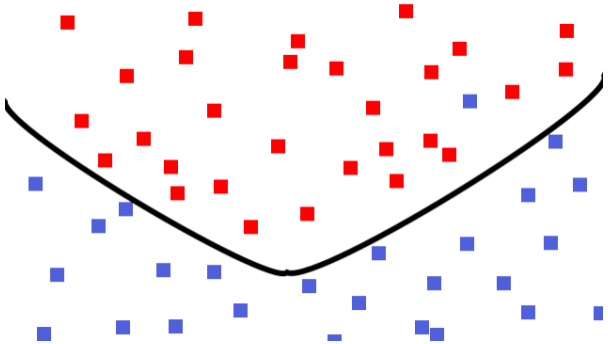
Professor at Stanford University; Research Lead at DeepMind, Mountain View

Topic: Reinforcement Learning

RL problem

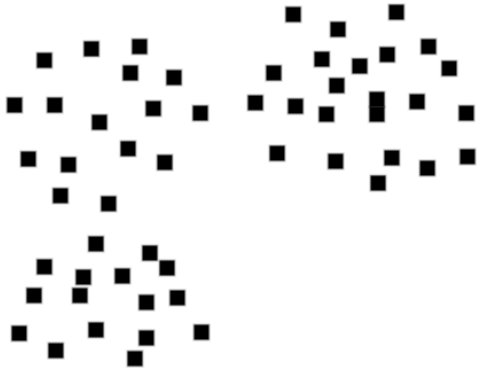


3 Modes of Learning

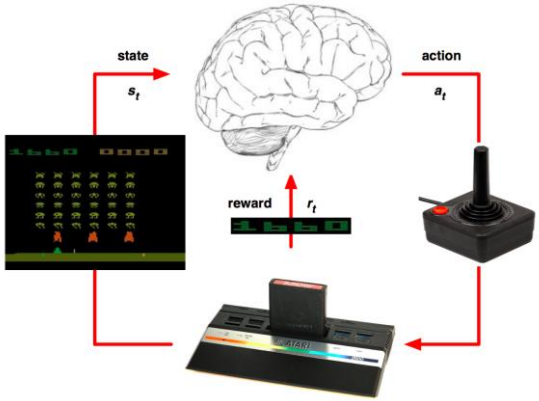


Supervised Learning

Reinforcement Learning

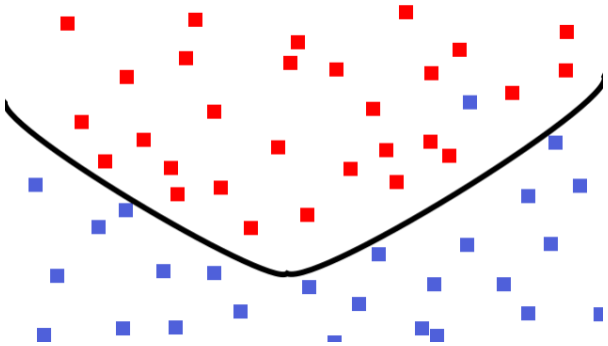


Unsupervised Learning



3 Modes of Learning

Supervised Learning

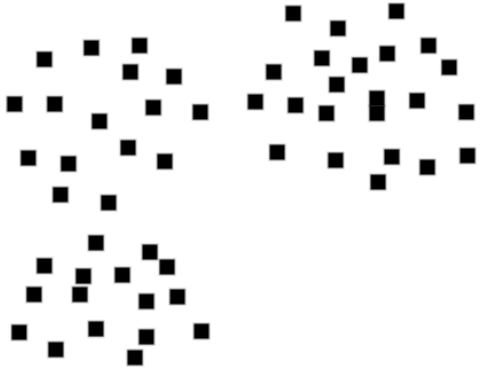


- Observe:
  - $(x_1, y_1), (x_2, y_2), \ldots$
- Objective:
  - Input an unseen x_{new}
  - What is y_{new}?

3 Modes of Learning

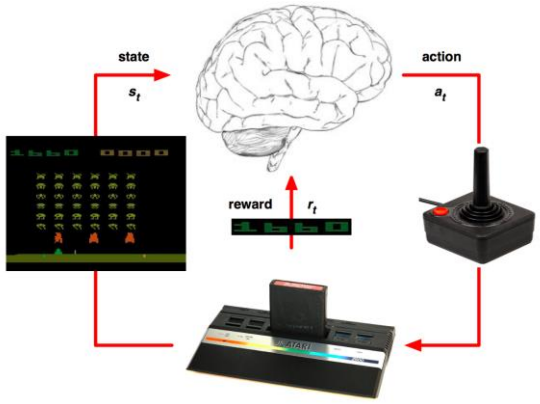
Unsupervised Learning

- Observe:
  - $x_1, x_2, x_3, x_4, \ldots$
- Objective:
  - What is P(x) ?
  - What is a *good* representation of x ?
  - What can we learn from P(x) ?



3 Modes of Learning

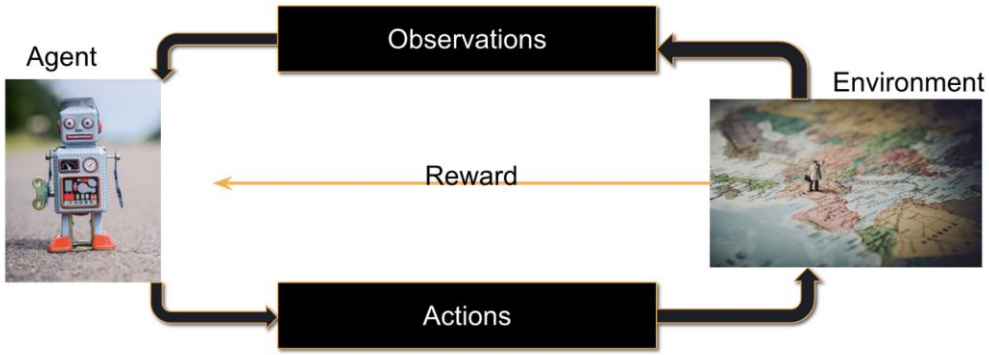
Reinforcement Learning (RL)



- Observe:
  - The states (x_1, x_2, x_3, …)
  - The reward (r_1, r_2, r_3, …)
- Can also take actions
  - a_1 , a_2 , a_3 , …
- What are the best actions?
  - Such that we will receive highest accumulative rewards

Difference between RL and other modes of learning

- Sequential decisions
- You have a goal vs
  You have means to get there
- No concept of “training set” and
  “test set”
- “Passive” vs “Active” learning



RL and Artificial General Intelligence

Yann LeCun

March 14, 2016 ·

Follow

Statement from a Slashdot post about the AlphaGo victory: "We know now that we don't need any big new breakthroughs to get to true AI"

That is completely, utterly, ridiculously wrong.

As I've said in previous statements: most of human and animal learning is unsupervised learning. If intelligence was a cake, unsupervised learning would be the cake, supervised learning would be the icing on the cake, and reinforcement learning would be the cherry on the cake. We know how to make the icing and the cherry, but we don't know how to make the cake.

We need to solve the unsupervised learning problem before we can even think of getting to true AI. And that's just an obstacle we know about. What about all the ones we don't know about?

#deeplearning #AI #AlphaGo

This is before he coined the term self-supervised learning for supervised learning

The Yann Lecunn's cake

How Much Information is the Machine Given during Learning?

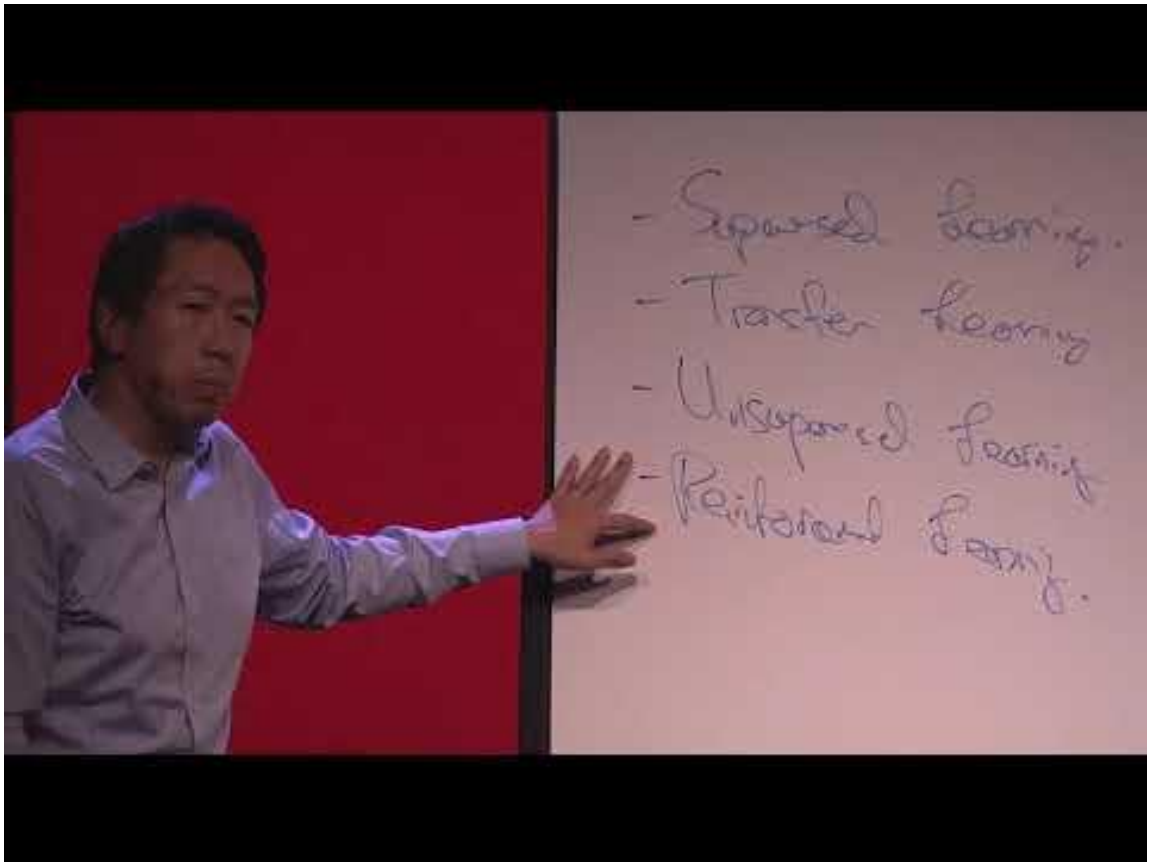
Y. LeCun

- **"Pure" Reinforcement Learning (cherry)**
  - The machine predicts a scalar reward given once in a while.
  - **A few bits for some samples**
- **Supervised Learning (icing)**
  - The machine predicts a category or a few numbers for each input
  - Predicting human-supplied data
  - **10→10,000 bits per sample**
- **Self-Supervised Learning (cake génoise)**
  - The machine predicts any part of its input for any observed part.
  - Predicts future frames in videos
  - **Millions of bits per sample**

https://drive.google.com/file/d/17w443t_5Atnwnu-iOrHKUPFik1pThyhx/view

RL and $$$

"The excitement and PR hype behind reinforcement learning is a bit disproportionate relative to the economic value it's creating today" - Andrew Ng



https://www.youtube.com/watch?v=NQK4ZY_gwKI&feature=youtu.be&t=1m16s

RL use cases

Go, chess, starcraft, dota, poker

Finance (https://www.jpmorgan.com/global/LOXM)...

Robotics...but...

https://research.googleblog.com/2016/03/deep-learning-for-robots-learning-from.html

https://towardsdatascience.com/applications-of-reinforcement-learning-in-real-world-1a94955bcd12
https://www.oreilly.com/ideas/practical-applications-of-reinforcement-learning-in-industry

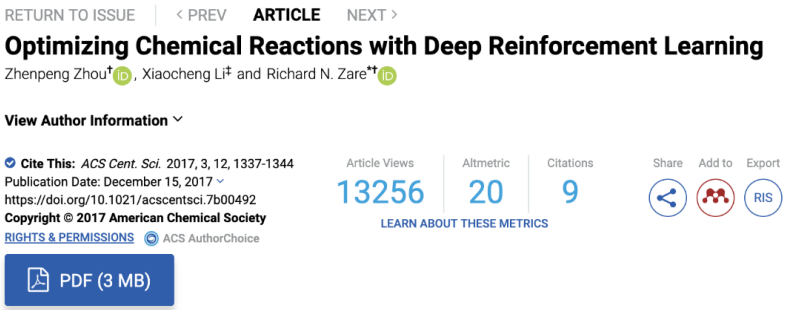
RL use cases

Data center and resource management (https://people.csail.mit.edu/alizadeh/papers/deeprm-hotnets16.pdf System configuration http://ranger.uta.edu/~jrao/papers/ICDCS09.pdf DRAM controller https://ieeexplore.ieee.org/abstract/document/4556714/ Recommender (Bandits) https://people.cs.umass.edu/~pthomas/papers/Barto2017.pdf)

Ad bidding (https://arxiv.org/abs/1701.02490)

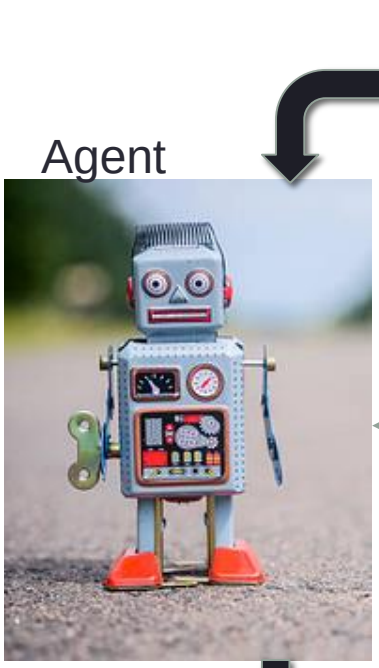
Chemistry (https://pubs.acs.org/doi/full/10.1021/acscentsci.7b00492)

Some other tasks that use algorithms from RL to help perform model training (autoML, REINFORCE)



ACS Central Science

RL framework



Agent

Observations

Reward

Actions

Environment

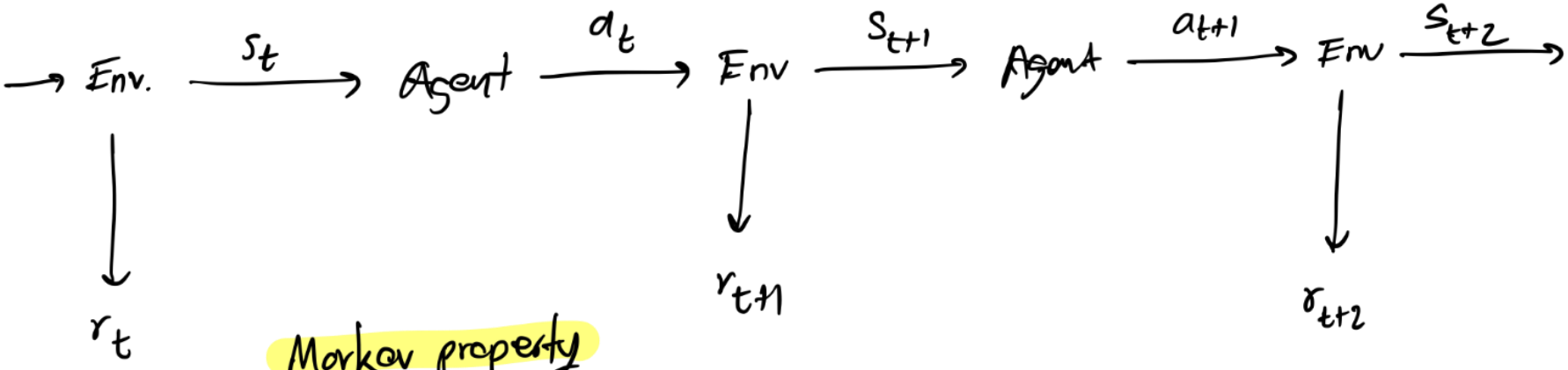
Learning through trial and error

RL framework

Reward (r_t)

State (s_t)

Action (a_t)



Markov property

$a_t = A(s_t)$

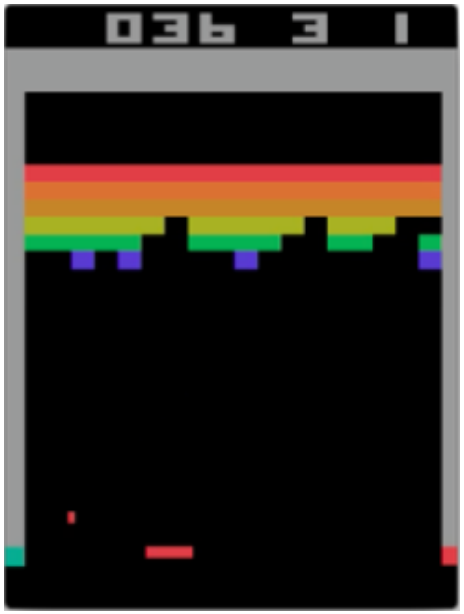
$r_{t+1} = R(s_t, a_t)$

$s_{t+1} = S(s_t, a_t)$

Rewards-based learning

- Maximise the rewards
- Can we design any desired behaviour with reward?

R_t = Δdistance



R_t = score

$$R_T = \begin{cases} 1 \text{ , win} \\ -1 \text{, lose} \end{cases}$$

The Environment

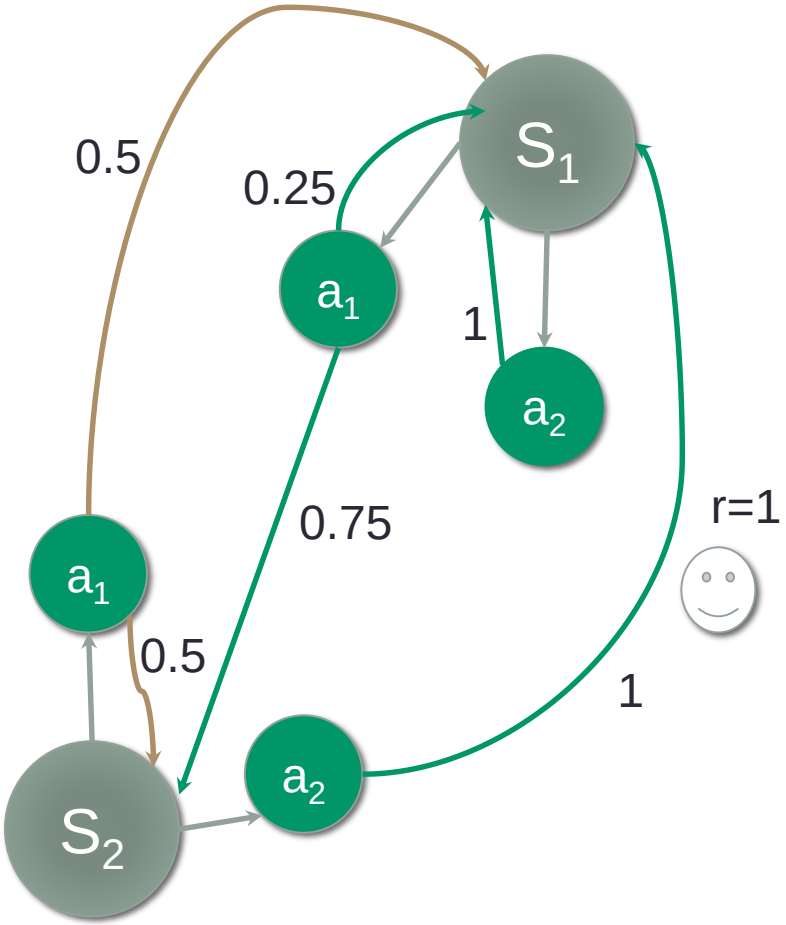
How can we model the environment?

Markov Decision Process (MDP)

- **S,A,P,R**,γ
- **S** – Set of states
- **A** – Set of actions
- **P** – Transition between states given an action

$$P^{a}_{s,s'} = \text{Prob}[s_{t+1}=s' \mid s_t=s,\ a_t=a]$$

- **R** – Rewards associated with actions and states
- γ – Discount factor

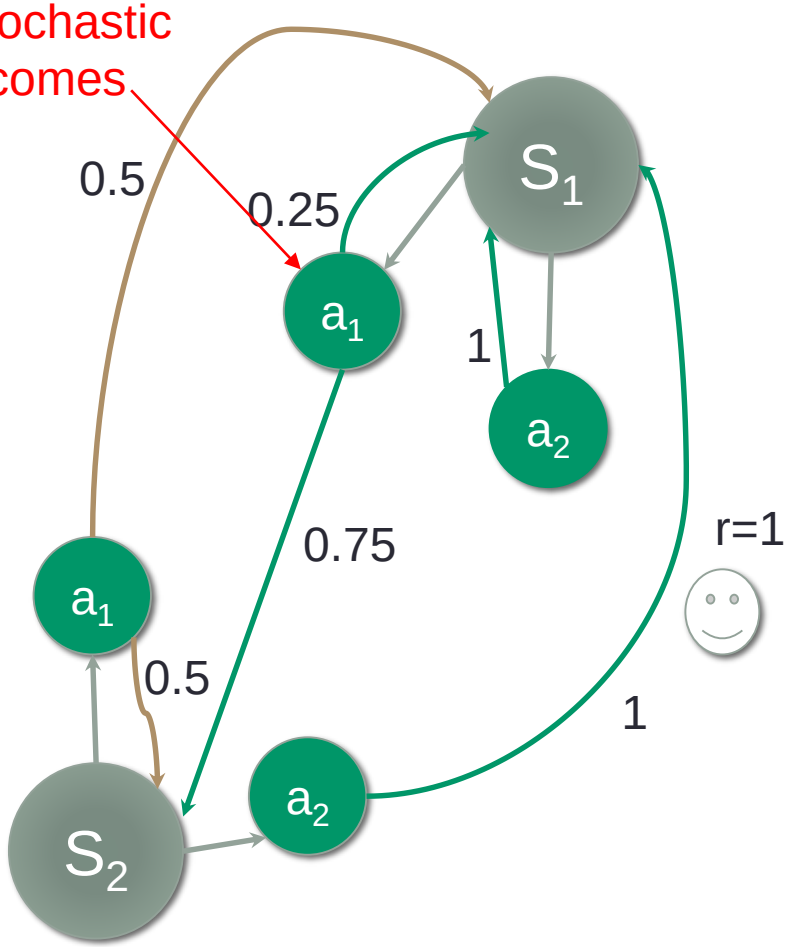


Markov Decision Process (MDP)

- **S,A,P,R**,γ
- **S** – Set of states
- **A** – Set of actions
- **P** – Transition between states given an action

$$P^{a}_{s,s'} = \text{Prob}[s_{t+1}=s' \mid s_t=s,\ a_t=a]$$

- **R** – Rewards associated with actions and states
- γ – Discount factor



Markov Property

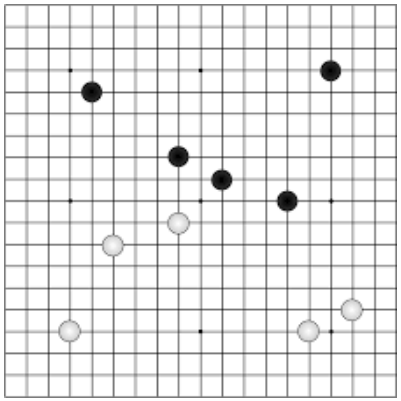
$$p(s_{t+1} | s_t, a_t)$$

- s_{t+1} depends only on s_t
- not s_{t-1}, not anything before
- this simplifies our situation!

Fog of war

But, is it true in every case?

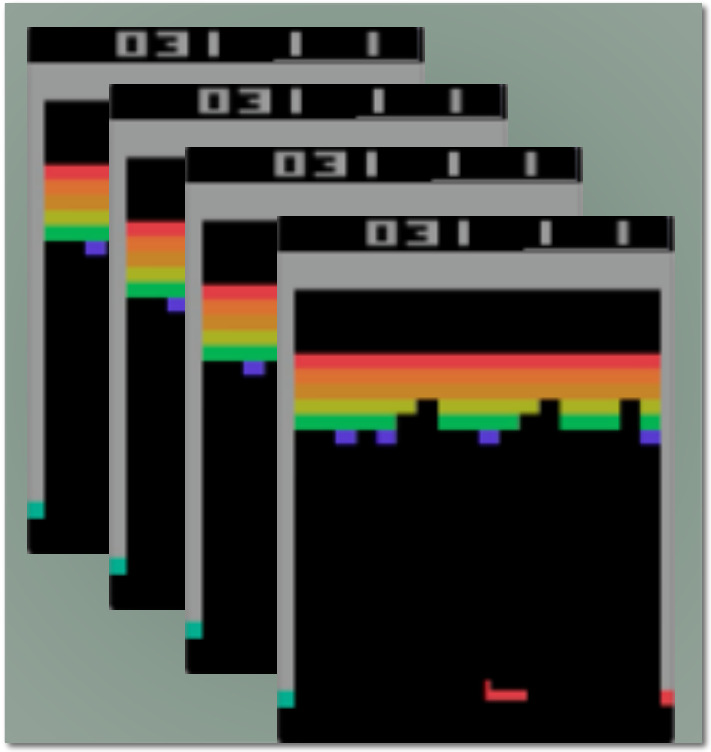
- It depends on your observed state
  - Fully observable state
  - Partially observable state



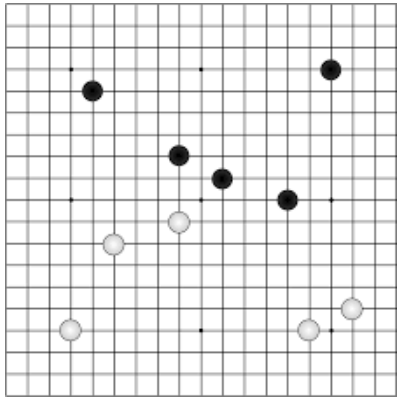
Fully observable

Fully Observable State

- Fully observable state: All information from the past is captured in the current state



For Go, a board position
For simple video games, stack multiple frames



The Agent

Policy

- Policy = a mapping from a state to an action
- Objective of RL is to find the ‘’*optimal*” policy!

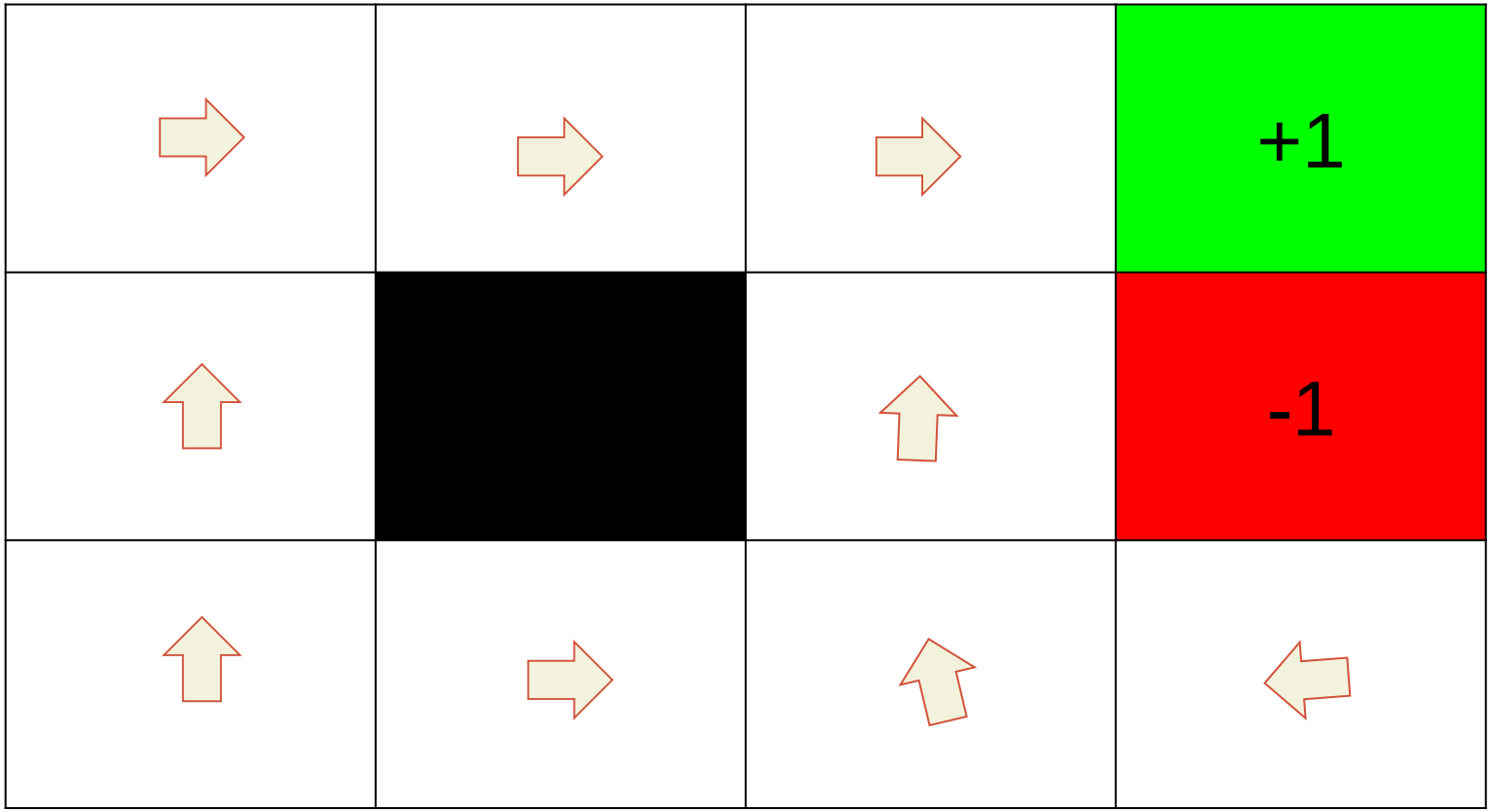
$$a = \pi(s)$$

$$a \sim \pi(s)$$

Can be either deterministic or stochastic

Policy

Example: tabular policy



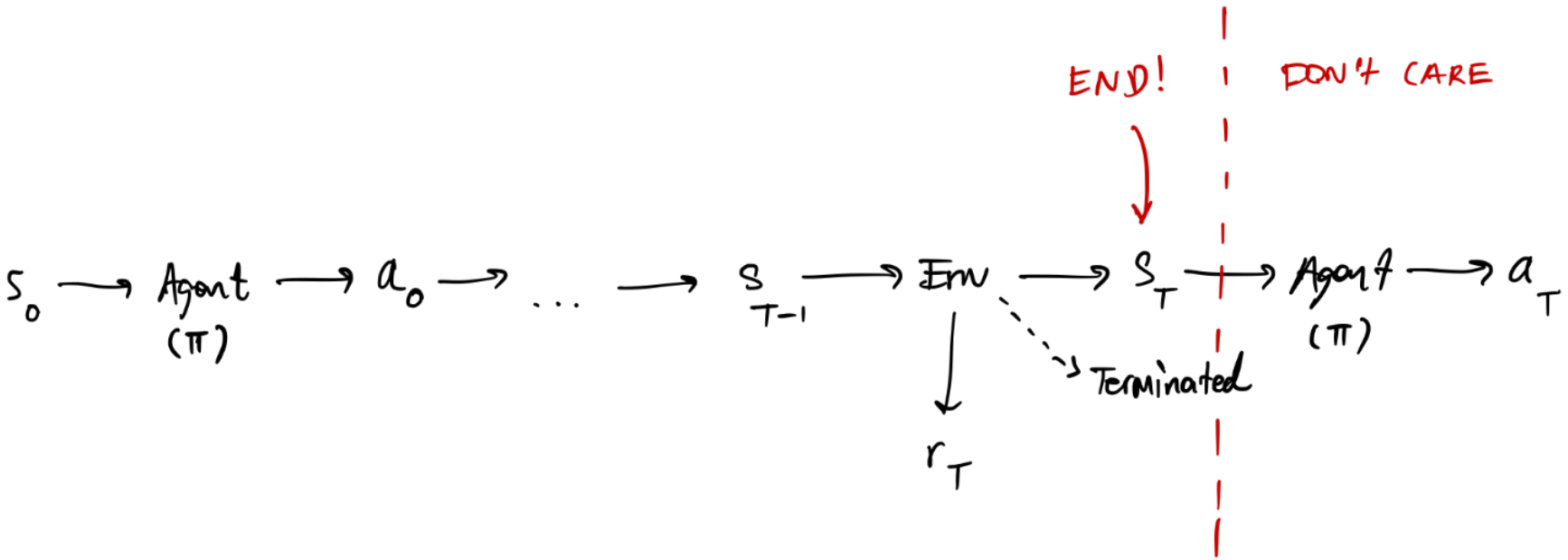
Learning

How do we find the best policy ?

Rollout (Our data)

Time	0	1	2	3	...	T-1	T
S	S_0	S_1	S_2	S_3	...	S_{T-1}	S_T
A	A_0	A_1	A_2	A_3	...	A_{T-1}	A_T
R	R_0	R_1	R_2	R_3	...	R_{T-1}	R_T
Done	0	0	0	0	...	0	1

Don't care



Return (Cumulative rewards)

- Return = cumulative rewards with discount

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_t$$

$r_t = 0$

$r_{t+10} = 1$

$r_{t+34} = -1$

What is learning?

- Use data to find/search for the best policy

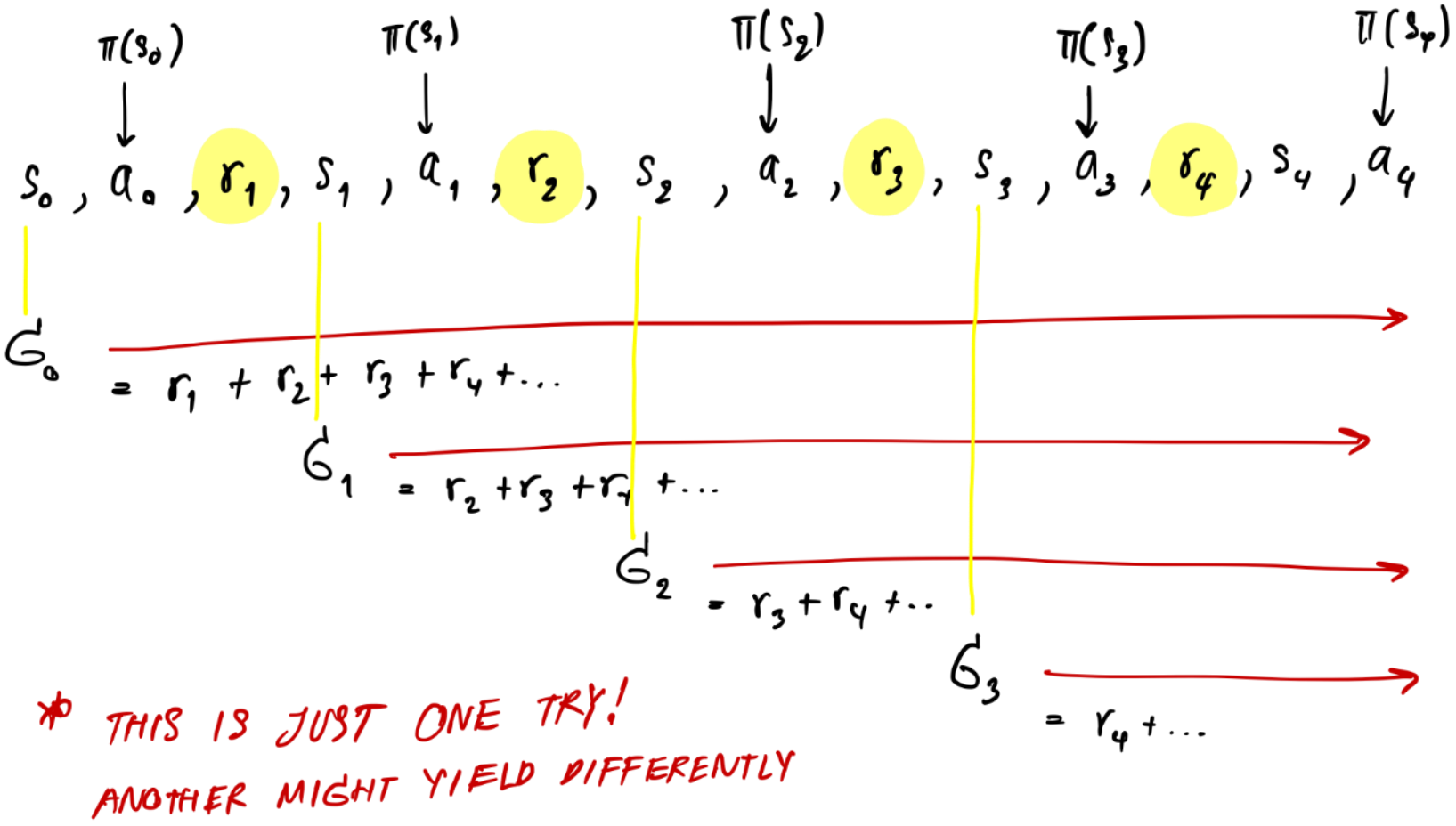
What is the best policy?

- Policy that give us the highest expected return!

$$G = r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$$

$$J(\pi) = E_\pi[G]$$

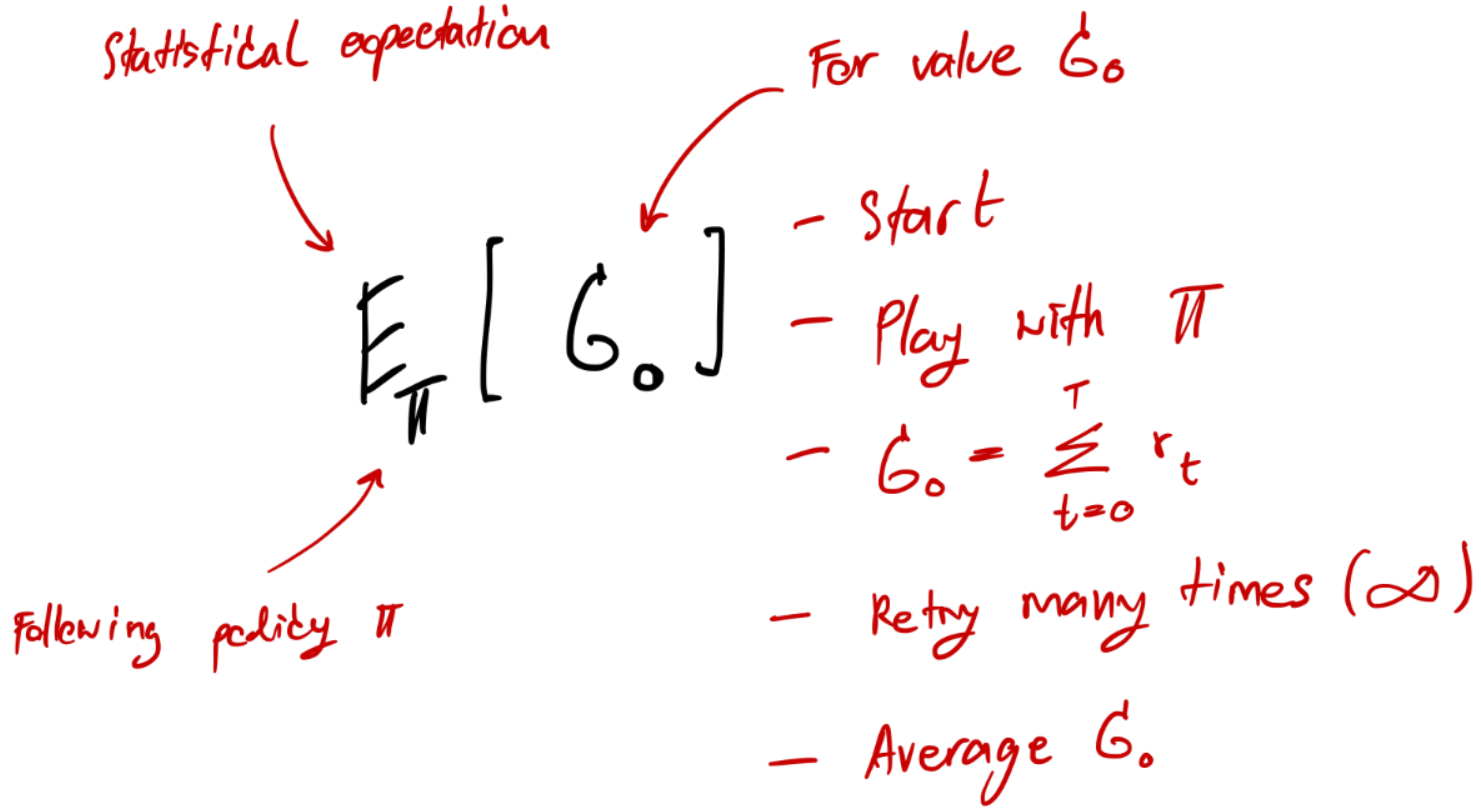
Return under a policy (G^{π})



* THIS IS JUST ONE TRY!
ANOTHER MIGHT YIELD DIFFERENTLY

Expected Return

How good “on average” is our return?



A naive learning method

1. Initialise a policy randomly
2. Evaluate the policy by running that policy multiple times
   a. which we then collect the returns of all the runs
3. Randomly initialise another policy
4. Evaluate the new policy
5. Keep the policy that have a higher expected return
6. Repeat 3-5

Intuitive. But very inefficient!

How to make it more efficient?

Categories of RL methods

Value-based vs Policy-based

Model-based vs Model-free

On-policy vs Off-policy

Trade-off on a spectrum

Valued-based RL

Q-learning algorithm

- Let's define a state value as
  - Expectation of the return after visit s and follow **π**

$$V^{\pi}(s) = E_{\pi}[G_t | s_t = s]$$

- Let's define a state-action value (Q-value) as
  - Expectation of the return after visit a state s, take action a

$$Q^{\pi}(s, a) = E_{\pi}[G_t | s_t = s, a_t = a]$$

$$V^{\pi}(s) = E_{\pi(s)}[Q^{\pi}(s, \pi(s))]$$

Q-learning algorithm

- There exist an optimal value function associate with an optimal policy,

$$V^*(s) = \max_\pi V^\pi(s) \quad \forall s \in S$$

- The optimal policy is the policy that achieves the highest value for every state

Q-learning algorithm

- It follows that

$$V^*(s) = \max_a [Q^*(s,a)]$$

- and ..

$$\pi^*(s) = argmax_a Q^*(s,a)$$

- Optimal actions can be found indirectly through Q-value

Value-based learning

2 Steps

- Improve Q/V
- Improve the policy

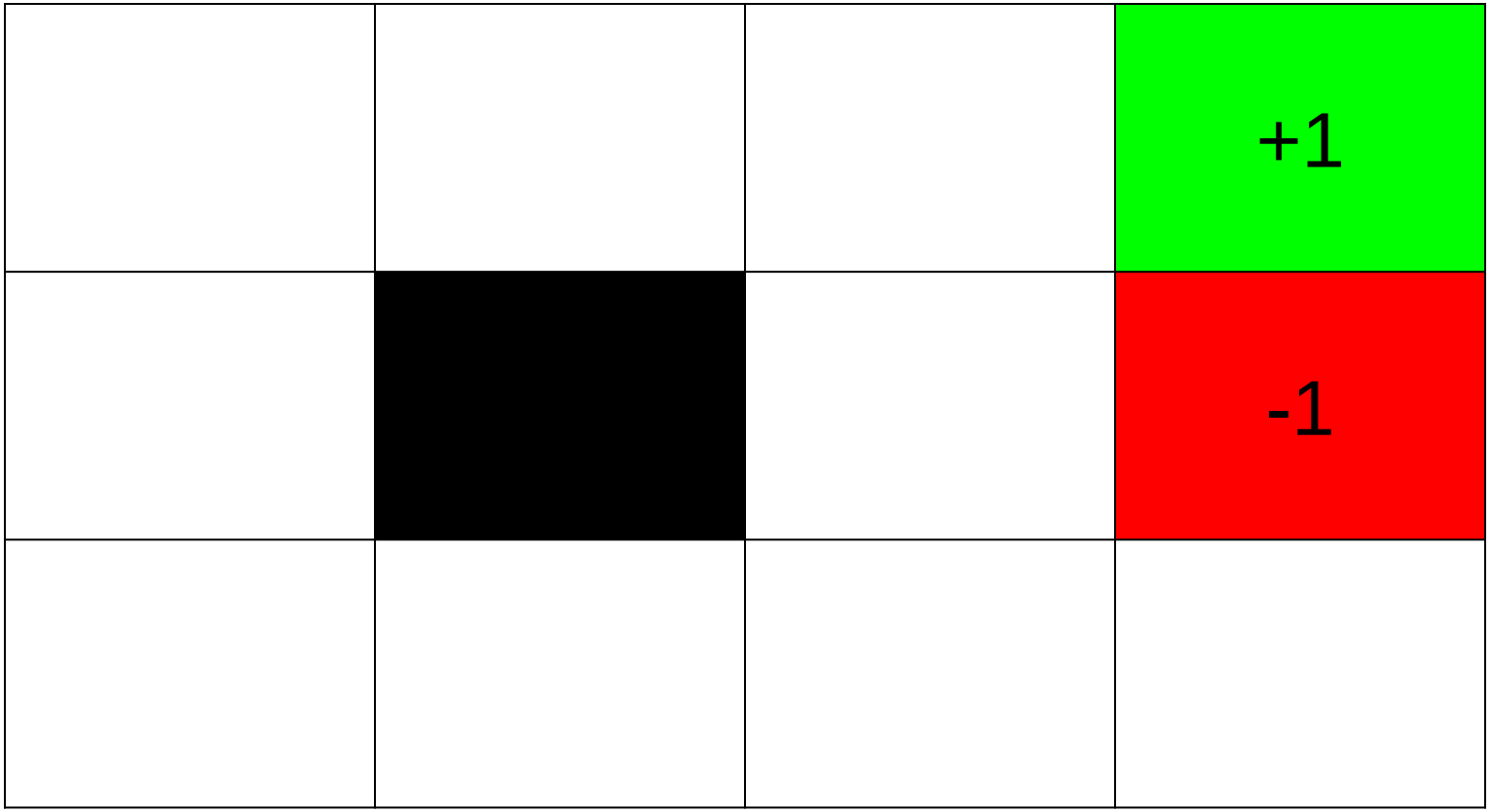
How to learn Q/V?

- Monte-Carlo
- Bootstrap

How to improve the policy?

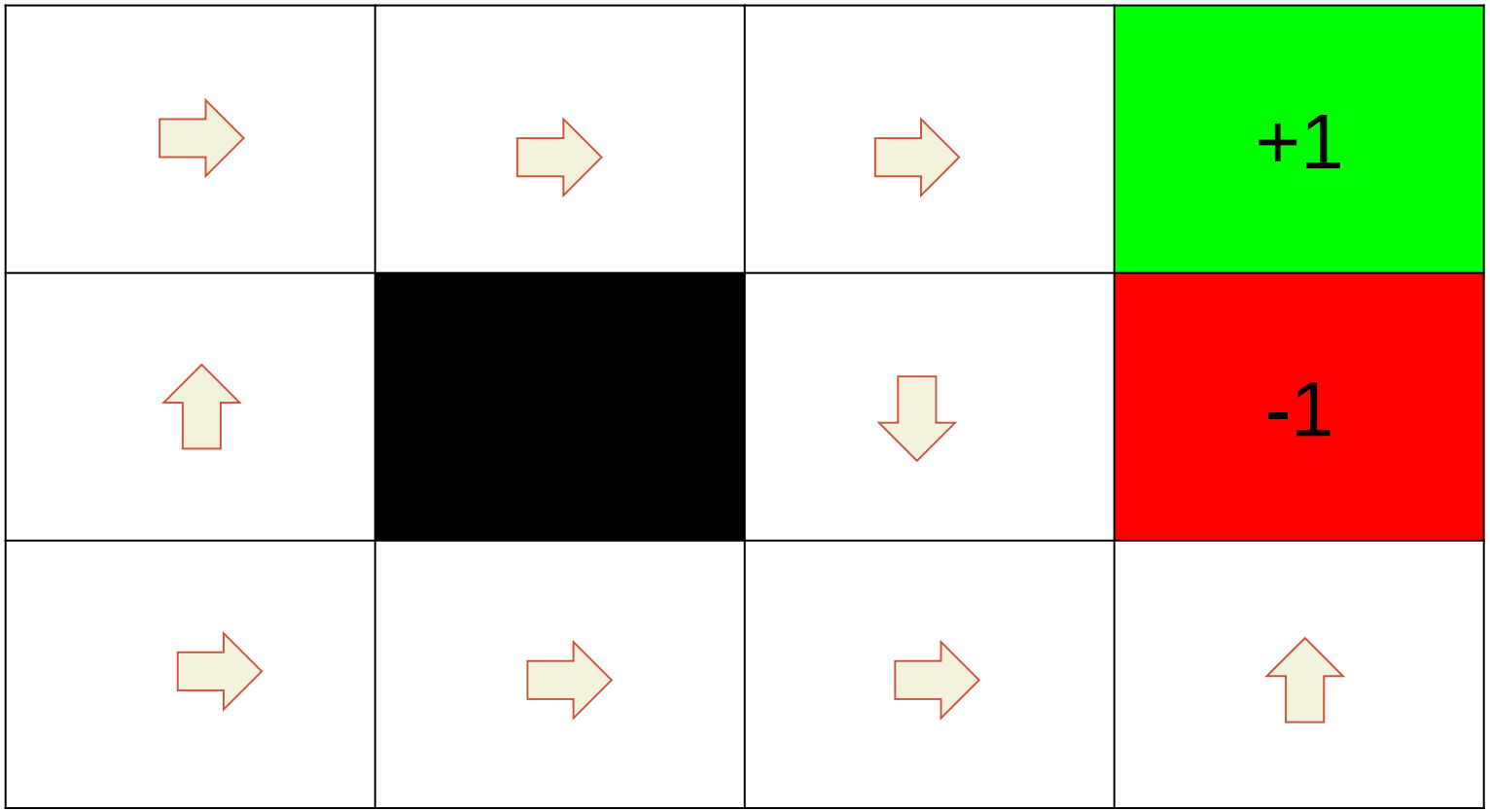
- Follows the best Q/V

Example, Tabular Q-learning



Monte-Carlo Estimator

- Initialise π

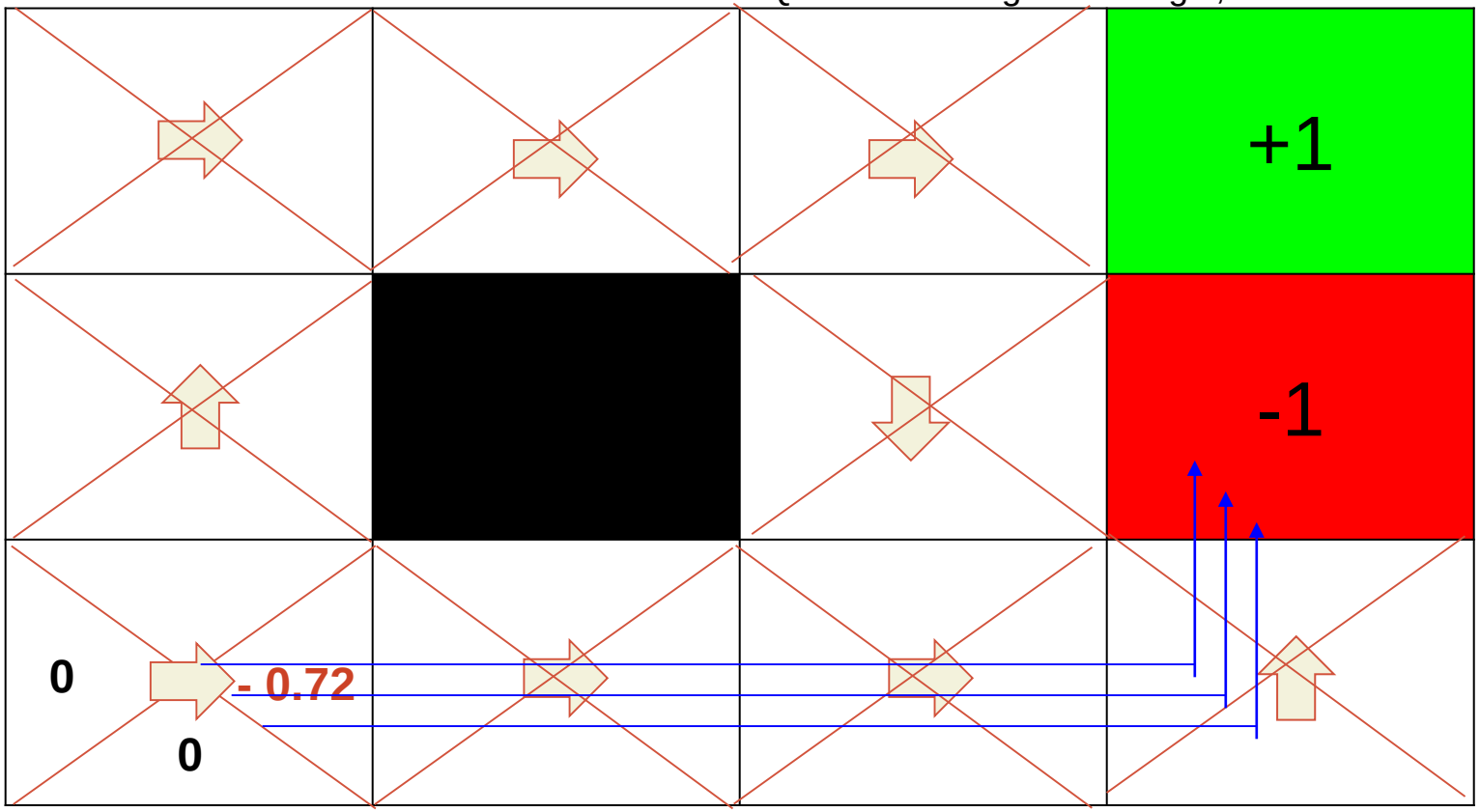


Monte-Carlo Estimator

$$Q^{\pi}(s,a) \approx r_0 + \gamma r_1 + \gamma^2 r_2 + .. + \gamma^n r_n$$

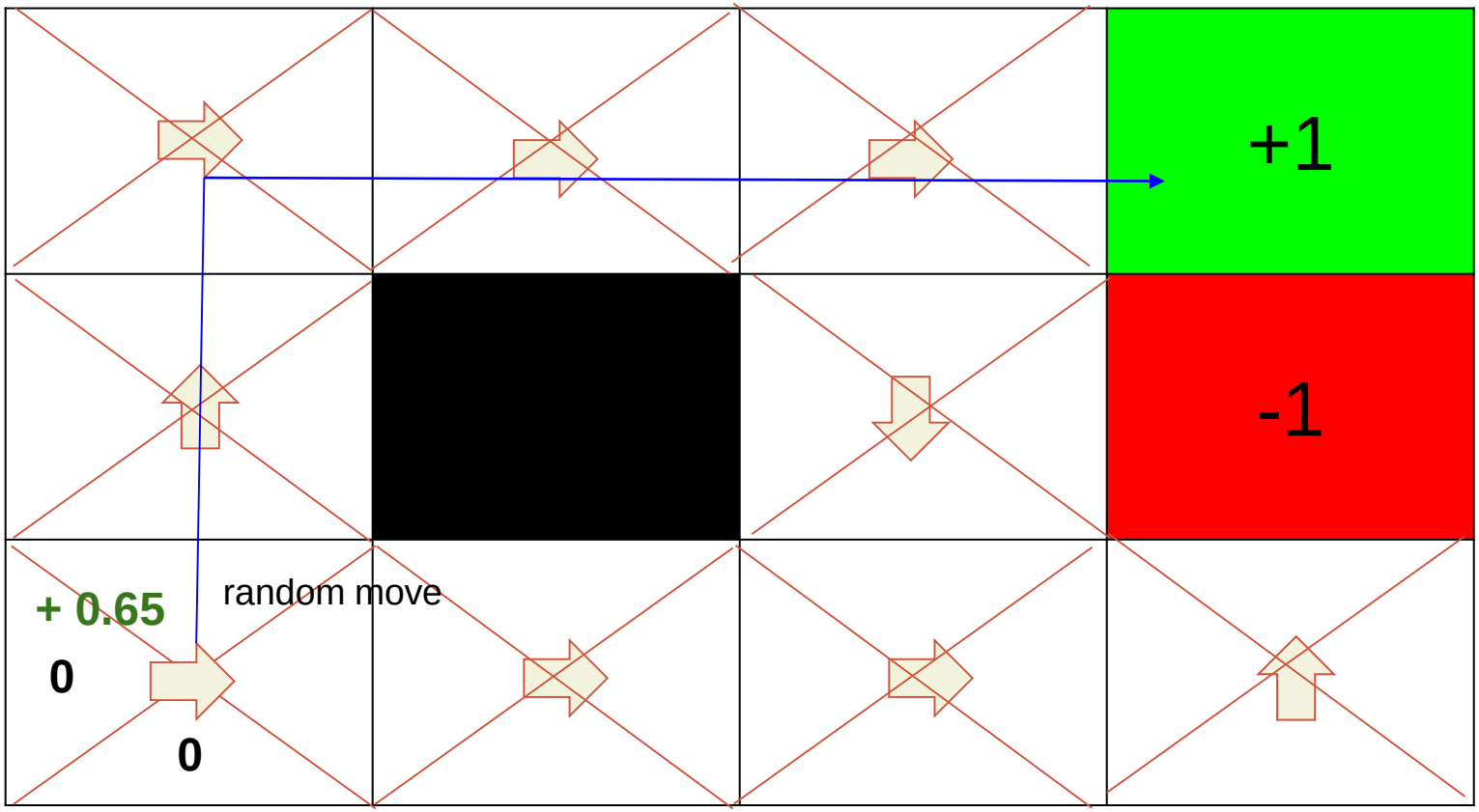
$$\gamma = 0.9$$

Running simulations until the end.
Estimate Q as the average of taking s,a



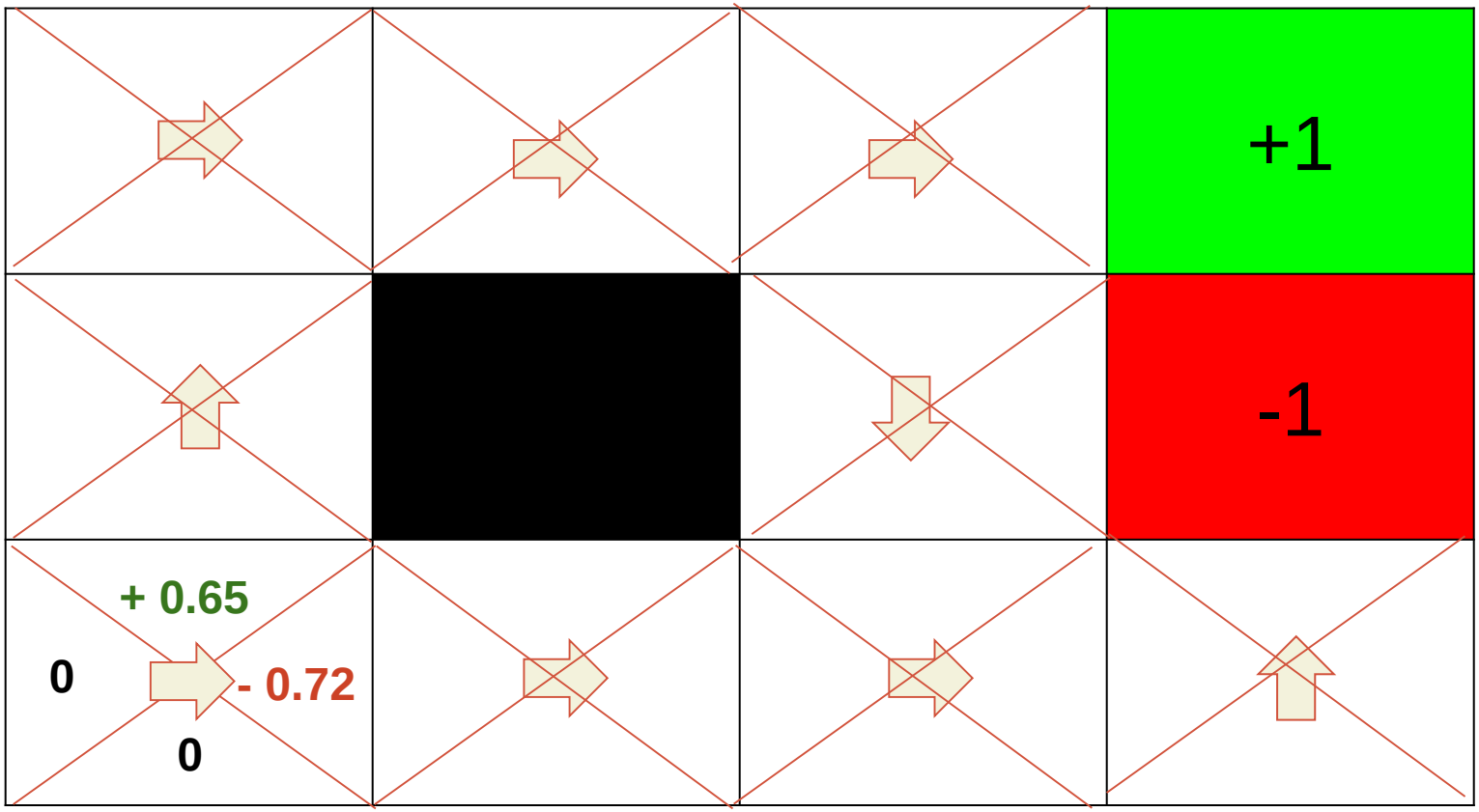
Exploration

Randomly explore sometimes to get better solutions
Have epsilon probability to take a random move. Take current policy move with 1-epsilon probability.



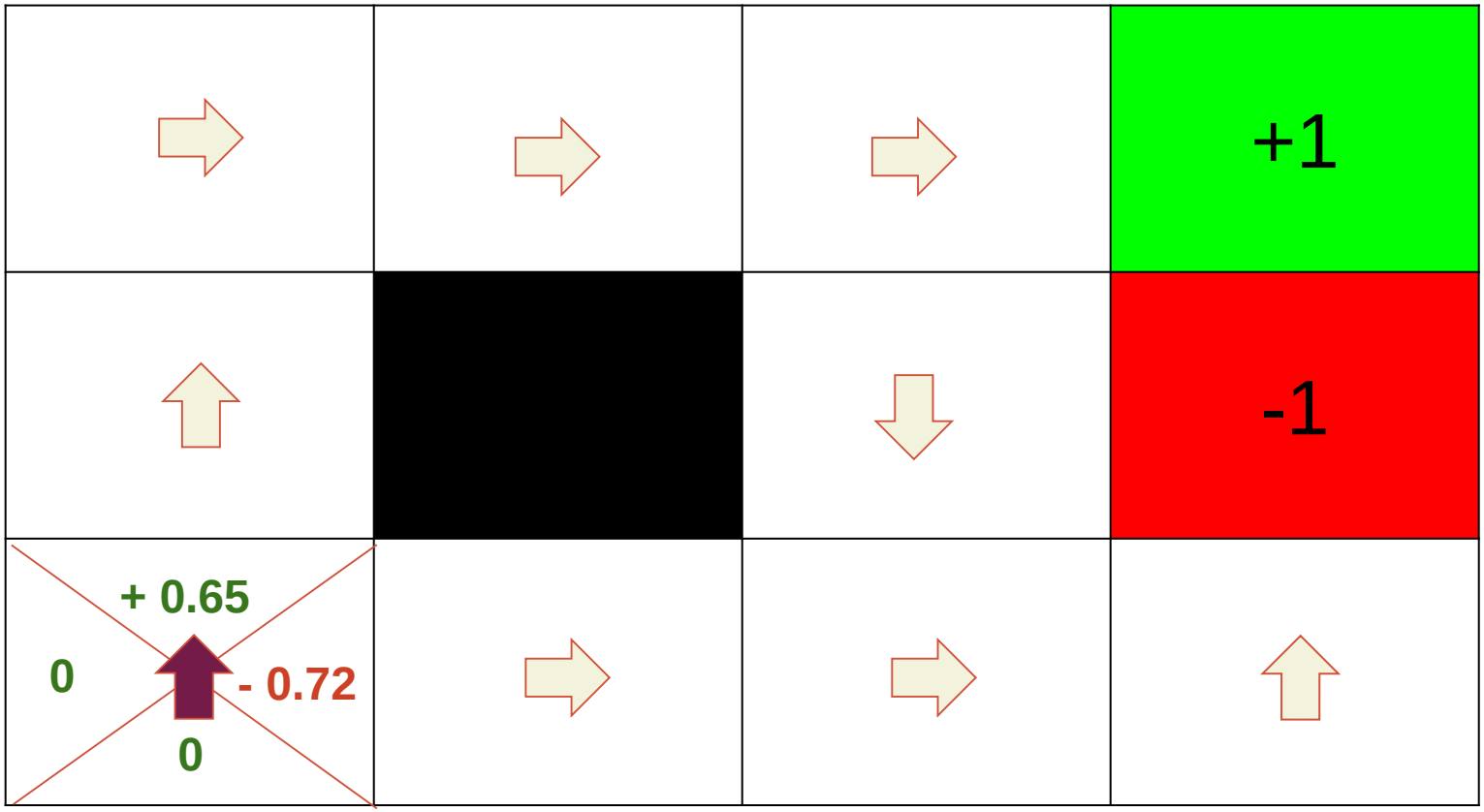
Monte-Carlo Estimator

Running many simulations until the end to estimate Q(s,a)



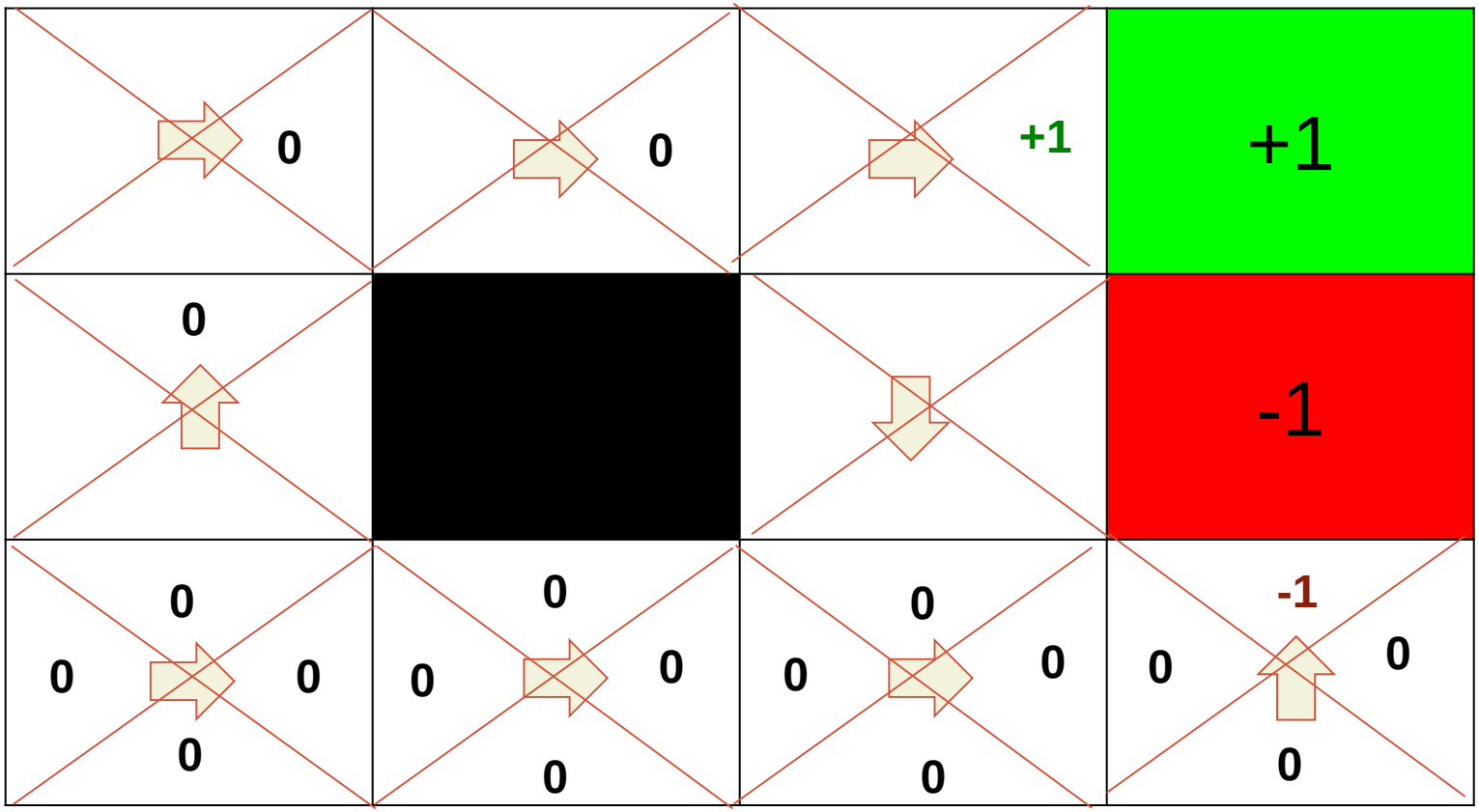
Policy Improvement

$$\pi'(s) = argmax_a Q(s,a)$$



Bootstrap Estimator

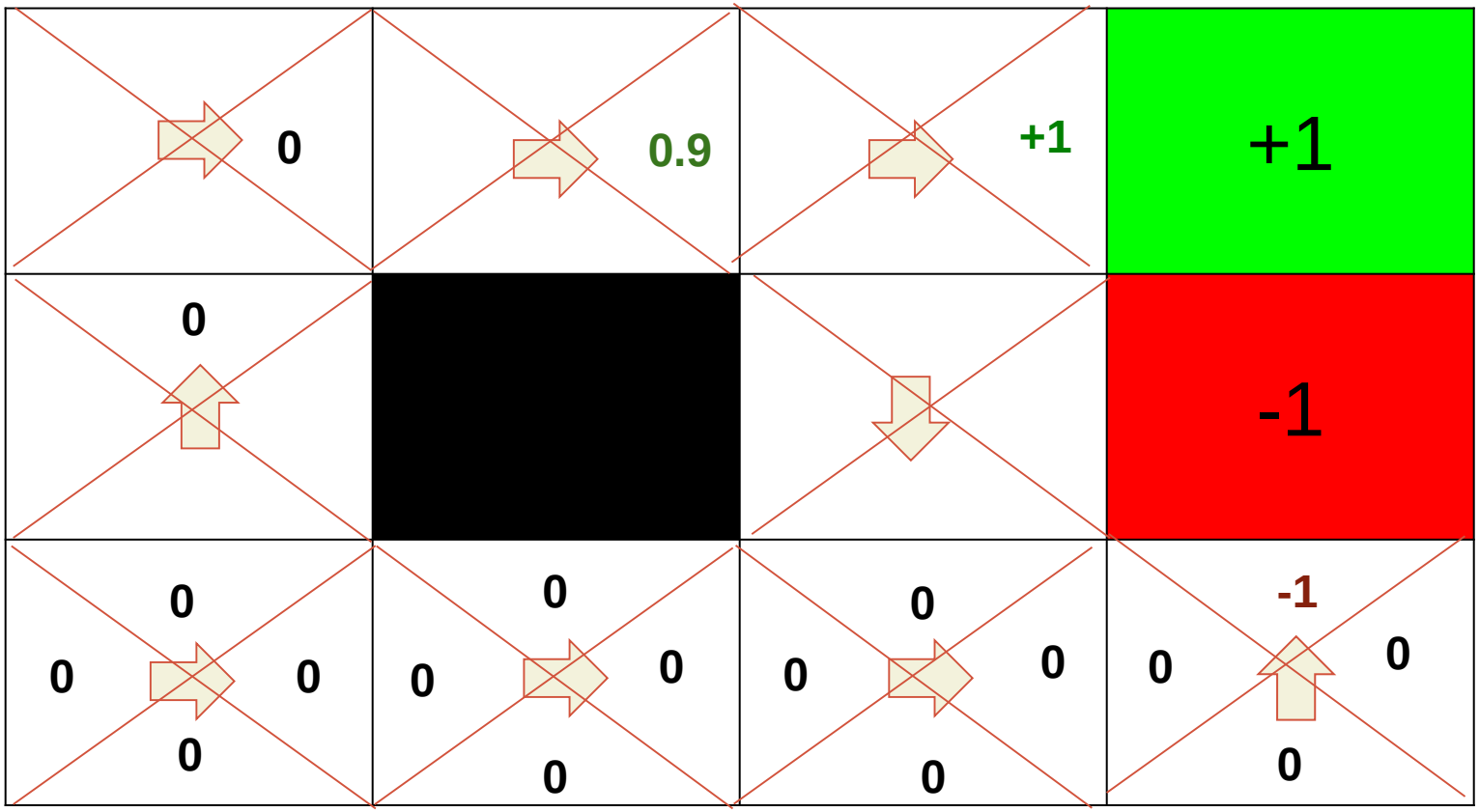
$$Q(s,a) \approx r_0 + \gamma \max_b Q(s',b)$$



Bootstrap Estimator

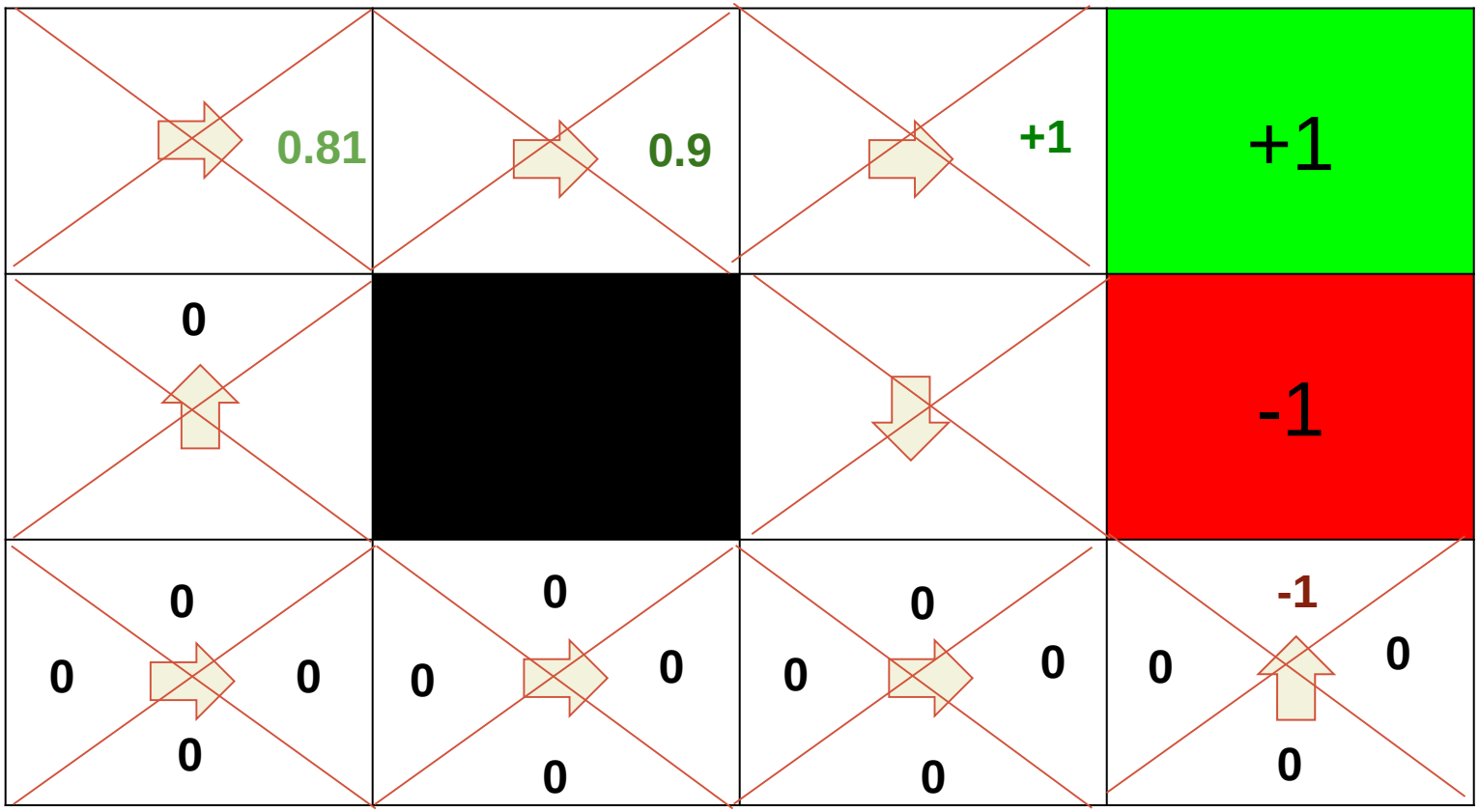
bellman equation

$$Q(s,a) \approx r_0 + \gamma \max_b Q(s',b)$$



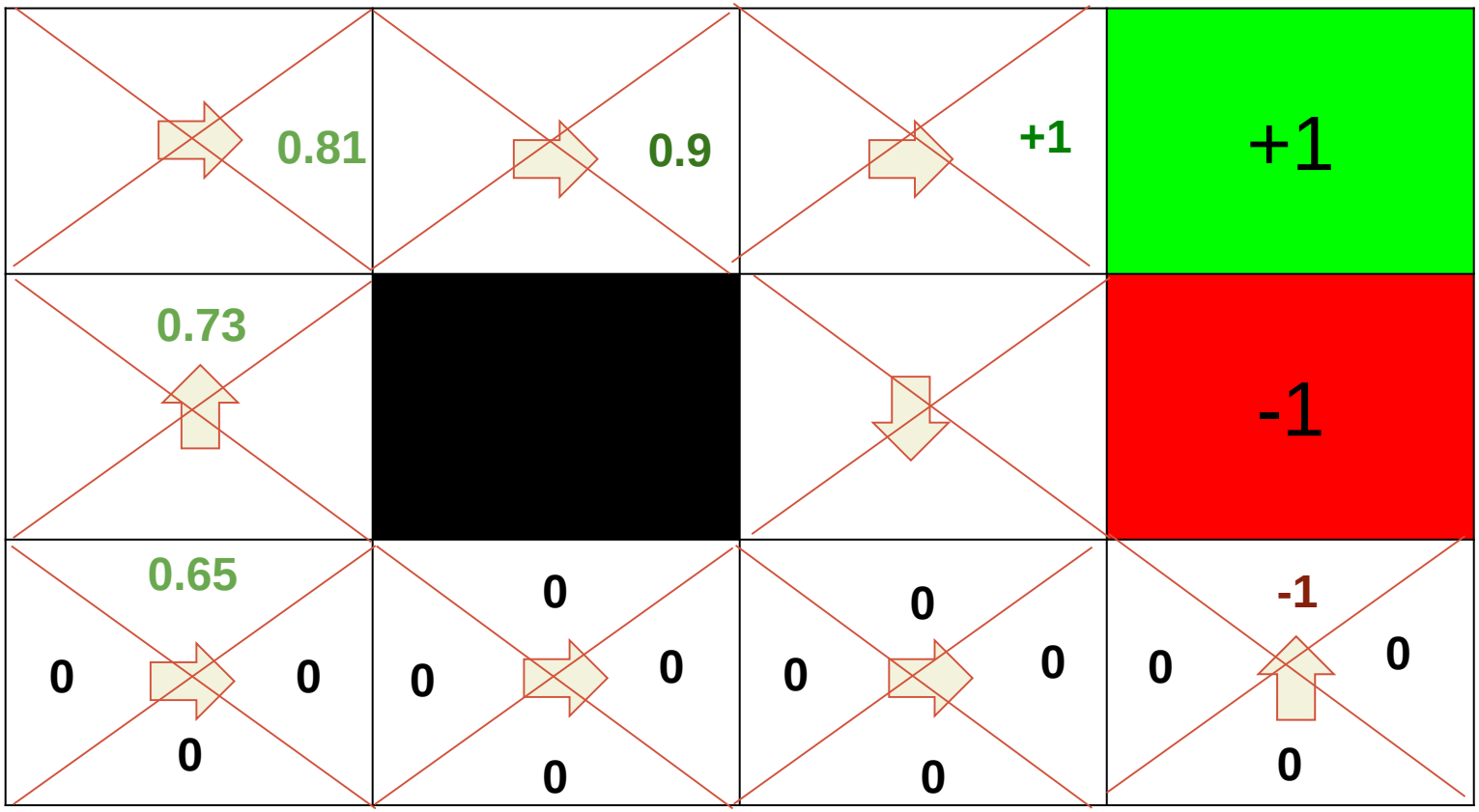
Bootstrap Estimator

$$Q(s,a) \approx r_0 + \gamma \max_b Q(s',b)$$

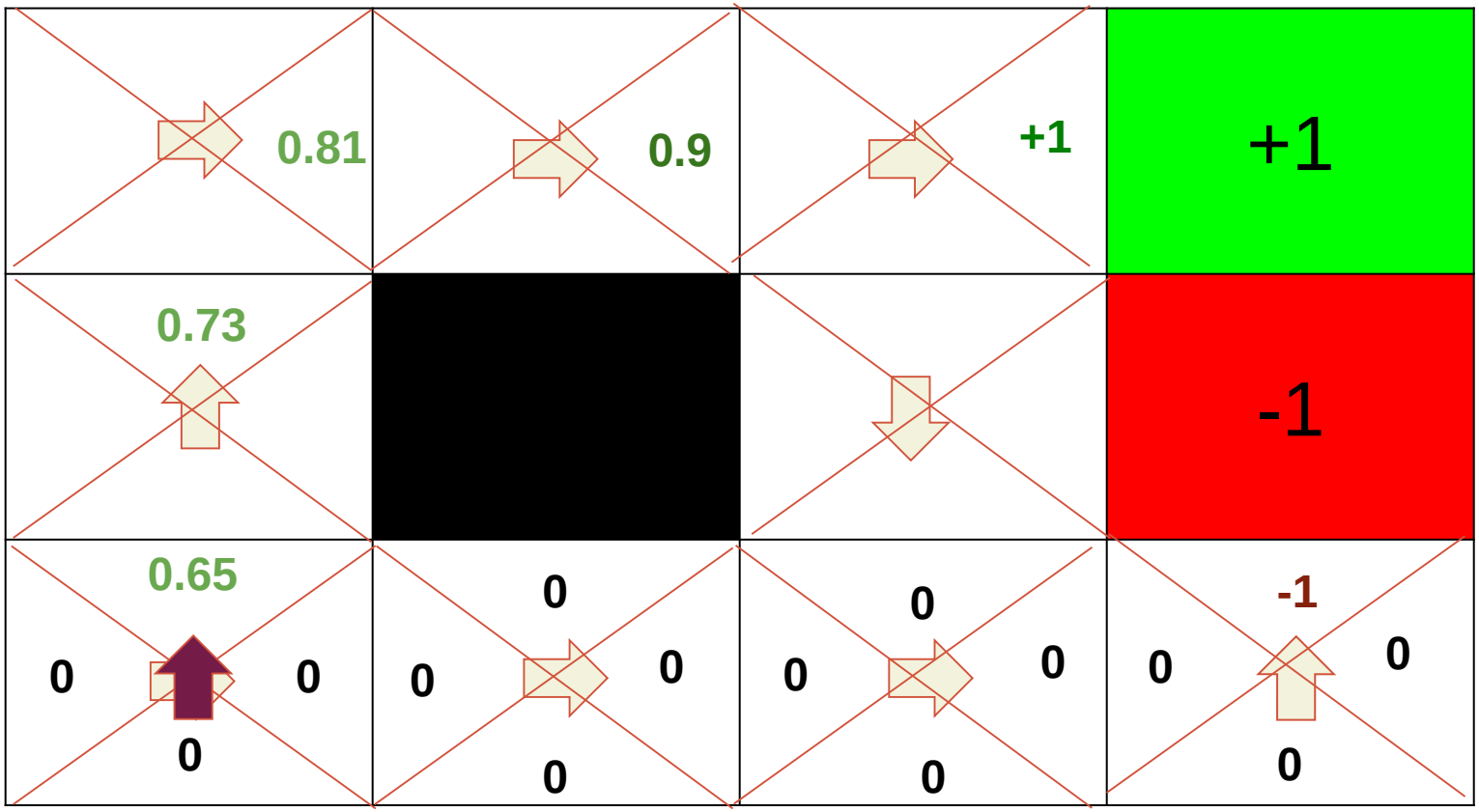


Bootstrap Estimator

$$Q(s,a) \approx r_0 + \gamma \max_b Q(s',b)$$



Policy Improvement



Bias and Variance in RL

What is bias of V_π estimation?

- Let $\hat{V}_\pi(s)$ be an estimate of $V_\pi(s)$
- $\hat{V}_\pi(s)$ is unbiased if:

$$\mathbb{E}_s\left[\hat{V}_\pi(s) - V_\pi(s)\right] = 0$$

What is variance of $\hat{V}_\pi(s)$ estimation?

$$\mathbb{V}\text{ar}\left[\hat{V}_\pi(s)\right] = \mathbb{E}_s\left[(\hat{V}_\pi(s) - \mathbb{E}_s\left[\hat{V}_\pi(s)\right])^2\right]$$

- High if $\hat{V}_\pi(s)$ fluctuates a lot

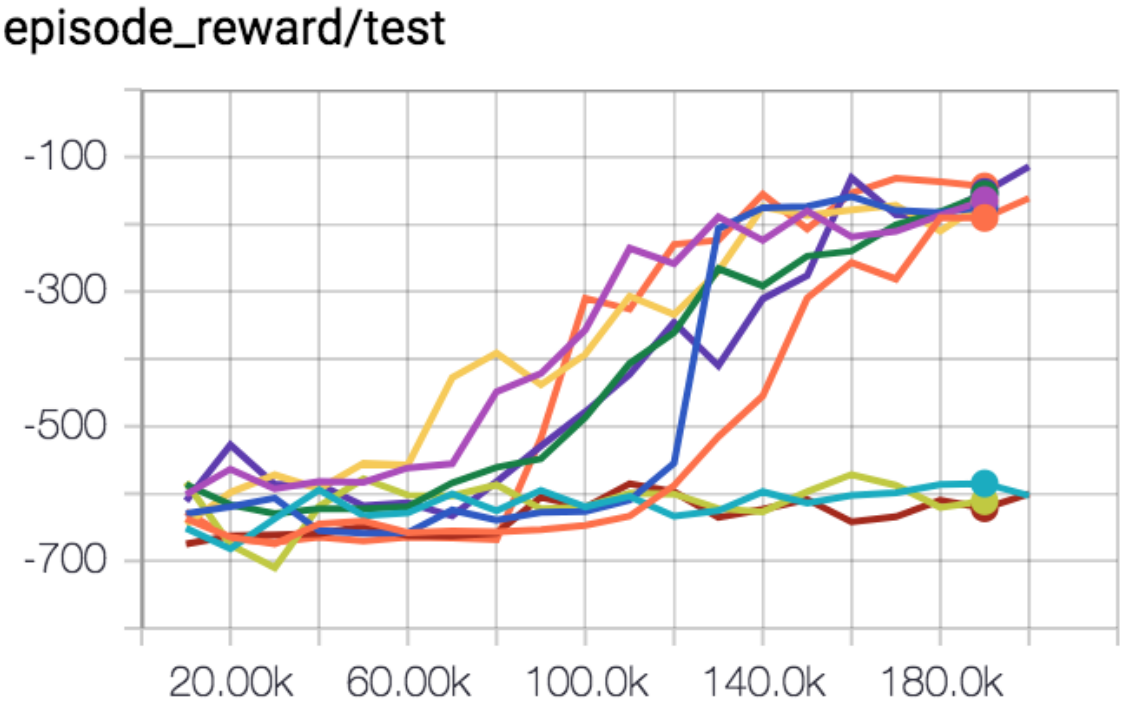
Bias and Variance

- Monte-Carlo estimate has high variance and low bias.
- Bootstrap estimate has higher bias but lower variance.

Problems with RL

Randomness

- Random initialization
- Random exploration
- Random environment

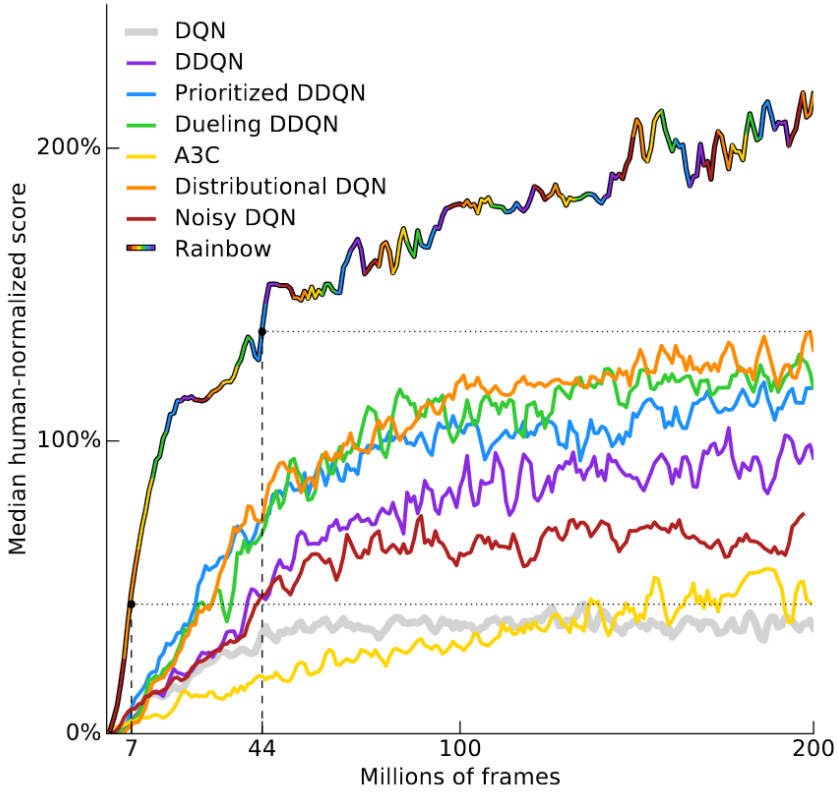


https://www.alexirpan.com/2018/02/14/rl-hard.html

Problems with (current) RL

Data inefficient

- Many use case can be better solved with supervised learning (efficiency and accuracy)



https://arxiv.org/abs/1710.02298

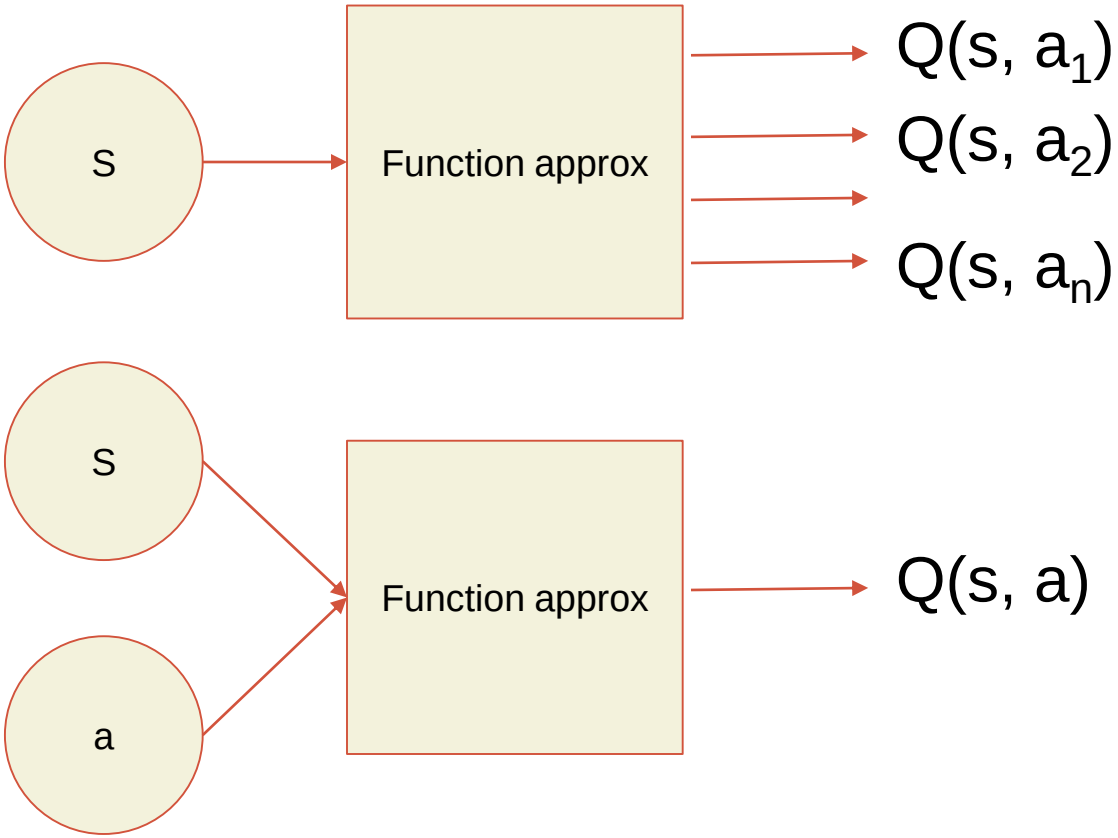
Function Approximator (FA)

- Tabular Q-value is impractical when the state-action space is large!
  - Need large memory
  - Impractical to fill up every cell

- Enter .. a function approximator

Function Approximator (FA)

- Instead of a table containing Q-value for every state and action, use a function that output Q-values.



Learning with FA

- With tabular Q-learning,
  - the act of learning = putting Q-value in the table
- With function approximator,
  - the act of learning = searching for the optimal parameters of the FA

Learning with FA

- How to adapt the parameters (weights) of the FA?
- Step 1: Define a loss function.
- Step 2: Optimise the weights to minimise the loss

Loss function

- What should be the loss function?
- Introducing Bellman's equations

$$V^{\pi}(s_t) = E_{\pi,P}[r_t + \gamma V^{\pi}(s_{t+1})]$$

$$Q^{\pi}(s_t) = E_{\pi,P}[r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1})]$$

- Bellman's optimality equations

$$Q^{*}(s_t) = E_{P}[r_t + \gamma \max_b Q^{*}(s_{t+1}, b)]$$

Loss function

- The Bellman's equation must hold for correct Q-value
- Rewrite the Bellman's optimality with our estimator (FA)

$$\hat{Q}(s_t) = E_P[r_t + \gamma \max_b \hat{Q}(s_{t+1}, b)]$$

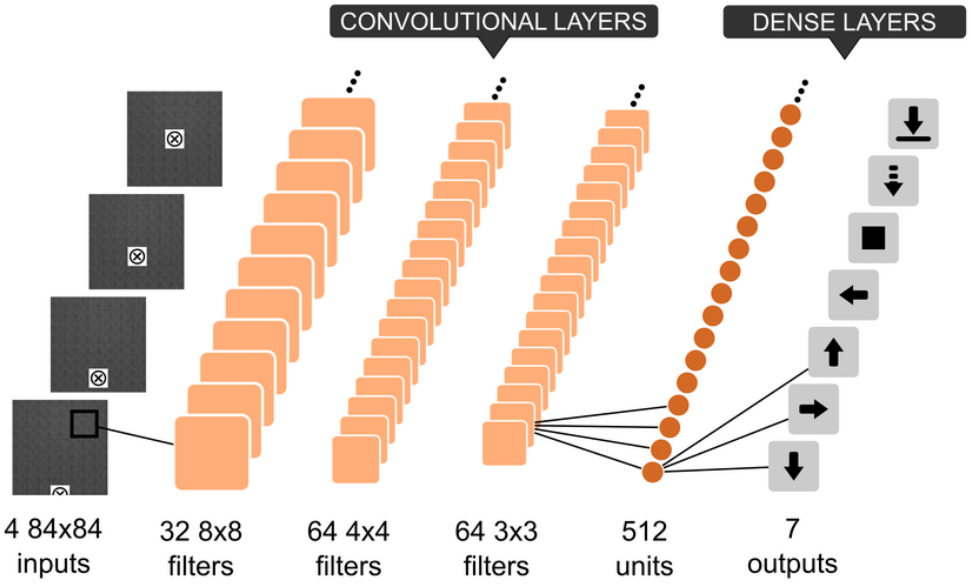
The estimator is correct if
the left hand side = right hand side

$$TD = r_t + \gamma \max_b \hat{Q}_\theta(s_{t+1}, b) - \hat{Q}_\theta(s_t)$$

``Temporal Difference error''

Temporal Difference Learning

- Use TD-error to guide learning
- Example
  - Deep Q-Networks (DQN)
    - Deep convolutional neural network as a function approximator
    - Optimise square TD-error



$$L(\theta) = (r_t + \gamma \max_b \hat{Q}_\theta(s_{t+1}, b) - \hat{Q}_\theta(s_t))^2$$

Policy-based methods

Policy gradient

Policy gradient

Q-Learning

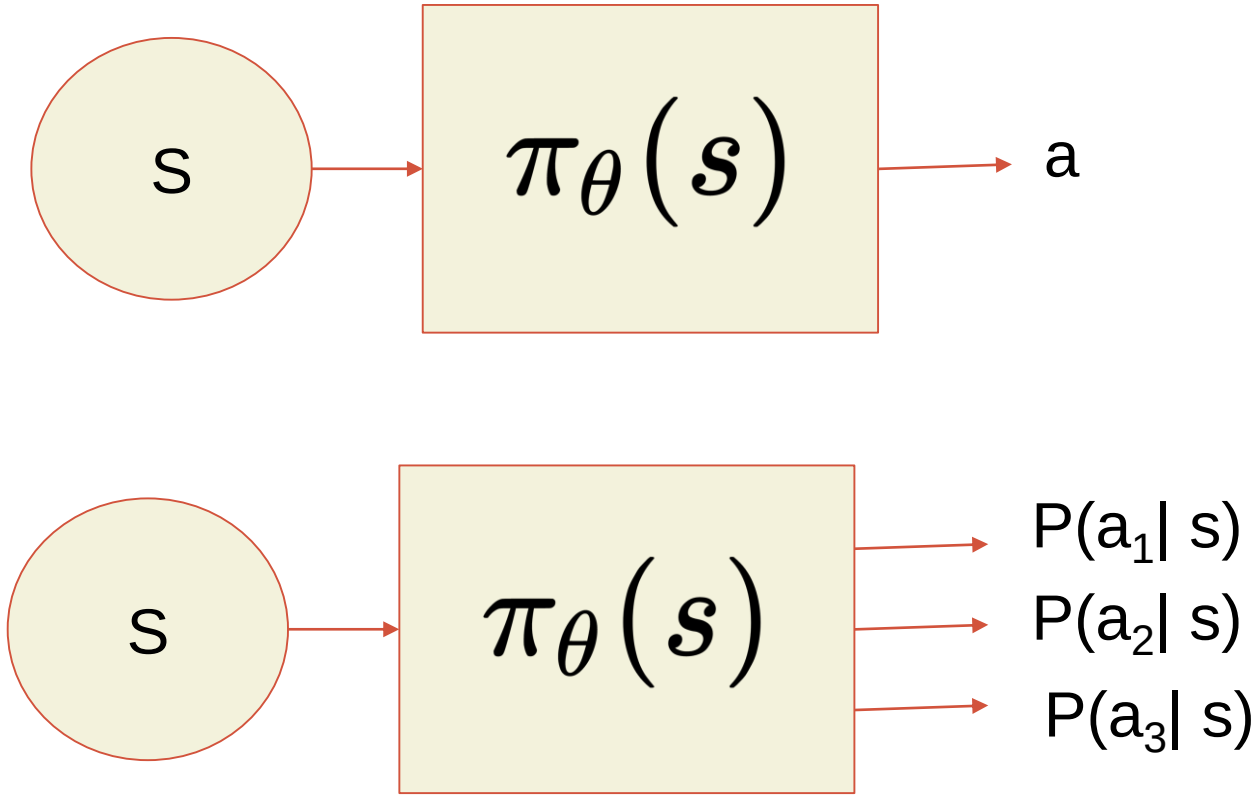
- policy is implicit
- if we already have Q, we have policy
- we just look at Q to get π

Policy gradient

- learns π directly explicitly
- use Q, V as a helper for learning π

Policy gradient

- Use Function Approximator to represent policy directly



Policy-based vs Value-based

Policy-based can learn continuous actions

Q-learning needs to $\text{argmax}_a Q(s,a)$

Policy-based can yield non-deterministic policy $P(a \mid s)$

Loss function for policy gradient

- Start-state objective

$$J(\theta) = E_{\pi(\theta)}[G|s_0] = V^{\pi}(s_0)$$

- Average-reward objective

$$J(\theta) = \sum_s d^{\pi}(s) \sum_a \pi(s,a) r(s,a)$$

* **d** is a stationary distribution of a Markov chain.

- One way to optimise these objectives is to use SGD.

Computing the gradient

Let's try to compute the gradient of the start-state objective

$$J(\theta) = E_{\pi_\theta}[G|s_0]$$

To evaluate this expectation, maybe we could try a one-sample Monte-Carlo estimator:

$$J(\theta) \approx r_0 + \gamma r_1 + \gamma^2 r_3 + \ldots$$

$$\nabla J(\theta) \approx \nabla_\theta [r_0 + \gamma r_1 + \gamma^2 r_3 + \ldots]$$ 

Doesn't quite work? The evaluated value does not depend on θ . Gradient can't be computed.

Computing the gradient

Maybe we can try change θ a little bit and find the difference?

$$J(\theta) \approx r_0 + \gamma r_1 + \gamma^2 r_3 + \dots$$

$$J(\theta + \delta\theta) \approx r'_0 + \gamma r'_1 + \gamma^2 r'_3 + \dots$$

$$\nabla J = \frac{J(\theta) - J(\theta + \delta)}{\delta}$$

Could work? But...

Looks very expensive and noisy to compute!

Maybe there is a better way?

Policy gradient

Let's start from the average-reward objective

$$J(\theta) = \sum_s d^{\pi}(s) \sum_a \pi_\theta(s,a) r(s,a)$$

For simplicity let's assume d(s) does not depend on θ

$$J(\theta) = \sum_s d(s) \sum_a \pi_\theta(s,a) r(s,a)$$

$$\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(s,a) r(s,a)$$

Almost there...

Policy gradient

REINFORCE trick!

$$\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \nabla_\theta \pi_\theta(s,a) r(s,a)$$

$$\nabla_\theta J(\theta) = \sum_s d(s) \sum_a \pi_\theta \nabla_\theta \log \pi_\theta(s,a) r(s,a)$$

We get this by sampling a playthrough using current policy (on-policy)

$$\nabla_\theta J(\theta) = E_\pi [\nabla_\theta \log \pi_\theta(s,a) r(s,a)]$$

$$\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(s,a) r(s,a)$$

Policy gradient theorem

There is a theorem…called policy gradient theorem say that we can replace $r(s,a)$ with $Q(s,a)$

$$\nabla_\theta J(\theta) = E_{\pi_\theta}\left[Q^{\pi_\theta}(s,a)\nabla_\theta log\pi_\theta(s,a)\right]$$

What is the gradient doing?

Goal maximize rewards

Push **π** towards directions of higher Q(s,a)

$$\nabla_\theta J(\theta) = E_{\pi_\theta}\left[Q^{\pi_\theta}(s,a)\nabla_\theta log\pi_\theta(s,a)\right]$$

Q(1,1) = 5

Q(1,2) = 2

- **π**(1,2) will have a higher weight. Policy gets push towards action 2

Notes on Policy gradient

Also known as REINFORCE or likelihood ratio.

Used by other ML fields when original loss is not differentiable (-Q in this case). Push network to produce lower loss.

$$\nabla_\theta J(\theta) = E_{\pi_\theta}[Q^{\pi_\theta}(s,a)\nabla_\theta log\pi_\theta(s,a)]$$

Example of non-differentiable functions

argmax (not maxpool)

sampling

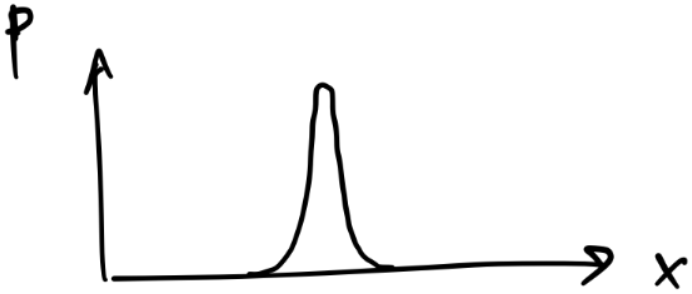
Encourage Exploration

- policy $\pi_\theta(a|s)$ could be too confident early
- Like, $\pi_\theta(a = a|s) = 1$
- This could lead to insufficient exploration
- Encourage exploration by “entropy term” $H(\pi_\theta)$
- We want to punish too low entropy
- New gradient rule: $\nabla_\theta J(\theta) \rightarrow \nabla_\theta J(\theta) + \nabla_\theta H(\pi_\theta)$

Entropy (H)

high entropy

low entropy



On policy and off policy algorithms

Q Learning: $Q^*(s_t, a_t) = r_{t+1} + \max_a Q^*(s_{t+1}, a)$

- you need a_t, s_t, r_{t+1}, s_{t+1} to satisfy the above equation
- you can get (a, s, r, s') from <u>any</u> *policy*
- Q learning is *off-policy*

Policy gradient: $\nabla_\theta J(\theta) = \mathbb{E}\left[Q_\pi(s,a)\nabla_\theta \log \pi_\theta(a|s)\right]$

- you need s, a and $Q_\pi(s,a)$
- Q_π needs to be from the current policy
- **s, a** needs to come from <u>current</u> policy
- policy gradient is *on-policy*

Notes on on vs off policy

Off-policy can learn from any policy

- Can use old experience
- Can use expert experience
- Sample efficient

On-policy can only learn from current policy

- Slow

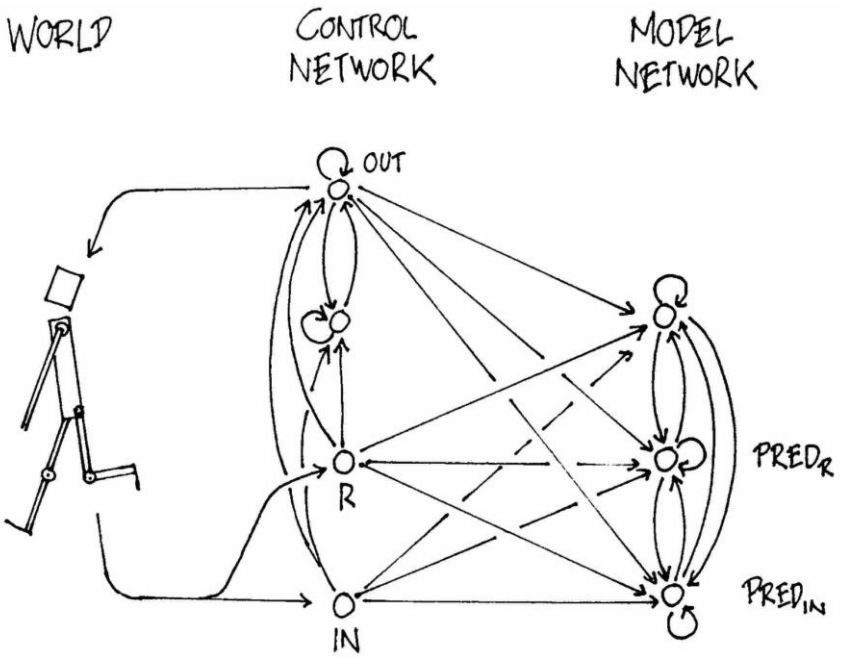
Model-based RL

RL's Model

RL's Model vs ML's Model

Think mental model that models the environment

Basically a model can guess the future or we have access to the environment so we can take alternative routes

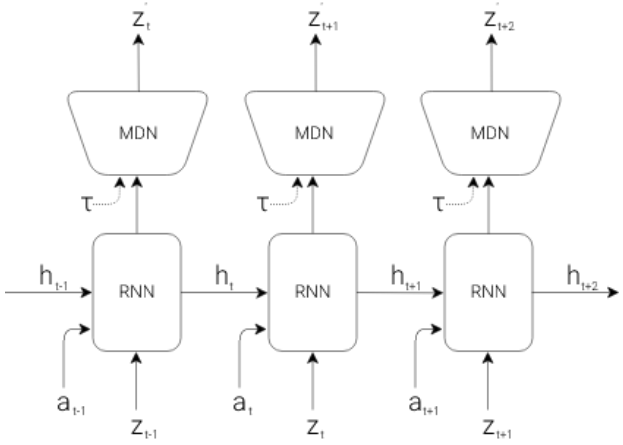


https://worldmodels.github.io/

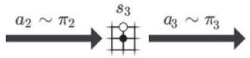
RL's Model

Guess the future

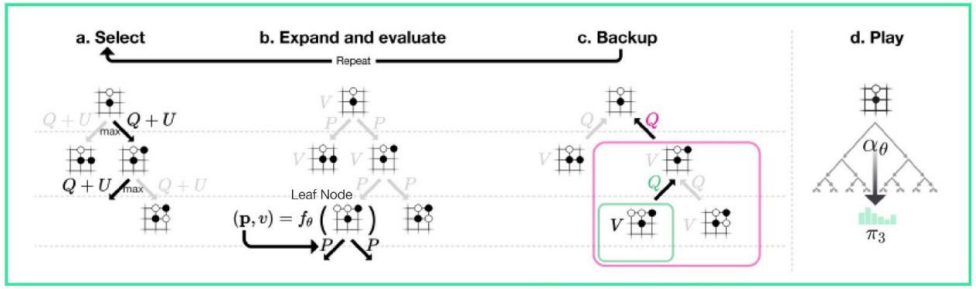
Or access to the environment so we can take alternative routes



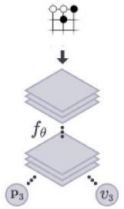
With a board position s_3



Use MCTS to compute π_3



Use f to compute p and v



Compute π_3 in MCTS

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s,a) + u(s,a))$$

$$u(s,a) = c_{\text{puct}} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$

$$(P(s,\cdot),\ v) = f_\theta(s)$$

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

$$W(s,a) = W(s,a) + v$$

$$Q(s,a) = \frac{W(s,a)}{N(s,a)}$$

$$\pi(a|s) = N(s,a)^{1/\tau} / \textstyle\sum_b N(s,b)^{1/\tau}$$

https://worldmodels.github.io/

https://medium.com/@jonathan_hui/monte-carlo-tree-search-mcts-in-alphago-zero-8a403588276a

Model-based RL

- In model-based RL, we first build the model of the environment
- Then use that model to directly search for the answer.
- The problem is … inaccurate model can give us bad policies…
- It is believed that if we can treat the uncertainty in the model correctly...model-based RL is the most efficient method!
- However, measuring uncertainty in the model is also very difficult.

Things to consider

When do we need RL?

- Your action affects the observation. Action has consequences (RL vs Bandit problem)
  - x_1 , a_1 ⟹ x_2
- The target behavior is difficult to be directly hard-coded.
  - How to move a snake robot?
- Collection of the data of a target behavior is difficult.

AlphaGo and why it works so well

Properties of the game Go

Deterministic
Fully observable
Rules are known (Model completely known)
Static
Fits MDP (Markov property)

Credit assignment problem

- An action can have consequences further away in time
- Some movements might not have any effect on the outcome

10 time steps later

$r_t = 0$

$r_{t+10} = 1$

Sparse reward problem

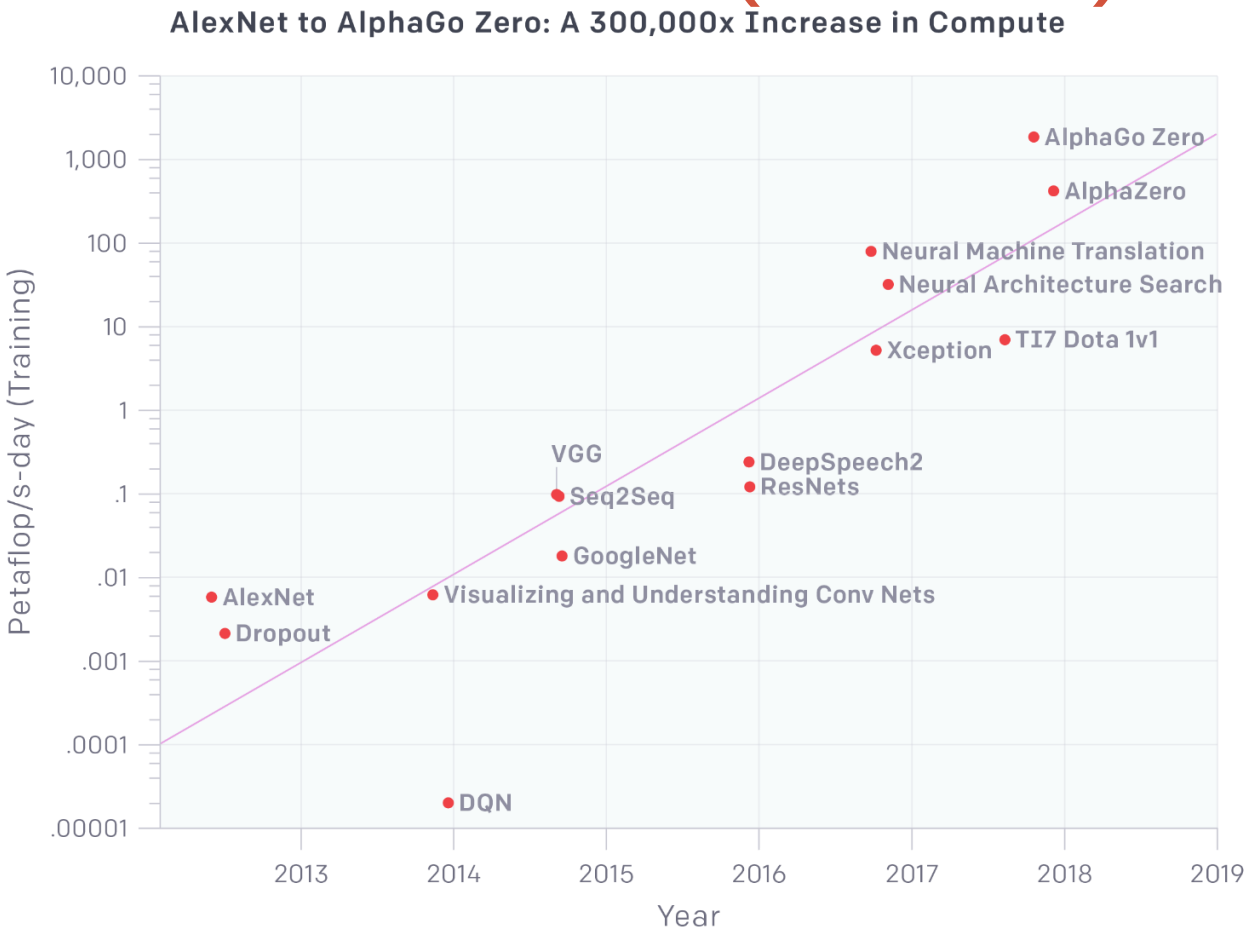
- Another problem is when rewards are sparse.
- Since model-free RL is just learning the correlations of trajectories and rewards… when there is no reward, RL cannot learn.
- Can we make it better?
  - Curiosity + intrinsic motivation?
  - Curriculum learning?
  - Hierarchical RL?

Designing the reward signal

- Reward design can be quite challenging..
- Naive reward design can lead to unexpected (cheating) behaviours!
- Example:

https://www.youtube.com/watch?v=tlOIHko8ySg

Problems with (current) RL

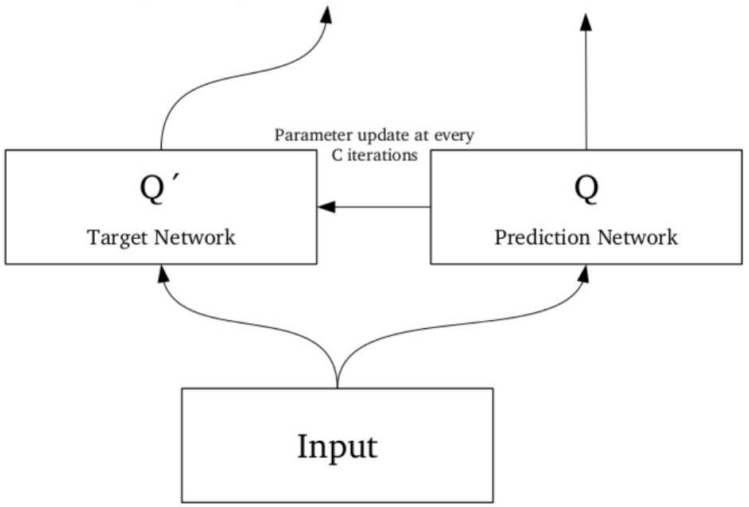


https://blog.openai.com/ai-and-compute/

Bias and Variance

- Bias and variance are really important in RL
- We want to reduce both of them as much as possible
  - DQN uses experience replay to reduce bias
  - DQN uses target network to reduce variance
  - Actor-Critic method uses baseline to reduce variance
  - Actor-Critic method uses parallel worker to reduce bias
  - etc.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i) \right)^2 \right]$$



https://www.slideshare.net/MuhammedKocaba/humanlevel-control-through-deep-reinforcement-learning-presentation

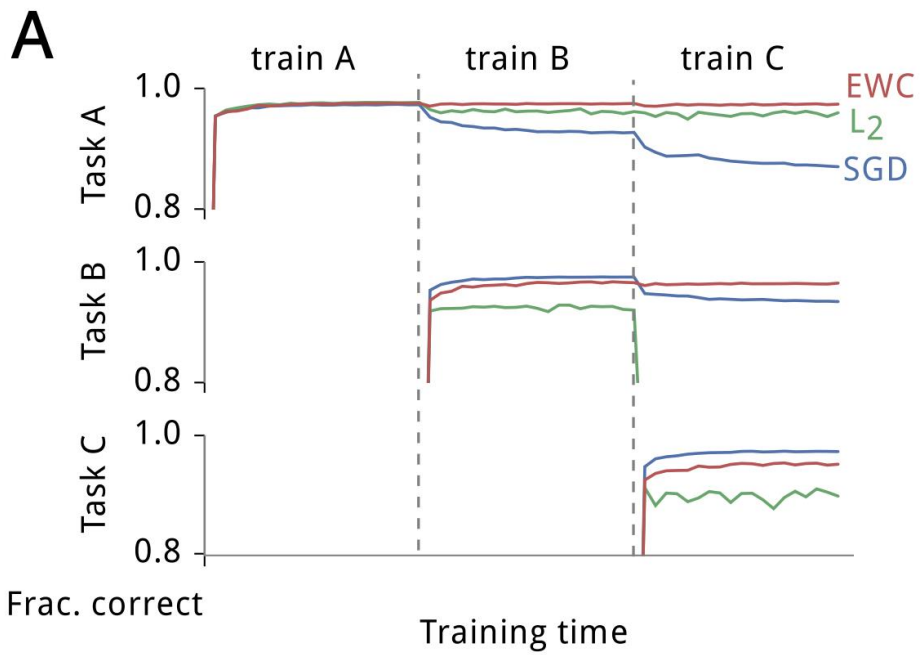
Exploration and Exploitation

- Many RL results assume that all states are visited infinitely often.
- Also, many RL algorithms are reduced into just an optimization problem.
- Therefore, nicely spread/informative data can help a lot!

- DQN uses epsilon-greedy for exploration
- Policy gradient uses entropy regularizer to encourage exploration

Optimisation problem

- Initialization problems
- Is SGD the best we can do?
- Catastrophic forgetting?



http://www.pnas.org/content/early/2017/03/13/1611835114.full.pdf?with-ds=yes

Current trends & open problems

- Intrinsic motivation, reward-bonus
- Imitation learning
- Multi-agent system and self-play
- Curriculum learning
- Model-based RL
- Robot learning + sim-to-real transfer learning
- etc...

Reinforcement learning

Elements of RL

- Environment, Agent, State, and MDP

Estimating Q

- Monte-Carlo, Bootstrap
- Deep learning as a function approximator

Policy learning

- Q-learning
- TD learning
- Policy gradient

Concepts

- Exploration vs Exploitation

Further learning

- Chula RL course
  - https://www.youtube.com/playlist?list=PLcBOyD1N1T-PyNUNA77lTYNCAeAMGxV5I
- Deepmind RL course 2021
  - https://www.deepmind.com/learning-resources/reinforcement-learning-lecture-series-2021

Project update

- 10 mins + 5 mins question
- Content
  - What are you doing?
  - Setup
  - Data
  - Method
  - Results (must have at least a baseline result! Or even a stupid baseline)
  - Next steps
- Must have enough detail for your classmates to understand
  - If you are working based on some technique never taught in class you must explain the technique