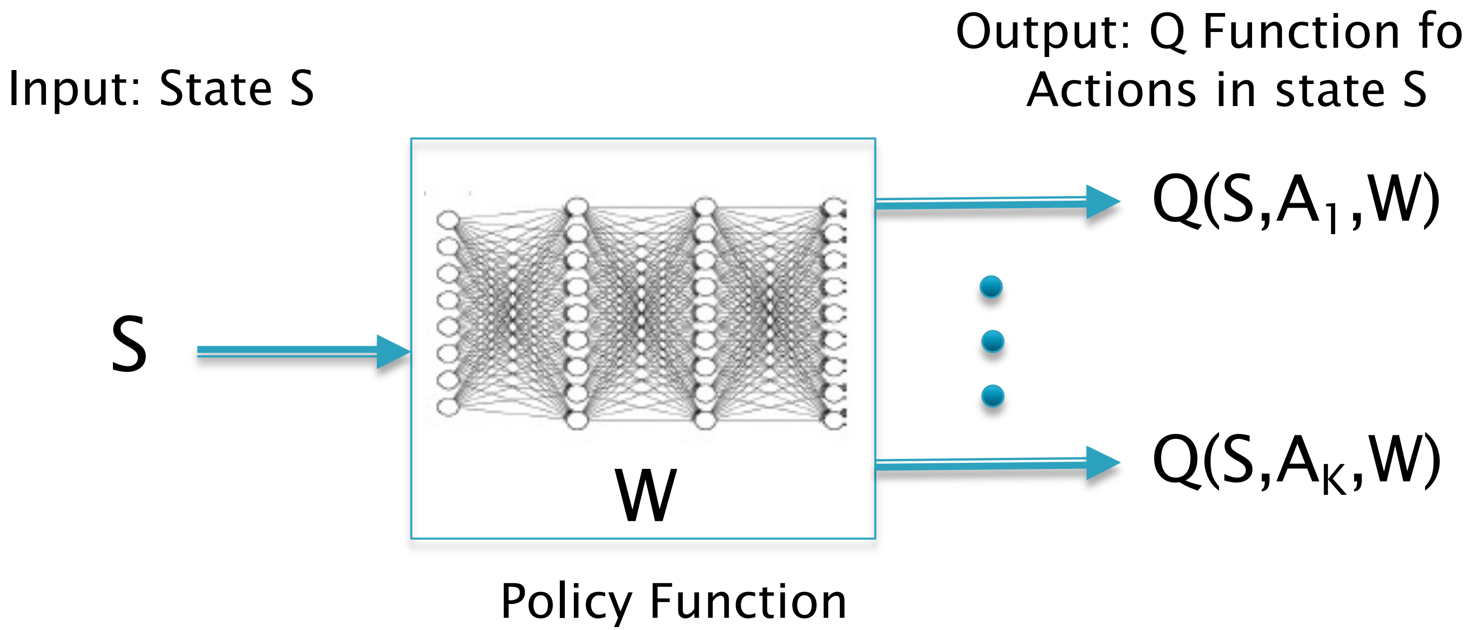
Policy Gradient Methods

Lecture 8

Subir Varma

Last Lecture



Focus was on Value Function Approximations

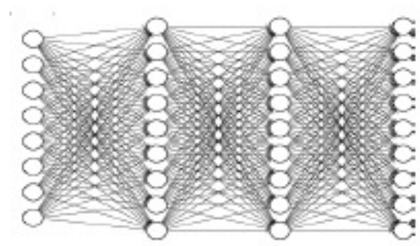
Policy was generated indirectly by taking the max of the Q functions

This Lecture

Input: State S

Output: Probability Distribution for Actions in state S

S



w

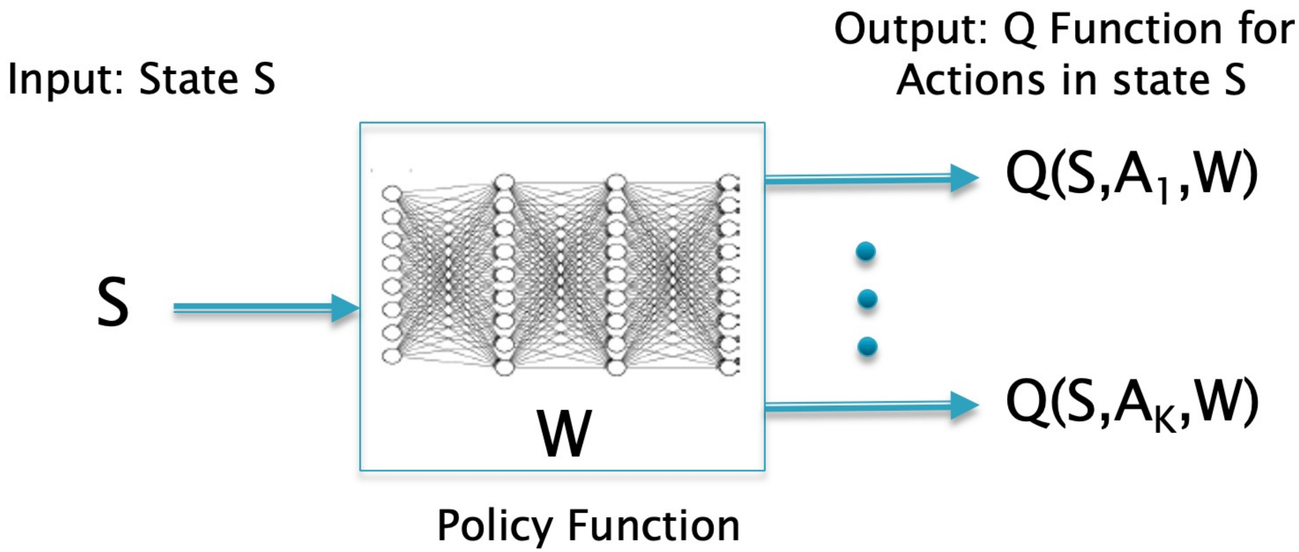
$\pi(A_1, S, W) = P(A_1|S, W)$

⋮

$\pi(A_K, S, W) = P(A_K|S, W)$

Generate the Optimal Policy Directly without using Q Functions

How to Train the Policy Network?



Basic Issue:
What Reward Function should we use for the
Policy Network such that it outputs the Optimal Policy

Reward Functions

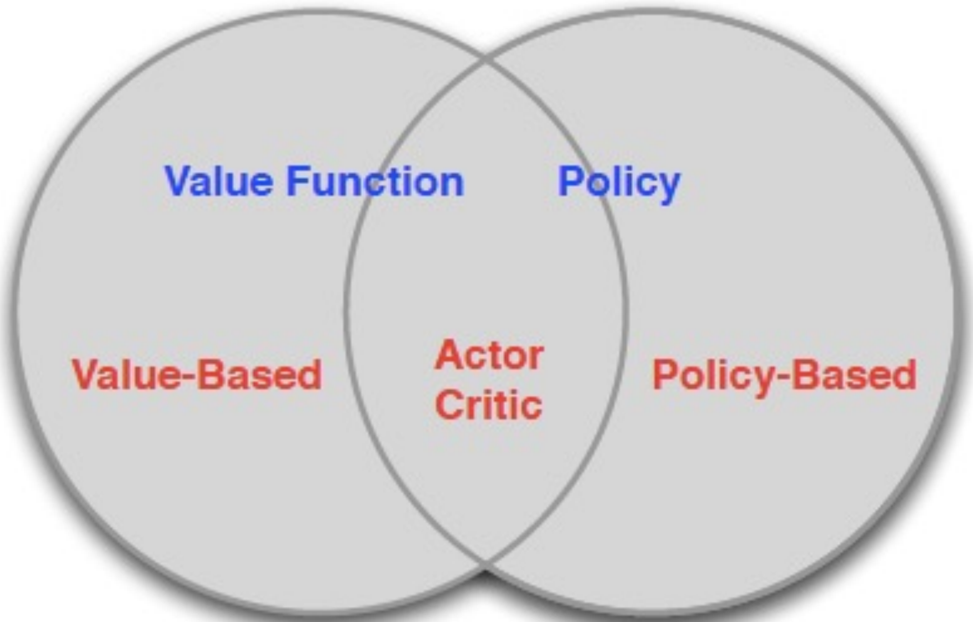
Three Techniques:

- The Reinforce Algorithm (also called Policy Gradients)
  - R = G(S)L, where G is the Reward to Go and L is the Reward Function used in Logistic Regression
  - Works only for MDPs with terminating Episodes
- Actor-Critic Algorithms
  - R = r + V_w(S) – V_w(S'), where r is the 1-step Reward and V_w is the NN computed Value Function for the MDP
  - Works for non-terminating MDPs
- Deterministic Policy Gradients Algorithm
  - R = Q_w(S,A), where Q_w(S,A) is the NN computed Q function for the MDP
  - Works for continuous Action Spaces

$$L(W) = \sum_{k=1}^{K} t_k \log y_k$$

Value Based and Policy Based RL

- Value Based
  - Learnt Value Function
  - Implicit policy (e.g. ϵ-greedy)
- Policy Based
  - No Value Function
  - Learnt Policy
- Actor-Critic
  - Learnt Value Function
  - Learnt Policy



MC
SARSA
Q-Learning
DQN

DPG

Reinforce

Pros and Cons of Policy Based RL

Advantages:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies

Disadvantages:

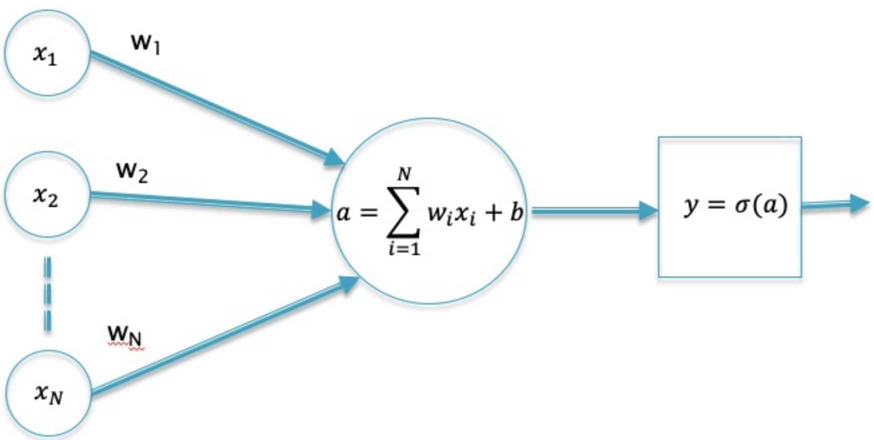
- Typically converge to a local rather than global optimum
- Evaluating a policy is typically inefficient and high variance

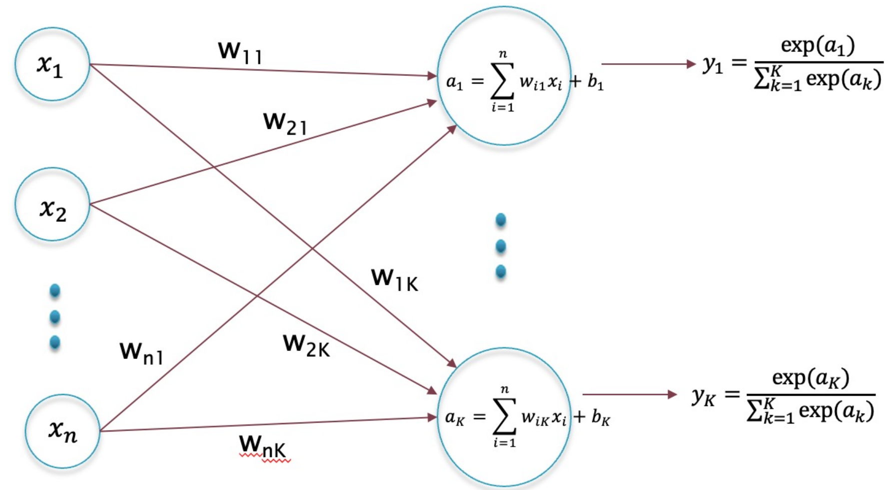
Typically required 10x the number of episodes to converge compared to DQN

Logistic Regression

Choose weights to Maximize Reward

$$L(W) = \frac{1}{M}\sum_{j=1}^{M}\sum_{k=1}^{K} t_k\,(j)\log y_k(j)$$



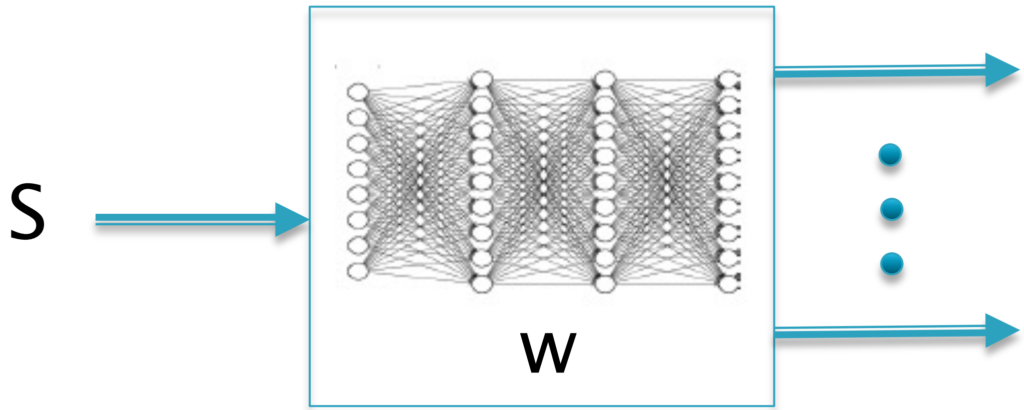


$$w_i \leftarrow w_i - \eta x_i(y - t)$$

$$w_{ik} \leftarrow w_{ik} - \eta x_i(y_k - t_k)$$

The Policy Gradient Algorithm (Reinforce)

Basic Idea behind Policy Gradients

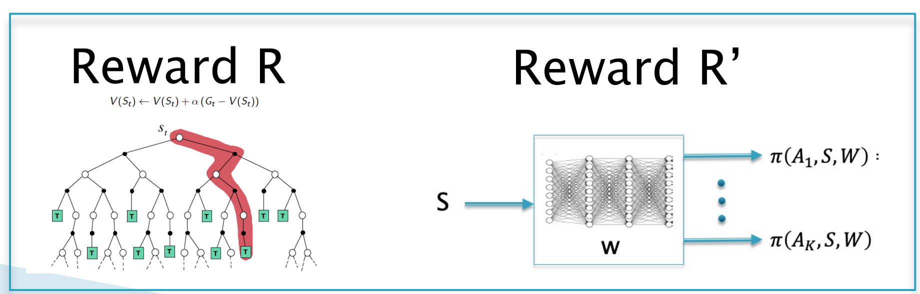


$$\pi(A_1, S, W) = P(A_1|S, W)$$

$$\pi(A_K, S, W) = P(A_K|S, W)$$

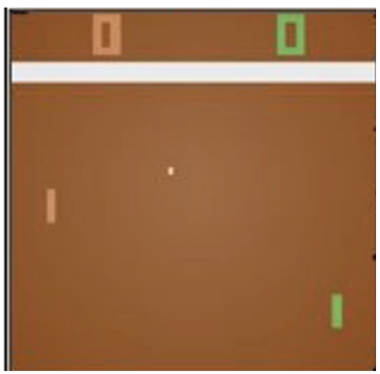
Problem: We want to find a Neural Network whose output gives the Optimal Policy that maximizes the RL Reward R

Find a Reward Function R' for the Neural Network, such that the the weights W that maximize R' also result in the Optimal Policy for the RL problem



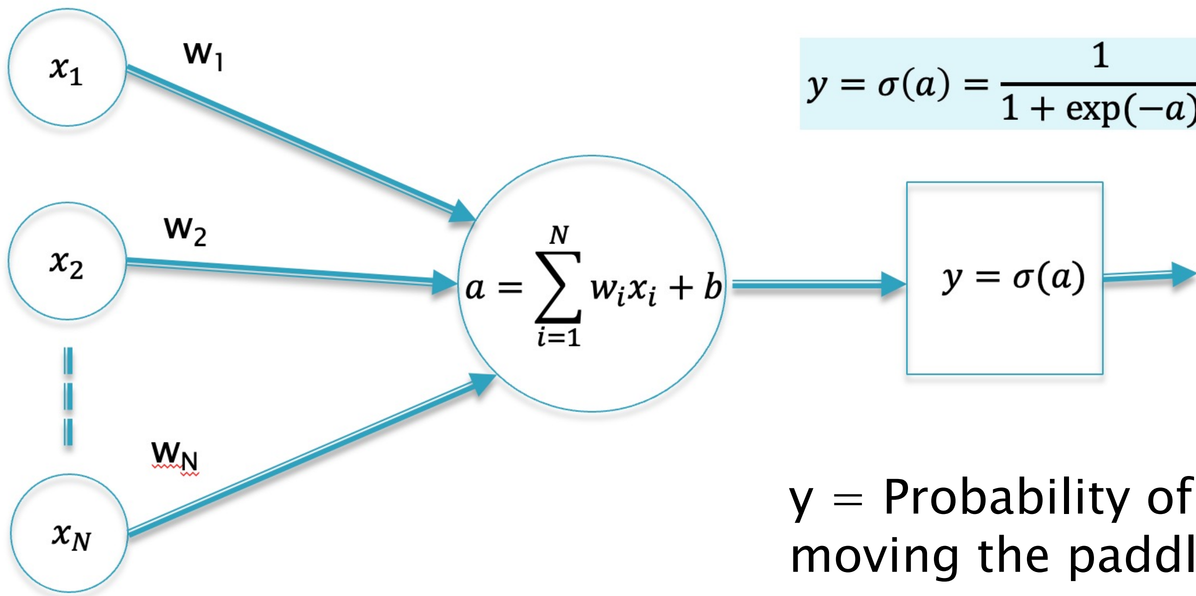
Playing Pong using Policy Gradients

Policy Network for Pong



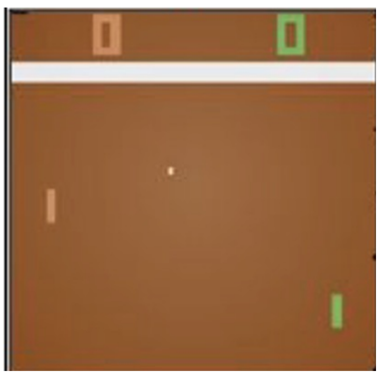
height width

[80 x 80]
array of
Pixels



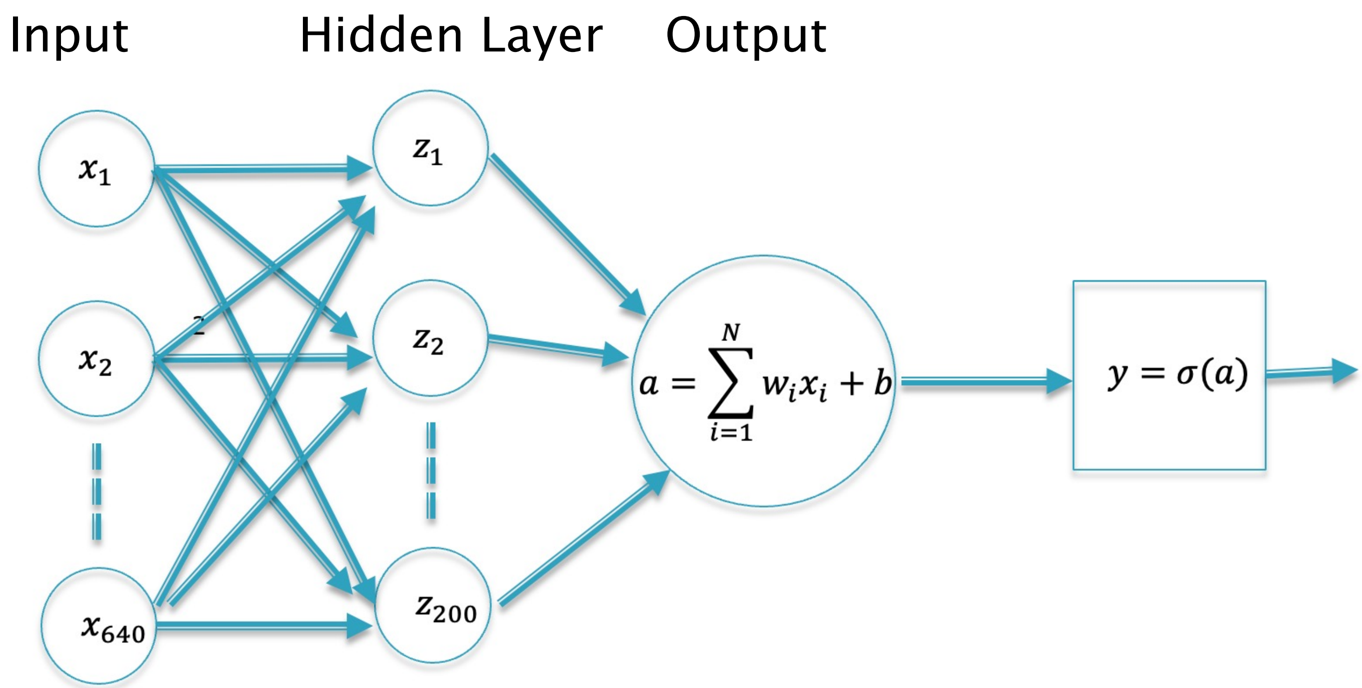
y = Probability of
moving the paddle UP

Policy Network for Pong



height width

[80 x 80]
array of
Pixels

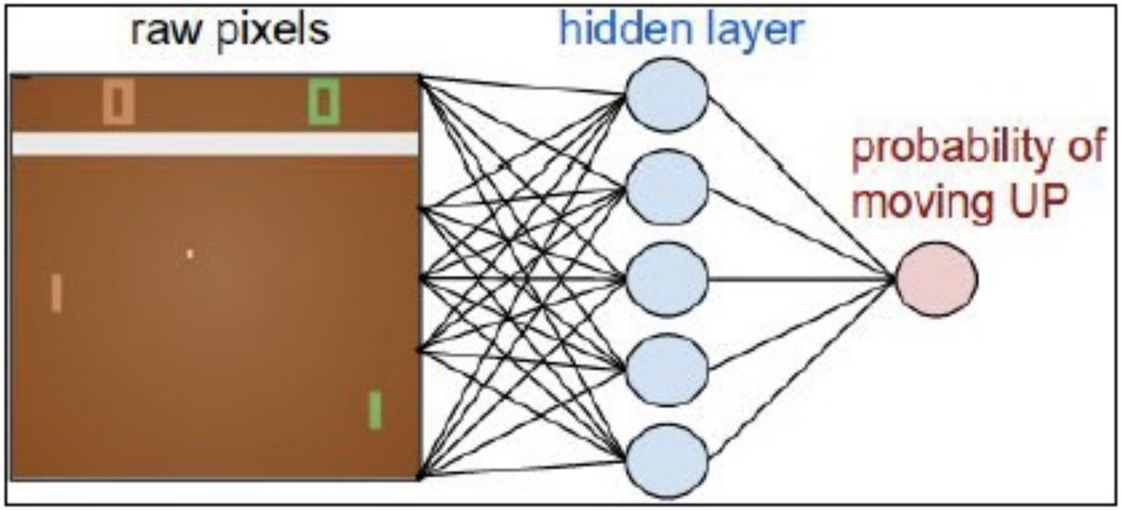


y = Probability of
moving the paddle UP

Policy Network for Pong

height width

[80 x 80]
array



E.g. 200 nodes in the hidden network, so:

[(80*80)*200 + 200] + [200*1 + 1] = **~1.3M parameters**

Layer 1 Layer 2

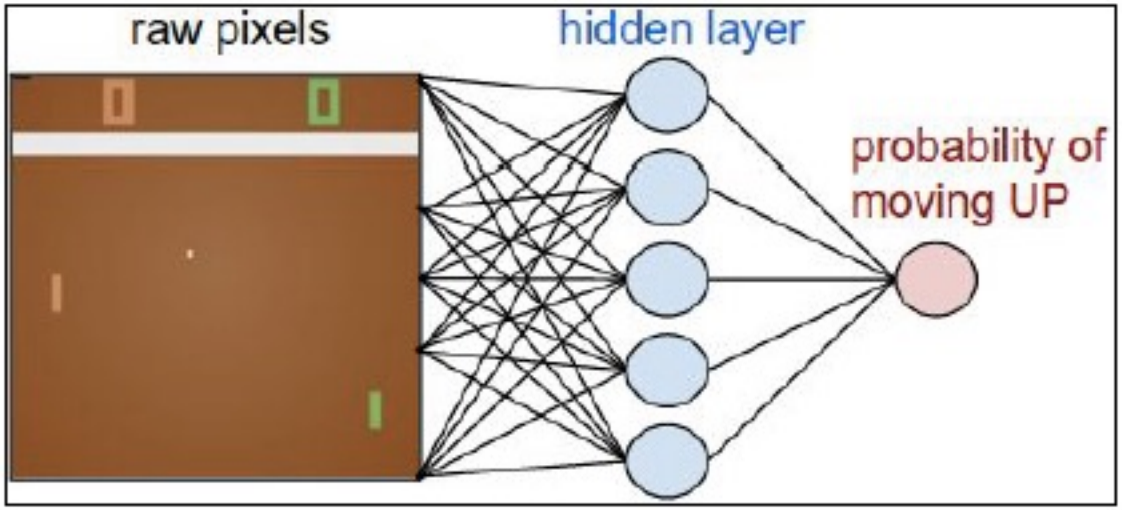
Policy Network for Pong

Neural Network with a single Hidden Layer

Game Screen

height width

[80 x 80] array of Pixels




```
h = np.dot(W1, x) # compute hidden layer neuron activations
h[h<0] = 0 # ReLU nonlinearity: threshold at zero
logp = np.dot(W2, h) # compute log probability of going up
p = 1.0 / (1.0 + np.exp(-logp)) # sigmoid function (gives probability of going up)
```

Suppose we had the training labels…
(we know what to do in any state)

(x1,UP)
(x2,DOWN)
(x3,UP)
...

Training Using
Supervised Learning

Labels

t = 1, UP
t = 0, DOWN

Suppose we had the training labels…
(we know what to do in any state)

(x1,UP)
(x2,DOWN)
(x3,UP)
…

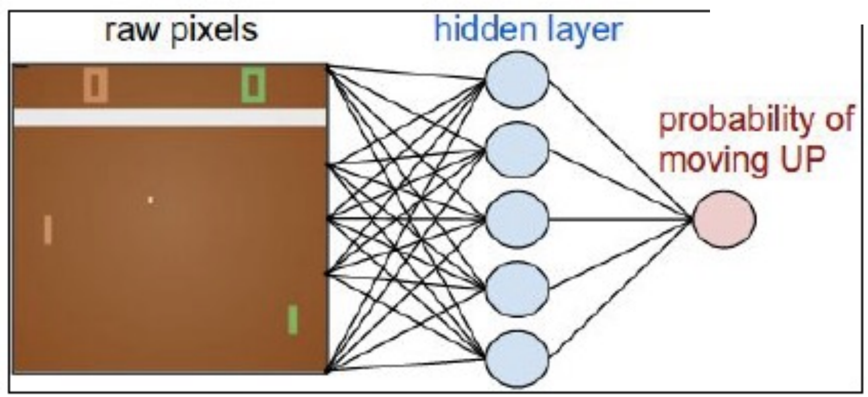
Maximize

Label

Network Output

$$L(W) = \frac{1}{M}\sum_{j=1}^{M}[t \log y + (1-t)\log(1-y)]$$

$$y(j) = \frac{1}{1+\exp(-\sum_{i=1}^{n} w_i x_i(j) - b)}$$

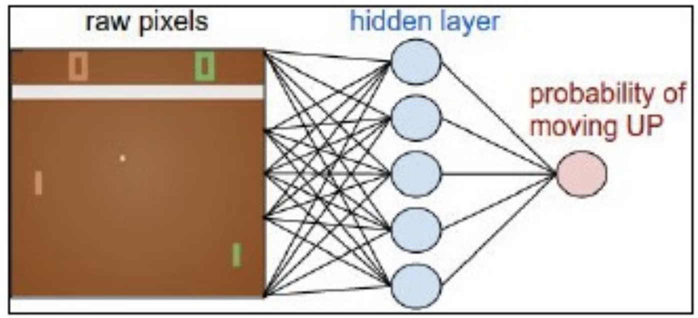


Except, we don't have labels…

Can't use Supervised Learning

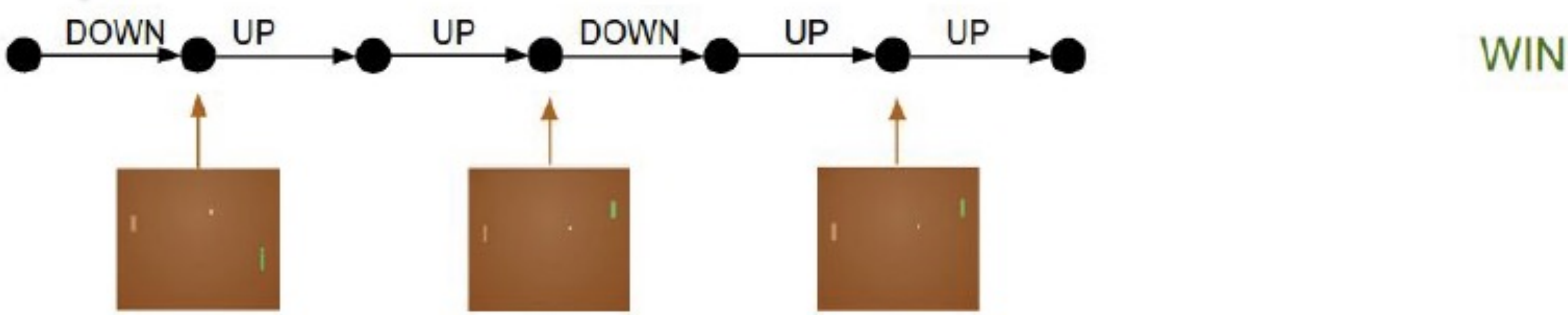
Let's just act according to our current policy...

As given by the Neural Network



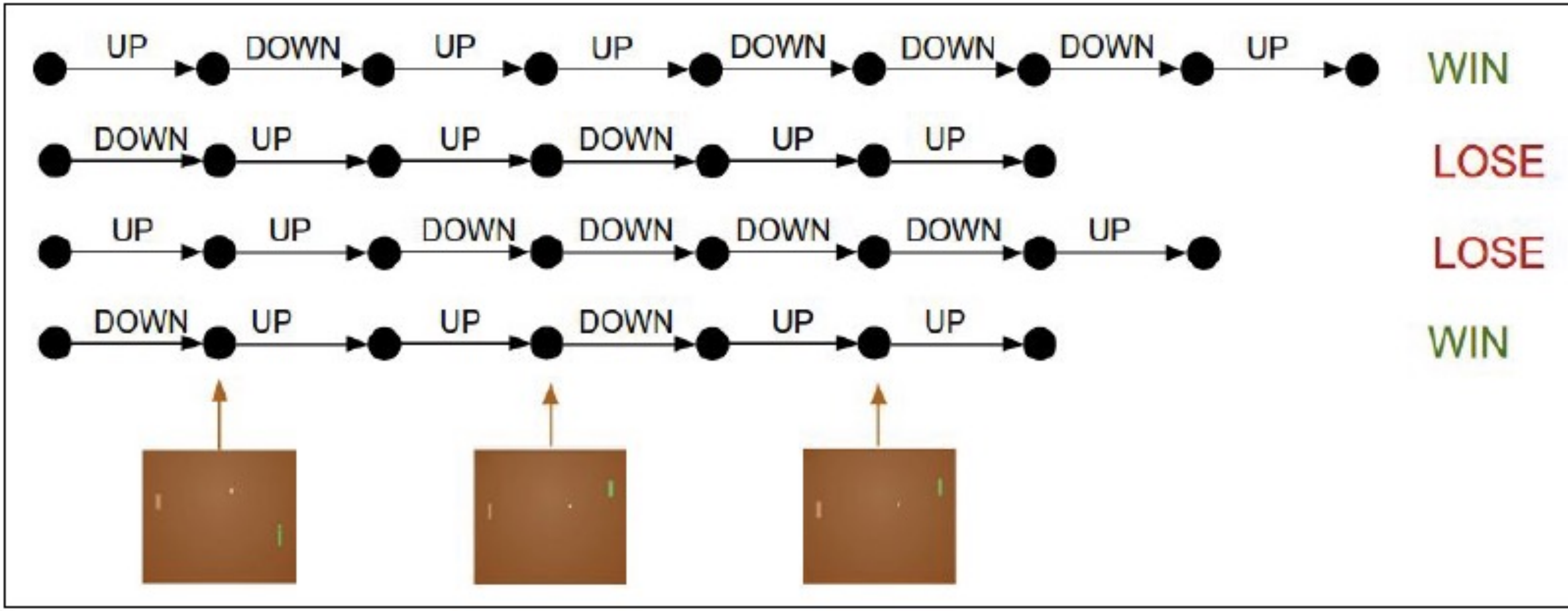
Rollout the policy and collect an episode

Actions are sampled from Network Output

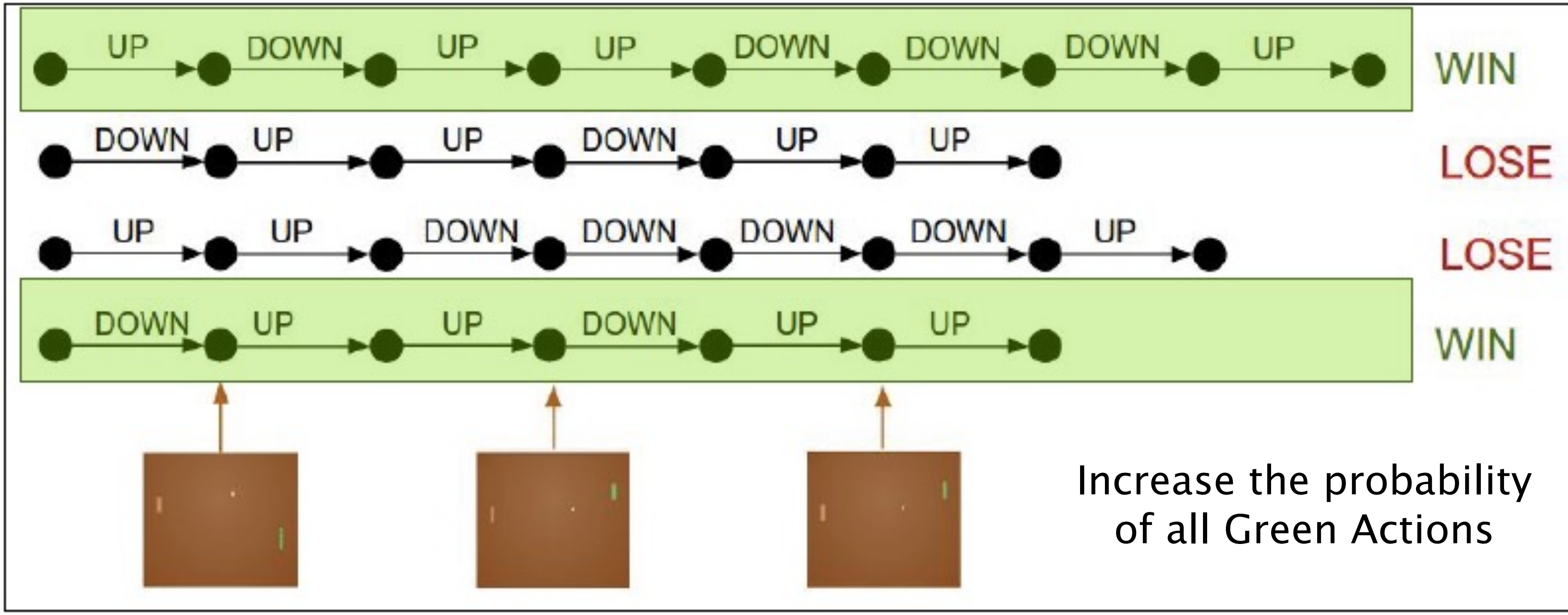


Collect many rollouts...

4 rollouts:



Assume that Every Action we took was correct!



Recall from Lecture 6: We can increase the probability of Action 1 by changing the weights as per

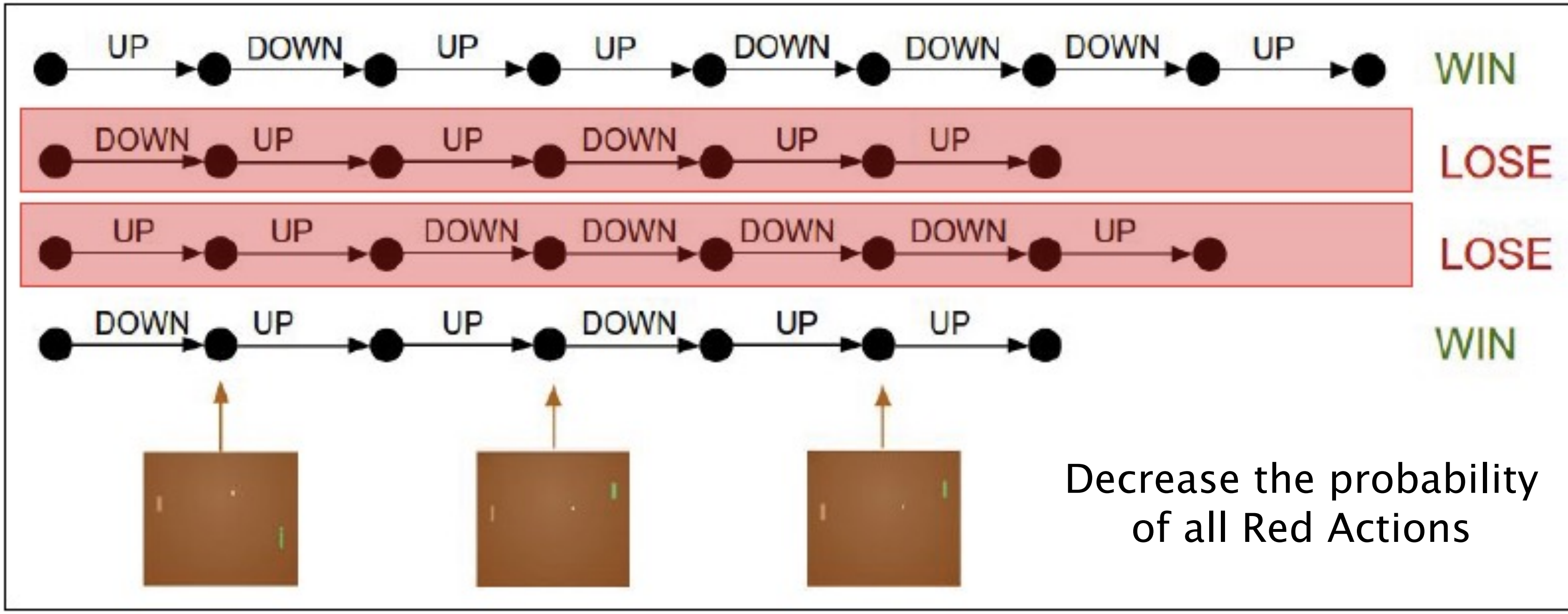
$$w_i \leftarrow w_i + \eta x_i (1 - y)$$

And we can increase the probability of Action 0 by changing the weights as per

$$w_i \leftarrow w_i - \eta x_i y$$

$$w_i \leftarrow w_i + \eta x_i (t - y)$$

Assume that Every Action we took was incorrect!



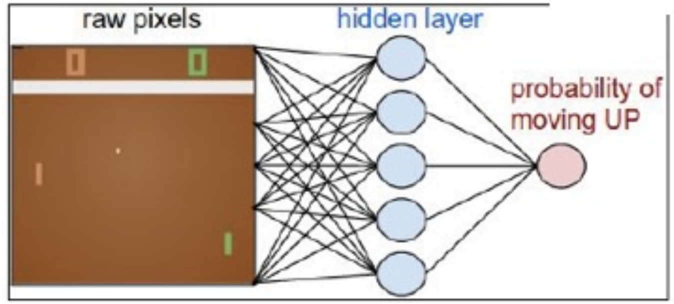
Recall from Lecture 6: We can decrease the probability of Action 1 by changing the weights as per

$$w_i \leftarrow w_i - \eta x_i(1-y)$$

And we can decrease the probability of Action 0 by changing the weights as per

$$w_i \leftarrow w_i + \eta x_i y$$

$$w_i \leftarrow w_i + \eta x_i(t - y)$$



Effectively, we are maximizing

$$J(W) = G\,[t \log y + (1-t)\log(1-y)] = \; GL$$

Where

- L is the Cross Entropy function used as a Reward Function for Logistic Regression
- G is the total Reward for a sample Monte Carlo Episode. In this example G = +1 or -1. We will show that in general G= $\sum_{j=1}^{T} R_j$

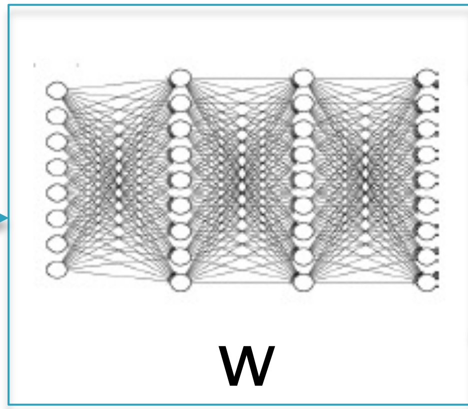
$$\boxed{w_i \leftarrow w_i + \eta G x_i (t - y)}$$

With K Actions

Output: Probability Distribution for Actions in state S

Input: State S

S



$\pi(A_1, S, W)$

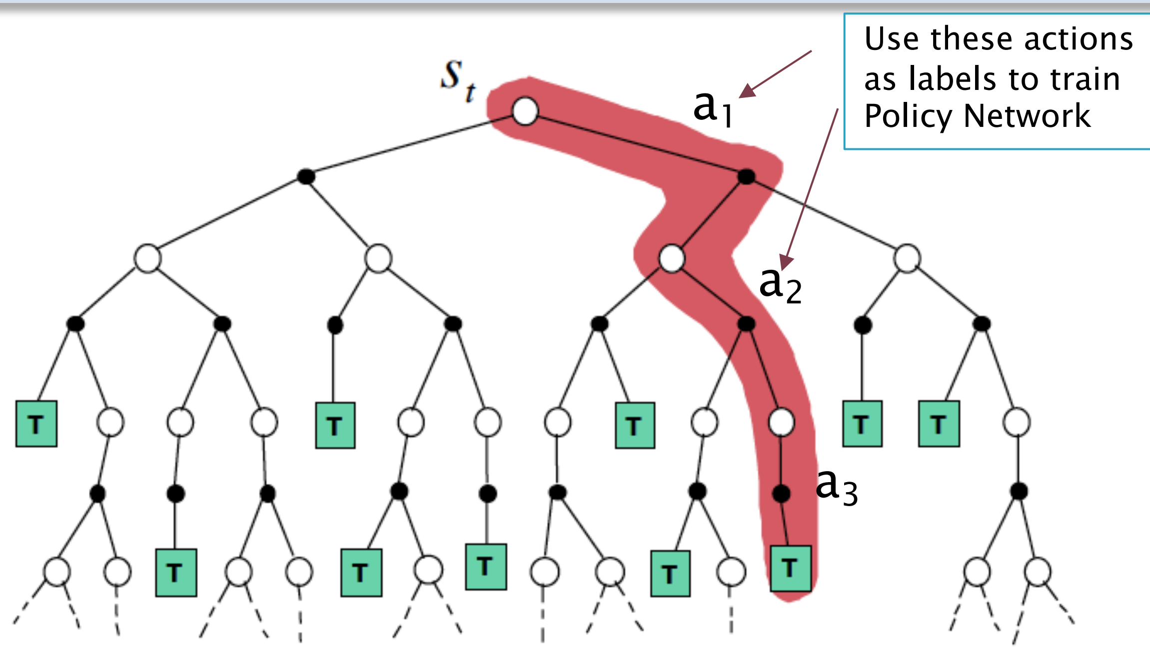
$\pi(A_K, S, W)$

Maximize $\quad J(W) = G \sum_{k=1}^{K} t_k \log \pi_k$

$$\boxed{w_{ik} \leftarrow w_{ik} + \eta G x_i (t_k - y_k)}$$

Monte Carlo Policy Gradients: Reinforce

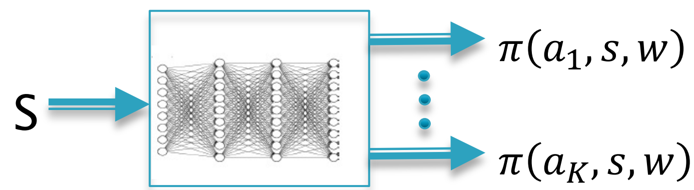
$$\mathrm{w} \leftarrow w + \eta \frac{\partial J}{\partial w} = w + \eta G \frac{\partial L}{\partial w}$$



Episode 1: Actions are chosen according to Policy $\pi 1$

Episode 2: Actions are chosen according to Improved Policy $\pi 2$

$\mathrm{J}(W) = G \sum_{k=1}^{K} t_k \log \pi_k = \mathrm{GL(W)}$



$\mathrm{G} = \sum_{j=1}^{T} R_j$: The total reward for the Episode

$w_{ik} \leftarrow w_{ik} + \eta G x_i (t_k - y_k)$

$w_{ik} \leftarrow w_{ik} + \eta G \dfrac{\partial L}{\partial w_{ik}}$

Choose Weights to Maximize GL(W)

The Reinforce Algorithm

REINFORCE:

Initialize policy parameters θ arbitrarily

for each episode $\{s_1, a_1, r_2, \cdots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

  for $t = 1$ to T - **do**

    $\theta \leftarrow \theta + \alpha$ $G \frac{\partial L_t}{\partial w}$

  endfor

endfor

return θ

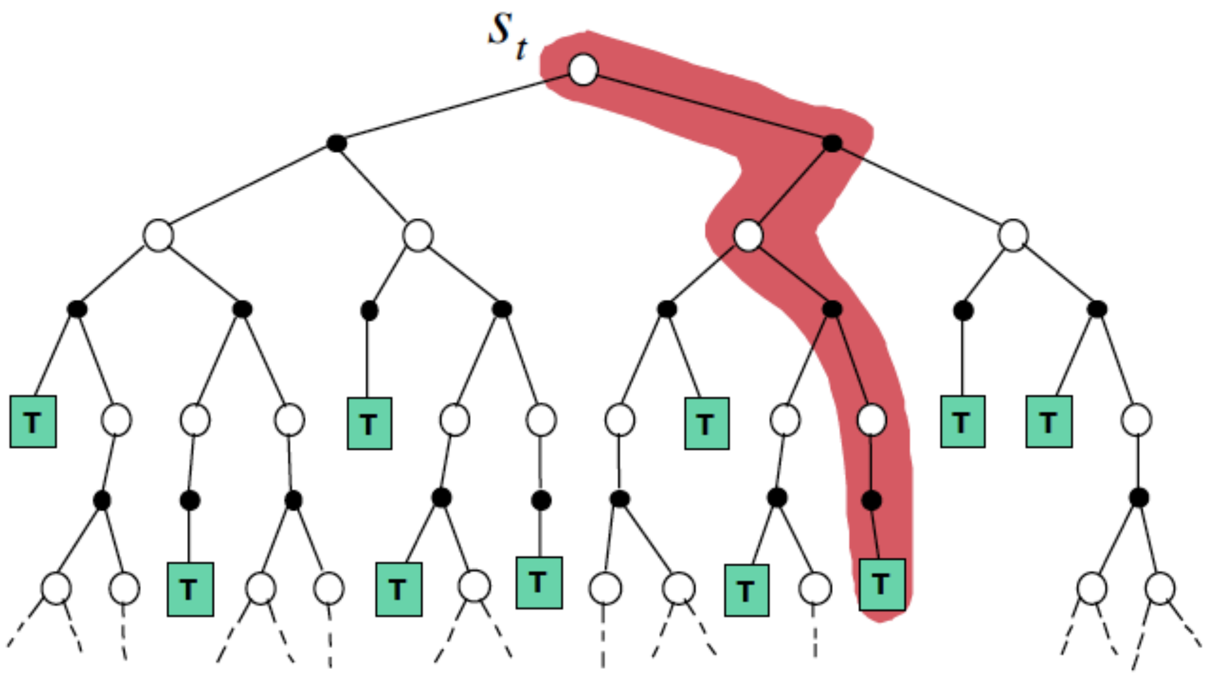
Total Reward for the Episode

Compute gradient using action a_t as the Label at time t

Derivation of Likelihood Ratio Formula

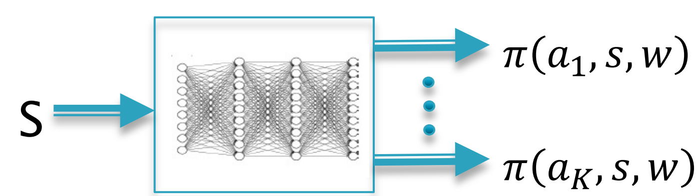
Monte Carlo: Policy Gradients

$$w \leftarrow w + \eta \frac{\partial J}{\partial w}$$



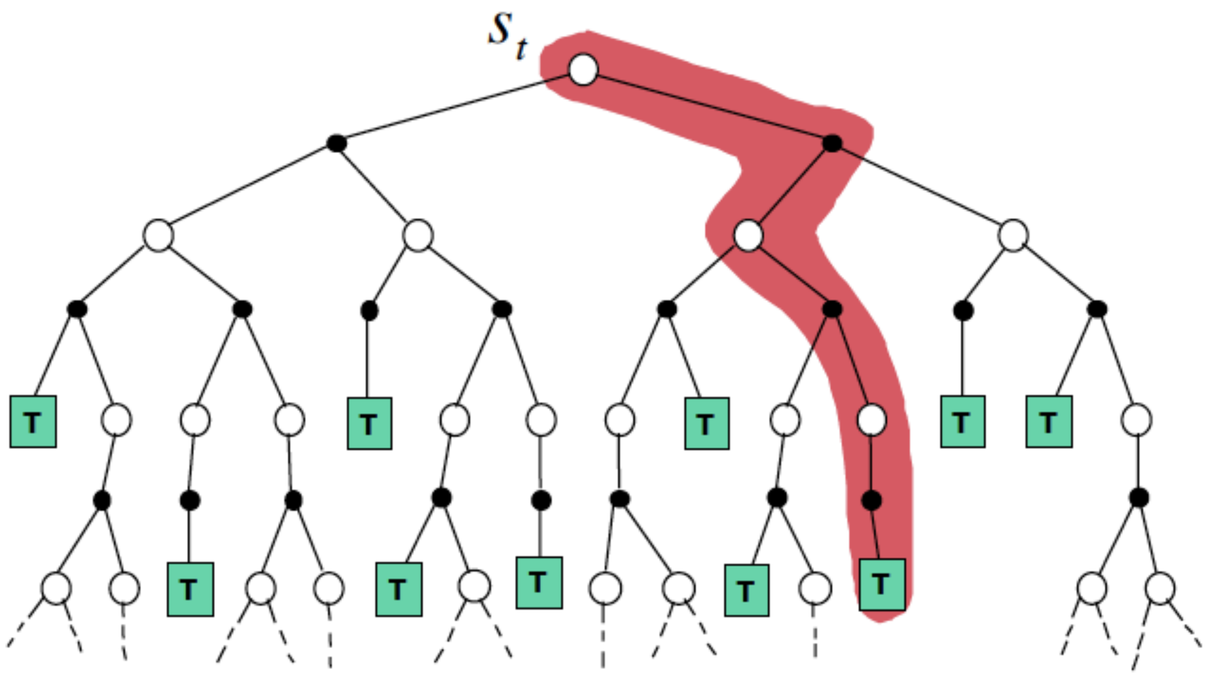
Episode 1: Actions are chosen according to Policy $\pi 1$

Episode 2: Actions are chosen according to Improved Policy $\pi 2$



Monte Carlo: Policy Gradients

$$w \leftarrow w + \eta \frac{\partial J}{\partial w}$$

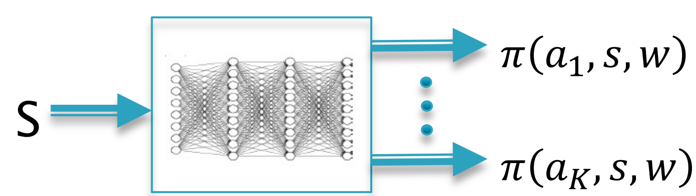


Episode 1: Actions are chosen according to Policy $\pi 1$

Episode 2: Actions are chosen according to Improved Policy $\pi 2$

What is an appropriate
Reward Function J for the Policy Network?

If we change the parameters w to optimize J, then the Policy should improve.



Reward Function for Policy Network

Given a history under some policy π:

$(s_1, a_1, r_1), (s_2, a_2, r_2), \ldots, (s_M, a_M, r_M)$

Total Expected Reward

$$J(W) = E_\pi[\sum_{t=1}^{M} r(S_t, A_t)]$$

$$\mathrm{R}(\tau_e) = \sum_{t=1}^{M} r(S_t, A_t)$$

This can be split up into episodes:

$$J(W) = \sum_{e=1}^{E} P^\pi(\tau_e, w)\, \mathrm{R}(\tau_e)$$

Total reward for episode τ_e

Probability of generating episode τ_e under Policy π

Reward Function for Policy Network

By splitting the transitions into episodes, the Reward Function can be estimated using sample episodes:

$$J(W) = \sum_{e=1}^{\varepsilon} P^{\pi}(\tau_e, w)\,\mathrm{R}(\tau_e) \approx \frac{1}{\mathcal{E}} \sum_{e=1}^{\mathcal{E}} R(\tau_e)$$

Algorithm:

1. Generate sample episodes using weights w (which results in policy π)
2. Use the data from the sample episodes to tweak the weights, so as to increase the Reward Function J

$$w \leftarrow w + \eta \frac{\partial J}{\partial w}$$

   This results in a new improved policy

3. Go back to step 1 and repeat

How do we compute this gradient ??

Estimating the Reward Gradient from Sample Episodes

J(W) $= E[\mathrm{R}(\tau_e)] = \sum_{e=1}^{\mathcal{E}} P^{\pi}(\tau_e, w)\,\mathrm{R}(\tau_e)$

$$\frac{\partial J(W)}{\partial w} = \sum_{e=1}^{\mathcal{E}} \frac{\partial P^{\pi}(\tau_e, w)}{\partial w}\mathrm{R}(\tau_e)$$

$$\frac{\partial J(W)}{\partial w} = \sum_{e=1}^{\mathcal{E}} \frac{P^{\pi}(\tau_e, w)}{P^{\pi}(\tau_e, w)}\frac{\partial P^{\pi}(\tau_e, w)}{\partial w}\mathrm{R}(\tau_e)$$

$$\frac{\partial J(W)}{\partial w} = \sum_{e=1}^{\mathcal{E}} P^{\pi}(\tau_e, w)\frac{\partial log P^{\pi}(\tau_e, w)}{\partial w}\mathrm{R}(\tau_e)$$

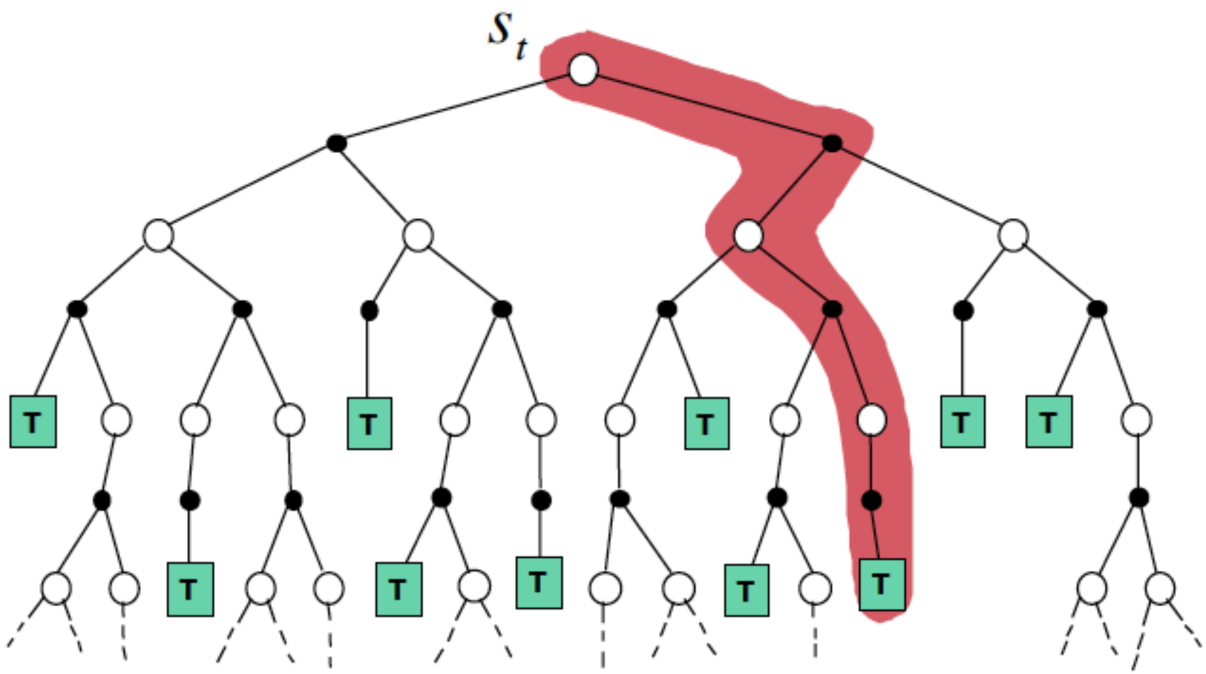
$$= E\left[\frac{\partial log P^{\pi}(\tau_e, w)}{\partial w}\mathrm{R}(\tau_e)\right]$$

$$\approx \frac{1}{\mathcal{E}}\sum_{e=1}^{\mathcal{E}} \frac{\partial log P^{\pi}(\tau_e, w)}{\partial w}\mathrm{R}(\tau_e)$$

How to compute this ?

Implies that the Gradient can be computed directly from data generated by sample episodes!!

Probability of Generating an Episode



$$P^{\pi}(\tau_e, w) = \prod_{i=1}^{T} P(s_{i+1}|s_i, a_i)\pi_W(a_i|s_i)$$

Reward Gradient can be Estimated Model Free

$$P^{\pi}(\tau_e, w) = \prod_{i=1}^{T} P(s_{i+1}|s_i, a)\pi_W(a_i|s_i)$$

$$\log P^{\pi}(\tau_e, w) = \log \prod_{i=1}^{T} P(s_{i+1}|s_i, a_i)\pi_W(a_i|s_i)$$

$$= \sum_{i=1}^{T} \log P(s_{i+1}|s_i, a_i) + \sum_{i=1}^{T} \log \pi_W(a_i|s_i)$$

$$\mathrm{L}(W) = [t(j)\log y(j) + (1 - t(j))\log(1 - y(j))]$$

$$\frac{\partial \log P^{\pi}(\tau_e, w)}{\partial w} = \sum_{i=1}^{T} \frac{\partial \log \pi_W(a_i|s_i)}{\partial w}$$

This step makes the Policy Gradient Algorithms Model Free!!

It follows that

$L(s_i, a_i)$

$$\frac{\partial J(W)}{\partial w} = E[\sum_{i=1}^{T} \frac{\partial \log \pi_W(a_i|s_i)}{\partial w} \mathrm{R}(\tau_e)]$$

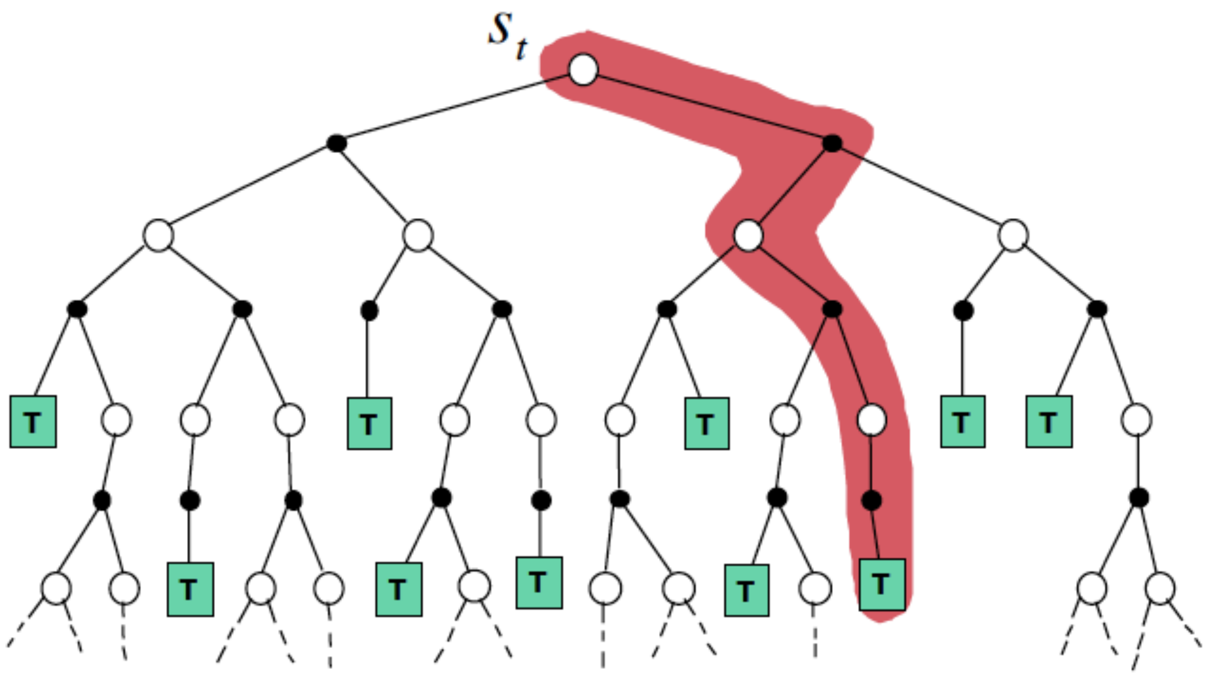
$$= E[\sum_{i=1}^{T} \frac{\partial L_i}{\partial w} R(\tau_e)] \approx \frac{1}{\mathcal{E}} \sum_{e=1}^{\mathcal{E}} (\sum_{i=1}^{T} \frac{\partial L_i}{\partial w}) R(\tau_e)$$

This gradient can potentially be evaluated by sampling without knowledge of the model

$$L(W) = \sum_{k=1}^{K} t_k \log \pi_k$$

Monte Carlo Policy Gradients: Reinforce

$$w \leftarrow w + \eta G \frac{\partial L}{\partial w}$$



Episode 1: Actions are chosen according to Policy $\pi 1$

Episode 2: Actions are chosen according to Improved Policy $\pi 2$

$$\frac{\partial J(W)}{\partial w} = \frac{1}{\mathcal{E}} \sum_{e=1}^{\mathcal{E}} \left(\sum_{i=1}^{T} \frac{\partial L_i}{\partial w} \right) G_e$$

Full Gradient Descent

$$\frac{\partial J(W)}{\partial w} = \frac{\partial L}{\partial w} \; G_e$$

Stochastic Gradient Descent

The Reinforce Algorithm

REINFORCE:

Initialize policy parameters θ arbitrarily

for each episode $\{s_1, a_1, r_2, \cdots, s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**

  for $t = 1$ to T - **do**

    $\theta \leftarrow \theta + \alpha$ $\frac{\partial L_i}{\partial w} G_e$

  endfor

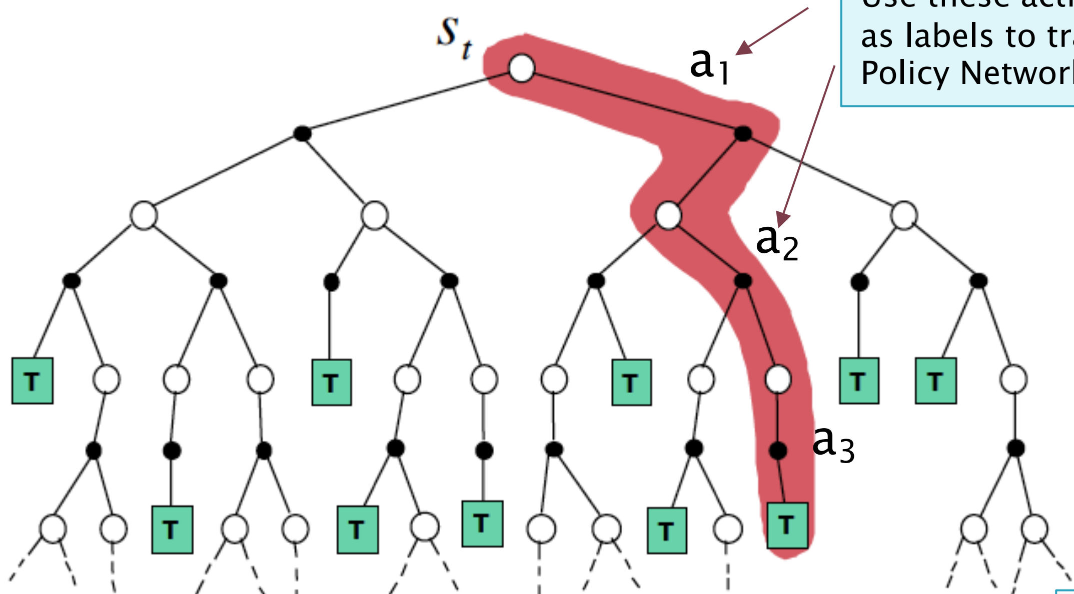
endfor

return θ

How to compute gradients ?

Monte Carlo Policy Gradients: Reinforce

$$w \leftarrow w + \eta \frac{\partial J}{\partial w} = w + \eta G \frac{\partial L}{\partial w}$$

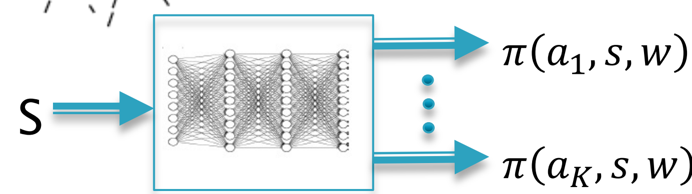


Use these actions as labels to train Policy Network

Episode 1: Actions are chosen according to Policy $\pi 1$

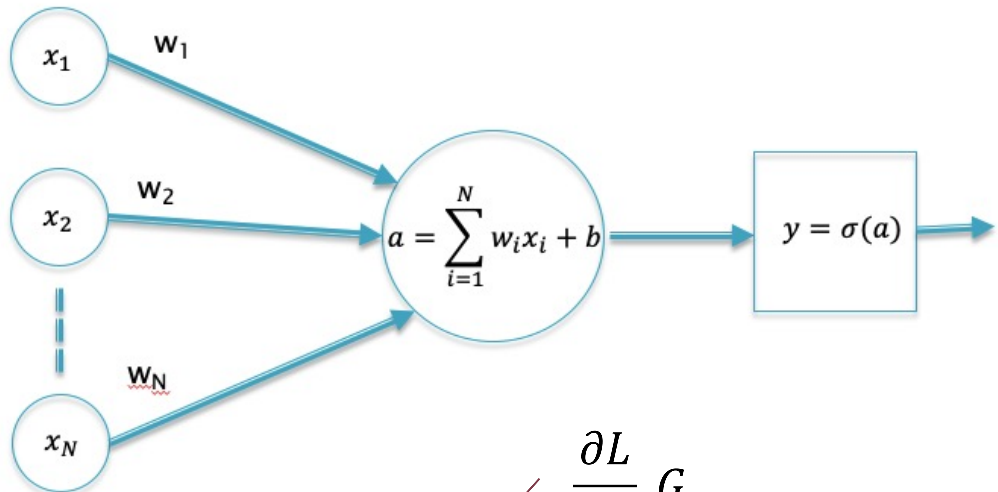
Episode 2: Actions are chosen according to Improved Policy $\pi 2$

$G_i = \sum_{j=1}^{T} R_j$: The total reward for the episode



If G for the episode is positive, then increase the probability of taking all actions in the episode and vice versa

The case of K = 2

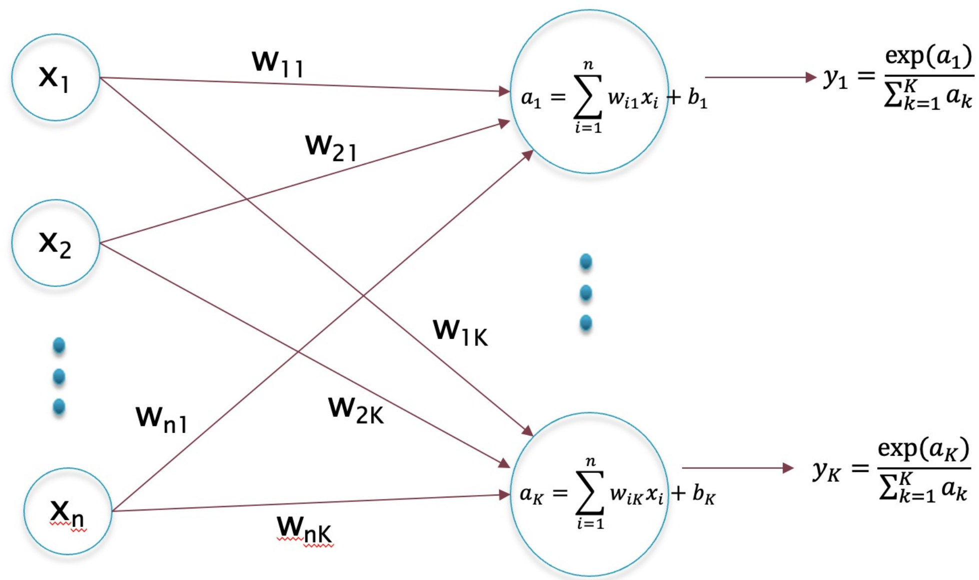


$$\frac{\partial L}{\partial w} \; G_e$$

$$w_i \leftarrow \begin{cases} w_i + \eta x_i(S) G(S)[1 - y(S)], & if \; A = 1 \\ w_i - \eta x_i(S) G(S) y(S), & if \; A = 0 \end{cases}$$

Where A is the Action that the Agent takes in State S
And G(S) is the Total Reward for Episode

The case of K > 2



$$w_{ik} \leftarrow \begin{cases} w_{ik} + \eta x_i(S) G(S)[1 - y_k(S)], & if\ k = A \\ w_{ik} - \eta x_i(S) G(S) y_k(S), & if\ k \neq A \end{cases}$$

Where A is the Action that the Agent takes in State S
And G(S) is the Total Reward for Episode

Gradient Computation in Supervised Learning

Maximum likelihood:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
loss = tf.reduce_mean(negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

Computes $\mathrm{L} = \sum_{k=1}^{K} t_k \log \pi_k$

Gradient Computation in Policy Gradients

Policy gradient:

```
# Given:
# actions - (N*T) x Da tensor of actions
# states - (N*T) x Ds tensor of states
# q_values – (N*T) x 1 tensor of estimated state-action values
# Build the graph:
logits = policy.predictions(states) # This should return (N*T) x Da tensor of action logits
negative_likelihoods = tf.nn.softmax_cross_entropy_with_logits(labels=actions, logits=logits)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

Computes $\mathrm{L} = \sum_{k=1}^{K} t_k \log \pi_k$

Computes $\mathrm{L} = (\sum_{\boldsymbol{i=1}}^{\boldsymbol{M}} \boldsymbol{r_i}) \sum_{k=1}^{K} t_k \log \pi_k$

Issues

- High Variance
- The algorithm takes a long time to converge
- It is an On-Policy algorithm: Existing training data cannot be reused

Variance Reduction

Techniques for Variance Reduction

1. Exploiting Causality: Reward to-go
2. Discounting
3. Baselines
4. Actor-Critic Algorithms

Exploiting Causality: Reward To-Go

$$J(W) = L(W) \sum_{j=1}^{T} R_j$$

Observation: Action taken at time i can only influence the rewards from i onwards

$$J(W) = L(W) \sum_{j=i}^{T} R_j$$

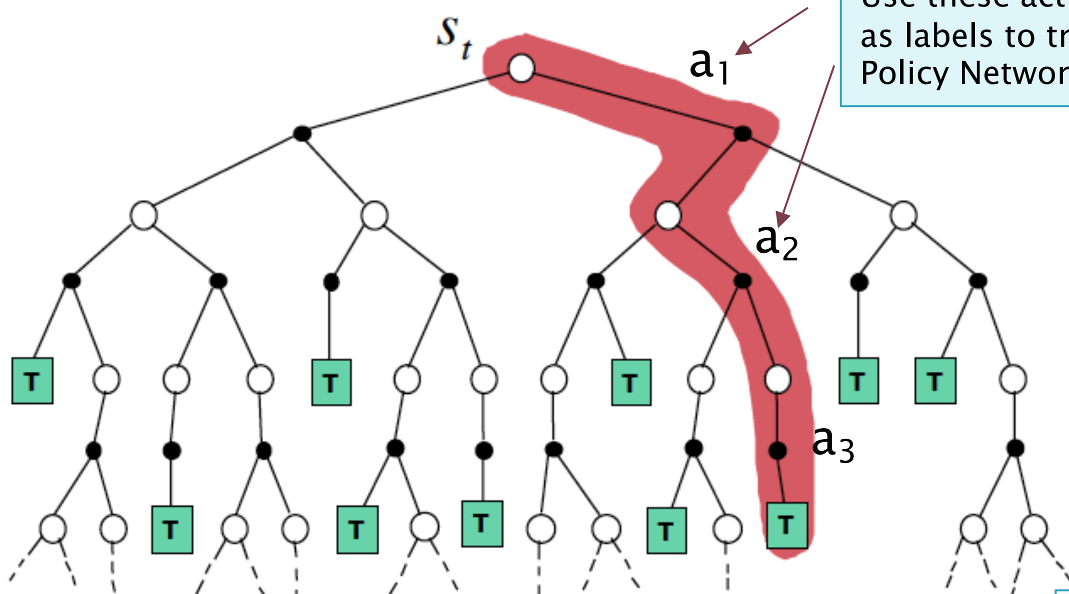
$$\boxed{\frac{\partial J(W)}{\partial w} = \sum_{i=1}^{T} \frac{\partial \log \pi_i}{\partial w} G_i}$$

Monte Carlo Policy Gradients: Reinforce

$$\text{w} \leftarrow w + \eta \frac{\partial J}{\partial w} = w + \eta G \frac{\partial L}{\partial w}$$

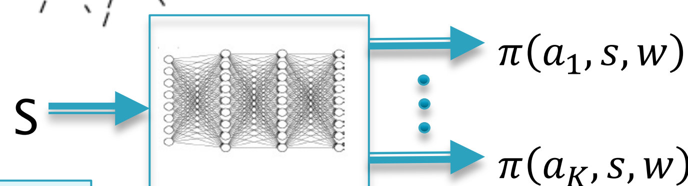
Use these actions as labels to train Policy Network

Episode 1: Actions are chosen according to Policy $\pi 1$



Episode 2: Actions are chosen according to Improved Policy $\pi 2$

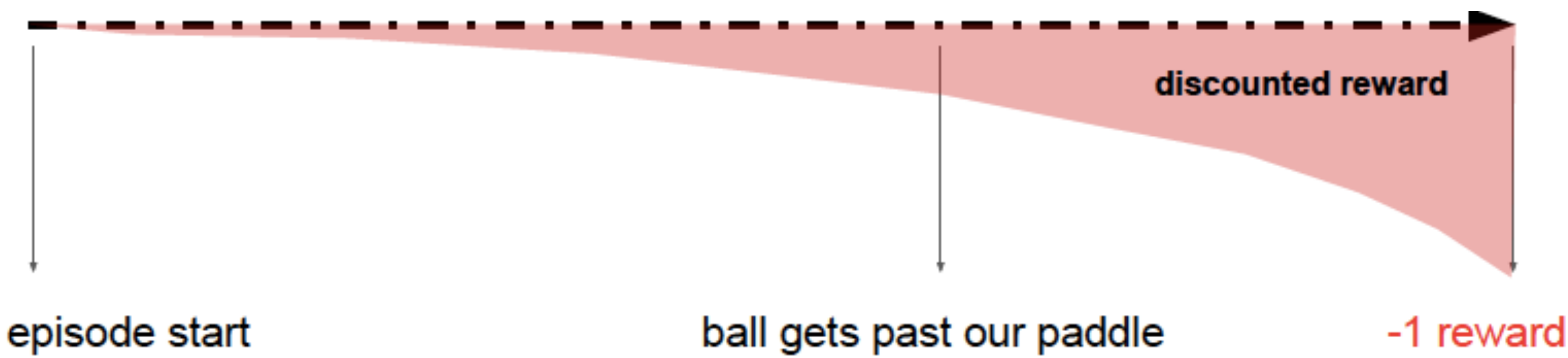
$G_i = \sum_{j=i}^{T} R_j$: The total reward from step i onwards



If G_i for action a_i is positive, then increase the probability of taking that action and vice versa

Discounting

$$G_e = \sum_{j=1}^{T} \beta^j R^j$$



Give more weight to Actions that occur near Reward

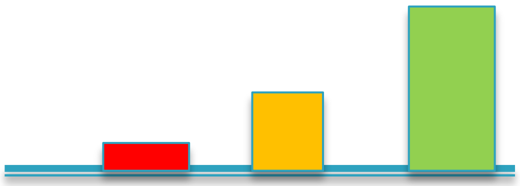
Baselines

Instead of $\rightarrow$ $$\frac{\partial J(W)}{\partial w} = \mathrm{E}\left(\sum_{i=1}^{T} \frac{\partial L_i}{\partial w} G_i\right)$$

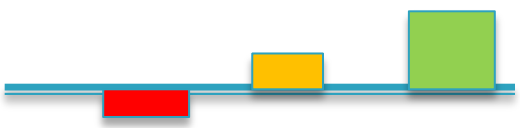
Do this $\rightarrow$ $$\frac{\partial J(W)}{\partial w} = \mathrm{E}\left(\sum_{i=1}^{T} \frac{\partial L_i}{\partial w} (G_i - b)\right)$$

We are only interested in how good the reward is compared to its average value, not its absolute value.

$$b = \frac{1}{M} \sum_{i=1}^{T\mathcal{E}} G_i$$



Is Baselining allowed?



Proof

$$\frac{\partial J(W)}{\partial w} = E\left[\frac{\partial logP^{\pi}(\tau_e, w)}{\partial w}(\mathrm{R}(\tau_e) - b)\right]$$

$$E\left[\frac{\partial logP^{\pi}(\tau_e, w)}{\partial w} b\right] = \int P^{\pi}(\tau_e, w)\frac{\partial logP^{\pi}(\tau_e, w)}{\partial w} b$$

$$= \int \frac{\partial P^{\pi}(\tau_e, w)}{\partial w} \mathrm{b} = \mathrm{b}\frac{\partial}{\partial w}\int P^{\pi}(\tau_e, w) = b\ \frac{\partial(1)}{\partial w} = 0$$

i.e. $\frac{\partial J(W)}{\partial w} = E\left[\frac{\partial logP^{\pi}(\tau_e, w)}{\partial w}(\mathrm{R}(\tau_e) - b)\right] = E\left[\frac{\partial logP^{\pi}(\tau_e, w)}{\partial w}\mathrm{R}(\tau_e)\right]$

Baseline Choices

$$\frac{\partial J(W)}{\partial w} = E\left[\sum_{i=1}^{M} \frac{\partial \log \pi\,(a_i|S_i)}{\partial w}(G_i - b(S_i))\right]$$

Baseline can be a function of S in general

1) $b = \frac{1}{M}\sum_{i=1}^{T\mathcal{E}} G_i$ (a constant)

2) **State-dependent expected return:**

$$b(s_t) = \mathbb{E}\left[r_t + r_{t+1} + r_{t+2} + \ldots + r_{H-1}\right] = V^{\pi}(s_t)$$

→ Increase logprob of action proportionally to how much its returns are better than the expected return under the current policy

$$\frac{\partial J(W)}{\partial w} = E\left[\sum_{i=1}^{M} \frac{\partial \log \pi\,(a_i|S_i)}{\partial w}(G_i - v_{\pi}(S_i))\right]$$

Actor-Critic Algorithms

What Problem Are We Solving?

1. Monte Carlo Policy Gradients work only for systems with terminating episodes.
   - What about systems in which the episodes do not terminate?
2. How can we further reduce the variance

$$\frac{\partial J(W)}{\partial w} = \frac{1}{\mathcal{E}} \sum_{e=1}^{\mathcal{E}} \sum_{i=1}^{T} \frac{\partial \log \pi_W(a_i|s_i)}{\partial w} \left(R_i(s_i, a_i) + R_{i+1}(s_{i+1}, a_{i+1}) + \cdots + R_T(s_T, a_T)\right)$$

Variance is high due to the sum of the rewards.

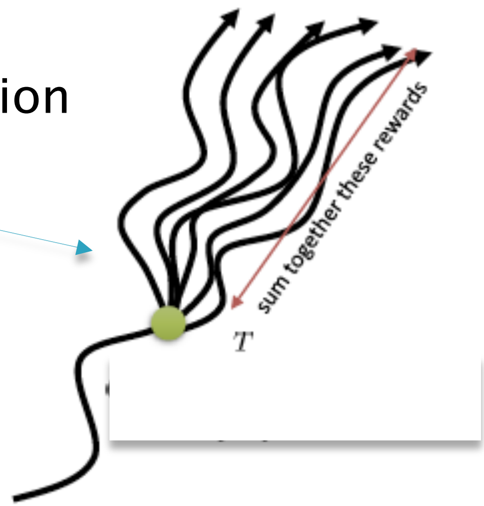
$$\frac{\partial J(W)}{\partial w} = \frac{1}{\mathcal{E}} \sum_{e=1}^{\mathcal{E}} \sum_{i=1}^{T} \frac{\partial \log \pi_W(a_i|s_i)}{\partial w} \; q_\pi(s_i, a_i)$$

Replace with Average Value!

(Re) Introducing the Q Function!

$$\frac{\partial J(W)}{\partial w} = \frac{1}{\mathcal{E}} \sum_{e=1}^{\mathcal{E}} \sum_{i=1}^{T} \frac{\partial \log \pi_W(a_i|s_i)}{\partial w} \; q_\pi(s_i, a_i)$$

Monte Carlo evaluation of $Q_\pi(s_i, a_i)$



What about the Baseline

$$\frac{\partial J(W)}{\partial w} = \frac{1}{\mathcal{E}} \sum_{e=1}^{\mathcal{E}} \sum_{i=1}^{T} \frac{\partial \log \pi_W(a_i|s_i)}{\partial w} \ [q_\pi(s_i, a_i) - v_\pi(s_i)]$$

Recall Bellman Expectation Equation:

$$v_\pi(s_i) = \sum_{j=1}^{K} \pi(a_j|s_i)\, q_\pi(s_i, a_j)$$

Answers the question:
If in state s_i I take action a_i, then how much better is this reward compared to the average reward over all actions?

The Advantage Function

Define the Advantage Function as

$$A_\pi(s_i, a_i) = q_\pi(s_i, a_i) - v_\pi(s_i)$$

$$\frac{\partial J(W)}{\partial w} = \frac{1}{\mathcal{E}} \sum_{e=1}^{\mathcal{E}} \sum_{i=1}^{T} \frac{\partial \log \pi_W(a_i|s_i)}{\partial w} A_\pi(s_i, a_i)$$

The better this estimate, the lower the variance

Estimating the Advantage Function

Note that

$A_\pi(s,a) = q_\pi(s,a) - v_\pi(s)$

From this equation it would seem that we need to estimate both q_π and v_π to estimate A_π. However …

Since

$q_\pi(s,a) = R(s,a) + \sum_{s'} P(s'|s,a) v_\pi(s')$

Approximately (along the sample path s,a,r,s')

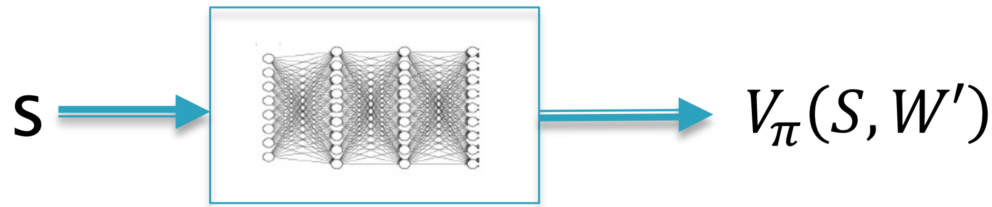
$q_\pi(s,a) \approx R(s,a) + v_\pi(s')$

So that

$A_\pi(s,a) \approx R(s,a) + v_\pi(s') - v_\pi(s)$

The Advantage Function can be estimated through just the Value Function

Estimating the Value Function with Neural Networks



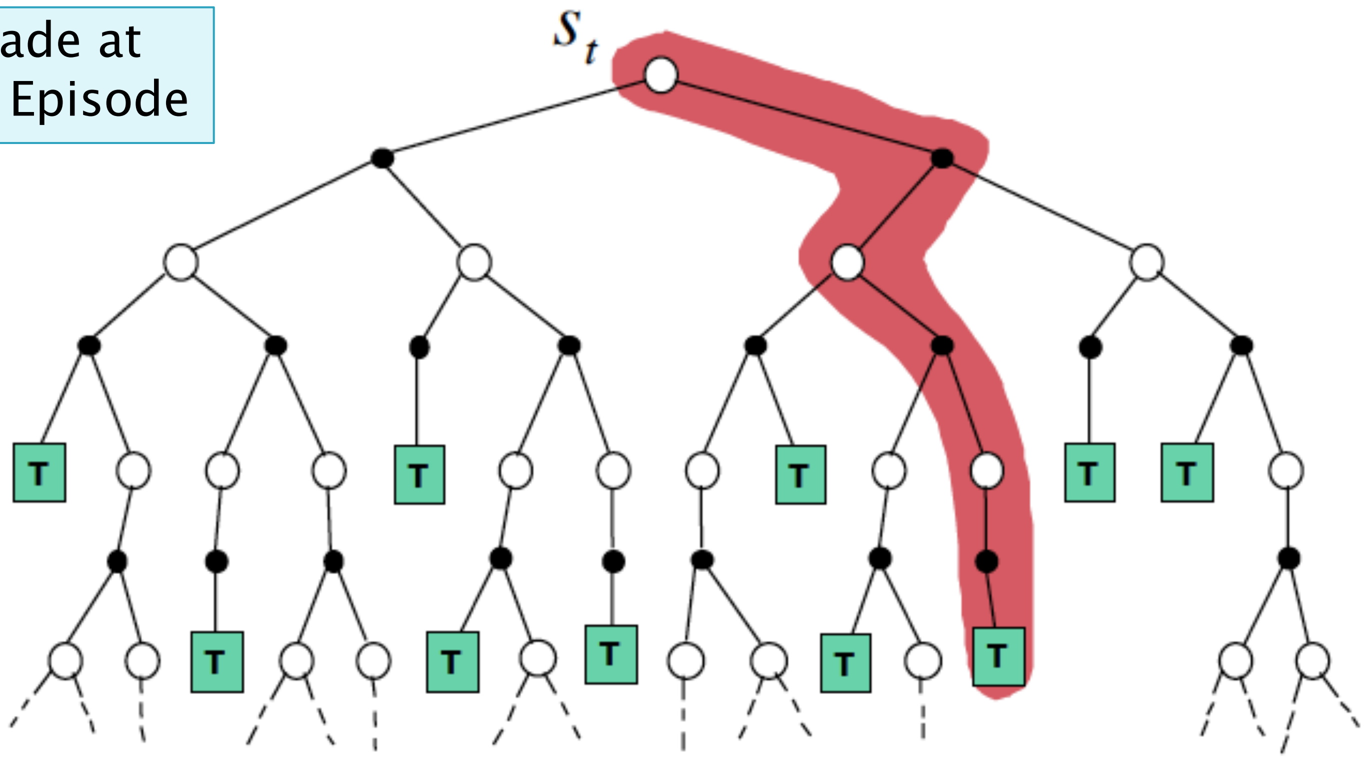
Two Techniques to train the Neural Network model:

- Monte Carlo
- Temporal Difference

Value Function Estimation using Monte Carlo

$$w_j \leftarrow w_j - \eta x_j [V(S_i, W) - G(S_i)]$$

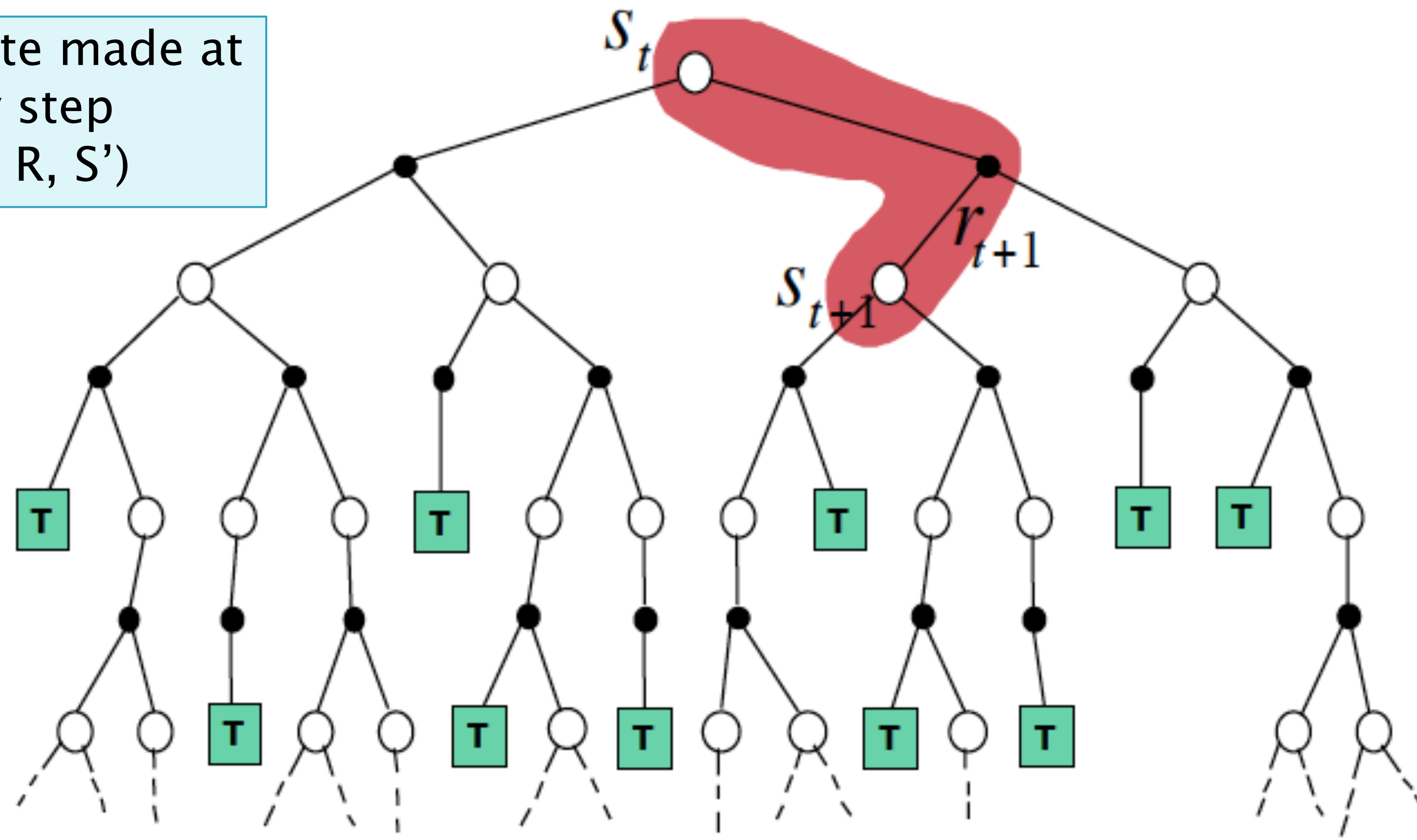
Update made at end of an Episode



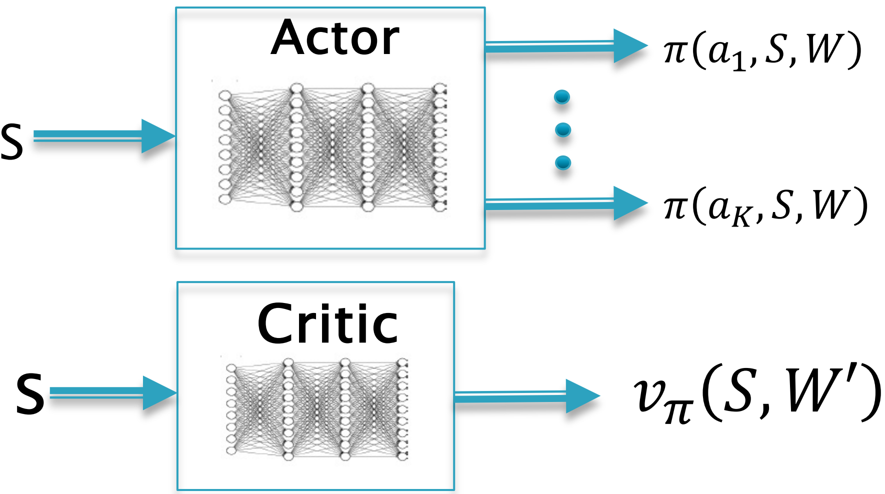
Value Function Estimation using Temporal Difference

$$w_j \leftarrow w_j - \eta x_j [V(S,W) - (R + \gamma V(S',W))]$$

Update made at
every step
(S, A, R, S')



Overall System: Online Actor Critic



$$\boxed{\boldsymbol{target = R + V_\pi(S', W')}}$$

$$\boxed{\boldsymbol{A_\pi(S,a) = R + V_\pi(S', W') - V_\pi(S, W')}}$$

$$\boxed{\frac{\partial J(W)}{\partial W} = \frac{\partial \log \pi_w(S,a)}{\partial W} A_\pi(S, W')}$$

In state S
Run Actor to Generate next Action A

↓

Take a step in the environment
to generate reward R and next state S'

↓

Use (S,A,R,S') to update the Critic's
Neural Network parameters.

↓

Use $V_\pi(S, W')$ to update the Actor's
neural Network Parameters
Set S' ← S

Online Actor–Critic Algorithm

online actor-critic algorithm:

1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a}|\mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
2. update $\hat{V}_\phi^\pi$ using target $r + \gamma \hat{V}_\phi^\pi(\mathbf{s}')$
3. evaluate $\hat{A}^\pi(\mathbf{s}, \mathbf{a}) = r(\mathbf{s}, \mathbf{a}) + \gamma \hat{V}_\phi^\pi(\mathbf{s}') - \hat{V}_\phi^\pi(\mathbf{s})$
4. $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(\mathbf{a}|\mathbf{s}) \hat{A}^\pi(\mathbf{s}, \mathbf{a})$
5. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Continuous Action Spaces: Deterministic Policy Gradients

Published as a conference paper at ICLR 2016

CONTINUOUS CONTROL WITH DEEP REINFORCEMENT LEARNING

Timothy P. Lillicrap*, Jonathan J. Hunt*, Alexander Pritzel, Nicolas Heess,
Tom Erez, Yuval Tassa, David Silver & Daan Wierstra
Google Deepmind
London, UK
`{countzero, jjhunt, apritzel, heess,`
`etom, tassa, davidsilver, wierstra} @ google.com`

ABSTRACT

We adapt the ideas underlying the success of Deep Q-Learning to the continuous action domain. We present an actor-critic, model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces. Using the same learning algorithm, network architecture and hyper-parameters, our algorithm robustly solves more than 20 simulated physics tasks, including classic problems such as cartpole swing-up, dexterous manipulation, legged locomotion and car driving. Our algorithm is able to find policies whose performance is competitive with those found by a planning algorithm with full access to the dynamics of the domain and its derivatives. We further demonstrate that for many of the tasks the algorithm can learn policies "end-to-end": directly from raw pixel inputs.

1 INTRODUCTION

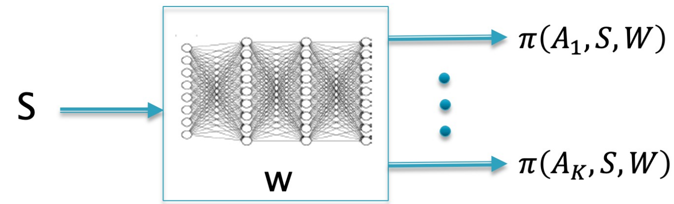
One of the primary goals of the field of artificial intelligence is to solve complex tasks from unprocessed, high-dimensional, sensory input. Recently, significant progress has been made by combining advances in deep learning for sensory processing (Krizhevsky et al., 2012) with reinforcement learning, resulting in the "Deep Q Network" (DQN) algorithm (Mnih et al., 2015) that is capable of human level performance on many Atari video games using unprocessed pixels for input. To do so, deep neural network function approximators were used to estimate the action-value function.

However, while DQN solves problems with high-dimensional observation spaces, it can only handle discrete and low-dimensional action spaces. Many tasks of interest, most notably physical control tasks, have continuous (real valued) and high dimensional action spaces. DQN cannot be straightforwardly applied to continuous domains since it relies on a finding the action that maximizes the action-value function, which in the continuous valued case requires an iterative optimization process at every step.

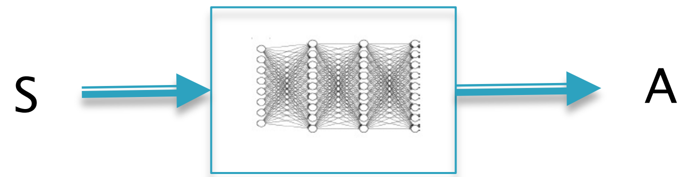
An obvious approach to adapting deep reinforcement learning methods such as DQN to continuous domains is to to simply discretize the action space. However, this has many limitations, most no-

Continuous Action Spaces

Instead of generating the Optimal Policy

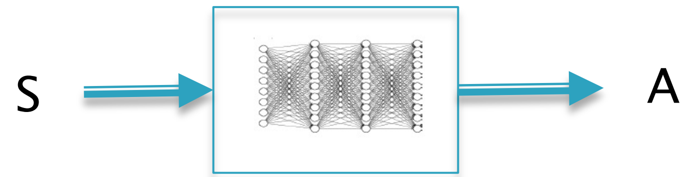


Generate the Optimal Action directly!



Continuous Action Spaces

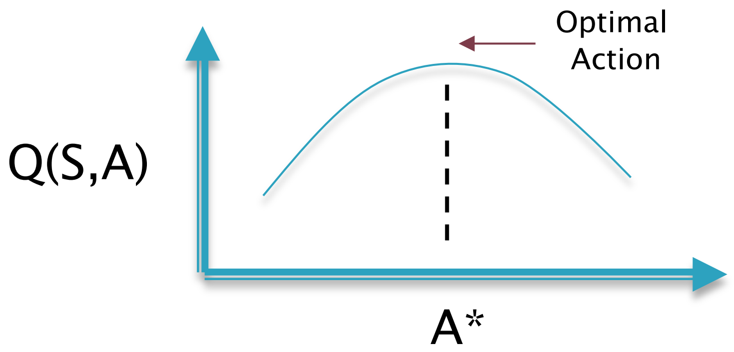
Generate the Optimal Action directly!



How to train the Policy Network for continuous Actions?

Hint: The Optimal Action is the one that maximizes the Q Function

Idea: Use the Q Function as the Reward to train the Policy Network



DPG Algorithm

Critic

S

A

Network 1
W_1

$Q(S,A,W_1)$

Actor

S'

Network 2
W_2

$A(S',W_2)$

Choose the Action that maximizes $Q(S,A,W_1)$

Use $Q(S,A,W_1)$ as the Reward Function for Network 2

$$W_2 \leftarrow W_2 + \eta \frac{\partial Q(S,A,W_1)}{\partial W_2}$$

$$\frac{\partial Q(S,A,W_1)}{\partial W_2} = \frac{\partial Q(S,A,W_1)}{\partial A} \frac{\partial A(S',W_2)}{\partial W_2}$$

DPG Algorithm

1. Choose Action A using Network 2

2. Take a step in the env and obtain the (S,A,R,S’) values

1. take some action $\mathbf{a}_i$ and observe $(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i)$, add it to $\mathcal{B}$
2. sample mini-batch $\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\}$ from $\mathcal{B}$ uniformly
3. compute $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}(\mathbf{s}'_j, \mu_{\theta'}(\mathbf{s}'_j))$ using *target* nets $Q_{\phi'}$ and $\mu_{\theta'}$
4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)(Q_\phi(\mathbf{s}_j, \mathbf{a}_j) - y_j)$
5. $\theta \leftarrow \theta + \beta \sum_j \frac{d\mu}{d\theta}(\mathbf{s}_j)\frac{dQ_\phi}{d\mathbf{a}}(\mathbf{s}_j, \mathbf{a})$
6. update ϕ' and θ' (e.g., Polyak averaging)

3. Choose the next Action A’ using Network 2

4. Improve the Q Value by improving Network 1

5. Improve the Action by improving Network 2 (Policy Improvement Step)