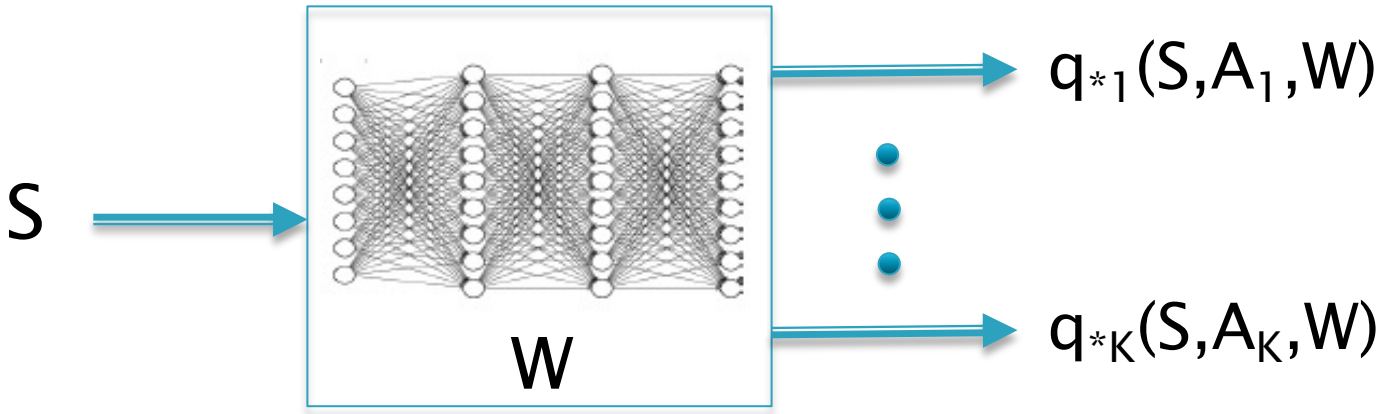
Policy Gradient Methods

Lecture 8

Subir Varma

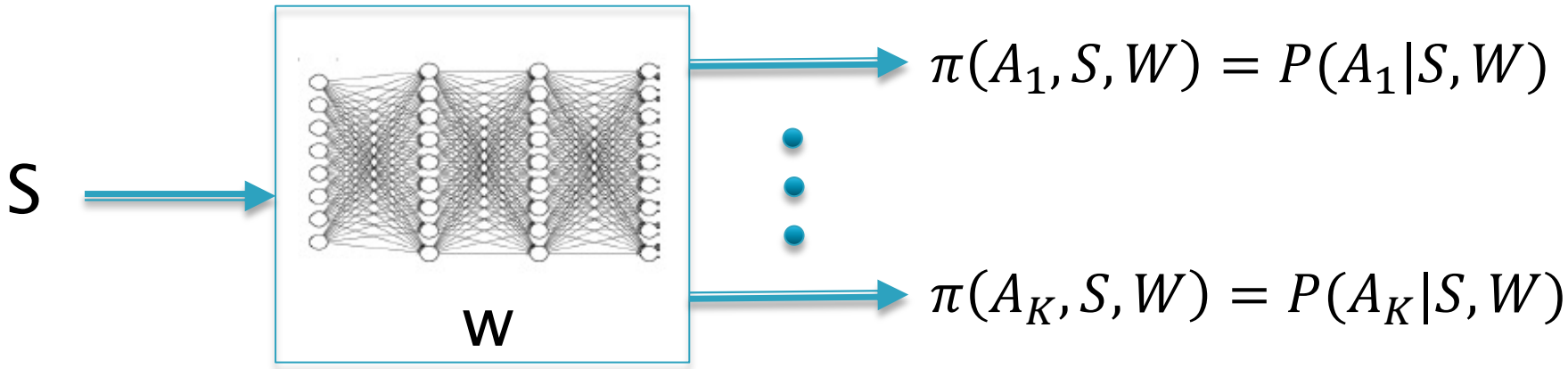
Last Lecture



Focus was on Value Function Approximations

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

This Lecture



Generate the Optimal Policy Directly without using Q Functions

Pros and Cons of Policy Based RL

Advantages:

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

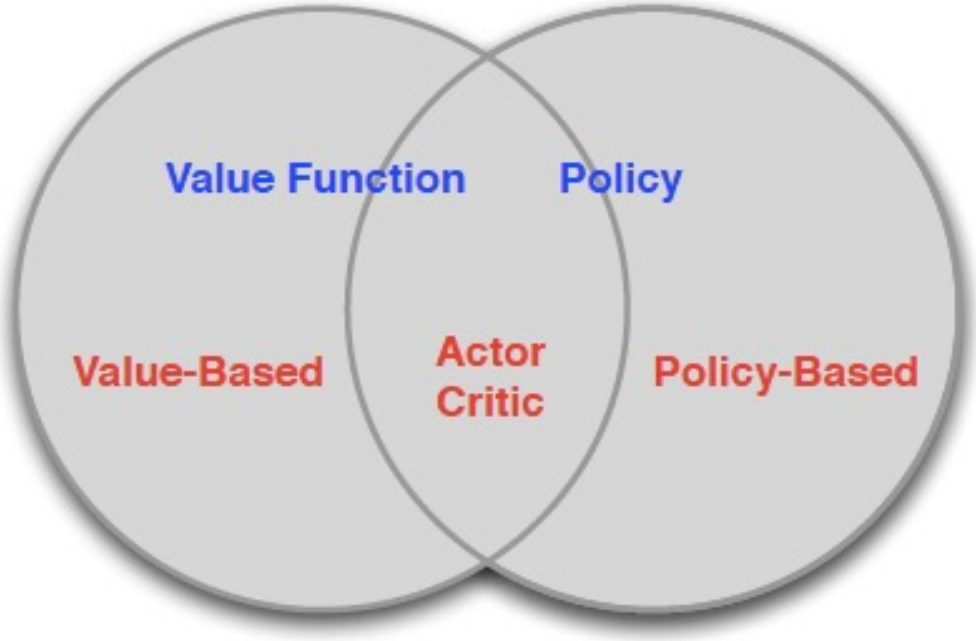
Disadvantages:

- 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

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

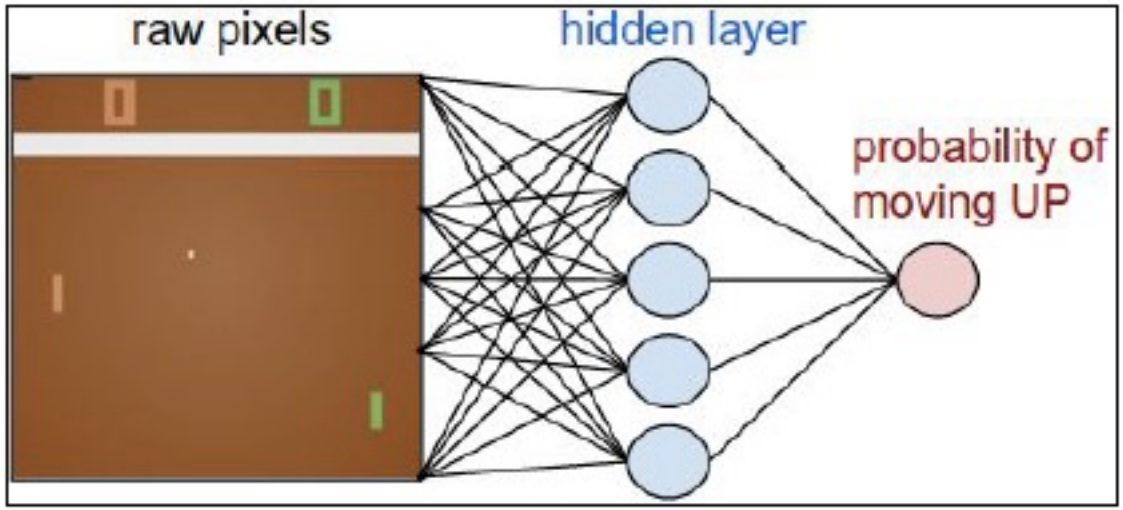
Reinforce

Playing Pong using Policy Gradients

Policy Network for Pong

height width

[80 x 80]
array of

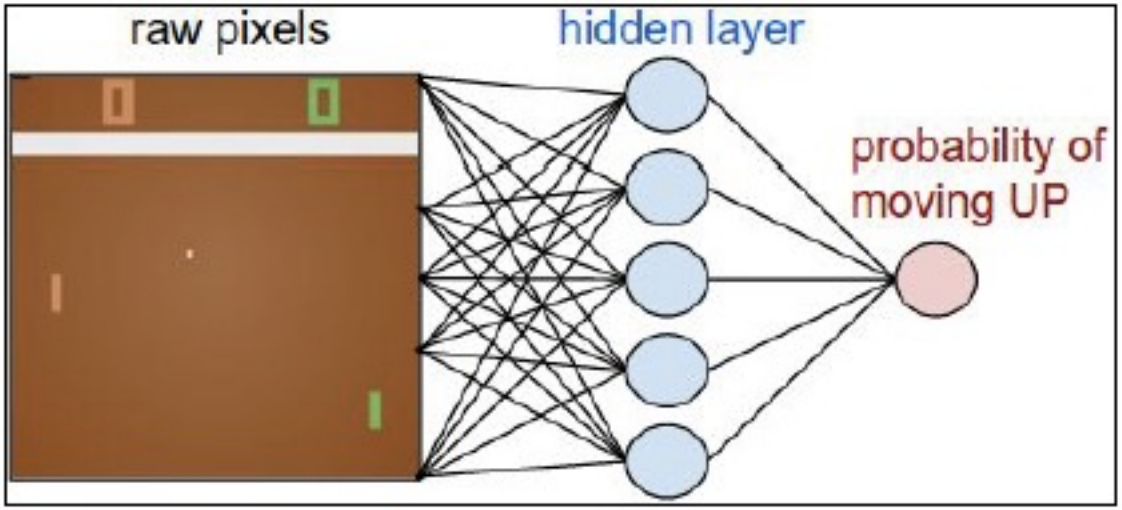



```
h = np.dot(W1, x) # compute hidden layer neuron activations
h[h<0] = 0 # ReLU nonlinearity: threshold at zero
logp = np.dot(W2, h) # compute log probability of going up
p = 1.0 / (1.0 + np.exp(-logp)) # sigmoid function (gives probability of going up)
```

Policy Network for Pong

height width

[80 x 80]
array



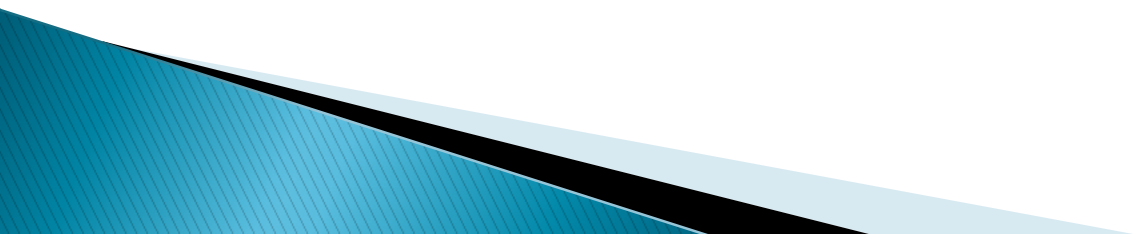
E.g. 200 nodes in the hidden network, so:

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

Layer 1 Layer 2

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

(x1,UP)
(x2,DOWN)
(x3,UP)
…



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)
…

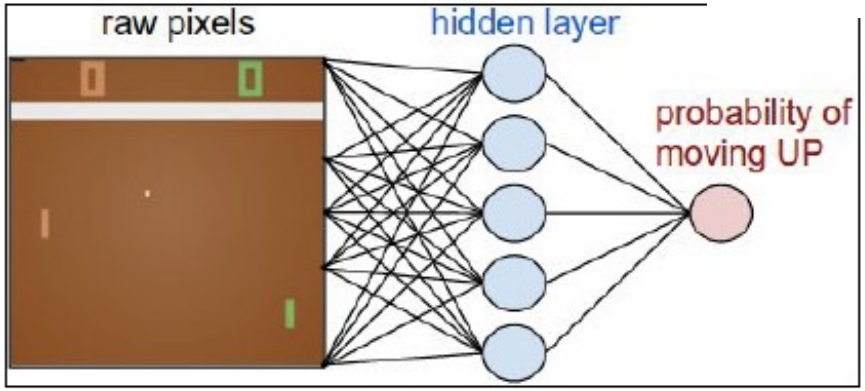
Maximize

Label

Network Output

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

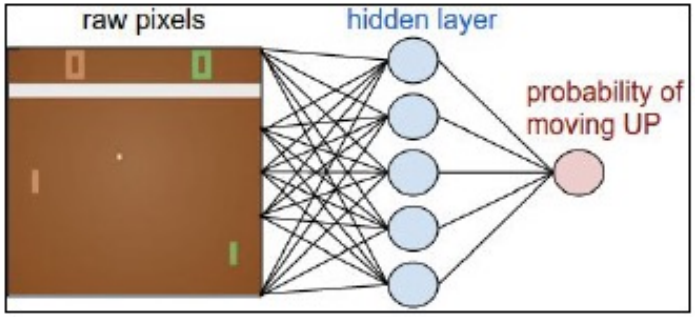
$$y(j) = \frac{1}{1+\exp(-\sum_{i=1}^{n} w_i x_i(j) - b)}$$



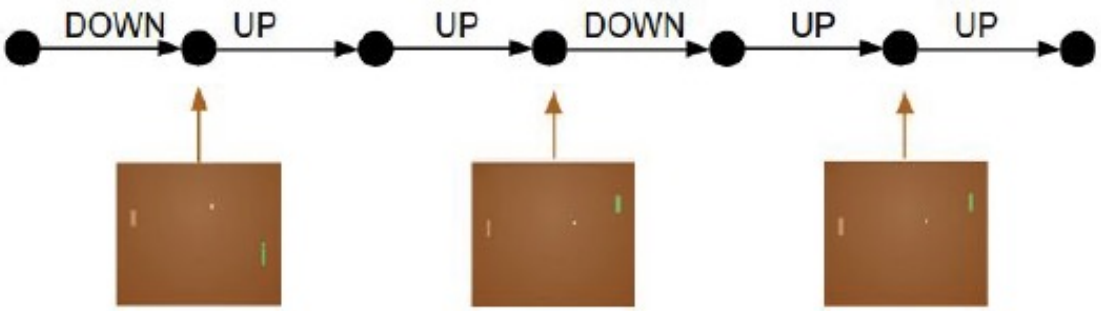
Except, we don't have labels…

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

Actions are sampled from Network Output



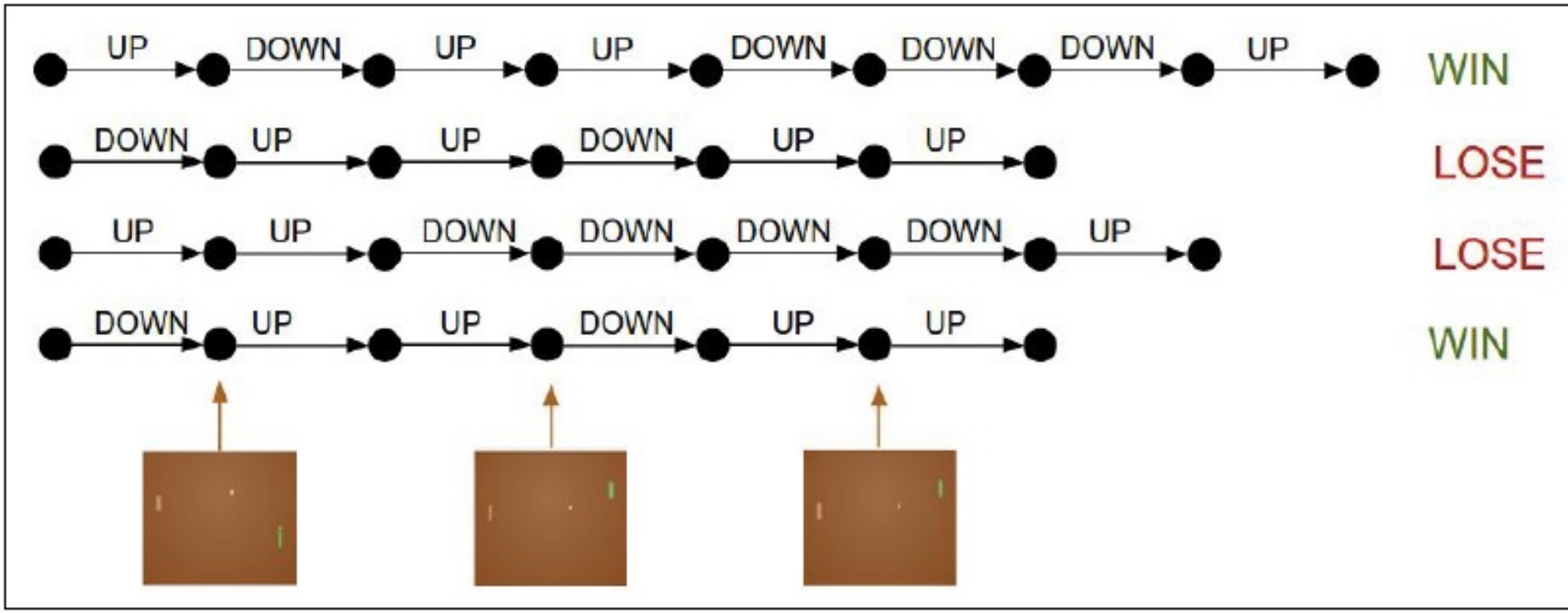
Rollout the policy and collect an episode



WIN

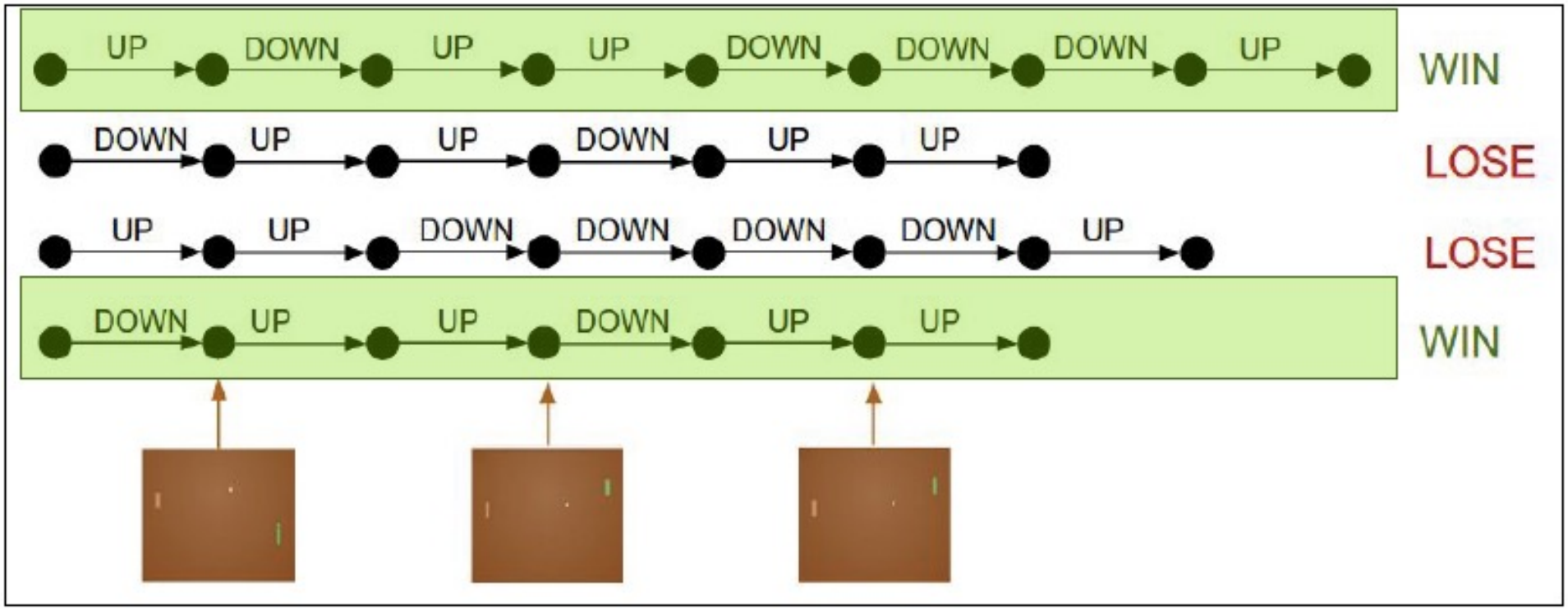
Collect many rollouts...

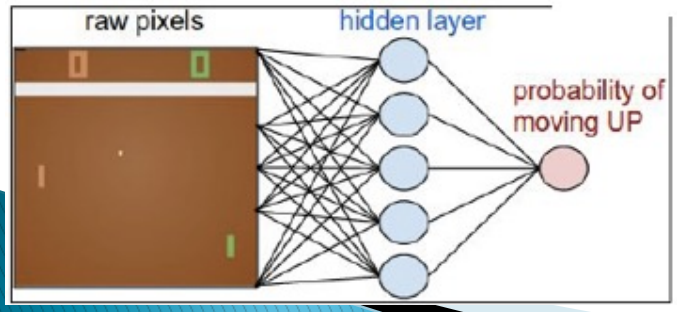
4 rollouts:



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

$$w_i \leftarrow w_i - \eta x_i(j)[y(j) - t(j)]$$





Pretend every action we took here was the correct label.

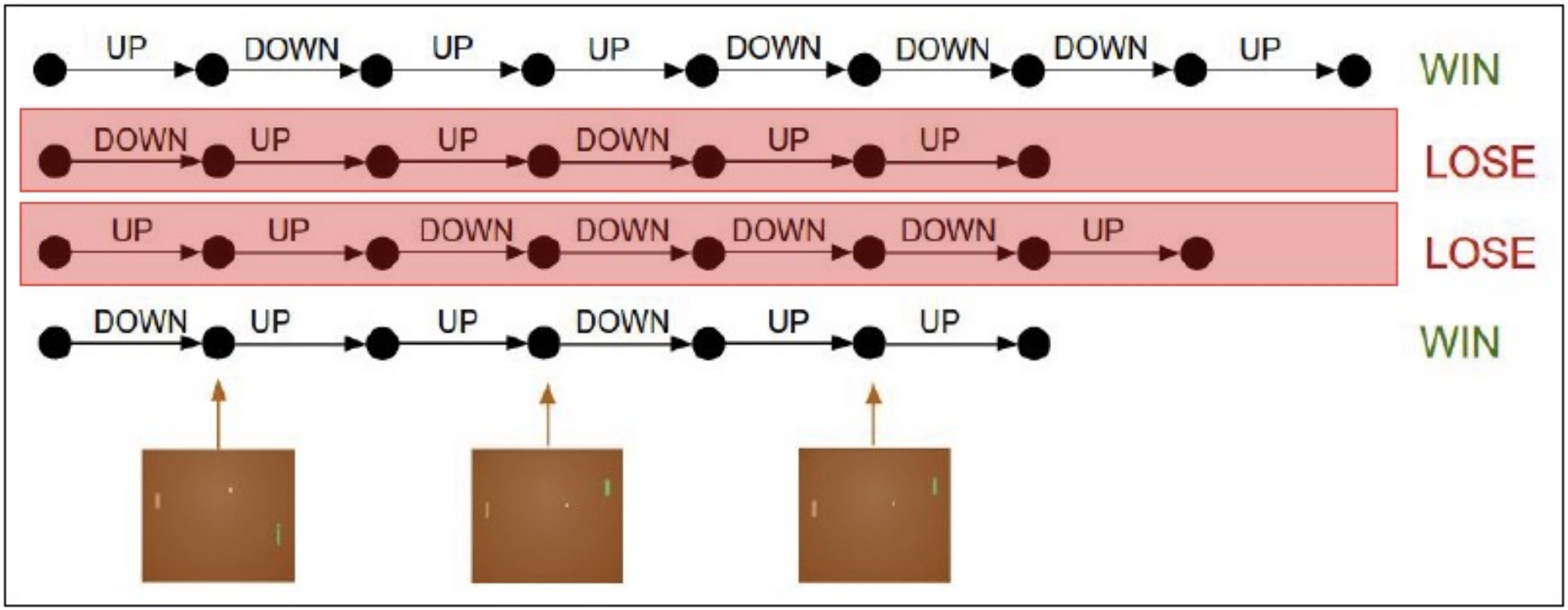
$$w_i \leftarrow \begin{cases} w_i + \eta G x_i(j)(1 - y(j)), & t = 1 \\ w_i - \eta G x_i(j) y(j), & t = 0 \end{cases}$$

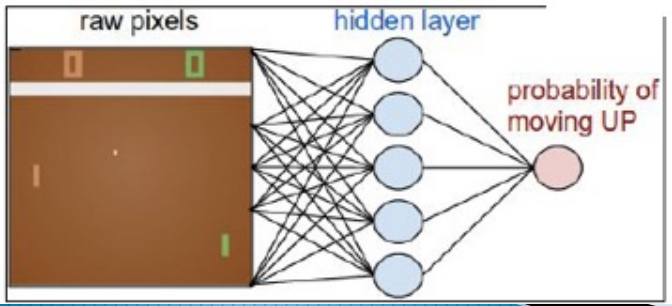
Increase the probability of the UP action

Increase the probability of the DOWN action

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

$$w_i \leftarrow w_i - \eta x_i(j)[y(j) - t(j)]$$





This can be accomplished by maximizing

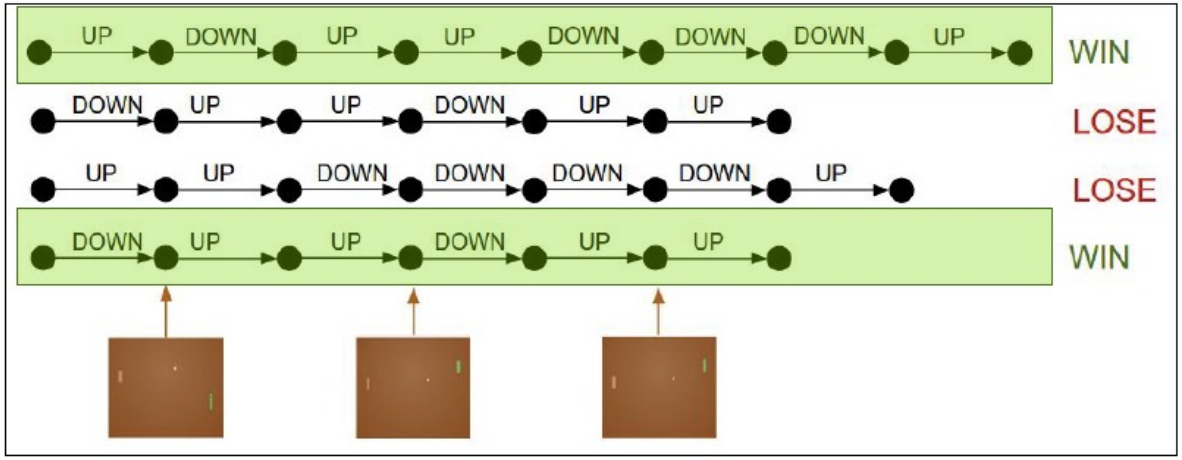
$$L(W) = -[t(j)\log y(j) + \big(1 - t(j)\big)\log(1 - y(j))]$$

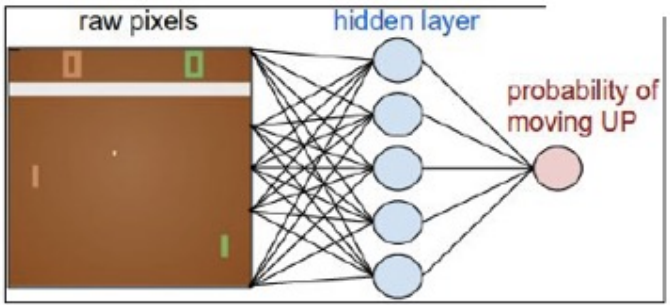
for LOSE episodes

$$w_i \leftarrow \begin{cases} w_i - \eta x_i(j)(1 - y(j)], t = 1 \\ w_i + \eta x_i(j)y(j), \quad t = 0 \end{cases}$$

Decrease the probability of the UP action

Decrease the probability of the DOWN action





Effectively, we are maximizing

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

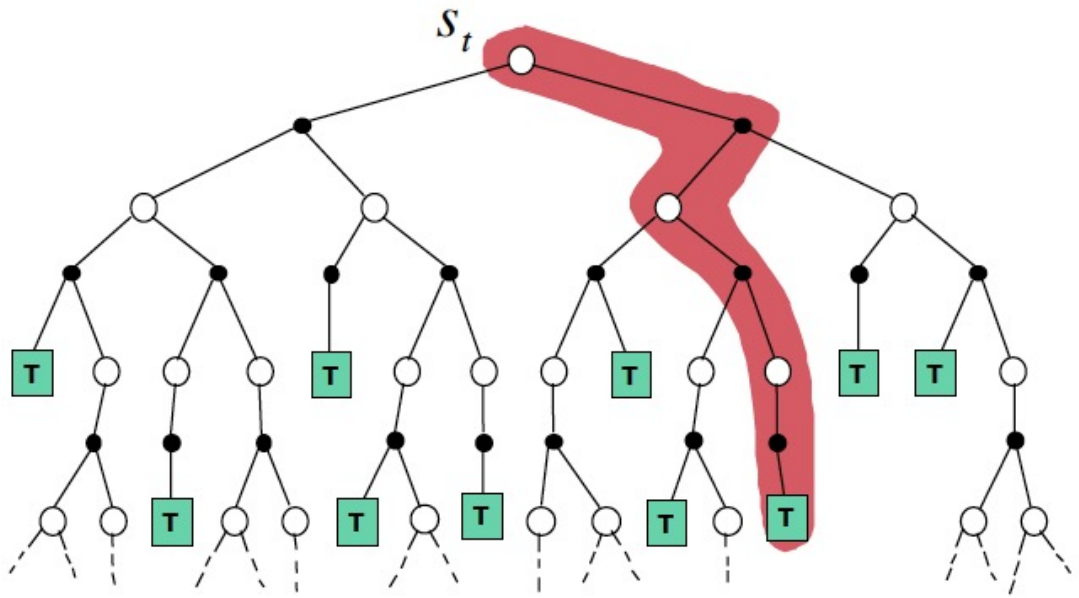
$$\frac{\partial J}{\partial w} = G\frac{\partial L}{\partial w}$$

L = Cross Entropy Function

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

Monte Carlo Policy Gradients: Reinforce

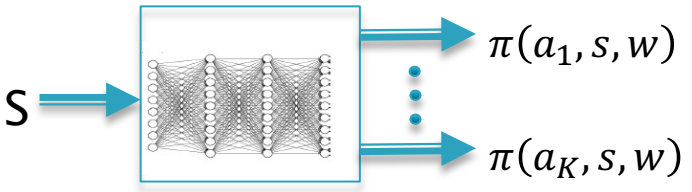
$$w \leftarrow w + \eta \frac{\partial J}{\partial w} = w + \eta G \frac{\partial L}{\partial w}$$



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

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

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



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

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

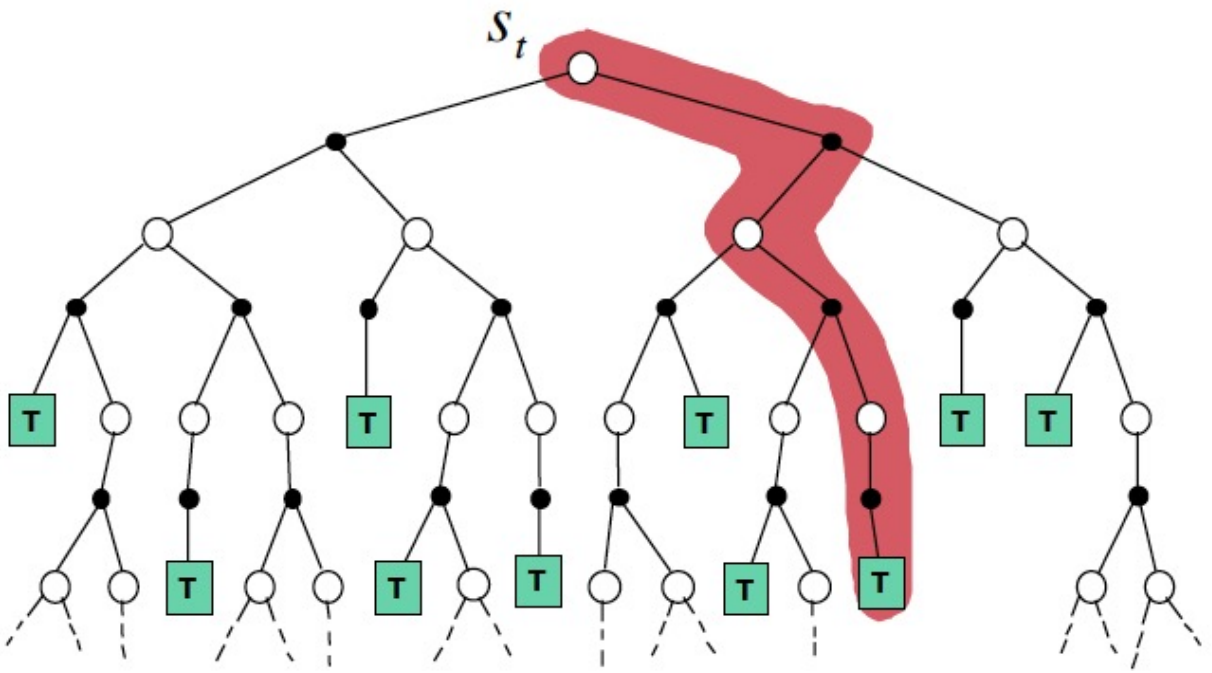
Total Reward for the Episode

Use action a_t as the Label at time t

Derivation of Likelihood Ratio Formula

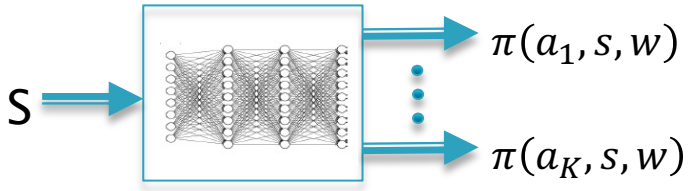
Monte Carlo: Policy Gradients

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



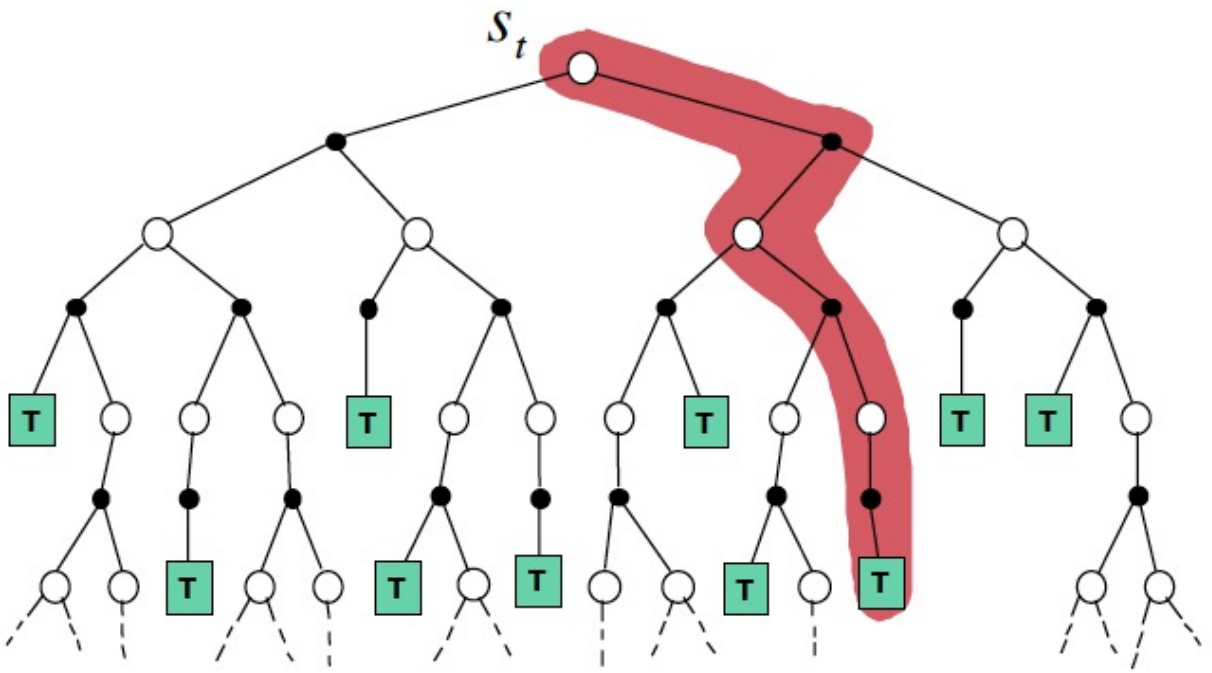
Episode 1: Actions are chosen according to Policy $\pi 1$

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



Monte Carlo: Policy Gradients

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

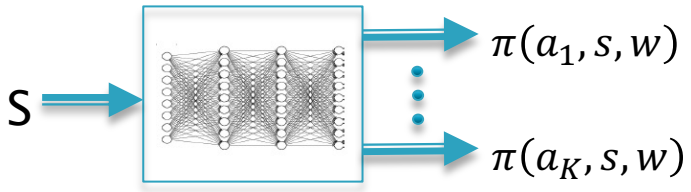


Episode 1: Actions are chosen according to Policy $\pi 1$

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

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

Total reward for episode τ_e

This can be split up into episodes:

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

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:

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

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

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

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

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

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

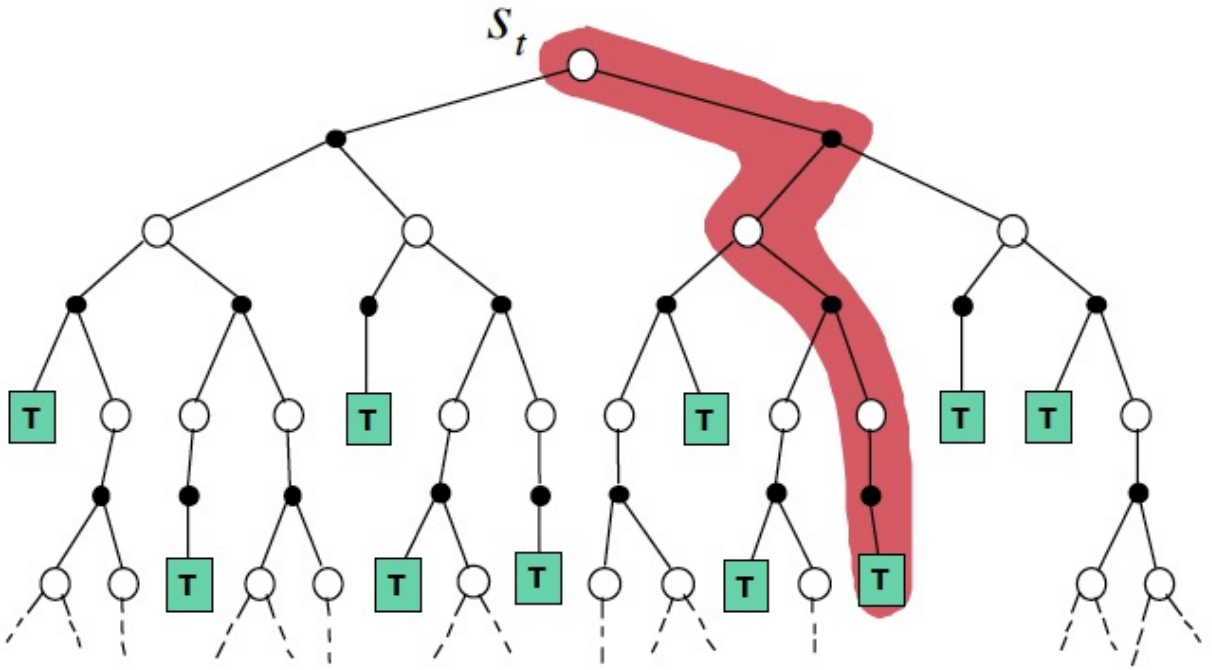
$$= E\left[\frac{\partial log P^{\pi}(\tau_e, w)}{\partial w} R(\tau_e)\right]$$

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

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

$L(s_i, a_i)$

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

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

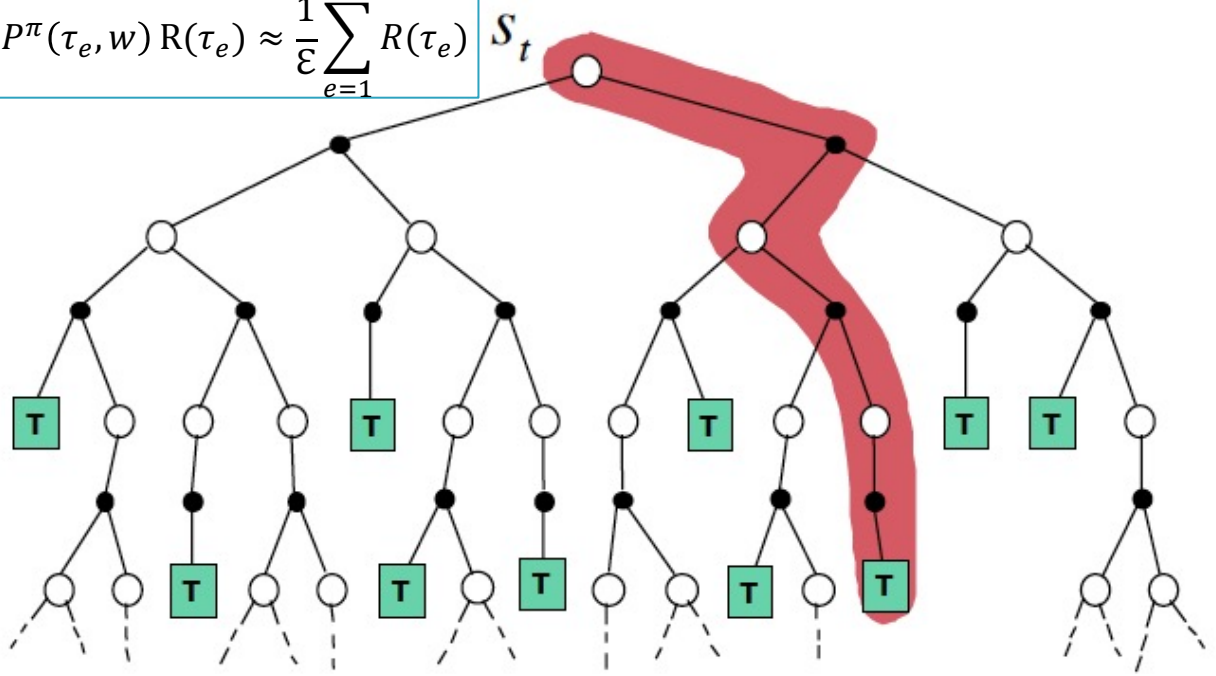
This gradient can potentially be evaluated by sampling without knowledge of the model

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

Monte Carlo Policy Gradients: Reinforce

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

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



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

$$\frac{\partial J(W)}{\partial w} = \frac{\partial L_i}{\partial w}\, G_e$$

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**

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

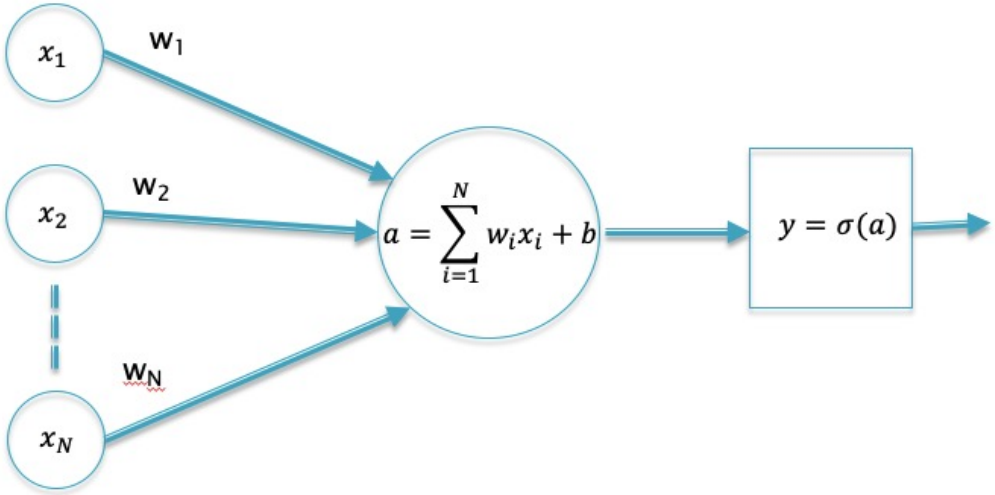
  endfor

endfor

return θ

How to compute gradients ?

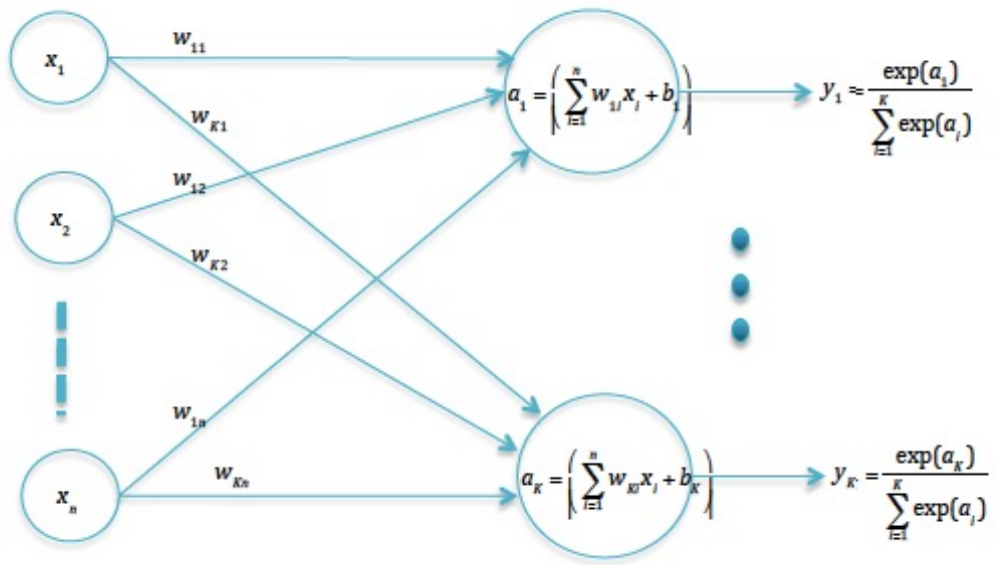
The case of K = 2



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

Where A is the Action that the Agent takes in State S
And G(S) is the Total Reward to Go from state S

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 to Go from state S

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

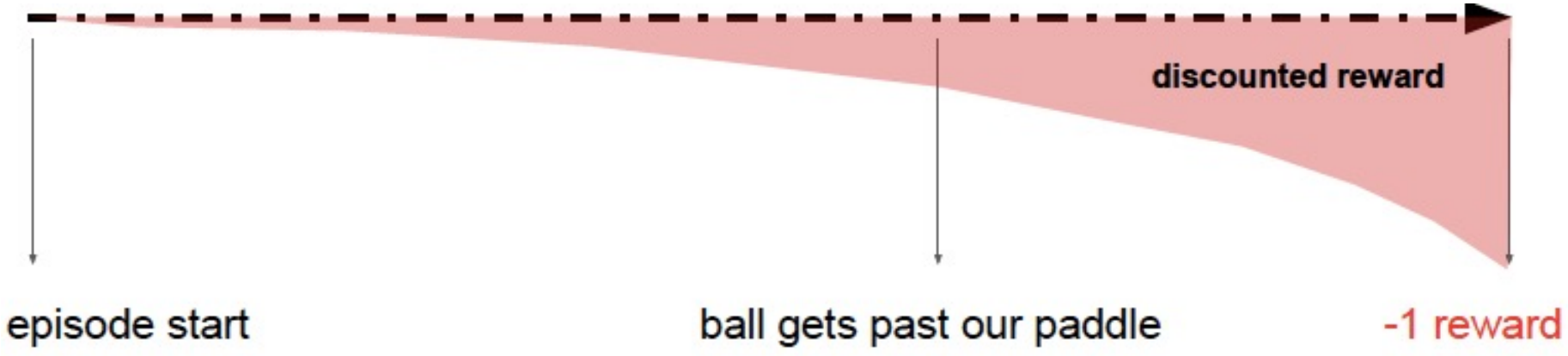
Variance Reduction

Techniques for Variance Reduction

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

Discounting

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



Give more weight to Actions that occur near Reward

Exploiting Causality: Reward To-Go

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

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

$$\frac{\partial J(W)}{\partial w} = \frac{1}{\mathcal{E}} \sum_{e=1}^{\mathcal{E}} \sum_{i=1}^{T} \frac{\partial \log \pi_i}{\partial w} \sum_{j=i}^{T} R_j$$

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

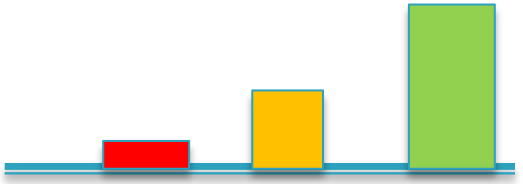
Baselines

Instead of → $$\frac{\partial J(W)}{\partial w} = \frac{1}{\mathcal{E}} \sum_{e=1}^{\mathcal{E}} \sum_{i=1}^{T} \frac{\partial L_i}{\partial w} G_i$$

Do this → $$\frac{\partial J(W)}{\partial w} = \frac{1}{\mathcal{E}} \sum_{e=1}^{\mathcal{E}} \sum_{i=1}^{T} \frac{\partial L_i}{\partial w} (G_i - b)$$

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 this allowed?

Proof

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

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

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

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

Baseline Choices

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

Baseline can be a function of S in general

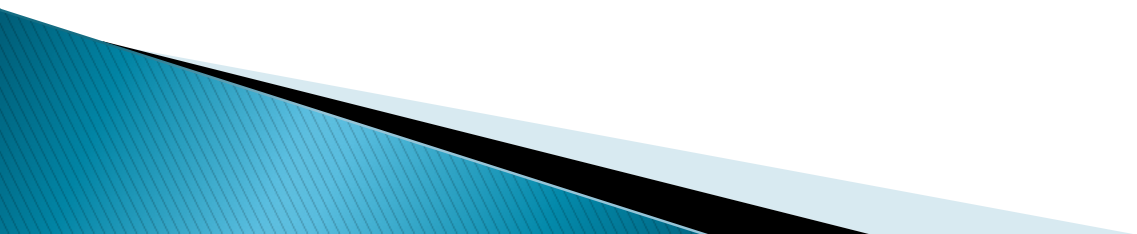
1) $b = \frac{1}{M}\sum_{i=1}^{T\mathcal{E}} G_i$ (a constant)

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

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.

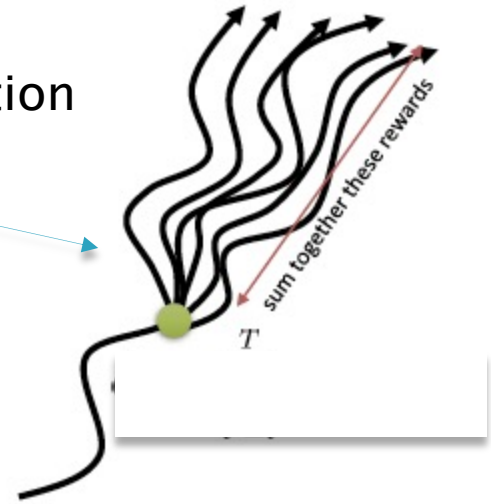
$$\frac{\partial J(W)}{\partial w} = \frac{1}{\mathcal{E}} \sum_{e=1}^{\mathcal{E}} \sum_{i=1}^{T} \frac{\partial \log \pi_W(a_i|s_i)}{\partial w} \, q_\pi(s_i, a_i)$$

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

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

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

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

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

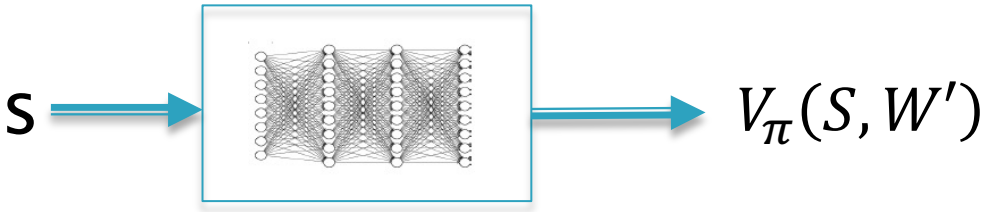
$q_\pi(s,a) \approx R(s,a) + v_\pi(s')$

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



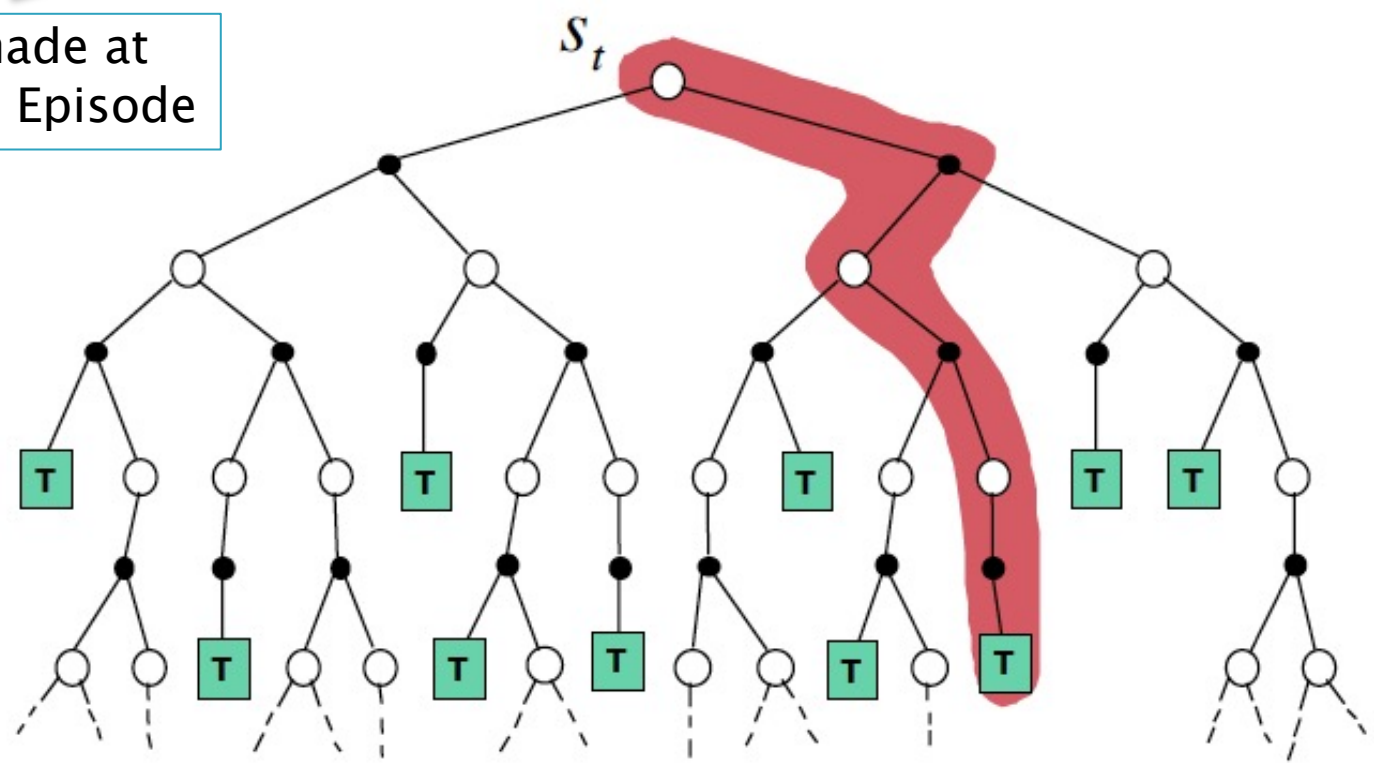
Two Techniques to train the Neural Network model:

- Monte Carlo
- Temporal Difference

Value Function Estimation using Monte Carlo

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

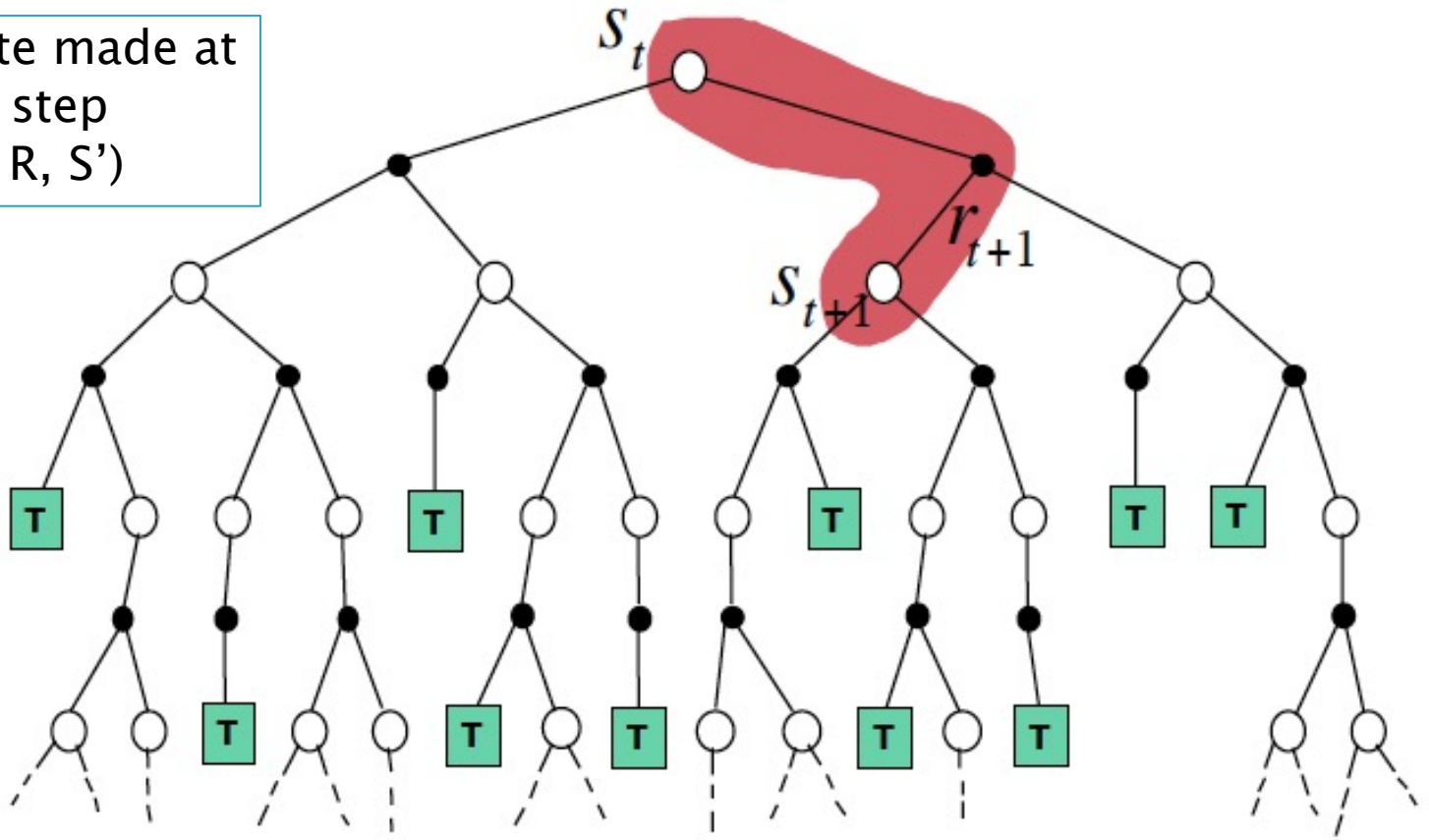
Update made at end of an Episode



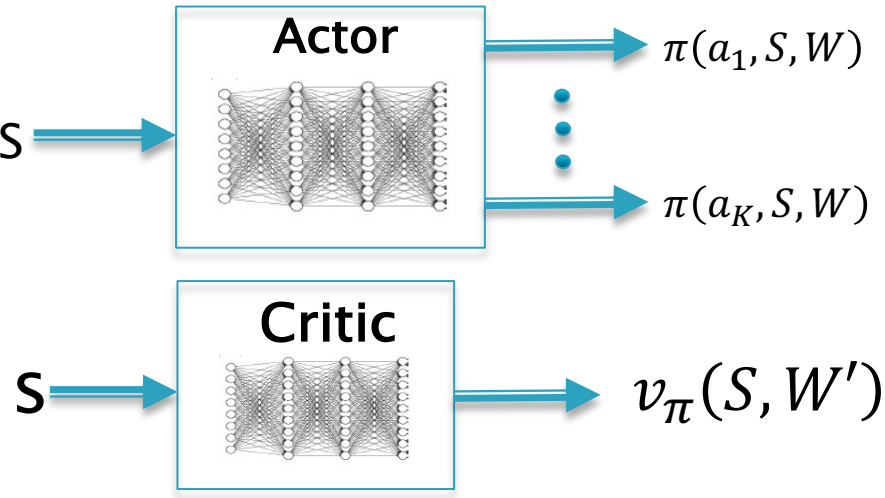
Value Function Estimation using Temporal Difference

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

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



Overall System: Online Actor Critic



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

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

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

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

Adding Discount

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