Policy Gradient Methods in Reinforcement Learning

Kabir Ahuja

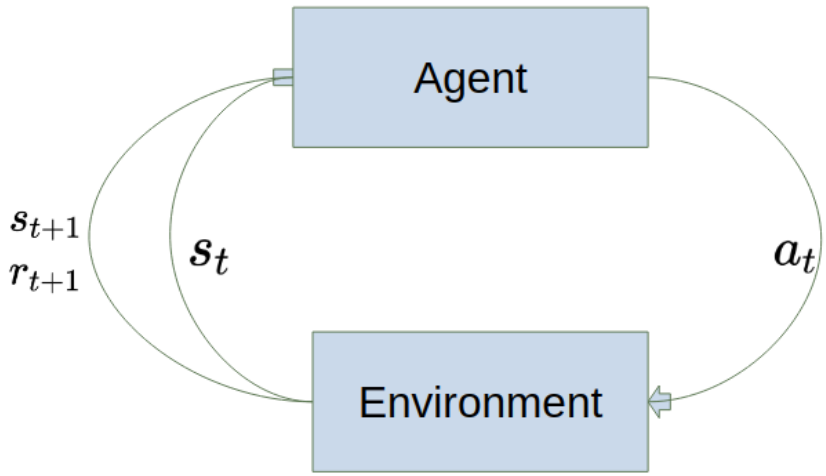
July 14, 2019

Outline

Outline

The Reinforcement Learning Problem

- Deals with agents learning to take decisions in an environment that provides numerical rewards.
- The goal of the agent is to maximize the cumulative reward obtained by interacting with environment



Markov Decision Process (MDP)

- Mathematical Formulation of RL problem.
- Based on the **Markov property** which means that the current state completely captures the state of the world. More concretely: $P(s_{t+1}|s_t) = P(s_{t+1}|s_t, s_{t-1}, ..., s_0)$
- Defined by $(S, A, \mathbb{R}, \mathbb{P}, \gamma)$
  - S : set of possible states
  - A : set of possible actions
  - $\mathbb{R}$: distribution of reward given (state, action) pair.
  - $\mathbb{P}$: transition probability which gives the distribution of next state given a (state, action) pair.
  - γ : discount factor

Policy

- A policy describes the behavior of an agent.
- It is a function that maps state to action.
- Deterministic Policy: $a_t = \pi(s_t; \theta)$
- Stochastic Policy: $a_t \sim \pi(a_t|s_t; \theta)$
- The goal of an RL agent is to learn a policy which maximizes the agent's cumulative discounted reward (also called return) which is defined as: $\sum_{t>0} \gamma^{t-1} r_t$

Value Functions

State Value Function

- A value function describes how good or bad a particular **state** is.
- It is defined by the expected cumulative reward starting from state s_t and following a policy π.
  $V_\pi(s_t) =$
  $\mathbb{E}_\pi[r_t + \gamma r_{t+1} + ... + \gamma^{T-t} r_T | s_t]$
- The optimal value function is defined as the maximum value function for all the policies.
  $V(s_t) = max_\pi(V_\pi(s_t))$

Action Value Function

- Describes how good a **state, action** pair is.
- It is defined as the expected cumulative reward obtained on starting from a state s_t, taking action a_t and then following a policy π
  $Q_\pi[s_t, a_t] =$
  $\mathbb{E}_\pi[r_t + \gamma r_{t+1} + ... + \gamma^{T-t} r_T | s_t, a_t]$
- The optimal Q value function can be defined as the maximum of q value functions for all the policies.
  $Q(s_t, a_t) = max_\pi(Q_\pi(s_t, a_t))$

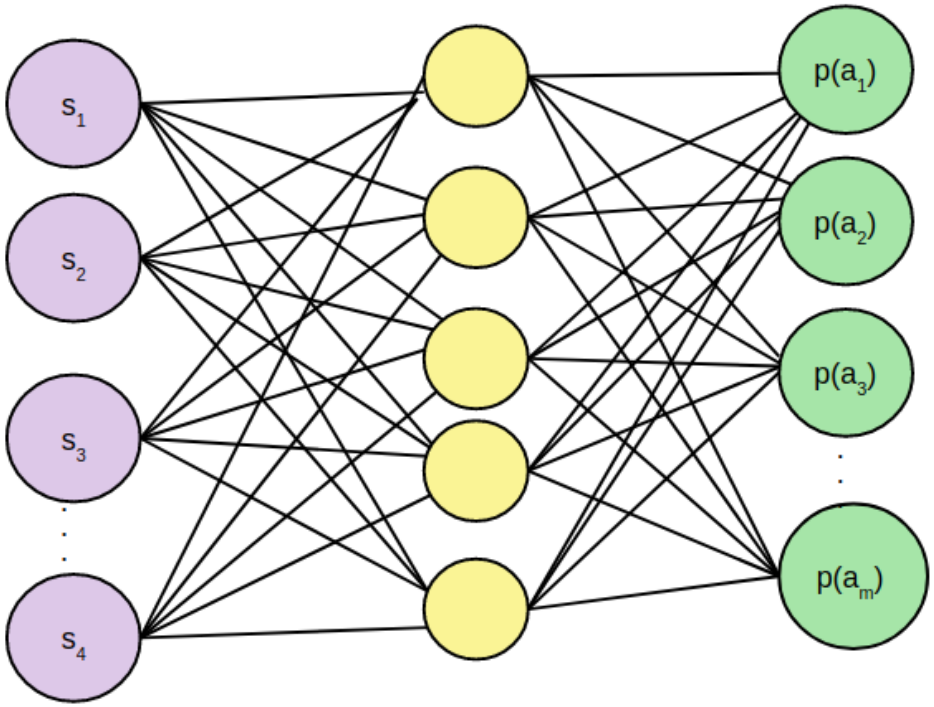
Solving the RL Problem

- Value based RL
  - Learn optimal action value function Q
  - Policy is the generated implicitly from the learned Q function (greedy, ϵ greedy)
  - Eg: Q Learning, SARSA, Monte Carlo Control etc.
- Policy based RL (Policy Gradient Methods)
  - Learn the policy π directly.
  - Eg: REINFORCE , A2C, TRPO, PPO, DDPG etc.

Outline

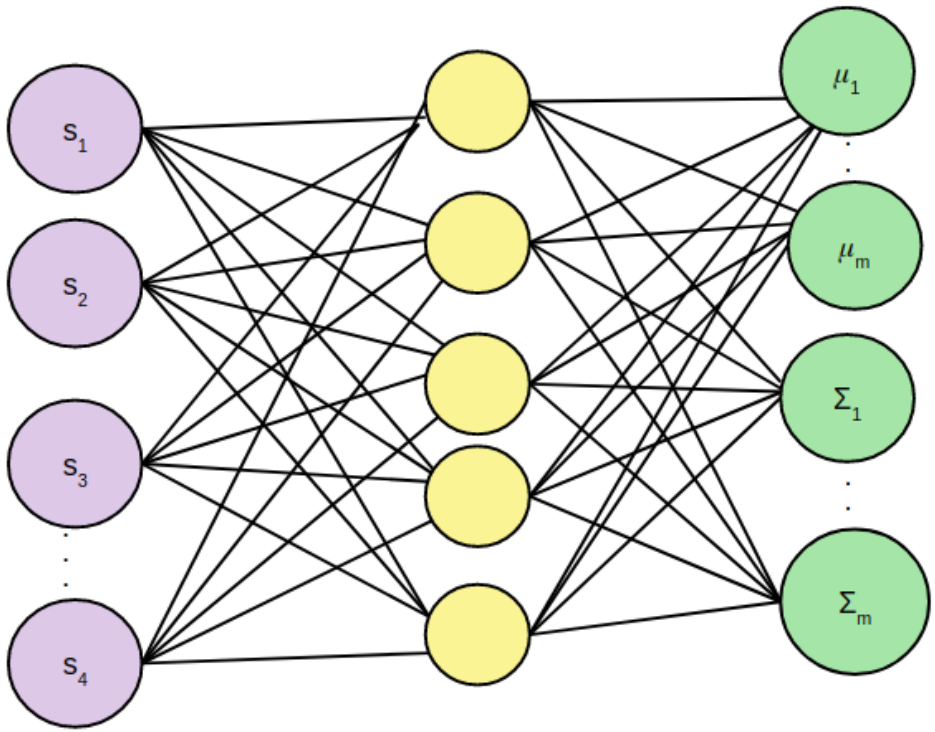
Parameterized Policies

- For the purpose of this lecture we will focus on policies parameterized by Neural Nets.
- For **Discrete action spaces**, the network takes the state representation as the input and outputs the probabilities of taking each action (Softmax Policy). Analogus to classification in Supervised Learning.



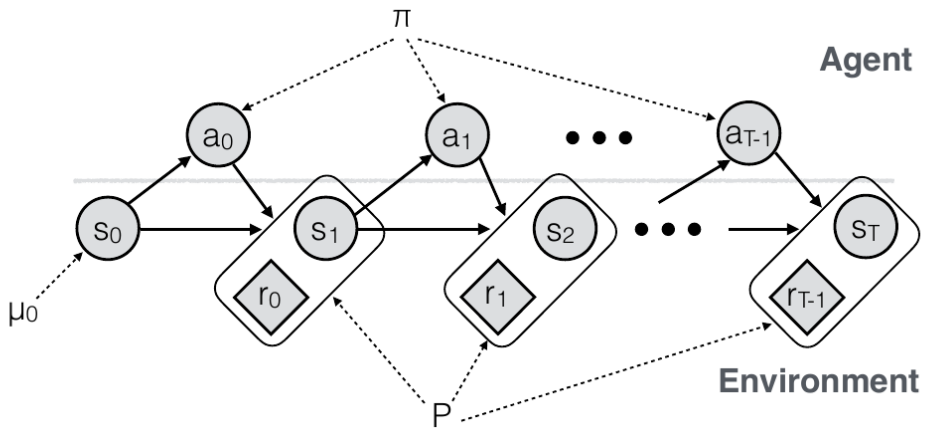
Parameterized Policies

- In case of **Continuous action spaces**, we assume our policy to be gaussian and the network takes the state as input and outputs the mean and diagonal covariance of the distribution. Analogus to regression in Supervised Learning.



Policy Optimization Objective

$$s_0 \sim \mu(s_0)$$
$$a_0 \sim \pi(a_0|s_0;\theta)$$
$$s_1 \sim \mathbb{P}(s_1|s_0,a_0)$$
$$r_0 \sim \mathbb{R}(s_0,a_0,s_1)$$
$$a_1 \sim \pi(a_1|s_1;\theta)$$
$$s_2 \sim \mathbb{P}(s_2|s_1,a_1)$$
$$r_1 \sim \mathbb{R}(s_1,a_1,s_2)$$
$$\ldots$$
$$a_{T-1} \sim \pi(a_{T-1}|s_{T-1};\theta)$$
$$s_T \sim \mathbb{P}(s_T|s_{T-1},a_{T-1})$$
$$r_{T-1} \sim \mathbb{R}(s_{T-1},a_{T-1},s_T)$$



[1] Image from John Schulman slides on Policy Gradients.

Policy Optimization Objective

Objective Function

maximize $\eta(\pi_\theta)$

where

$$\begin{aligned}\eta(\pi_\theta) &= \mathbb{E}_\tau[r_0 + \gamma r_1 + \gamma^2 r_2 + ... + \gamma^{T-1} r_{T-1} | \pi_\theta] \\ &= \mathbb{E}_\tau[R(\tau) | \pi_\theta]\end{aligned} \tag{1}$$

and τ denotes a trajectory $(s_0, a_0, r_0, s_1, a_1, r_1, ..., s_T, a_{T-1}, r_{T-1})$

Intuition

By maximizing this objective function we:

- Make the good trajectories more probable.
- By making good trajectories more probable we also make good actions more probable.

Solving the Optimization Problem

- Note that our objective function is not a direct function of policy parameters θ
- To optimize the function using gradient based methods we need a way to estimate the gradients of the objective with respect to θ
- **Score function gradient estimator** is used for this purpose.

Score Function Gradient Estimator

- Consider an optimization problem of maximizing $\mathbb{E}_{x\sim p(x|\theta)}[f(x)]$ with respect to θ

$$\begin{aligned}
\nabla_\theta \mathbb{E}_x[f(x)] &= \nabla_\theta \sum_x p(x|\theta) f(x) \\
&= \sum_x \nabla_\theta p(x|\theta) f(x) \\
&= \sum_x p(x|\theta) \frac{\nabla_\theta p(x|\theta)}{p(x|\theta)} f(x) \\
&= \sum_x p(x|\theta) \nabla_\theta \log p(x|\theta) f(x) \\
&= \mathbb{E}_x[f(x) \nabla_\theta \log p(x|\theta)]
\end{aligned} \tag{2}$$

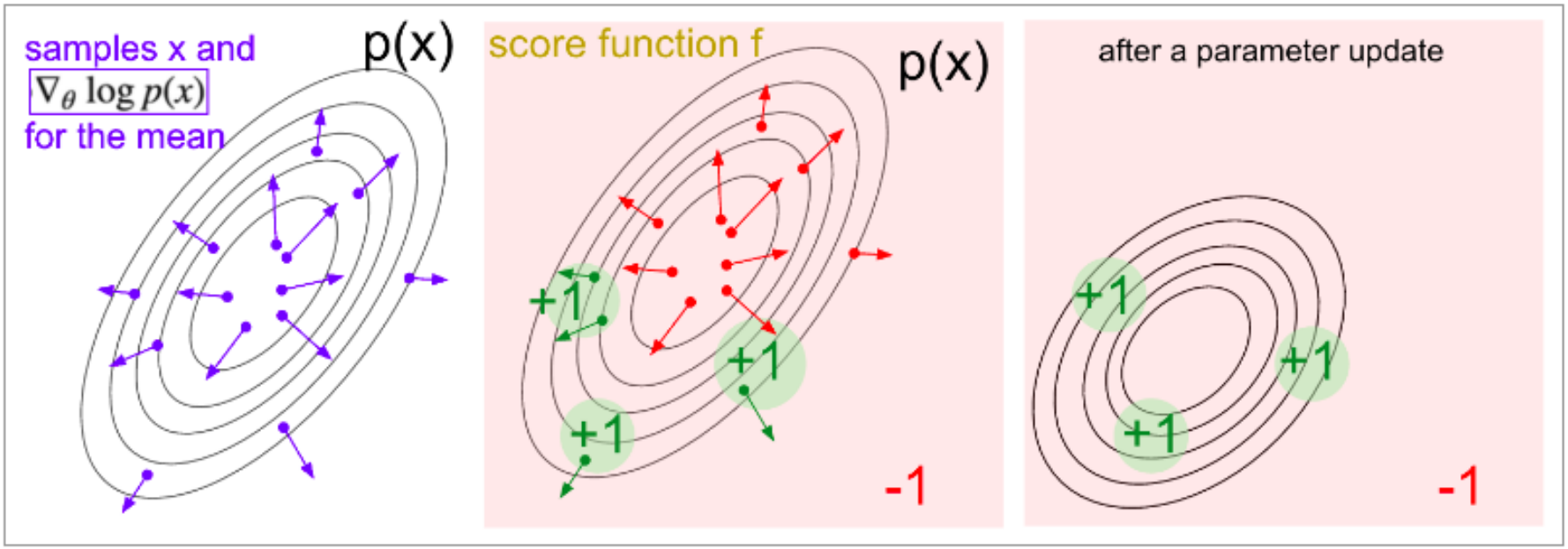
Score Function Gradient Estimator

Intuition

Lets denote the score function estimator as $g_i = f(x_i)\nabla_\theta \log p(x_i|\theta)$

- Let $f(x)$ denotes how good a sample x is.
- Moving in the direction of g pushes the probability of x in the proportion of how good it is ($f(x)$).
- A sample which leads to a small value of $f(x)$ will lead to adjustment of parameters θ such that its log probability is reduced as compared to the samples with higher $f(x)$ and vice versa.
- $f(x)$ **need not be a continuous or differentiable function**.

Score Function Gradient Estimator



A visualization of the score function gradient estimator. **Left**: A gaussian distribution and a few samples from it (blue dots). On each blue dot we also plot the gradient of the log probability with respect to the gaussian's mean parameter. The arrow indicates the direction in which the mean of the distribution should be nudged to increase the probability of that sample. **Middle**: Overlay of some score function giving -1 everywhere except +1 in some small regions (note this can be an arbitrary and not necessarily differentiable scalar-valued function). The arrows are now color coded because due to the multiplication in the update we are going to average up all the green arrows, and the *negative* of the red arrows. **Right**: after parameter update, the green arrows and the reversed red arrows nudge us to left and towards the bottom. Samples from this distribution will now have a higher expected score, as desired.

[1] Image from Andrej Karpathy's blog (https://karpathy.github.io/2016/05/31/rl/)

Score Function Gradient Estimator for Policies

- Remember our objective function was: $\max_\theta \mathbb{E}_\tau[R(\tau)]$
- The random variable x in this case is the trajectory τ denoted by $(s_0, a_0, r_0, s_1, a_1, r_1, ..., s_T, a_{T-1}, r_{T-1})$
- And the function f is the total cumulative reward. $f(\tau) = R(\tau) = \sum_t \gamma^t r_t$
- The probability of the trajectory can be computed as:

$$p(\tau) = \mu(s_0)\pi(a_0|s_0;\theta)\mathbb{P}(s_1|s_0,a_0)\pi(a_1|s_1;\theta)\mathbb{P}(s_2|s_1,a_1)...$$

$$p(\tau) = \mu(s_0)\prod_{t=0}^{T-1}\pi(a_t|s_t;\theta)\mathbb{P}(s_{t+1}|s_t,a_t)$$

$$\log p(\tau) = \log \mu(s_0) + \sum_{t=0}^{T-1}\log \pi(a_t|s_t;\theta) + \sum_{t=0}^{T-1}\log \mathbb{P}(s_{t+1}|s_t,a_t)$$

Score Function Gradient Estimator for Policies

$$\nabla_\theta \log p(\tau) = \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t;\theta)$$

$$\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau[R\nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t;\theta)]$$

$$\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau[(\sum_{t=0}^{T-1} \gamma^t r_t)(\nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t;\theta))]$$

Intuition: Good trajectories i.e. the trajectories with higher values of R will serve as supervised examples like in the case of classification or regression.

Score Function Gradient Estimator for Policies

- A slightly better estimate can be obtained by rearranging the terms in the expression above and we obtain:
  $\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau[\nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t;\theta) \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'}]$
- The second term in the expression can be interpreted as an estimate of action value function (Q)
  $\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau[\nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t;\theta) Q(s_t, a_t)]$

Practical Implementation with Autodiff

- Collect n different trajectories
  $\tau_i = (s_0^i, a_0^i, r_0^i, s_1^i, a_1^i, r_1^i, ..., s_T^i, a_{T-1}^i, r_{T-1}^i)$
- For each trajectory τ_i compute Q value estimates
  $Q(s_t^i, a_t^i) = \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'}^i$
- The log probabilities can be obtained from the output of neural network f.
  - For Discrete Action Spaces: $\log \pi(a_t^i | s_t^i; \theta) = \log f(s_t^i; \theta)$
  - For Continuous Action Spaces:

$$\mu_t^i, \sigma_t^i = f(s_t; \theta)$$

$$\log \pi(a_t^i | s_t^i; \theta) = \frac{-(a_t^i - \mu_t^i)^2}{2\sigma_t^{i2}} - \frac{1}{2} \log(2\pi^k \sigma_t^i)$$

Practical Implementation with Autodiff

- Define the surrogate loss as: $L_{surr}(\theta) = -\frac{\sum_{i=1}^{n}\sum_{t=0}^{T-1} Q(s_t^i, a_t^i)\log\pi(a_t|s_t;\theta))}{n}$
- Find policy gradient estimate: $g(\theta) = \nabla_\theta L_{surr}(\theta)$
  - Pytorch: loss.backward()
  - Tensorflow : tf.train.Optimizer().minimize(loss)
- Plug $g(\theta)$ in your favourite optimizer SGD/ ADAM
- This algorithm is called REINFORCE or Vanilla Policy Gradient

Collecting Trajectories

Initialize sets S, A, R, P for storing states, actions, rewards and neural network outputs.

Sample initial state s_0 from environment E

t = 0

Repeat till episode **terminates**

Sample action a_t from $\pi(a_t|s_t, \theta)$ defined using the outputs of neural network $f(s_t)$. (Multinomial Distribution for discrete case and Gaussian for continuous)

Execute action a_t in environment E and obtain reward r_t and new state s_{t+1}

Store $s_t, a_t, r_t, f(s_t)$ in the sets S, A, R, P

t := t+1

REINFORCE Algorithm

Initialize policy parameters θ

for iteration = 1,2... **do**

Collect a set of trajectories using current policy π.

At each time step for each trajectory compute the approximate Q value as: $Q(s_t^i, a_t^i) = \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'}^i$

Compute the log probabilities $\log \pi(a_t|s_t;\theta)$ using the output of the neural net.

Compute the surrogate loss using log probabilities and Q values.

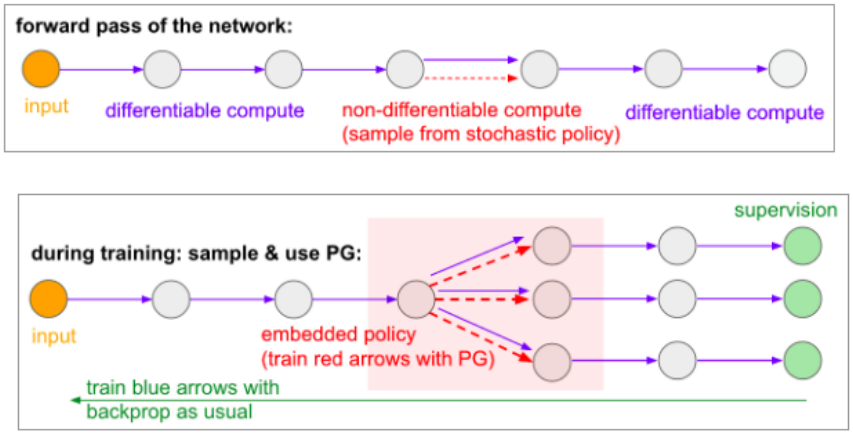
Compute the gradient $g(\theta)$ of the loss function and update the parameters of policy: For eg. SGD Update: $\theta = \theta - \alpha g(\theta)$

end for

Outline

Upside of Policy Gradients

- One of the most attractive thing about policy gradients is that the reward function need not be differentiable with respect to the policy parameters.
- This provides us with a framework to handle non differentiable computation when working with neural networks.
- Such situation often arises when we require sampling from a probability distribution during the forward pass or when we require to optimize a non differentiable objective function.



[1] Image from Andrej Karpathy's blog (https://karpathy.github.io/2016/05/31/rl/)

Case Study: Optimizing BLEU for NMT

- For training Machine Translation systems we optimize the negative log likelihood of the predictions of decoder.
- However NLL loss might not be the best measure of the quality of translation.
- For evaluation of the translation by an NMT model we often use metrics like BLEU.
- However using standard supervised learning approaches we can not directly optimize BLEU since it requires the access to the predicted tokens, not the probabilities.
- To make matters worse BLEU calculation is non differentiable so even if we manage to select the tokens by a differentiable operation (max pool) it will still be problematic.

Case Study: Optimizing BLEU for NMT

- Policy Gradients to the rescue!
- We can represent the machine translation problem as an MDP:
  - State s_t : The state at time step t will be given by the input token to the decoder at time t and the hidden state of the decoder h_t which encompasses information from the previous time steps of decoder as well as the hidden states of encoder.
  - Action a_t : The action at time step t is given by the prediction of decoder at time t. This hence becomes a case of discrete action space and we can use a softmax policy.
  - Reward r_t: We use a delayed reward in this case where 0 reward is given at all time steps and at final step reward equal to the BLEU score between the prediction and the ground truth is used.
  - Transition $\mathbb{P}$: Transition in this case will be given by the input token at the next time step and the updated hidden state of RNN h_t

Case Study: Optimizing BLEU for NMT

- Now we can simply use the REINFORCE algorithm to train our translation system.
- Please note that the action space for this problem is quite large (equal to the number of words in the vocabulary).
- Directly training the model with REINFORCE will be very slow and might not even converge to a good policy.
- People often first train their model using the standard supervised learning approach and then fine tune it using policy gradients.

Outline

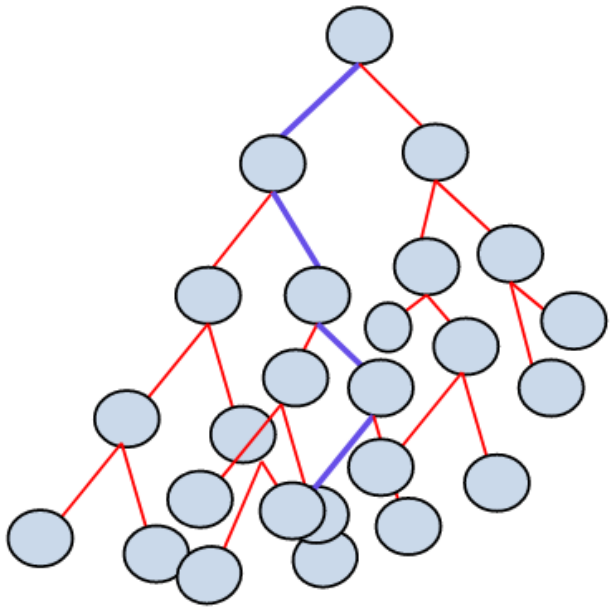
Downside of Policy Gradients

Vanilla Policy Gradients suffers from 2 serious problems:

- High variance of gradient estimates
- Highly sample inefficient

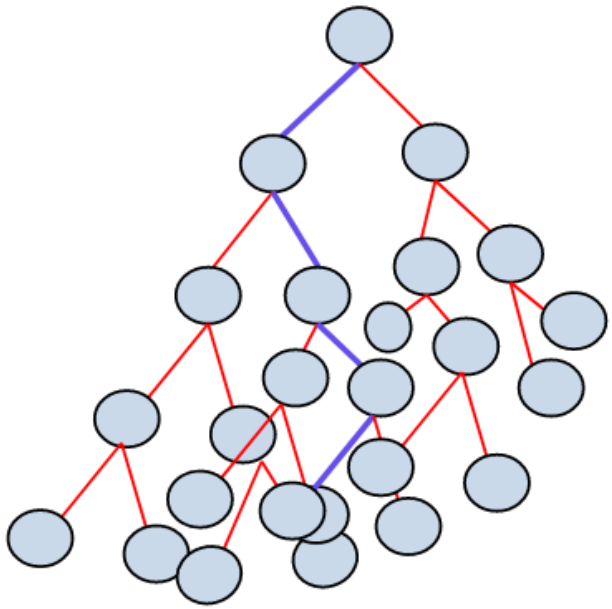
High Variance of gradient estimates

- As we saw before the policy gradient is given by: $\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau[\nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t;\theta) \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'}]$
- However during implementation we estimate this expectation using only a few examples which leads to a very high variance in the estimate, especially for long trajectories and high dimensional action spaces.
- One natural way of reducing variance is to collect a lot of trajectories before calculating the policy gradient.



High Variance of gradient estimates

- Using discount factor also reduces the variance to an extent since it diminishes the importance of future rewards.
- However these 2 tricks are not always sufficient to solve the problem.
- One way of reducing variance which is mostly used in practice is by subtracting the Q value estimate with an appropriate baseline.
- Actor critic methods (A2C, A3C) learn a baseline to reduce the variance. (Will be discussed in next tutorial)



Sample Efficiency

- The vanilla policy gradient algorithm has a very bad sample efficiency.
- The collected trajectories can only be used for a single gradient update.
- After making the update we throw away the data that was collected and collect new experience using the updated policy for the next update.
- In many cases collecting data might be very expensive (for eg. In Robotics)

Sample Efficiency

- To better understand why this problem occurs, lets look again at the equation of policy gradients:
  $\nabla_\theta \mathbb{E}_\tau[R] = \mathbb{E}_\tau[\nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t;\theta) \sum_{t'=t}^{T-1} \gamma^{t'} r_{t'}]$
- The expectation is over the trajectory distribution which is a function of policy.
- Once we update the policy, the trajectory distribution will change too.
- Hence the collected samples will no longer be valid to compute this expectation.
- Variants of vanilla policy gradients like TRPO, PPO (next tutorial) modify the policy gradient to enable multiple policy updates using the collected samples.
- Off policy variants like DDPG enables the use of experience replay which helps in using the samples from past experience to make policy updates.

References

1. John Schulman's slides on Policy Gradients: http://rail.eecs.berkeley.edu/deeprlcoursesp17/docs/lec2.pdf
2. David Silver's slides on Policy Gradients: http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching_files/pg.pdf
3. Andrej Karpathy's Blog on Policy Gradients: https://karpathy.github.io/2016/05/31/rl/
4. Lijun Wu, Fei Tian, Tao Qin, Jianhuang Lai, Tie-Yan Liu: "A Study of Reinforcement Learning for Neural Machine Translation" (2018)

Thank you!