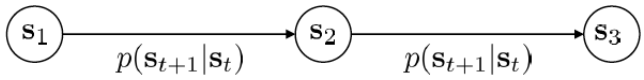
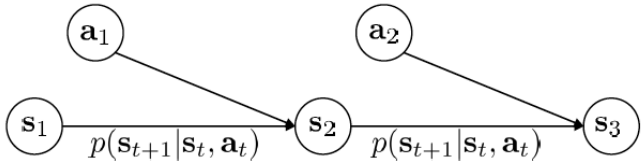
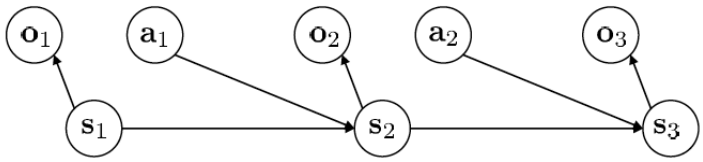
Lecture 4: Introduction to Reinforcement Learning

Definitions

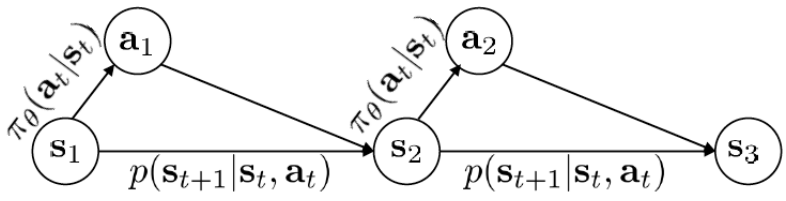
- $\pi_\theta\left(\mathbf{a}_t \mid \mathbf{o}_t\right)$ - policy (partially observed)
- $\pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)$ - policy (fully observed)
- **Markov Chain**: $\mathcal{M}=\{\mathcal{S}, \mathcal{T}\}$
  - 

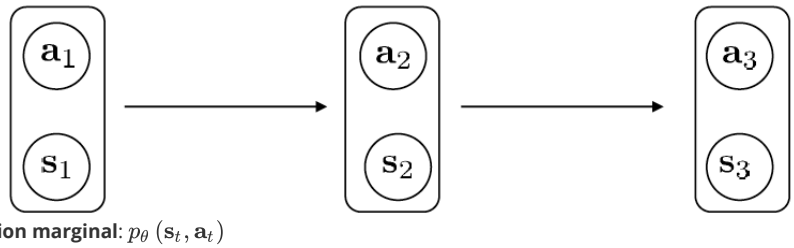
  - $\mathcal{S}$ - state space
  - $\mathcal{A}$ - action space
- **Markov Decision Process**: $\mathcal{M}=\{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$
  - 

  - $\mathcal{T}$ - transition operator
  - r - reward function: $r: \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$
- **partially observed Markov Decision Process**: $\mathcal{M}=\{\mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T}, \mathcal{E}, r\}$
  - 

  - $\mathcal{O}$ - observation space
  - $\mathcal{E}$ - emission probability - $p(o_t \mid s_t)$

RL objective

- $p_\theta(\tau)=p_\theta\left(\mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T\right)=p\left(\mathbf{s}_1\right) \prod_{t=1}^T \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right) p\left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t\right)$
  - $\theta^\star=\arg \max _\theta E_{\tau \sim p_\theta(\tau)}\left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]$
- Markov Chain on $(\mathbf{s}, \mathbf{a})$
  - $p\left(\left(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}\right) \mid\left(\mathbf{s}_t, \mathbf{a}_t\right)\right)=p\left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t\right) \pi_\theta\left(\mathbf{a}_{t+1} \mid \mathbf{s}_{t+1}\right)$
  - 

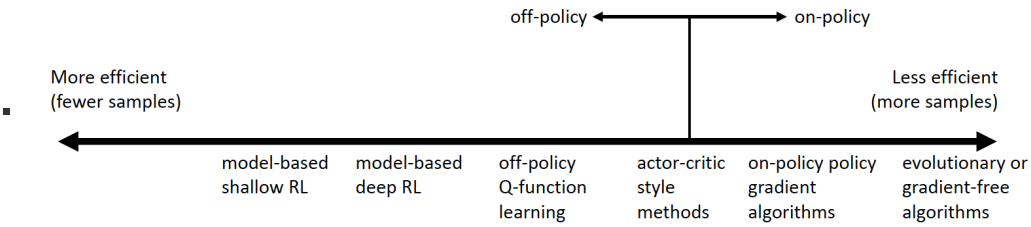
  - 

- **state-action marginal**: $p_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right)$
- **finite horizon case**: $\theta^\star=\arg \max _\theta E_{\tau \sim p_\theta(\tau)}\left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]=\arg \max _\theta \sum_{t=1}^T E_{\left(\mathbf{s}_t, \mathbf{a}_t\right) \sim p_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right)}\left[r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]$
- **infinite horizon case**: $\theta^\star=\arg \max _\theta E_{\tau \sim p_\theta(\tau)}\left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right] \rightarrow \arg \max _\theta E_{(\mathbf{s}, \mathbf{a}) \sim p_\theta(\mathbf{s}, \mathbf{a})}[r(\mathbf{s}, \mathbf{a})]$ (in the limit as $T \rightarrow \infty$)
  - stationary distribution: $\mu=p_\theta\left(\mathbf{s}_t, \mathbf{a}_t\right)$
    - $\mu=\mathcal{T} \mu$ => $(\mathcal{T}-\mathbf{I}) \mu=0$ =>μ is eigenvector of $\mathcal{T}$ with eigenvalue 1.
      - $\mathcal{T}$ is state-action trainsition operator

RL Algorithms

- *Algorithm*:

```
Repeat:
  Generate Samples (i.e. run the policy)
  fit a model / estimate the return
  improve the policy
```

- **Q-function**: $Q^{\pi}\left(\mathbf{s}_{t}, \mathbf{a}_{t}\right)=\sum_{t^{\prime}=t}^{T} E_{\pi_{\theta}}\left[r\left(\mathbf{s}_{t^{\prime}}, \mathbf{a}_{t^{\prime}}\right) \mid \mathbf{s}_{t}, \mathbf{a}_{t}\right]$
- **Value function**: $V^{\pi}\left(\mathbf{s}_{t}\right)=\sum_{t^{\prime}=t}^{T} E_{\pi_{\theta}}\left[r\left(\mathbf{s}_{t^{\prime}}, \mathbf{a}_{t^{\prime}}\right) \mid \mathbf{s}_{t}\right]=E_{\mathbf{a}_{t} \sim \pi\left(\mathbf{a}_{t} \mid \mathbf{s}_{t}\right)}\left[Q^{\pi}\left(\mathbf{s}_{t}, \mathbf{a}_{t}\right)\right]$
  - RL objective in terms of V: $E_{\mathbf{s} \sim p(\mathbf{s})}\left[V^{\pi}\left(\mathbf{s}_{1}\right)\right]$
- *Key ideas*:
  - If we have policy π and know $$Q^{\pi}\left(\mathbf{s}, \mathbf{a}\right)$, then we can improve the policy
  - compute gradient to increase probability od good action $\mathbf{a}$
    - if $$Q^{\pi}\left(\mathbf{s}, \mathbf{a}\right) > V^{\pi}\left(\mathbf{s}\right)$, then $\mathbf{a}$ is better than average =>
- *Types of RL algorithms*:
  - **Policy gradients**: directly differentiate the objective $\theta^{\star}=\arg \max _{\theta} E_{\tau \sim p_{\theta}(\tau)}\left[\sum_{t} r\left(\mathbf{s}_{t}, \mathbf{a}_{t}\right)\right]$
    - eg. REINFORCE, Natural PG, Trust Region Policy Gradient (TRPO)
  - **Value-based**: estimate V or Q function of the optimal policy (no explicit policy)
    - eg. Q-learning, DQN, TD Learning, Fitted Value Iteration
  - **Actor-Critic**: estimate V or Q function of the current policy, use it to improve policy
    - eg. Asynchronous advantage actor-critic (A3C), Soft actor critic (SAC)
  - **Model-based RL**: estimate the transition model, and then use it for planning (no explicit policy) or to improve a policy or something else.
    - eg. Dyna, Guided Policy Search
- Why so many RL algorithms?
  - Different tradeoffs
    - *Sample efficiency*
      - 

      - **off-policy vs on-policy** - for on-policy, will need to - new samples every time policy is changed.
      - don't forget the wall-clock time for each iteration.
    - *Stability & ease of use* - convergence?
      - Value function fitting
        - At best, minimizes error of fit ("Bellman error") - Not the same as expected reward
        - At worst, doesn't optimize anything - Many popular deep RL value fitting algorithms are not guaranteed to converge to anything in the nonlinear case
      - Model based RL
        - Model minimizes error of fit - This will converge
        - No guarantee that better model = better policy
      - Policy gradient
        - The only one that actually performs SGD on the true objective
  - Different assumptions
    - Stochastic or deterministic?
    - Continuous or discrete?
    - Episodic or infinite horizon?
    - Common assumptions:
      - full observability
        - generally by value-function fitting methods

        - episodic learning
            - often by pure PG methods
            - some model-based RL methods
        - continuity or smoothness
            - some continuous value-function fitting methods
            - often by model-based RL methods
    - Different things are easy or hard in different settings
        - Easier to represent the policy?
        - Easier to represent the model?

Lecture 5: Policy Gradients

The policy gradient algorithm

- RL objective $\theta^{\star} = \arg\max_{\theta} E_{\tau \sim p_{\theta}(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]$
    - finite-horizon: $\theta^{\star} = \arg\max_{\theta} \sum_{t=1}^{T} E_{(\mathbf{s}_t, \mathbf{a}_t) \sim p_{\theta}(\mathbf{s}_t, \mathbf{a}_t)} \left[r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]$
    - infinite-horizon: $\theta^{\star} = \arg\max_{\theta} E_{(\mathbf{s}, \mathbf{a}) \sim p_{\theta}(\mathbf{s}, \mathbf{a})} \left[r\left(\mathbf{s}, \mathbf{a}\right)\right]$
- *episode reward*: $r(\tau) = \sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)$
- *objective*: $J(\theta) = E_{\tau \sim p_{\theta}(\tau)}\left[r(\tau)\right] = E_{\tau \sim p_{\theta}(\tau)} \left[\sum_t r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]$
- *gradient*: $\nabla_{\theta} J(\theta) = E_{\tau \sim \pi_{\theta}(\tau)}\left[\nabla_{\theta} \log \pi_{\theta}(\tau) r(\tau)\right] = E_{\tau \sim \pi_{\theta}(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right)\left(\sum_{t=1}^{T} r\left(\mathbf{s}_t, \mathbf{a}_t\right)\right)\right]$
  * using log-derivative trick: $\nabla_{\theta} \pi_{\theta}(\tau) = \pi_{\theta}(\tau) \nabla_{\theta} \log \pi_{\theta}(\tau)$
  * also $\nabla_{\theta} \log \pi_{\theta}(\tau) = \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_t \mid \mathbf{s}_t\right)$
- Sample to evaluate $J(\theta)$ and $\nabla_{\theta} J(\theta)$
    - $J(\theta) \approx \frac{1}{N} \sum_i r\left(\tau_i\right) = \frac{1}{N} \sum_i \sum_t r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)$
    - $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\nabla_{\theta} \log \pi_{\theta}(\tau_i) r(\tau_i)\right) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\right)\left(\sum_{t=1}^{T} r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)\right)$
    - All expectations can be similarly evaluated!
- **REINFORCE**
    1. Repeat:
        1. sample $\left\{\tau^i\right\}$ from $\pi_{\theta}\left(\mathbf{a}_t \mid \mathbf{s}_t\right)$ (run the policy)
        2. $\nabla_{\theta} J(\theta) \approx \sum_i \left(\sum_t \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_t^i \mid \mathbf{s}_t^i\right)\right)\left(\sum_t r\left(\mathbf{s}_t^i, \mathbf{a}_t^i\right)\right)$
        3. $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$
- *Policy Gradient v/s Maximum Likelihood (ML)*
    - PG: $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\nabla_{\theta} \log \pi_{\theta}(\tau_i) r(\tau_i)\right) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\right)\left(\sum_{t=1}^{T} r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)\right)$
    - ML: $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\nabla_{\theta} \log \pi_{\theta}(\tau_i)\right) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\right)$
    - basically good stuff made more likely and bad stuff less.
- *Partial Observability* -> simply replace $\pi_{\theta}\left(\mathbf{a}_t \mid \mathbf{s}_t\right)$ with $\pi_{\theta}\left(\mathbf{a}_t \mid \mathbf{o}_t\right)$
    - $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_{i,t} \mid \mathbf{o}_{i,t}\right)\right)\left(\sum_{t=1}^{T} r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)\right)$
    - works as we never used the Markov property!
- problem: *High Variance*

Basic Variance Reduction

- **Causality**: policy at time t' cannot affect reward at time $t < t'$
    - Let $\hat{Q}_{i,t} = \sum_{t'=t}^{T} r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right)$
    - $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\right)\left(\sum_{t=1}^{T} r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)\right) = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right) \hat{Q}_{i,t}$
- **Baselines**
    - Use $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\nabla_{\theta} \log \pi_{\theta}(\tau_i)\left(r(\tau_i) - b\right)\right)$
    - subtracting baselines is unbiased in expectation: $E\left[\nabla_{\theta} \log \pi_{\theta}(\tau) b\right] = 0$
    - potential baselines:
        - $b = \frac{1}{N} \sum_{i=1}^{N} r(\tau)$ - *average reward*
        - $b = \frac{E\left[g(\tau)^2 r(\tau)\right]}{E\left[g(\tau)^2\right]}$ - *optimal baseline* that minimizes variance $Var_{\tau \sim p_{\theta}(\tau)}\left[g(\tau)(r(\tau) - b)\right]$
            - $g(\tau) = \nabla_{\theta} \log \pi_{\theta}\left(\tau\right)$

Off-policy Learning using Importance Sampling

- PG is on-policy
- **Importance sampling (IS)**: $E_{x\sim p(x)}[f(x)] = \int p(x)f(x)dx = E_{x\sim q(x)}\left[\frac{p(x)}{q(x)}f(x)\right]$
- Using IS, $J(\theta) = E_{\tau\sim\bar{\pi}(\tau)}\left[\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)}r(\tau)\right]$
  - where $\frac{\pi_\theta(\tau)}{\bar{\pi}(\tau)} = \frac{p(\mathbf{s}_1)\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)}{p(\mathbf{s}_1)\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)p(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t)} = \frac{\prod_{t=1}^{T}\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\prod_{t=1}^{T}\bar{\pi}(\mathbf{a}_t|\mathbf{s}_t)}$
- To estimate value at some *new* parameters θ'
  - $J(\theta') = E_{\tau\sim\pi_\theta(\tau)}\left[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)}r(\tau)\right]$
  - $\nabla_{\theta'}J(\theta') = E_{\tau\sim\pi_\theta(\tau)}\left[\frac{\pi_{\theta'}(\tau)}{\pi_\theta(\tau)}\nabla_{\theta'}\log\pi_{\theta'}(\tau)r(\tau)\right]$
  - problem: Exponential scaling in T
  - solution 1:
  - solution 2: apply IS to state action marginal $(\mathbf{s}_{i,t},\mathbf{a}_{i,t}) \sim \pi_\theta(\mathbf{s}_t,\mathbf{a}_t)$ and ignore state portion
    - on-policy PG: $\nabla_{\theta'}J(\theta') \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(\mathbf{a}_{i,t}\mid\mathbf{s}_{i,t})\hat{Q}_{i,t}$
    - off-policy PG: $\nabla_{\theta'}J(\theta') \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\frac{\pi_{\theta'}(\mathbf{s}_{i,t},\mathbf{a}_{i,t})}{\pi_\theta(\mathbf{s}_{i,t},\mathbf{a}_{i,t})}\nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_{i,t}\mid\mathbf{s}_{i,t})\hat{Q}_{i,t}$
      - Apply bayes rule and ignore the state portion
        - $\nabla_{\theta'}J(\theta') \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\frac{\pi_{\theta'}(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}{\pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})}\nabla_{\theta'}\log\pi_{\theta'}(\mathbf{a}_{i,t}\mid\mathbf{s}_{i,t})\hat{Q}_{i,t}$

Pseudocode (with discrete actions)

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)
```

Policy Gradients

```
# 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 (reward to go Q_t)
# 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)
```

Practical Considerations

- remember that gradient has high variance and will be noisy
- use larger batch size
- learning rates, optimizers

Lecture 6: Actor-Critic Algorithms

- $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta\log\pi_\theta(\mathbf{a}_{i,t}\mid\mathbf{s}_{i,t})\hat{Q}^{\pi}_{i,t}$ has high variance

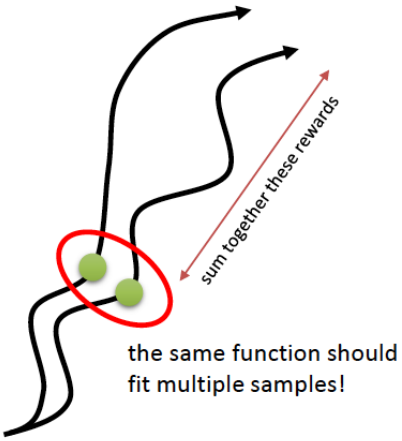
    - lower variance will allow **bigger steps** and **faster learning**.
- $\hat{Q}_{i,t} = \sum_{t'=t}^{T} r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right)$ is **single-sample** estimate of expected reward for taking $\mathbf{a}_{i,t}$ at $\mathbf{s}_{i,t}$ ("reward to go")
    - *unbiased but high variance*
- **state-action value function**: $Q^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right) \mid \mathbf{s}_t, \mathbf{a}_t\right]$ - true *expected* reward-to-go
- **state value function**: $V^{\pi}\left(\mathbf{s}_t\right) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right) \mid \mathbf{s}_t\right] = E_{\mathbf{a}_t \sim \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)}\left[Q^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]$ - total reward from $\mathbf{s}_t$
- **advantage**: $A^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right) = Q^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right) - V^{\pi}\left(\mathbf{s}_t\right)$ - how much better than average $\mathbf{a}_t$ is
- $\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right) A^{\pi}\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)$
    - *biased but lower vairance*
        - Using a lower variance "reward-to-go" and a state-dependent baseline $V\left(\mathbf{s}_t\right)$
    - better the estimate $A^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right)$, lower the variance -> usually use an approximation

Value-Function fitting

- What to fit - Q, V or A?
    - Tradeoff: Can get V and A from Q but it depends on both s and a.
- $Q^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right) = r\left(\mathbf{s}_t, \mathbf{a}_t\right) + \sum_{t'=t+1}^{T} E_{\pi_\theta}\left[r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right) \mid \mathbf{s}_t, \mathbf{a}_t\right] = r\left(\mathbf{s}_t, \mathbf{a}_t\right) + E_{\mathbf{s}_{t+1} \sim p\left(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t\right)}\left[V^{\pi}\left(\mathbf{s}_{t+1}\right)\right]$
- Use a single sample estimate for second part (unbiased)
    - $Q^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right) \approx r\left(\mathbf{s}_t, \mathbf{a}_t\right) + V^{\pi}\left(\mathbf{s}_{t+1}\right)$
    - $A^{\pi}\left(\mathbf{s}_t, \mathbf{a}_t\right) \approx r\left(\mathbf{s}_t, \mathbf{a}_t\right) + V^{\pi}\left(\mathbf{s}_{t+1}\right) - V^{\pi}\left(\mathbf{s}_t\right)$
    - Can now fit only V! => don't need to fit a function approximator that takes both state and action

Policy Evaluation

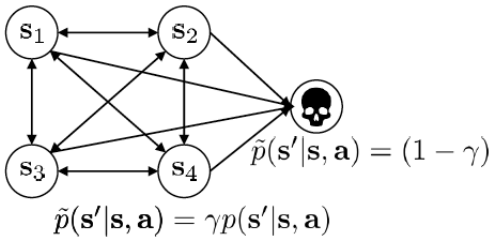
- Policy evaluation - Fitting value function to policy
    - $V^{\pi}\left(\mathbf{s}_t\right) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right) \mid \mathbf{s}_t\right]$
    - $J(\theta) = E_{\mathbf{s}_1 \sim p\left(\mathbf{s}_1\right)}\left[V^{\pi}\left(\mathbf{s}_1\right)\right]$
- Estimate V^{π}?
    - $V^{\pi}\left(\mathbf{s}_t\right) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t'=t}^{T} r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right)$ - requires resetting the simulator to $\mathbf{s}_t$
    - $V^{\pi}\left(\mathbf{s}_t\right) \approx \sum_{t'=t}^{T} r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right)$ - single sample estimate - **Monte Carlo target**
        - unbiased but higher variance
        - but still works because the same function approximator (NN) is fitted for all states => needs to generalize and some sharing happens between similar states.
        -



        - **Monte Carlo Evaluation**
            - $V^{\pi}\left(\mathbf{s}_t\right) \approx \sum_{t'=t}^{T} r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right)$
            - training data: $\left\{\left(\mathbf{s}_{i,t}, y_{i,t}\right)\right\}$ where $y_{i,t} = \sum_{t'=t}^{T} r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right)$
            - supervised regression: $\mathcal{L}(\phi) = \frac{1}{2} \sum_i \left\|\hat{V}_\phi^{\pi}\left(\mathbf{s}_i\right) - y_i\right\|^2$
    - $y_{i,t} \approx r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right) + \hat{V}_\phi^{\pi}\left(\mathbf{s}_{i,t+1}\right)$: **Bootstrapped Target** or **TD Target**
        - Approximate the reward after current timestep using the fitted value function V^{π} from the previous training iteration
        - Biased because
            - the critic is not perfect (not fitted yet)
            - plus used the previous critic

    - But **lower variance** and likely better

Discount Factors

- $y_{i,t} \approx r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)+\gamma \hat{V}_{\phi}^{\pi}\left(\mathbf{s}_{i,t+1}\right)$ -> better to get rewards sooner rather than later
    - Useful for infinite horizon problems -> infinite summation
    - Also useful for finite horizon problems as if rewards are always positive -> V^{π} will keep getting bigger
    - changes the MDP -> add a *death state*
    - 

- Monte Carlo policy gradients with discount factors
    - option 1: $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\left(\sum_{t'=t}^{T} \gamma^{t'-t} r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right)\right)$
    - option 2: $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \gamma^{t-1} \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\left(\sum_{t=t}^{T} \gamma^{t'-t} r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right)\right)$
        - later steps contribute less to the gradient and hence matter less
        - correct formulation for the MDP with deadth state
    - option 1 is used in practice
        - with critic: $\nabla_{\theta} J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\left(r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)+\gamma \hat{V}_{\phi}^{\pi}\left(\mathbf{s}_{i,t+1}\right)-\hat{V}_{\phi}^{\pi}\left(\mathbf{s}_{i,t}\right)\right)$
        - the larger the summation, the higher the variance
        - γ allows to tradeoff bias and variance
        - γ allows an approximation to truly *infinite horizon average reward formulation* (one without a death state) while avoiding the infinite sums (returns).
        - refer: Bias in natural actor-critic algorithms

Actor-Critic Algorithm (with discount)

- **Batch Actor-Critic Algorithm**:
    1. Repeat:
        1. sample $\left\{\mathbf{s}_i, \mathbf{a}_i\right\}$ from $\pi_\theta(\mathbf{a} \mid \mathbf{s})$ (run it on the robot)
        2. fit $\hat{V}_{\delta}^{\pi}(\mathbf{s})$ to sampled reward sums
        3. evaluate $\hat{A}^{\pi}\left(\mathbf{s}_i, \mathbf{a}_i\right)=r\left(\mathbf{s}_i, \mathbf{a}_i\right)+\gamma \hat{V}_{\phi}^{\pi}\left(\mathbf{s}_i'\right)-\hat{V}_{\phi}^{\pi}\left(\mathbf{s}_i\right)$
        4. $\nabla_{\theta} J(\theta) \approx \sum_i \nabla_{\theta} \log \pi_{\theta}\left(\mathbf{a}_i \mid \mathbf{s}_i\right) \hat{A}^{\pi}\left(\mathbf{s}_i, \mathbf{a}_i\right)$
        5. $\theta \leftarrow \theta+\alpha \nabla_{\theta} J(\theta)$
- **Online Actor-Critic Algorithm:**
    1. Repeat:
        1. take action $\mathbf{a} \sim \pi_\theta(\mathbf{a} \mid \mathbf{s})$, get $(\mathbf{s}, \mathbf{a}, \mathbf{s}', r)$
        2. update $\hat{V}_{\phi}^{\pi}$ using target $r+\gamma \hat{V}_{\phi}^{\pi}\left(\mathbf{s}'\right)$
        3. evaluate $\hat{A}^{\pi}(\mathbf{s}, \mathbf{a})=r(\mathbf{s}, \mathbf{a})+\gamma \hat{V}_{\phi}^{\pi}\left(\mathbf{s}'\right)-\hat{V}_{\phi}^{\pi}(\mathbf{s})$
        4. $\nabla_{\theta} J(\theta) \approx \nabla_{\theta} \log \pi_{\theta}(\mathbf{a} \mid \mathbf{s}) \hat{A}^{\pi}(\mathbf{s}, \mathbf{a})$
        5. $\theta \leftarrow \theta+\alpha \nabla_{\theta} J(\theta)$
    - Architecture?
        - Two network design: One each for $\hat{V}_{\phi}^{\pi}\left(\mathbf{s}'\right)$ and $\pi_\theta(\mathbf{a} \mid \mathbf{s})$
            - simple and stable but no shared features
        - Shared network design
            - shared features but need to take care of gradients for shared part
- Online Actor-Critic in practice
    - single sample batch is bad (too high variance) - want larger batch size
    - **synchronized parallel actor-critic**
        - synchronisation at the end of each iteration
    - **asynchronous parallel actor-critic**
        - use a parameter server
        - easier to implement as no need to synchronise
        - used in practice

- Critics as state-dependent baselines
  - Actor-Critic: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\left(r\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)+\gamma \hat{V}_\phi^\pi\left(\mathbf{s}_{i,t+1}\right)-\hat{V}_\phi^\pi\left(\mathbf{s}_{i,t}\right)\right)$
    - lowest variance but biased
  - Policy Gradient: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\left(\left(\sum_{t'=t}^{T}\gamma^{t'-t} r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right)\right)-b\right)$
    - unbiased but higher variance (because single sample estimate)
  - middle ground: $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\left(\left(\sum_{t'=t}^{T}\gamma^{t'-t} r\left(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}\right)\right)-\hat{V}_\phi^\pi\left(\mathbf{s}_{i,t}\right)\right)$
    - unbiased and lower variance than PG (baseline V^π is closer to rewards)
- **Control Variates**: action-dependent baselines
  - $\hat{A}^\pi\left(\mathbf{s}_t, \mathbf{a}_t\right)=\sum_{t'=t}^{\infty}\gamma^{t'-t} r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right)-Q_\phi^\pi\left(\mathbf{s}_t, \mathbf{a}_t\right)$
    - not correct - goes to zero in expectation if critic is correct!
  - $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta\left(\mathbf{a}_{i,t} \mid \mathbf{s}_{i,t}\right)\left(\hat{Q}_{i,t}-Q_\phi^\pi\left(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}\right)\right)+\frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta E_{\mathbf{a}\sim\pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_{i,t}\right)}\left[Q_\phi^\pi\left(\mathbf{s}_{i,t}, \mathbf{a}_t\right)\right]$
    - Added an additional expectation term
      - was zero for baseline and state-dependent baseline
    - Should be possible to compute the expectation analytically as intractable to do so using samples
      - Eg. multivariate normal policy and Q quadratic in $\mathbf{a}$
    - Leads to even lower variance than using V^π as baseline (and still unbiased).
- **Multi-Step Returns**
  - Bootstrapped target has lower variance but higher bias if value is wrong (it always is)
  - Monte Carlo target has no bias but much higher variacne (single sample estimarte)
  - $\hat{A}_n^\pi\left(\mathbf{s}_t, \mathbf{a}_t\right)=\sum_{t'=t}^{t+n}\gamma^{t'-t} r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right)+\gamma^n \hat{V}_\phi^\pi\left(\mathbf{s}_{t+n}\right)-\hat{V}_\phi^\pi\left(\mathbf{s}_t\right)$
    - Use n-step returns where n is small enough to avoid high variance.
    - lower variance as $\gamma^n \hat{V}_\phi^\pi\left(\mathbf{s}_{t+n}\right)$ becomes more heavily discounted as $n \to \infty$.
    - Combines the best of bootstrapped target and monte-carlo target.
  - **Generalised Advantage Estimation**
    - $\hat{A}_{\mathrm{GAE}}^\pi\left(\mathbf{s}_t, \mathbf{a}_t\right)=\sum_{n=1}^{\infty} w_n \hat{A}_n^\pi\left(\mathbf{s}_t, \mathbf{a}_t\right)$ - Weighted combination of n-step returns
      - With $w_n=(1-\lambda)\lambda^{n-1}$ - exponential falloff - prefer cutting earlier (less variance)
      - $\hat{A}_{\mathrm{GAE}}^\pi\left(\mathbf{s}_t, \mathbf{a}_t\right)=\sum_{t'=t}^{\infty}(\gamma\lambda)^{t'-t}\delta_{t'}$ where $\delta_{t'}=r\left(\mathbf{s}_{t'}, \mathbf{a}_{t'}\right)+\gamma \hat{V}_\phi^\pi\left(\mathbf{s}_{t'+1}\right)-\hat{V}_\phi^\pi\left(\mathbf{s}_{t'}\right)$
        - λ has similar effect to γ - variance reduction
        - $\lambda=0$ gives advantage with bootstrapped target
        - $\lambda=1$ gives advantage with monte-carlo target
        - $0<\lambda<1$ makes a compromise between bias and variance

Papers

- Classic papers
  - Sutton, McAllester , Singh, Mansour (1999). Policy gradient methods for reinforcement learning with function approximation: actor critic algorithms with value function approximation
- Deep reinforcement learning actor critic papers
  - Mnih , Badia , Mirza, Graves, Lillicrap , Harley, Silver, Kavukcuoglu (2016). Asynchronous methods for deep reinforcement learning: A3C parallel online actor critic
  - Schulman, Moritz, L., Jordan, Abbeel (2016). High dimensional continuous control using generalized advantage estimation: batch mode actor critic with blended Monte Carlo and function approximator returns
  - Gu, Lillicrap , Ghahramani , Turner, L. (2017). Q Prop: sample efficient policy gradient with an off policy critic: policy gradient with Q function control variate

Lecture 7: Value Function Methods

Value-based Methods

- High Level **Policy Iteration Algorithm**
  1. Repeat:
     1. evaluate $A^\pi(\mathbf{s}, \mathbf{a})$ or $V^\pi(\mathbf{s}, \mathbf{a})$ <- fit a model
     2. set $\pi \leftarrow \pi'$ <- improve implicit policy
- Policy Evaluation using Dynamic Programming (DP)

    - bootstrapped update: $V^{\pi}(\mathbf{s}) \leftarrow E_{\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})}\left[r(\mathbf{s}, \mathbf{a}) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \mathbf{a})}\left[V^{\pi}\left(\mathbf{s}'\right)\right]\right]$
    - simplified: $V^{\pi}(\mathbf{s}) \leftarrow r(\mathbf{s}, \pi(\mathbf{s})) + \gamma E_{\mathbf{s}' \sim p(\mathbf{s}'|\mathbf{s}, \pi(\mathbf{s}))}\left[V^{\pi}\left(\mathbf{s}'\right)\right]$
        - using current estimate of V^{π} and deterministic policy $\pi(\mathbf{s}) = \mathbf{a}$
            - for any fully observed MDP, there exists a optimal policy that is deterministic
                - not true for partially observerd MDPs.
    - Use this to evaluate $V^{\pi}(\mathbf{s}, \mathbf{a})$ (step 1 in policy iteration).
- Can skip the policy (step 2) and compute value or Q-function directly
- **Value iteration Algorithm**
    1. Repeat:
        1. set $Q(\mathbf{s}, \mathbf{a}) \leftarrow r(\mathbf{s}, \mathbf{a}) + \gamma E\left[V\left(\mathbf{s}'\right)\right]$ <- fit a model
        2. set $V(\mathbf{s}) \leftarrow \max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a})$ <- improve implicit policy
    - Maintaining the table for $V(\mathbf{s})$ may be intractable => need function approximation
- **Fitted Value Iteration Algorithm**
    1. Repeat:
        1. set $\mathbf{y}_i \leftarrow \max_{\mathbf{a}_i}\left(r\left(\mathbf{s}_i, \mathbf{a}_i\right) + \gamma E\left[V_{\phi}\left(\mathbf{s}_i'\right)\right]\right)$
        2. set $\phi \leftarrow \arg\min_{\phi} \frac{1}{2} \sum_i \left\|V_{\phi}\left(\mathbf{s}_i\right) - \mathbf{y}_i\right\|^2$
    - supervised regression
    - instead of enumerating all possible states can just sample states
- *Problem*: step 1 requires enumerating all actions from each state which requires knowing transition dynamics
- *Solution*: iterate on Q instead of V
    - approximate $E\left[V\left(\mathbf{s}_i'\right)\right] \approx \max_{\mathbf{a}'} Q_{\phi}\left(\mathbf{s}_i', \mathbf{a}_i'\right)$
    - computing Q values for each action doesn't require knowing the transition dynamics

Q-value based Methods

- Full **Fitted Q Iteration Algorithm**:
    1. Repeat:
        1. collect dataset $\left\{\left(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}_i', r_i\right)\right\}$ using some policy - parameters: dataset size N, collection policy
        2. Repeat K times: - parameter: K
            1. set $\mathbf{y}_i \leftarrow r\left(\mathbf{s}_i, \mathbf{a}_i\right) + \gamma \max_{\mathbf{a}_i'} Q_{\phi}\left(\mathbf{s}_i', \mathbf{a}_i'\right)$
            2. set $\phi \leftarrow \arg\min_{\phi} \frac{1}{2} \sum_i \left\|Q_{\phi}\left(\mathbf{s}_i, \mathbf{a}_i\right) - \mathbf{y}_i\right\|^2$ - parameter: gradient step S
    - pros:
        - doesn't require simulation of actions!
        - works even for *off-policy* samples (unlike actor critic)
        - only *one network*, no high variance policy gradient
    - cons:
        - no convergence guarantees for non linear function approximation
    - *Q-learning* is a special case of this algorithm.
    - the $\max_{a_i'}$ in step 2,1 improves policy in tabular case - no guarantees with function approximation (NN)
    - **Bellman Error**: $\mathcal{E} = \frac{1}{2} E_{(\mathbf{s}, \mathbf{a}) \sim \beta}\left[\left(Q_{\phi}(\mathbf{s}, \mathbf{a}) - \left[r(\mathbf{s}, \mathbf{a}) + \gamma \max_{\mathbf{a}'} Q_{\phi}\left(\mathbf{s}', \mathbf{a}'\right)\right]\right)^2\right]$
        - step 2.2 minimizes this
- **Q-Learning: Online Q-Iteration Algorithm**:
    1. Repeat:
        1. take some action $\mathbf{a}_i$ and observe $\left(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}_i', r_i\right)$
        2. $\mathbf{y}_i = r\left(\mathbf{s}_i, \mathbf{a}_i\right) + \gamma \max_{\mathbf{a}'} Q_{\phi}\left(\mathbf{s}_i', \mathbf{a}_i'\right)$
        3. $\phi \leftarrow \phi - \alpha \frac{dQ_{\phi}}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)\left(Q_{\phi}\left(\mathbf{s}_i, \mathbf{a}_i\right) - \mathbf{y}_i\right)$
    - Problem: batch size is 1 - bad for high-dimensional parametric classes (eg. NN).
- **Exploration Policy**
    - epsilon-greedy policy
    - Boltzmann Exploration: $\pi\left(\mathbf{a}_t \mid \mathbf{s}_t\right) \propto \exp(Q_{\phi}\left(\mathbf{s}_t, \mathbf{a}_t\right))$
        - biases against sub-optimal policies with large -ve Q.

Value Function Learning Theory

- Let operator $\mathcal{B}$: $\mathcal{B}V = \max_{\mathbf{a}} r_{\mathbf{a}} + \gamma \mathcal{T}_{\mathbf{a}} V$
    - *Contraction* in l_{∞} norm: $\|\mathcal{B}V - \mathcal{B}\bar{V}\|_{\infty} \leq \gamma \|V - \bar{V}\|_{\infty}$ for all V and $\bar{V}$

    - V^* is a fixed point of $\mathcal{B}$
        - always exists, is unique and corresponds to optimal policy
- Let operator $\mathcal{C}$: $\mathcal{C}Q = r + \gamma\mathcal{T}\max_{\mathbf{a}} Q$
    - Also a *Contraction* in l_∞ norm
    - Q^* is a fixed point of $\mathcal{C}$
        - always exists, is unique and corresponds to optimal policy
- Let operator Π : $\Pi V = \arg\min_{V' \in \Omega} \frac{1}{2}\sum \|V'(\mathbf{s}) - V(\mathbf{s})\|^2$
    - Π is a l_2 projection on Ω (eg.neural nets)
    - **Contraction** in l_2 norm: $\|\Pi V - \Pi\bar{V}\|_2 \leq \gamma\|V - \bar{V}\|_2$ for all V and $\bar{V}$
- *Value iteration*: $V \leftarrow \mathcal{B}V$
    - Converges to V^* (in tabular case) because $\mathcal{B}$ is a contraction.
        - With $\bar{V} = V^\star$, $\|\mathcal{B}V - V^\star\|_\infty \leq \gamma\|V - V^\star\|_\infty$
- *Fitted Value iteration*: $V \leftarrow \Pi\mathcal{B}V$
    - does not converge as $\Pi\mathcal{B}$ is not a contraction of any kind.
        - not in general and often not in practice
- *Fitted Q-iteration*: $V \leftarrow \Pi\mathcal{C}V$
    - also does not converge as $\Pi\mathcal{C}$ is not a contraction of any kind
    - *Can still be made to work because NNs are powerful function approximator and can make the error in Π very small.*

Lecture 8: Deep RL with Q-Functions

Making Online Q-learning work with NNs

- Problem 1: Batch size is 1 and samples in subseqeunt steps are correlated
    - **Parallel Q-learning** - synchronised or asynchronous
        - Pro: larger batch size
        - Con: doesn't fully resolve correlated samples' issue.
    - **Replay Buffer**
        - Samples no longer correlated
        - Multiple samples in the batch (low variance gradient)
        - Need to periodically feed the replay buffer
- Problem 2: Moving target (no gradient through target value)
    - Q-learning with **Target Networks** (and Replay Buffer)
        1. Repeat:
            1. save target network parameters: $\phi' \leftarrow \phi$
            2. Repeat N times::
                1. collect dataset $\left\{\left(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i\right)\right\}$ using some policy, add it to $\mathcal{B}$
                2. Repeat K times:
                    1. sample a batch $\left(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i\right)$ from $\mathcal{B}$
                    2. $\phi \leftarrow \phi - \alpha\sum_i \frac{dQ_\phi}{d\phi}(\mathbf{s}_i, \mathbf{a}_i)\left(Q_\phi\left(\mathbf{s}_i, \mathbf{a}_i\right) - \left[r\left(\mathbf{s}_i, \mathbf{a}_i\right) + \gamma\max_{\mathbf{a}'} Q_{\phi'}\left(\mathbf{s}'_i, \mathbf{a}'_i\right)\right]\right)$
        - Use an earlier ϕ as ϕ'
        - Targets don't change in inner loop
        - Perfectly well defined supervised regression like in Fitted Q-iteration
- **DQN**
    1. Repeat
        1. take some action $\mathbf{a}_i$ and observe $\left(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i\right)$, add it to $\mathcal{B}$
        2. sample mini-batch $\left\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\right\}$ from $\mathcal{B}$ uniformly
        3. compute $y_j = r_j + \gamma\max_{\mathbf{a}'_j} Q_{\phi'}\left(\mathbf{s}'_j, \mathbf{a}'_j\right)$ using target network $Q_{\phi'}$
        4. $\phi \leftarrow \phi - \alpha\sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)\left(Q_\phi\left(\mathbf{s}_j, \mathbf{a}_j\right) - y_j\right)$
        5. update ϕ' : copy ϕ every N steps
    - Much more stable due to target networks and replay buffers
- Problem with Target Network: uneven lag between ϕ and ϕ'
    - Solution: **Polyak averaging**: $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$
        - $\tau = 0.999$ works well

Generalised view of Q-learning

- Both Q-learning (with target networks and replay buffer) and Fitted Q-iteration have these processes running parallelly but at different rates:
  - Process 1: data collection (using exploration policy) (and eviction of old data)
  - Process 2: target update (potentially using polyak averaging)
  - Process 3: Q-function regression (supervised regression)
- Online Q learning: evict immediately, process 1, process 2, and process 3 all run at the same speed
- DQN: process 1 and process 3 run at the same speed, process 2 is slow
- Fitted Q iteration: process 3 in the inner loop of process 2, which is in the inner loop of process 1

Tricks for improving Q learning in practice

Double Q-Learning

- *Problem*: Predicted reward seems to be systematically larger than actual reward
- *Cause*:
  - The max-term in $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}\left(\mathbf{s}'_j, \mathbf{a}'_j\right)$ overestimates the next value
    - end-up paying less attention to the rewards
  - $E\left[\max\left(X_1, X_2\right)\right] \geq \max\left(E\left[X_1\right], E\left[X_2\right]\right)$
  - Cause: value and selected action both come according to $Q_{\phi'}$ which is imperfect/noisy
    - $\max_{\mathbf{a}'} Q_{\phi'}\left(\mathbf{s}', \mathbf{a}'\right) = Q_{\phi'}\left(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi'}\left(\mathbf{s}', \mathbf{a}'\right)\right)$
- *Solution*: **Double Q-learning**
  - decorrelate noise using different network to choose action and evaluate value
  - Uses two networks:
    - $Q_{\phi_A}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_B}\left(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_A}\left(\mathbf{s}', \mathbf{a}'\right)\right)$
    - $Q_{\phi_B}(\mathbf{s}, \mathbf{a}) \leftarrow r + \gamma Q_{\phi_A}\left(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi_B}\left(\mathbf{s}', \mathbf{a}'\right)\right)$
  - In practice, use current and target network: $y = r + \gamma Q_{\phi'}\left(\mathbf{s}', \arg\max_{\mathbf{a}'} Q_{\phi}\left(\mathbf{s}', \mathbf{a}'\right)\right)$
    - sufficiently decorrelated

Multi-step returns in off-policy Q-learning

- $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}\left(\mathbf{s}'_j, \mathbf{a}'_j\right)$
  - First term matters more early on when $Q_{\phi'}$ is bad otherwise second term matters more
- $y_{j,t} = \sum_{t'=t}^{t+N-1} \gamma^{t-t'} r_{j,t'} + \gamma^N \max_{\mathbf{a}_{j,t+N}} Q_{\phi'}\left(\mathbf{s}_{j,t+N}, \mathbf{a}_{j,t+N}\right)$
  - Pros:
    - less biased target values when Q values are inaccurate
    - typically faster learning, especially early on
  - Cons:
    - only actually correct when learning on policy because the first term comes from the exploration policy
      - solution 1: ignore - often works well for smal N
      - solution 2: dynamically choose N to get only on policy data
        - works well when data mostly on policy, and action space is small
          - and the behavior policy takes same action as the target policy
      - solution 3: importance sampling
- reference: *Safe and efficient off-policy reinforcement learning*, Munos et al. '16

Q-learning with Continuous Action Spaces

- problem: max over actions
  - $\pi\left(\mathbf{a}_t \mid \mathbf{s}_t\right) = \begin{cases} 1 \text{ if } \mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q_\phi\left(\mathbf{s}_t, \mathbf{a}_t\right) \\ 0 \text{ otherwise} \end{cases}$
  - target value $y_j = r_j + \gamma \max_{\mathbf{a}'_j} Q_{\phi'}\left(\mathbf{s}'_j, \mathbf{a}'_j\right)$
    - particularly bad as in inner training loop
- solution 1: **optimization**
  - option 1: gradient based optimization (e.g., SGD) a bit slow in the inner loop
  - option 2: stochastic optimization

        - action space typically low dimensional
        - simple solution:
            - $\max_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}) \approx \max \left\{ Q\left(\mathbf{s}, \mathbf{a}_1\right), \ldots, Q\left(\mathbf{s}, \mathbf{a}_N\right) \right\}$
                - $\left(\mathbf{a}_1, \ldots, \mathbf{a}_N\right)$ sampled from some distribution (e.g., uniform)
                - very simple and efficiently parallelizable
                - inaccurate
        - Cross Entropy Method (CEM)
            - simple iterative stochastic optimization
        - CMA-ES
            - substantially less simple iterative stochastic optimization
        - works for upto 40 dimensions
- solution 2: **Analytic optimisation using easily maximizable Q-functions**
    - option 1: NAF: Normalized Advantage Functions
        - $Q_\phi(\mathbf{s}, \mathbf{a}) = -\frac{1}{2}(\mathbf{a} - \mu_\phi(\mathbf{s}))^T P_\phi(\mathbf{s}) \left(\mathbf{a} - \mu_\phi(\mathbf{s})\right) + V_\phi(\mathbf{s})$
            - quadratic in a and arbitrarily complex in s
        - $\arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a}) = \mu_\phi(\mathbf{s})$
        - $\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a}) = V_\phi(\mathbf{s})$
        - Pros:
            - no change in algorithm
            - just as efficient as Q-learning
        - Cons:
            - loses representational power - Q may be highly non-quadratic in a
- solution 3: **learn an approximate maximizer**
    - The hard part is the $\arg\max$: $\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a}) = Q_\phi\left(\mathbf{s}, \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})\right)$
    - idea: train another network $\mu_\theta(\mathbf{s})$ such that $\mu_\theta(\mathbf{s}) \approx \arg\max_{\mathbf{a}} Q_\phi(\mathbf{s}, \mathbf{a})$
        - want to solve $\theta \leftarrow \arg\max_\theta Q_\phi\left(\mathbf{s}, \mu_\theta(\mathbf{s})\right)$
        - $\frac{dQ_\phi}{d\theta} = \frac{d\mathbf{a}}{d\theta}\frac{dQ_\phi}{d\mathbf{a}}$
        - following this gradient will learn the required "argmaxer" $\mu_\theta(s)$
    - new target $y_j = r_j + \gamma Q_{\phi'}\left(\mathbf{s}'_j, \mu_\theta\left(\mathbf{s}'_j\right)\right) \approx r_j + \gamma Q_{\phi'}\left(\mathbf{s}'_j, \arg\max_{\mathbf{a}'} Q_{\phi'}\left(\mathbf{s}'_j, \mathbf{a}'_j\right)\right)$
    - **DDPG** (Lillicrap et al., ICLR 2016):
        1. Repeat
            1. take some action $\mathbf{a}_i$ and observe $\left(\mathbf{s}_i, \mathbf{a}_i, \mathbf{s}'_i, r_i\right)$, add it to $\mathcal{B}$
            2. sample mini-batch $\left\{\mathbf{s}_j, \mathbf{a}_j, \mathbf{s}'_j, r_j\right\}$ from $\mathcal{B}$ uniformly
            3. compute $y_j = r_j + \gamma Q_{\phi'}\left(\mathbf{s}'_j, \mu_{\theta'}\left(\mathbf{s}'_j\right)\right)$ using target nets $Q_{\phi'}$ and $\mu_{\theta'}$
            4. $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(\mathbf{s}_j, \mathbf{a}_j)\left(Q_\phi\left(\mathbf{s}_j, \mathbf{a}_j\right) - y_j\right)$
            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)

Practical tips for Q-learning

- Q learning takes some care to stabilize
    - Test on easy, reliable tasks first, make sure your implementation is correct
- Large replay buffers help improve stability
    - Looks more like fitted Q iteration
- It takes time, be patient might be no better than random for a while
- Start with high exploration (epsilon) and gradually reduce
- Bellman error gradients can be big; clip gradients or user Huber loss
- Double Q learning helps a lot in practice, simple and no downsides
- N step returns also help a lot, but have some downsides
- Schedule exploration (high to low) and learning rates (high to low), Adam optimizer can help too
- Run multiple random seeds and average results, it's very inconsistent between runs

Lecture 8: Advanced Policy Gradients

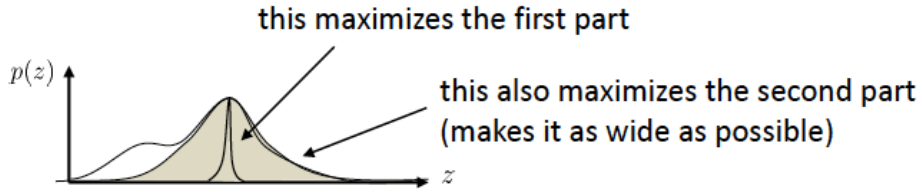
- **REINFORCE**
  1. Repeat:
     1. sample $\left\{\tau^i\right\}$ from $\pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)$ (run the policy)
     2. $\nabla_\theta J(\theta) \approx \sum_i\left(\sum_t \nabla_\theta \log \pi_\theta\left(\mathbf{a}_t^i \mid \mathbf{s}_t^i\right)\right)\left(\sum_t r\left(\mathbf{s}_t^i, \mathbf{a}_t^i\right)\right)$
     3. $\theta \leftarrow \theta+\alpha \nabla_\theta J(\theta)$
- Proximal Policy Gradient: Paper, Blog
  - Use importance sampling directly
- Natural Policy Gradient: Paper
  - Use natural Gradient
- Trust Region Policy Optimization
  - Use natural Gradient
  - Fix ϵ and derive α
  - Efficient Fischer-vector products without building the Fischer Information Matrix explicitly

Lecture 9: Optimal Control and Planning

Lecture 13: Variational Inference and Generative Models

- Probabilistic Models: $p(x)$ or $p(y|x)$
- Latent variable models
  - If $p(x)$ (or $p(y|x)$) is a complex distribution, it can be represented as a non-linear transformation of simple distribution (eg. Gaussian).
  - To this, we introduce a latent variable z.
  - Case 1: z is discrete and takes on a small number of values
    - $p(x)=\sum_z p(x|z)p(z)$
    - $p(y|x)=\sum_z p(y|x,z)p(z)$
    - How to train the model?
      - Can write down the sum and optimize it directly
      - Or can use EM
  - Case 2: z is continuous
    - $p(x)=\int_z p(x|z)p(z)$
    - $p(y|x)=\int_z p(y|x,z)p(z)$
    - Can have both $p(x|z)$ and $p(z)$ be any easy distribution. eg. Gaussian
      - Sample $z \sim p(z)$ and $p(x \mid z)=\mathcal{N}\left(\mu_{\mathrm{nn}}(z), \sigma_{\mathrm{nn}}(z)\right)$ where $\mu_{nn}(z)$ and $\sigma_{nn}(z)$ can be complex functions of z. Eg. Neural Networks.
    - How to train the model?
      - the model: $p_\theta(x)=\int p_\theta(x \mid z) p(z) dz$
      - the data: $\mathcal{D}=\{x_1, x_2, x_3, \ldots, x_N\}$
      - maximum likelihood fit: $\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_i \log p_\theta\left(x_i\right)=\arg\max_\theta \frac{1}{N} \sum_i \log(\int p_\theta\left(x_i \mid z\right) p(z) dz)$
      - The integral is completely intractable!
      - Idea: "guess" most likely z given x_i and pretend it's the right one. As there are many possible values, use the distribution $p(z|x_i)$
      - $\theta \leftarrow \arg\max_\theta \frac{1}{N} \sum_i E_{z \sim p(z|x_i)}\left[\log p_\theta\left(x_i, z\right)\right]$
      - To calculate $p(z|x_i)$ we use Variational Inference.
- **Variational Inference**
  - Approximate $p(z|x_i)$ with $q_i(z)=\mathcal{N}\left(\mu_i, \sigma_i\right)$
  - Now can get a bound on $\log p(x_i)$ (using Jensen's Inequality)
    - $$\begin{aligned}\log p(x_i) &= \log E_{z \sim q_i(z)}\left[\frac{p\left(x_i \mid z\right) p(z)}{q_i(z)}\right] \\ &\geq E_{z \sim q_i(z)}\left[\log \frac{p\left(x_i \mid z\right) p(z)}{q_i(z)}\right] \\ &= E_{z \sim q_i(z)}\left[\log p\left(x_i \mid z\right)+\log p(z)\right]+\mathcal{H}(q_i) \\ &= \mathcal{L}_i\left(p, q_i\right)\end{aligned}$$
  - $\mathcal{L}_i\left(p, q_i\right)$ is known as the **Variational Lower Bound (VLBO)** or **Evidence Lower Bound (ELBO)**
  - Relation to **KL-Divergence**
    - $D_{\mathrm{KL}}(q \| p)=E_{x \sim q(x)}\left[\log \frac{q(x)}{p(x)}\right]=E_{x \sim q(x)}[\log q(x)]-E_{x \sim q(x)}[\log p(x)]=-E_{x \sim q(x)}[\log p(x)]-\mathcal{H}(q)$

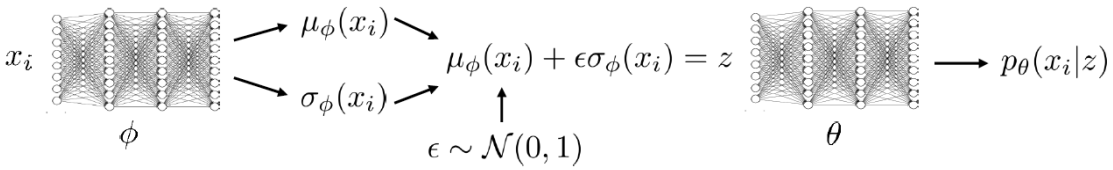
-



  - *Minimising* $D_{\mathrm{KL}}\left(q_i\left(z\right)\|p\left(z \mid x_i\right)\right)$ *w.r.t.* q_i *is equivalent to maximising* $\mathcal{L}_i\left(p, q_i\right)$*!*
  - $$\begin{aligned} D_{\mathrm{KL}}\left(q_i\left(z\right)\|p\left(z \mid x_i\right)\right) &= E_{z\sim q_i(z)}\left[\log \frac{q_i(z)}{p\left(z \mid x_i\right)}\right] = E_{z\sim q_i(z)}\left[\log \frac{q_i(z)p\left(x_i\right)}{p\left(x_i, z\right)}\right] \\ &= -E_{z\sim q_i(z)}\left[\log p\left(x_i \mid z\right) + \log p(z)\right] + E_{z\sim q_i(z)}\left[\log q_i(z)\right] + E_{z\sim q_i(z)}\left[\log p\left(x_i\right)\right] \\ &= -\mathcal{L}_i\left(p, q_i\right) + \log p\left(x_i\right) \end{aligned}$$
- As $\log p_\theta(x_i) \geq \mathcal{L}_i\left(p, q_i\right)$
  - we can solve, $\theta \leftarrow \arg\max_\theta \frac{1}{N}\sum_i \mathcal{L}_i\left(p, q_i\right)$
  - instead of $\theta \leftarrow \arg\max_\theta \frac{1}{N}\sum_i \log p_\theta(x_i)$
- *Algorithm*:
  - for each x_i (or mini-batch):
    - calculate $\nabla_\theta \mathcal{L}_i\left(p, q_i\right)$
      - sample $z \sim q_i\left(z\right)$
      - $\nabla_\theta \mathcal{L}_i\left(p, q_i\right) \approx \nabla_\theta \log p_\theta\left(x_i \mid z\right)$
    - $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}_i\left(p, q_i\right)$
    - $\phi_i \leftarrow \phi_i + \alpha \nabla_{\phi_i} \mathcal{L}_i\left(p, q_i\right)$
  - Above ϕ_i are the parameters for q_i
    - $\phi_i = [\mu_i, \sigma_i]$ if $q_i(z) = \mathcal{N}\left(\mu_i, \sigma_i\right)$
- *Problem*: If we use a separate q_i for each sample, we'll have too many parameters!
  - Eg. if $q_i(z) = \mathcal{N}\left(\mu_i, \sigma_i\right)$, then there are $|\theta| + (|\mu_i| + |\sigma_i|) \times N$ parameters!
- *Solution*: *Amortized Varitional Inference*

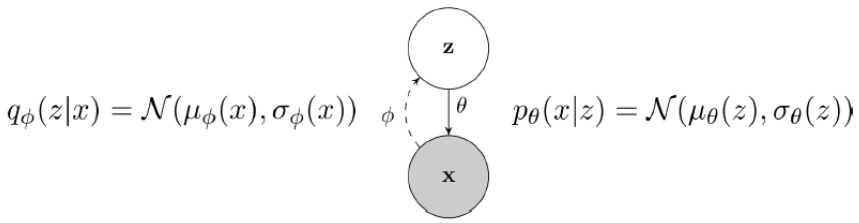
- **Amortised Variational Inference**
  - Train a network $q_i(z) = q_\phi\left(z \mid x_i\right) \approx p\left(z \mid x_i\right)$
    - Eg. $q_\phi(z \mid x) = \mathcal{N}\left(\mu_\phi(x), \sigma_\phi(x)\right)$
  - We thus have two networks, $p_\theta(x|z)$ for $p(x|z)$ and $q_\phi(z|x)$ for $p(z|x)$
  - **Algorithm**:
    - for each x_i (or mini-batch):
      - calculate $\nabla_\theta \mathcal{L}_i\left(p, q_i\right)$
        - sample $z \sim q_i\left(z\right)$
        - $\nabla_\theta \mathcal{L}_i\left(p, q_i\right) \approx \nabla_\theta \log p_\theta\left(x_i \mid z\right)$
      - $\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{L}_i\left(p, q_i\right)$
      - $\phi \leftarrow \phi + \alpha \nabla_\phi \mathcal{L}_i\left(p, q_i\right)$
  - How to compute $\nabla_\phi \mathcal{L}_i$ where $\mathcal{L}_i = E_{z\sim q_\phi(z|x_i)}\left[\log p_\theta\left(x_i \mid z\right) + \log p(z)\right] + \mathcal{H}\left(q_\phi\left(z \mid x_i\right)\right)$?
    - Second term is easy: Just lookup the formula for entropy of Gaussian (assuming $q_\phi(z \mid x) = \mathcal{N}\left(\mu_\phi(x), \sigma_\phi(x)\right)$)
    - First term is of the form $J(\phi) = E_{z\sim q_\phi(z|x_i)}\left[r\left(x_i, z\right)\right]$ so we can use *Policy Gradients*
      - $\nabla J(\phi) \approx \frac{1}{M}\sum_j \nabla_\phi \log q_\phi\left(z_j \mid x_i\right) r\left(x_i, z_j\right)$
      - *Problem*: Policy Gradient has high-variance
      - *Solution*: Unlike RL where we don't know the model, here we can use the *Reparametrisation Trick*!
  - **Reparametrisation Trick**
    - With $q_\phi(z \mid x) = \mathcal{N}\left(\mu_\phi(x), \sigma_\phi(x)\right)$,
      - set $z = \mu_\phi(x) + \epsilon\sigma_\phi(x)$
      - where $\epsilon \sim \mathcal{N}(0, 1)$
    - Now, $$\begin{aligned} J(\phi) &= E_{z\sim q_\phi(z|x_i)}\left[r\left(x_i, z\right)\right] \\ &= E_{\epsilon\sim\mathcal{N}(0,1)}\left[r\left(x_i, \mu_\phi\left(x_i\right) + \epsilon\sigma_\phi\left(x_i\right)\right)\right] \end{aligned}$$
    - Thus to estimate $\nabla_\phi J(\phi)$:
      - sample $\epsilon_1, \ldots, \epsilon_M$ from $\mathcal{N}(0, 1)$ (a single sample works well!)
      - $\nabla_\phi J(\phi) \approx \frac{1}{M}\sum_j \nabla_\phi r\left(x_i, \mu_\phi\left(x_i\right) + \epsilon_j\sigma_\phi\left(x_i\right)\right)$

- **Another way to look at it** (the way to implement it)
  - $$\begin{aligned}\mathcal{L}_i &= E_{z\sim q_\phi(z|x_i)}\left[\log p_\theta\left(x_i \mid z\right) + \log p(z)\right] + \mathcal{H}\left(q_\phi\left(z \mid x_i\right)\right) \\ &= E_{\epsilon\sim\mathcal{N}(0,1)}\left[\log p_\theta\left(x_i \mid \mu_\phi\left(x_i\right) + \epsilon\sigma_\phi\left(x_i\right)\right)\right] - D_{\mathrm{KL}}\left(q_\phi\left(z \mid x_i\right) \| p(z)\right) \\ &\approx \log p_\theta\left(x_i \mid \mu_\phi\left(x_i\right) + \epsilon\sigma_\phi\left(x_i\right)\right) - D_{\mathrm{KL}}\left(q_\phi\left(z \mid x_i\right) \| p(z)\right)\end{aligned}$$
  - Second term is easy:
    - Generally $q_\phi\left(z \mid x_i\right)$ and $p(z)$ have a simple form (eg. Gaussian)
    - and $D_{\mathrm{KL}}\left(q_\phi\left(z \mid x_i\right) \| p(z)\right)$ has a convenient analytical form.
    - So can simply compute its gradient using autodiff.
  - First term is harder and requires the *Reparamtrisation Trick* and is and is generally approximated with a single sample.
- *Architecture*:



- **Reparameterization trick vs. policy gradient**
  - Policy gradient $\nabla J(\phi) \approx \frac{1}{M}\sum_j \nabla_\phi \log q_\phi\left(z_j \mid x_i\right) r\left(x_i, z_j\right)$
    - (+) Can handle both discrete and continuous latent variables
    - (-) High variance, requires multiple samples & small learning rates
  - Reparameterization trick $\nabla_\phi J(\phi) \approx \frac{1}{M}\sum_j \nabla_\phi r\left(x_i, \mu_\phi\left(x_i\right) + \epsilon_j\sigma_\phi\left(x_i\right)\right)$
    - (+) Very simple to implement
    - (+) Low variance
    - (-) Only continuous latent variables

- **Variational Autoencoder**
  -



  - *training*: $\max_{\theta,\phi} \frac{1}{N}\sum_i \log p_\theta\left(x_i \mid \mu_\phi\left(x_i\right) + \epsilon\sigma_\phi\left(x_i\right)\right) - D_{\mathrm{KL}}\left(q_\phi\left(z \mid x_i\right) \| p(z)\right)$
    - First term: Autoencoder objective
    - Second term: Encoder should encode the observations x_i into latent distributions that are similar to the prior ($p(z)$)
  -
  - *sampling*:
    - $$\begin{aligned}z &\sim p(z) \\ x &\sim p(x \mid z)\end{aligned}$$
    - Why does sampling work?
      - $\mathcal{L}_i = E_{z\sim q_\phi(z|x_i)}\left[\log p_\theta\left(x_i \mid z\right)\right] - D_{\mathrm{KL}}\left(q_\phi\left(z \mid x_i\right) \| p(z)\right)$
      - The second term in the objective forces q_ϕ to embed x_i into z with distribution close to the prior $p(z)$ so if we sample $z \sim p(z)$, then $x \sim p(x \mid z)$ should give x that look like the data.