# REINFORCE

David S. Rosenberg

Bloomberg ML EDU

November 25, 2019

# Introduction

# Reinforcement Learning: What's it good for?

- Applications of reinforcement learning:
  - Video game playing
  - Self-driving cars
  - Go, Chess, Checkers, etc.
  - Dialogue systems
- What do these things have in common?
- Agent must decide what to do next, but there isn't always a single right answer.
- We often cannot directly assess performance of a single action, but only a whole sequence of actions.
  - in a game, did we win or lose?
  - did the car crash or successfully reach its destination?
  - did the customer in the dialogue system click "yes – I am satisfied with this AI interaction"

# RL Techniques for Supervised Learning

- More recently, RL is being applied to supervised learning problems.
- With RL, we can handle non-differentiable or black-box loss functions.
- With RL, we eliminate the "exposure bias" problem we get in learning sequence generators
- Perhaps it's also improving results when there are multiple right answers.
- Today we'll discuss a particular algorithm for RL called REINFORCE.

# Contents

# Highly Simplified RL Setup

# Review: Supervised Learning Framework

## The Spaces

- $\mathcal{X}$: input space
- $\mathcal{Y}$: outcome space
- $\mathcal{A}$: action space

## Prediction Function (or "decision function")

A **prediction function** (or **decision function**) gets input $x \in \mathcal{X}$ and produces an action $a \in \mathcal{A}$ :

$$\begin{array}{rccc} f: & \mathcal{X} & \to & \mathcal{A} \\ & x & \mapsto & f(x) \end{array}$$

## Loss Function

A **loss function** evaluates an action in the context of the outcome $y$.

$$\begin{array}{rccc} \ell: & \mathcal{A} \times \mathcal{Y} & \to & \mathbf{R} \\ & (a, y) & \mapsto & \ell(a, y) \end{array}$$

# Review: Risk Minimization

**Definition**

The **risk** of a prediction function $f : \mathcal{X} \to \mathcal{A}$ is

$$R(f) = \mathbb{E}\ell(f(x), y).$$

In words, it's the **expected loss** of $f$ on a new exampe $(x, y)$ drawn randomly from $P_{\mathcal{X} \times \mathcal{Y}}$.

- Ideally, we'd find a **Bayes prediction function** that achieves the *minimal risk* among all possible functions:

$$f^* \in \arg\min_f R(f).$$

# Simplified RL Setup

## The Spaces

- $\mathcal{X}$: input space
- $\mathcal{A}$: action space

## Policy

A **policy** takes input $x \in \mathcal{X}$ and produces a **distribution** on actions $a \in \mathcal{A}$ :

$$\begin{array}{rccc} \pi: & \mathcal{X} & \to & \text{Distributions on } \mathcal{A} \\ & x & \mapsto & \pi(a \mid x) \end{array}$$

## Reward Function

A **reward function** evaluates an action in the context of the input $x$.

$$\begin{array}{rccc} r: & \mathcal{A} \times \mathcal{X} & \to & \mathbf{R} \\ & (a, x) & \mapsto & r(a, x) \end{array}$$

# SGD for CPMs vs REINFORCE

# Conditional Probability Modeling (CPM)

- Input space $\mathcal{X}$
- Label space $\mathcal{Y}$
- Hypothesis space of functions $x \mapsto p(y \mid x; \theta)$
- Parameterized by $\theta \in \Theta$
- For any $\theta$ and $x$, $p(y \mid x; \theta)$ is a distribution on $\mathcal{Y}$.
- (mathematically, no different from a policy)

# Conditional Probability Modeling

- Given training set $\mathcal{D} = ((x_1, y_1), \ldots, (x_n, y_n)$ iid from $P_{\mathcal{X} \times \mathcal{Y}}$.
- Maximum likelihood estimation for dataset

$$\begin{aligned} \theta &\in \arg\max_{\theta \in \Theta} \prod_{i=1}^{n} p(y_i \mid x_i; \theta) \\ \iff \theta &\in \arg\max_{\theta \in \Theta} \sum_{i=1}^{n} \log\left[p(y_i \mid x_i; \theta)\right] \end{aligned}$$

# Gradient Descent Steps

- Let's consider a standard SGD step for observation $(x_i, y_i)$ in a conditional likelihood model.

$$\Delta\theta = \alpha \nabla_\theta \log p(y_i \mid x_i, \theta)$$

  for some learning rate $\alpha > 0$.
- In words: adjust $p(y_i \mid x_i, \theta)$ to put more probability mass on **correct output** $y_i$

# Reinforcement Learning Setting

- In reinforcement learning, we are not provided with the "right answer" $y_i$ during training.
- We get input $x_i$
- We take a random action by sampling $y \sim p(y \mid x_i; \theta)$
- We get reward $r(y, x_i)$.

- We want to adjust $\theta$ to increase the expected rewards we get.

# The REINFORCE Update

- The REINFORCE update is as follows:

$$\Delta\theta = \alpha r(y) \nabla_\theta \log p(y \mid x_i, \theta),$$

  where $y$ is sampled randomly from $p(y \mid x_i, \theta)$, the policy for $x_i$.

- Compare to MLE step:

$$\Delta\theta = \alpha \nabla_\theta \log p(y_i \mid x_i, \theta),$$

  where $y_i$ is the label corresponding $x_i$.

- In maximum likelihood, we're making the correct action more likely.
- In REINFORCE, we're making actions with big rewards relatively more likely than those with small rewards.

# Deriving REINFORCE

# Formalize our Problem Setting

- We assume the following data generating distributions:

$$\begin{aligned} \text{input } x &\sim P_x \\ \text{action } a|x &\sim \pi_\theta(\cdot \mid x) \\ \text{reward } r \mid a, x &\sim P_{r|a,x} \end{aligned}$$

- In general, $P_x$ and $P_{r|a,x}$ are known.
- We know the **policy** $\pi_\theta(\cdot \mid x)$
  - gives action distribution conditioned on input $x$
- We want to find $\theta$ giving a policy that maximizes $J(\theta) = \mathbb{E}_\theta[r]$.

# Work the Objective Function

- Suppose we have a discrete action space:

$$\begin{aligned} J(\theta) &= \mathbb{E}_\theta[r] \\ &= \mathbb{E}^x\left[\mathbb{E}^a_\theta\left[\mathbb{E}[r \mid a, x] \mid x\right]\right] \\ &= \mathbb{E}^x\left[\sum_{a \in \mathcal{A}} \pi_\theta(a \mid x)\mathbb{E}[r \mid a, x]\right] \end{aligned}$$

- And now we take the gradient...

# Gradient of Objective Function

- Clever trick: $\nabla_\theta \pi_\theta(a \mid x) = \pi_\theta(a \mid x) \nabla_\theta \log \pi_\theta(a \mid x)$
- For a given $\theta$, we want to find direction to increase $J(\theta)$:

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \mathbb{E}^x \left[ \sum_{a \in \mathcal{A}} \pi_\theta(a \mid x) \mathbb{E}[r \mid a, x] \right] \\
&= \mathbb{E}^x \left[ \sum_{a \in \mathcal{A}} \nabla_\theta [\pi_\theta(a \mid x)] \mathbb{E}[r \mid a, x] \right] \\
&= \mathbb{E}^x \left[ \sum_{a \in \mathcal{A}} \pi_\theta(a \mid x) \nabla_\theta [\log \pi_\theta(a \mid x)] \mathbb{E}[r \mid a, x] \right] \text{ (clever trick)} \\
&= \mathbb{E}^x [\mathbb{E}_\theta^a (\nabla_\theta [\log \pi_\theta(a \mid x)] \mathbb{E}^r [r \mid a, x])] \text{ (payoff of clever trick)} \\
&= \mathbb{E}_\theta^{x,a} [\mathbb{E}^r [r \nabla_\theta [\log \pi_\theta(a \mid x)] \mid a, x]] \\
&= \mathbb{E}_\theta^{x,a,r} [r \nabla_\theta [\log \pi_\theta(a \mid x)]]
\end{aligned}
$$

# Monte Carlo Approximation to the Gradient

- So we have the gradient w.r.t. the policy:

$$\nabla_\theta J(\theta) = \mathbb{E}_\theta^{x,a,r}\left[r \nabla_\theta \left[\log \pi_\theta(a \mid x)\right]\right].$$

- How do we evaluate this?
- Let's use a Monte Carlo approximation to the gradient:

$$\begin{aligned} \nabla_\theta J(\theta) &= \mathbb{E}_\theta^{x,a,r}\left[r \nabla_\theta \left[\log \pi_\theta(a \mid x)\right]\right] \\ &\approx \frac{1}{N} \sum_{i=1}^{N} r_i \nabla_\theta \left[\log \pi_\theta(a_i \mid x_i)\right] \end{aligned}$$

for $(x_1, a_1, r_1), \ldots, (x_N, a_N, r_N)$ a sample of $N$ rounds with the same policy $\theta$.

# Approximation is Unbiased, but Variance?

- Note that our approximation

$$\frac{1}{N}\sum_{i=1}^{N} r_i \nabla_\theta \left[\log \pi_\theta(a_i \mid x_i)\right]$$

  has expectation

$$\mathbb{E}_\theta^{x,a,r}\left[r\nabla_\theta \left[\log \pi_\theta(a \mid x)\right]\right] = \nabla_\theta J(\theta)$$

- So we have an unbiased estimate of the gradient.
- However, it turns out that it can have "high variance."
  - ("high variance" is in quotes because the gradient is a vector)
- Later we'll apply some tricks to control the variance, which is necessary in practice.

# REINFORCE = Monte Carlo Policy Gradient

## REINFORCE algorithm

1. Initialize policy $\theta = \theta_0$.
2. Repeat:
   1. Play $N$ rounds with policy $\theta$, giving $(x_1, a_1, r_1), \ldots, (x_N, a_N, r_N)$.
   2. Increment $\theta$ by

$$\theta \leftarrow \theta + \alpha \left[ \frac{1}{N} \sum_{i=1}^{N} r_i \nabla_\theta \left[ \log \pi_\theta(a_i \mid x_i) \right] \right]$$

# Reward Baseline

# Subtracting a Baseline from Reward

- Our objective function is
$$J(\theta) = \mathbb{E}_\theta(r).$$
- Suppose we introduce a new reward function $r_0 = r - b$, for constant $b$.
- Then
$$J_0(\theta) = \mathbb{E}_\theta(r_0) = \mathbb{E}_\theta(r) - b.$$
- Obviously, $J(\theta)$ and $J_0(\theta)$ have the same optimal $\theta$.
- But they'll have different optimization paths.
- Can certain $b$ lead to better optimization paths?

# Subtracting a Baseline

- The increment to $\theta$ is

$$\frac{\alpha}{N}\sum_{i=1}^{N} r_i \nabla_\theta \log \pi_\theta(a_i \mid x_i).$$

- Note that each summand $r_i \nabla_\theta [\log \pi_\theta(a_i \mid x_i)]$ is random.
- We will show that subtracting a **baseline** $b_i$ from the reward doesn't change the EV:

$$\begin{aligned}\mathbb{E}[(r_i - b_i)\nabla_\theta[\log \pi_\theta(a_i \mid x_i)]] &= \mathbb{E}[r_i \nabla_\theta \log \pi_\theta(a_i \mid x_i)] - b_i \underbrace{\mathbb{E}[\nabla_\theta \log \pi_\theta(a_i \mid x_i)]}_{=0} = \\ &= \mathbb{E}[r_i \nabla_\theta \log \pi_\theta(a_i \mid x_i)]\end{aligned}$$

# Zero Expectation Step

- Let $p_\theta(a)$ be a distribution on $a$, parameterized by $\theta$.
- Then $\mathbb{E}[\nabla_\theta \log p_\theta(a)] = 0$.
- **Proof:** (for case that $a$ is discrete)

$$\begin{aligned}
\mathbb{E}[\nabla_\theta \log p_\theta(a)] &= \mathbb{E}\left[\frac{\nabla_\theta p_\theta(a)}{p_\theta(a)}\right] \\
&= \sum_{a \in \mathcal{A}} p_\theta(a) \left[\frac{\nabla_\theta p_\theta(a)}{p_\theta(a)}\right] \\
&= \sum_{a \in \mathcal{A}} \nabla_\theta p_\theta(a) \\
&= \nabla_\theta \left[\sum_{a \in \mathcal{A}} p_\theta(a)\right] \\
&= \nabla_\theta [1] = 0
\end{aligned}$$

# Zero Expectation Step

- So

$$\begin{aligned}\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_i \mid x_i)\right] &= \mathbb{E}^{x_i}\left[\mathbb{E}^{a_i}\left[\nabla_\theta \log \pi_\theta(a_i \mid x_i) \mid x_i\right]\right] \\ &= \mathbb{E}^{x_i}[0] = 0.\end{aligned}$$

- This completes the proof that

$$\mathbb{E}\left[(r_i - b_i)\nabla_\theta\left[\log \pi_\theta(a_i \mid x_i)\right]\right] = \mathbb{E}\left[r_i \nabla_\theta \log \pi_\theta(a_i \mid x_i)\right]$$

- So, the expected step is independent of baseline $b_i$.

# What to use for the baseline?

- We're summing random vectors of the form

$$(r_i - b_i)\,\nabla_\theta \left[\log \pi_\theta(a_i \mid x_i)\right].$$

- Each is an unbiased estimate of $\nabla_\theta J(\theta)$.
- But we're told to worry about "high variance."
- But what is the "variance"?
- First, note that this expression is generally a **vector**.
- So there is no scalar "variance" we can just try to optimize.
- So raise your eyebrows if you see a derivation of the $b$ that gives "minimal variance."

# How to choose the baseline $b_i$ in $(r_i - b_i)\nabla_\theta[\log \pi_\theta(a_i \mid x_i)]$?

- Let $g_{ij} = (\nabla_\theta[\log \pi_\theta(a_i \mid x_i)])_j$ be the $j$th component of the gradient.
- If $g_{ij}$ and $r_i$ were independent (which they're NOT), then

$$\begin{aligned}\operatorname{Var}((r_i - b_i)g_{ij}) &= [\mathbb{E}(r_i - b_i)]^2 \operatorname{Var}(g_{ij}) + \Big[\underbrace{\mathbb{E}g_{ij}}_{=0}\Big]^2 \operatorname{Var}(r_i) + \operatorname{Var}(r_i)\operatorname{Var}(g_{ij}) \\ &= [\mathbb{E}r_i - b_i]^2 \operatorname{Var}(g_{ij}) + \operatorname{Var}(r_i)\operatorname{Var}(g_{ij}).\end{aligned}$$

- So choosing $b_i \approx \mathbb{E}r_i$ seems like a good thing to do.
- Can estimate $\mathbb{E}r_i$ e.g. by using historical rewards.

# Input-Dependent Baselines

- What if we generally get lower rewards $r_i$ for some inputs $x_i$ than others?
- Can we have the baseline $b_i$ depend on the input $x_i$?
- Yes!
- You can go back through our argument and change all the expectations to expectations conditional on $x_i$ and you will see that we still get unbiased estimates when we use a function $b_i = b(x_i)$ as a baseline.

# Learning the Baseline

- One can actually try to learn to predict the reward for a given input $x_i$, as a baseline.
- We can learn it at the same time as we learn our policy.
- We could use $b_\phi(x)$ as a baseline, where $\phi$ is learned to minimize $\left(r_i - b_\phi(x_i)\right)^2$.
- This is the approach suggested in Sutton's book.

# Self-Critical Baseline

- Here's another clever way to set a baseline:
- Find (or approximate) the action that is optimal under our policy:

$$a_i^* \approx \arg\max_a \pi_\theta(a|x_i),$$

  and then use the reward $r(a_i^*)$ as a baseline.

- Intuition is that, if the current action performs better than the action our policy says is best, then we should make the current action more likely.
- But if it performs worse than what are policy says is best, let's make it less likely.

- A reasonable idea and seems to performs well in practice (at least for sequence prediction).

# REINFORCE for Sequence Prediction

# Application: Sequence-to-Sequence Models

- Consider machine translation.
- e.g. Conditioned on sentence in English, produce a distribution on sentences in French.
- Model is $\pi_\theta(y \mid x)$, where $x$ is an English sentence and $y$ is a French sentence.
- This is typically trained as a conditional probability model using maximum likelihood.
- As usual, that means finding

$$\theta^* = \arg\min_\theta \pi_\theta(y \mid x).$$

- Seems reasonable...
- But how do we actually measure performance for machine translation?

# Application: Sequence-to-Sequence Models

- Suppose we are assessing performance of our MT model on a test set.
- We get input $x_i$.
- We run beam search with our current model $\pi_\theta(y \mid x)$ and produce a sequence $y'$.
- Suppose $y'$ is a perfect translation of $x_i$, but it's different from the gold sequence $y_i$.
- We'd like to give credit for this translation.
- I don't think there's a great way to do this in an automated way.

# But there is BLEU score

- A frequent measure of translation quality is BLEU score.
- Let's not discuss the details of BLEU score.
- For our purposes, sufficient to know that
  - BLEU takes a proposed translation and a ground truth and gives a numerical score
- BLEU score is computed by an algorithm and is **not differentiable.**
- Perhaps it would make sense to train a model to optimize directly for BLEU score?
- We can use REINFORCE for that.

# Exposure Bias

- Our sequence models are all **autoregressive**.
- We condition on tokens previously predicted tokens to predict the next token.
- During max likelihood training, we're always conditioning on the gold label.
- During test, we're conditioning on a predicted label.
- **Our model never trains using its own predictions as input.**

# Exposure Bias

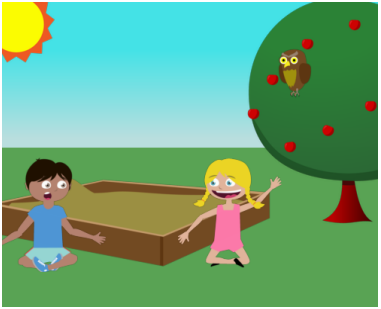
- This is a known issue with maximum likelihood training of sequence models.
- There is a family of approaches called "learning to search" that address this issue.
- e.g. SEARN, DAgger, AggreVaTe, LOLS, etc.
- But RL addresses this approach as well...
- We only condition on previous predictions during training.
- We don't even have the ground truth label to use, except as part of the reward function.

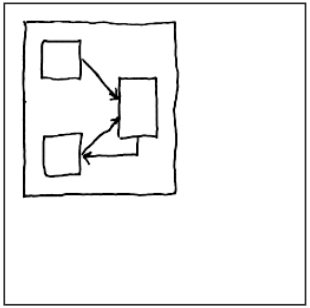
# Usually we pre-train with maximum likelihood

- Suppose we want to train seq2seq with BLEU score as reward.
- Our version of REINFORCE is sufficient for this task.
- We could, for example, use the self-critical baseline.
- In practice, we usually pretrain our model with maximum likelihood, then switch to RL.

# Image to Sequence

# Image to Sequence Problems





```
<object>
  <supercategory>C-1</supercategory>
  <category>CS-3<\category>
  <x-coordinate>120</x-coordinate>
  <y-coordinate>240</y-coordinate>
  <depth>1</depth>
  <flip>0</flip>
</object>
<object>....
```

```
<object>
  <category>Rectangle<\category>
  <x1-coordinate>7</x1-coordinate>
  <y1-coordinate>1</y1-coordinate>
  <x2-coordinate>11</x2-coordinate>
  <y2-coordinate>16</y2-coordinate>
</object>
<object>....
```

# How to Evaluate?

- Re-render for exact match
  - Very challenging metric
  - Doesn't work for hand-drawn shapes
- Two specifications can be very different, yet render to very similar things. (identifiability)
- Two images may look very different (e.g. at the pixel level), but have similar specifications
  - e.g. by changing a color
- We can evaluate performance in image space and in specification space.

# Image Space Measure

- We can measure performance in image space with

$$d_{img} = \|I - \Psi(I^R)\|_2^2,$$

  where $I$ is the original image vector and $I^R$ is the rendering of the predicted image.

- For the noisy shapes dataset, $\Psi$ is a Gaussian blurring function.
- For the abstract scene dataset, $\Psi$ is identity function.

- Why isn't this differentiable?

- Computing $I^R$ uses a graphics renderer...

- Though there are differentiable renderers now... but that's another story.

# Specification Space Measure: IOU Reward

- Our specifications break down into "objects".
- We can look for exact matches between prediction and ground truth at the object level.
- For numeric attributes, we divide range into 20 bins of equal size
  - consider it a match if the bin is correct
- Can summarize matches with precision, recall, F1, etc.
- A common summary in this scenario is **intersection-over-union** (IOU)....

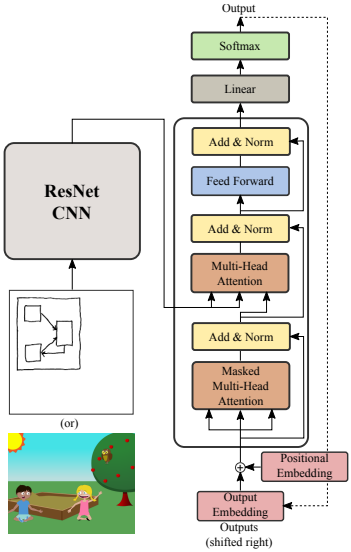
# Intersection over Union

- Let $\{o_i\}_{i=1}^m$ and $\{o_j^*\}_{j=1}^n$ represent the objects in predicted and ground-truth specifications, respectively.
- Then the IOU reward is defined as follows:

$$r_{iou} = \frac{\text{count}(\{o_i\}_{i=1}^m \cap \{o_j^*\}_{j=1}^n)}{\text{count}(\{o_i\}_{i=1}^m \cup \{o_j^*\}_{j=1}^n)}$$

- Roughly speaking, IOU gives credit for predicting objects that exactly match objects in the ground truth
- Penalizes both for predicting objects that do not match ground truth objects and for failing to predict objects that are part of the ground truth.

# Our Model: ResNet to Transformer Decoder



# Results: Cross-Entropy Loss (i.e. Maximum Likelihood)

| Model | Recons. Error | IOU |
|---|---|---|
| Cross-Entropy Loss | | |
| Image2LSTM+atten. | 15.70 | 32.06 |
| Image2Transformer | 10.92 | 58.54 |

- reconstruction error corresponds to the image distance
- average error

# Results: Reinforcement Learning

| Model | Recons. Error | IOU |
|---|---|---|
| Cross-Entropy Loss | | |
| Image2LSTM+atten. | 15.70 | 32.06 |
| Image2Transformer | 10.92 | 58.54 |
| Image2Transformer with Reinforce Loss | | |
| IOU Reward | 10.50 | 61.29 |
| Recons. Reward | **9.99** | 62.44 |
| IOU + Recons. | 10.04 | **62.45** |

# Full Reinforcement Learning Setting

# Markov Decision Processes (Sutton Chapter 3)

- Learner / decision maker is called the **agent**
- Agent interacts with the **environment**
- Each time step $t = 0, 1, 2, 3, \ldots$,
  - agent receives a state $s_t \in \mathcal{S}$.
  - agent selects an action $a_t \in \mathcal{A}$
  - agent receives a numerical reward $r_{t+1} \in \mathbf{R}$
- We get a **trajectory**: $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, \ldots$

# MDPs, continued

- The **dynamics** of the MDP are given by the conditional probability distribution:

$$p(s_t, r_t \mid s_{t-1}, a_{t-1})$$

- Gives distribution of reward and next state given previous state and action.
- This conditional distribution completely characterizes the MDP.
- The dynamics describe how the world evolves and reacts to our actions.
- Says nothing about what our actions are.

# Episodic Learning

- Often problem breaks up into "**episodes**" or "**trials**".
- Sometimes we get a single reward at the end of each episode
  - as in sequence prediction
- But now we'll consider the general case.
- For an episode there is a final time step $T$
  - need not be the same in every episode
  - it's typically random

# REINFORCE for this setting

- Define the **reward to go** as rewards received after action $a_t$:

$$g_t = \sum_{i=t+1}^{T} r_i.$$

- Then gradient step for this setting is approximated by

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[ \sum_{t=1}^{T} g_t \nabla \log \left[ \pi_\theta(a_{i,t} \mid s_{i,t}) \right] \right].$$

- If we only get reward at end of episode, then $g_1 = \cdots = g_T = r$.
- Reduces to our case earlier.

# What's the impact?

- Compare

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T}\left(\sum_{j=t+1}^{T} r_{ij}\right)\nabla \log\left[\pi_\theta(a_{i,t} \mid s_{i,t})\right]\right]$$

- to

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left[\sum_{t=1}^{T}\left(\sum_{j=1}^{T} r_{ij}\right)\nabla \log\left[\pi_\theta(a_{i,t} \mid s_{i,t})\right]\right].$$

- Note that the reward to go $\left(\sum_{j=t+1}^{T} r_{ij}\right)$ will typically be smaller than the full reward.
- Thus we can view this variation as a variation reduction technique!
- This is the form derived in Sutton and Barto's Chapter 13.