Algorithmic Foundations of Interactive Learning

Spring 2025

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Instructors
Zhiwei Steven Wu, J. Andrew Bagnell, Gokul Swamy

This is a compilation of lightly edited scribe notes which has not been subjected to the higher standard required for a publication. It therefore likely contains many errors.

Table of Contents

Part I

Online Learning

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 2: Intro to Online Learning / Weighted Majority

Lecturer: Drew Bagnell
Scribe: Gokul Swamy

2.1 The Problem of Induction

Machine learning (and perhaps all of science) can be viewed as attempting to solve the problem of *induction*: making predictions about the future conditioned on the past.

$$\begin{tikzcd}
\text{Prior} \arrow[r] & \text{Inference} \arrow[r] & \text{Prediction} \\
\text{Data} \arrow[ur] & &
\end{tikzcd}$$

Figure 2.1: We can view induction as the problem of turning what we know into a prediction about something we don't know. What assumptions are required for this to be possible?

Throughout the centuries, there have been a variety of views of the problem of induction:

- **Laplace:** If the sun rises for 1000 straight days, we should conclude that it has a high chance of rising on the 1001st day.
- **Hume (responding to the above):** If a chicken has been fed for 1000 days, should it conclude it won't become dinner tomorrow?

In this course, we will take the *no-regret view* (which grew out of the late 20th century): we will make no assumptions about how well the past predicts the future and attempt to do as well anyone could hope to. Note the contrast between this and the *statistical* view in standard machine learning, where we assume we will be tested on the same distribution our training data was drawn from. As we will explore later in the course, one of the key challenges of interactive learning is being able to predict accurately even under distribution shift, and the no-regret view will give us a rigorous theoretical foundation for doing so.

2.2 Prediction with Expert Advice

For this lecture, we will focus primarily on the setting of *prediction with expert advice.* Specifically, consider predicting a sequence of binary labels for T rounds, i.e. $p_t \in \{0, 1\}$. We will assume access to a set of experts $\mathcal{E}$ with $|\mathcal{E}| = N$ to help us with this task, e.g.:

- e_0: always predicts 0
- e_1: always predicts 1
- e_2: predict what happened at the last round
- e_3 predicts $0, 1, 0, 1, \dots$

We will assume there is some ground truth sequence of labels $y = (y_1, \dots, y_T)$ that is not necessarily chosen in advance (i.e. it can be chosen adversarially in response to what our algorithm predicts). We will judge algorithms based on the sum of per-round losses, where

$$\ell_t(p_t, y_t) = \mathbb{1}[p_t \neq y_t]. \tag{2.1}$$

2.2.1 Regret

Without any assumptions, this problem is quite hard. We will instead attempt to compete with the best *fixed* expert in hindsight. Specifically, we define regret as follows:

Definition 1 ((Static) Regret)

$$Regret(T) = \sum_t^T \ell_t(p_t) - \ell_t(e_t^\star) = \min_{e \in \mathcal{E}} \sum_t^T \ell_t(p_t) - \ell_t(e_t). \tag{2.2}$$

We say an algorithm achieves no-regret if it drives the time-averaged regret to zero, i.e. if

$$\lim_{T \to \infty} \frac{Regret(T)}{T} \to 0. \tag{2.3}$$

This may still seem like a lot to ask for (we haven't made any assumptions on how the y_t's are picked!). However, as we will spend the next few lectures exploring, there are a plethora of algorithms that satisfy the no-regret property, some of which you are probably already familiar with (e.g. gradient descent).

2.2.2 Follow-The-Leader (FTL)

We will first present what might be the most approach to online learning and then argue why it doesn't quite achieve no-regret. Let us define follow-the leader as the algorithm that picks the best expert in hindsight, i.e.

Definition 2 (Follow-The-Leader (FTL)) *Follow-The-Leader selects*

$$p_t = \arg\min_{e \in \mathcal{E}} \sum_{\tau}^{t-1} \ell_t(e). \tag{2.4}$$

For a moment, let us set $\mathcal{E} = \{e_0, e_1\}$ and tie-break in favor of e_0. Let us run this algorithm against an unforgiving world:

e_0 Correct	e_1 Correct	p_t	y_t
0	0	0	1
0	1	1	0
1	1	0	1

Figure 2.2: If an adversary were to choose y_t in a way to cause us maximal pain, they could cause us to suffer linear regret (i.e. be wrong at every timestep even though no individual expert is).

As we will discuss in future lectures, the core issue with FTL is that it is unstable – it switches its predictions too easily, which makes it possible for an adversary to take advantage of it.

2.2.3 Halving Algorithm

For simplicity, we're now going to assume there is a perfect expert, i.e. $\exists e^{\star} \in \mathcal{E}$ s.t. $\sum_t^T \ell_t(e_t^{\star}) = 0$. Let's now try a smarter approach:

Definition 3 (Halving Algorithm) *The Halving Algorithm maintains a set of experts $\mathcal{E}_t$ at each round. $\mathcal{E}_0 = \mathcal{E}$. If an expert ever makes a mistake, it is removed from the set. At each round, p_t is set to the majority vote across $\mathcal{E}_t$.*

Observe that the total number of mistakes (i.e. total regret) the above algorithm can suffer is $\log_2(N)$. This is because if we ever make a mistake, we eliminate half the experts in the last round's $\mathcal{E}_t$ and we can only do this process $\log_2(N)$ times. This is sometimes referred to as a *mistake bound*. Observe that this strategy is effectively weighting an expert by zero if they ever make a mistake.

2.2.4 Deterministic Weighted Majority

If we want to handle the case where no expert may be perfect, we need to adopt a less aggressive weighting scheme:

Definition 4 (Deterministic Weighted Majority (DWM)) *DWM maintains a weight w_i for each $e \in \mathcal{E}$. It predicts*

$$p_t = \mathbb{1}\left[\sum_{i:e_i(t)=1} w_i^t \geq \sum_{i:e_i(t)=0} w_i^t\right] \tag{2.5}$$

(i.e. takes a weighted majority vote). Whenever an expert is wrong, it sets $w_i^{t+1} = 0.5w_i^t$.

We remark that this recovers the halving algorithm if we change the 0.5 to 0 and a standard majority vote if we instead change to 1.

We now prove a regret bound for this algorithm:

Lemma 1 (DWM Regret Bound) *Let m denote the number of mistakes DWM makes and $m^\star$ denote the number of mistakes the best expert makes. Then, we have*

$$m \leq \log_2\left(\frac{4}{3}\right)(m^\star + \log_2(N)). \tag{2.6}$$

Proof: Observe that for DWM to make a mistake, half of the experts (in terms of weight) need to be wrong and after such a mistake, this half looses half of their weight. This means that every time the algorithm makes a mistake, the full set of experts loose at least $\frac{1}{4}$ of their total weight W. This directly implies that

$$N\left(\frac{3}{4}\right)^m \geq W \geq \left(\frac{1}{2}\right)^{m^\star}. \tag{2.7}$$

Re-arranging terms gives the regret bound. ■

We remark that the above regret bound is tight via a variant of the switching construction from 2.2.2.

Unfortunately, $\log_2\left(\frac{4}{3}\right) \approx 2.41 > 1$, so the above algorithm doesn't quite achieve no-regret.

2.2.5 Randomized Weighted Majority

To actually achieve no-regret, we will now switch to predicting a *distribution* over experts, and outputting what an expert sampled from that distribution said. While it might not be immediately obvious why this is a good idea, in later lectures where we discuss game solving, the reasoning behind this shift will become more transparent.

Definition 5 (Randomized Weighted Majority (RWM)) *RWM initializes all weights to 1, i.e.* $w_i^0 = 1$*. Define distribution* $p_t(i) = w_t(i)/\sum_j^N w_t(j)$*. RWM samples from this distribution and predicts what the sampled expert predicts. If an expert is wrong, it sets* $w_i^{t+1} \leftarrow \beta \cdot w_i^t$*.*

We won't prove this in this lecture, but we can bound the *expected* regret as

$$\mathbb{E}[m] \leq \frac{m^\star \ln(1/\beta) + \ln N}{\beta}. \tag{2.8}$$

For an appropriate choice of β, this algorithm will achieve no-regret. Observe that because we're now shelling out to an expert, we no longer need to restrict ourselves to the binary prediction setting.

2.2.6 Generalized Weighted Majority

For our final algorithm of this series, we will study Generalized Weighted Majority, for which we will assume $\ell_t \in [0, 1]$:

Definition 6 (Generalized Weighted Majority (GWM)) *GWM is identical to RWM but it sets*

$$w_i^{t+1} \leftarrow w_i^y \cdot \exp(-\varepsilon \ell_t(e_i)), \tag{2.9}$$

where ϵ *can be thought of as a kind of learning rate.*

Again, without proof, we will state that

$$\mathbb{E}[Regret(T)] \leq \varepsilon \sum_t^T \ell_t(e_t^\star) + \frac{\ln(N)}{\varepsilon} \leq T\varepsilon + \frac{\ln(N)}{\varepsilon}, \tag{2.10}$$

where the second inequality comes from the scale of the loss. If we take the gradient of the above w.r.t. ε and set it equal to zero, we can find that the optimal learning rate is $O\left(\frac{1}{\sqrt{T}}\right)$. Plugging in this $\varepsilon^\star$ gives us a final regret bound of

$$\mathbb{E}[Regret(T)] \leq O(\sqrt{T}) + O(\sqrt{T}) \ln(N). \tag{2.11}$$

Because all terms are sub-linear in T, we have proved the above algorithm achieves no-regret. Huzzah! We remark that without further assumptions, the $\sqrt{T}$ and $\ln N$ scaling are unavoidable up to constants.

2.2.7 Cover's Universal Portfolio

We now present an example to perhaps temper the excitement the above example may have evoked in the mind of an avaricious reader. What if, for a moment, we consider each expert a stock in the stock market, and our problem to invest optimally (i.e. compete with the best stock in hindsight). Why does the no-regret property not imply one of the preceding algorithms could do this trivially? Observe that all the no-regret property says is that

$$\frac{1}{T}\sum_{t}^{T} \ell_t(p_t) - \ell_t(e^\star) \to 0, \tag{2.12}$$

i.e. for the losses ℓ_t we *actually observed*, the regret goes to zero. It does not imply that for any counter-factual set of losses we could have observed we would be able to drive regret to zero. The reason this causes issues in stock trading is that if one puts enough money into a particular stock, they change the value of that stock (i.e. the sequence of p_t we choose influences the set of ℓ_t we observe). Thus, while the no-regret property implies we do well *conditioned on our choices*, it does not guarantee we do as well as any investor could have done in the market (as they may induce a different set of ℓ_t with their choices of p_t).

2.3 Convexity

We now define two concepts we will use repeatedly throughout the course.

Definition 7 (Convex Set) *A set $\mathcal{X}$ is convex if, $\forall a, b \in \mathcal{X}$ and $\forall \alpha \in [0,1]$,*

$$\alpha a + (1 - \alpha b) \in \mathcal{X}. \tag{2.13}$$

In words, this definition is saying that there is a straight line to every other point in the set that is contained in the set.

Definition 8 (Convex Function) *A function f is convex if, $\forall x, y \in dom(f)$ and $\forall \alpha \in [0,1]$,*

$$f(\alpha x + (1-\alpha)y) \geq \alpha f(x) + (1-\alpha)f(y) \tag{2.14}$$

Equivalently, a function is convex if its epigraph (the set of points above the function) is a convex set.

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 3: Information Theory and Maximum Entropy

Lecturer: Drew Bagnell
Scribe: Zora Wang, Lindia Tjuatja

3.1 Recap: Regret

Recall that the *regret* of some algorithm is the difference in loss l between the algorithm and the best expert $e^\star$ over all timesteps $t \in [T]$:

$$\text{Regret(T)} = \sum_{t}^{T} \ell_t(\text{alg}_t) - \ell_t(e^\star) \tag{3.15}$$

As we saw in the previous lecture, there exist algorithms (such as weighted majority) where the regret is sub-linear in T. We say an algorithm is *no-regret* when

$$\frac{\text{Regret}(T)}{T} \to 0. \tag{3.16}$$

For example, for weighted majority, $\text{Regret}(T) \leq \mathcal{O}(\log(N)\sqrt{T})$, where N is the number of experts we are selecting between.

3.2 Application: Linear Classifiers

We now explore how we can apply the idea of *learning from expert advice* to the standard machine learning problem of learning a linear classifier. Intuitively, we will partition the parameter space into a set of "experts", and attempt to compete with the best of these experts (i.e., the best classifier).

Suppose we have a linear classifier with parameters $\theta \in \Theta \subset \mathbb{R}^d$ with some input $x \in \mathbb{R}^d$:

$$f_\theta(x) = \begin{cases} \theta^T x \geq 0 : \text{Output } 1 \\ \theta^T x < 0 : \text{Output } -1 \end{cases} \tag{3.17}$$

As mentioned above, one way can frame this problem as learning from expert advice by partitioning Θ into a set of experts and running, say, multiplicative weights to pick a classifier.

Observe that from the definition of f_θ, only the direction and not the scale of θ matters. Thus, we can set Θ to be the unit sphere in d dimensions without loss of generality. If we split each of the d dimensions into b points, we get $\mathcal{O}(b^d)$ experts, roughly speaking. Recalling that the regret of WM scales logarithmically with N, it immediately follows that we pay linearly in the dimension, d. While that is nice, note that we need to do an *exponential* amount of work per iteration of WM because the number of experts still scales exponentially in d, making this a rather computationally inefficient approach.

3.3 Convexity

We now explore a few definitions related to convexity.

Definition 9 (Convex Function) *A function f is convex if, $\forall \alpha_1, \alpha_2 > 0$ s.t. $\alpha_1 + \alpha_2 = 1$ and $x_1, x_2 \in dom(f)$, we have:*

$$\sum_i^2 \alpha_i f(x_i) \geq f\left(\sum_i^2 \alpha_i x_i\right). \tag{3.18}$$

In words, this definition is saying that the line connecting two points on a convex function is on or above the function. Note that this also implies that all linear functions are convex – strong convexity excludes lines. We can generalize this observation via induction: given any 3 points on a convex function $(x_1, f(x_1)), (x_2, f(x_2)), (x_3, f(x_3))$, we can trace out a triangle by taking different linear combinations of the weights which lies above the function:

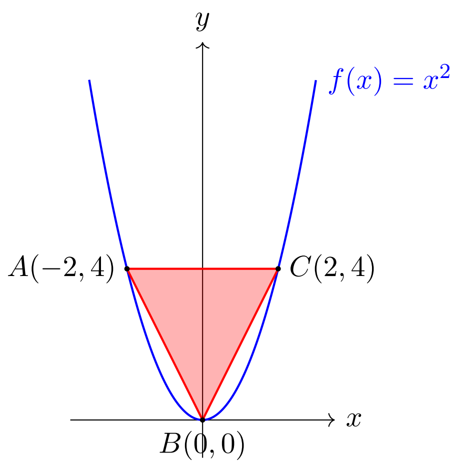


Figure 3.3: We can trace out the triangle in red with different linear combinations of points A, B, C (i.e. $\alpha_1, \alpha_2, \alpha_3 \geq 0$ with $\sum_i^3 \alpha_i = 1$. Convexity means this triangle lies above the function (other than at the 3 intersecting points).

The more general version of this above observation for an arbitrary number of points is known as Jensen's Inequality:

Lemma 2 *For a convex function f and probability distribution $p \in \Delta(dom(f))$, we have*

$$\mathbb{E}_p[f(x)] \geq f(E_p[f(x)]). \tag{3.19}$$

In words, the average value of a convex function lies at or above the evaluation of the function at the average input. [1]

3.4 Information Theory

Information theory is a theory of compressing, communicating, and quantifying information, like a message. We now outline some basic quantities in information theory.

3.4.1 Entropy, Cross-Entropy, and KL-Divergence

Consider a random variable X with sample space $\{a, b, c, d\}$. Let's say $P(X = a) = \frac{1}{2}$, $P(X = b) = \frac{1}{4}$, $P(X = c) = \frac{1}{8}$, and $P(X = d) = \frac{1}{8}$. Let's try to encode samples from

[1]If you forget which way Jensen's Inequality goes, a convenient heuristic is that it is the reverse of whatever you need to complete the proof.

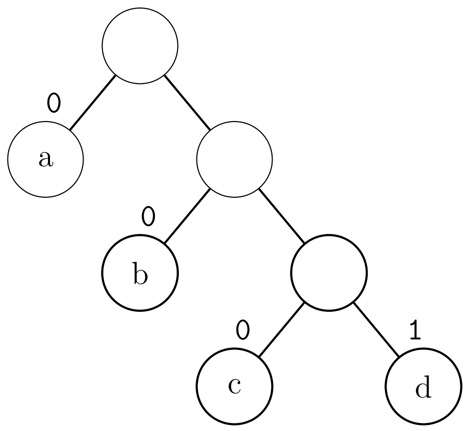


Figure 3.4: Prefix tree representation of optimal code of $P(X)$.

$P(X)$ in binary. Specifically, our goal will be to design a code (i.e. a mapping from the sample space to binary strings) to minimize the expected number of bits required to encode a sample from $P(X)$ is minimized.

Table 3.1 shows an optimal code for X. As one might expect, more probable values of X have shorter representations. Furthermore, this code is *instantaneously decodable*, i.e. we can, by reading the bits in sequence, tell when a character has completed without the need to insert explicit delimiters. This is why we don't use, say, 1 as the encoding of any value. This also means that we can write the code in a tree-structured manner, and is why it is sometimes referred to as a *prefix code*. Also observe that we set the length of the encoding of some member of our alphabet to be $\log_2(\frac{1}{P(X=\cdot)})$ – this value is sometimes refered to as the *information content* or *surprisal* of the outcome.

Table 3.1: Optimal prefix code for $P(X)$

Value	**Prob.**	**Code**	**Length**
a	1/2	`0`	1
b	1/4	`10`	2
c	1/8	`110`	3
d	1/8	`111`	3

To calculate the expected number of bits required to encode a sample using this optimal scheme, we can simply weight the information content of each out come by the probability of its occurrence. This value is known as *(Shannon) entropy*:

Definition 10 *Given a probability distribution $p \in \Delta(\mathcal{X})$, we can define the (Shannon)* *entropy of p as*

$$\mathbb{H}(p) = \mathbb{E}_{x \sim p}[-\log_2(p(x)]. \tag{3.20}$$

Observe that entropy does not depend on permutations of the alphabet, nor the dimension of the points in the distribution. The *maximum entropy* distribution (i.e., the most uncertain) is the uniform distribution. Thus, for discrete outcome spaces, $\mathbb{H}(p) \leq \log_2(N)$, where N is the number of possible outcomes. We visualize the special case of a Bernoulli random variable below.

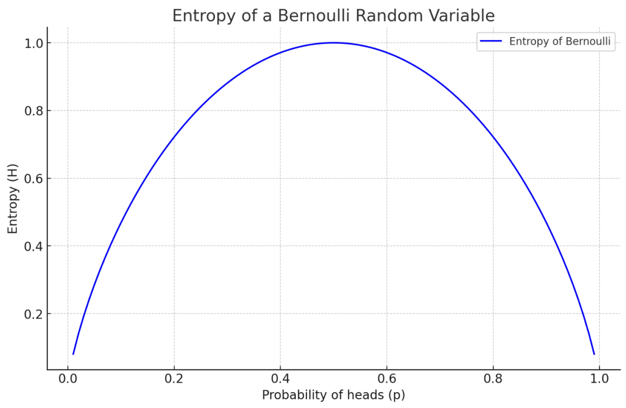


Figure 3.5: We plot the entropy of a $\textbf{Bern}(p)$ random variable of as a function of p. Observe that entropy is maximized at $p = 0.5$ (i.e., a fair coin) with value $\log_2 2 = 1$.

Above, we coded $P(X)$ according to $P(X)$. What if we instead code some $Q(X)$ according to $P(X)$? We can write the expected length of our code by changing the distribution we're taking the expectation with respect to, a quantity known as *cross entropy*:

Definition 11 *Given distributions $P, Q \in \Delta(\mathcal{X})$, we can define the cross entropy from Q to P as*

$$\mathbb{H}(Q||P) = \mathbb{E}_{x \sim Q}[\,-\log_2 P(x)]. \tag{3.21}$$

Clearly, this is minimized when $P = Q$, recovering entropy.

The amount of suboptimality incurred by encoding with the wrong probability distribution (i.e., the difference between entropy and cross-entropy) is known as the *KL Divergence*:

Definition 12 *Given distributions $P, Q \in \Delta(\mathcal{X})$, we can define the cross entropy from Q to P as*

$$\mathbb{D}_{KL}(Q||P) = \mathbb{E}_{x \sim Q}\left[\log_2 \frac{Q(x)}{P(x)}\right] = \mathbb{H}(P||Q) - \mathbb{H}(Q). \tag{3.22}$$

Intuitively, this is a measure of how many extra bits we need to pay for picking the wrong encoding scheme (P instead of Q). The KL divergence is minimized to zero only $P = Q$ (and only when this is true). This is sometimes known as Gibbs' Inequality:

Lemma 3 *Given any distributions $P, Q \in \Delta(\mathcal{X})$, $\mathbb{D}_{KL}(P||Q) \geq 0$.*

Proof: We proceed via Jensen's:

$$-\mathbb{D}_{\text{KL}}(P||Q) = -\sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \tag{3.23}$$

$$\leq \log\left(\sum_{x \in \mathcal{X}} P(x) \frac{Q(x)}{P(x)}\right) \tag{3.24}$$

$$= \log\left(\sum_{x \in \mathcal{X}} Q(x)\right) = \log(1) = 0. \tag{3.25}$$

Thus, $\mathbb{D}_{\text{KL}}(P||Q) \geq 0$. ■

We can view learning a probability distribution as attempting to maximally compress some set of samples. More formally, we can view *maximum likelihood estimation* (MLE) as minimizing the KL Divergence from reality to our model. In this sense, MLE is an information-theoretic algorithm.

3.5 The Principle of Maximum Entropy

We now consider a problem that, while perhaps seeming somewhat abstract at first, will underlie many of the concepts we will explore in this course: *give partial information about some distribution (e.g. its mean), what is the right choice of distribution to pick out of all that match the known information?* The **Principle of Maximum Entropy** says to pick the highest entropy distribution that satisfies the given set of constraints. As we will discuss more, one justification for MaxEnt is *minimax optimality*: if an adversary were to pick another distribution that satisfies the given constraint, we'd suffer the least pain the worst case by picking the MaxEnt distribution. In this sense, we get a form of robustness.

Example 1: 8-Sided Die. Consider rolling a die with 8 faces. What is the optimal coding scheme for the outcome?

MaxEnt says to pick the distribution that maximizes entropy subject to the given information. Given we have no information, the MaxEnt distribution would be the uniform distribution, which gives us an expected code length / entropy of 3. We won't prove this in class but, if an adversary were to view our code and then pick a true distribution that matched the given information, MaxEnt would prevent us from having a high KL divergence. Intuitively, we'll never put 0 probability mass on some outcome, or the adversary could make us output a code of infinite length by putting any weight on it.

Ok, that was a fairly boring example. Let's add in some known information. Using f to denote the number of dots on the top face, $\mathbb{E}_p[f]$ tells us how many dots we should expect, on average.

Example 2: Biased 8-sided Die. Let's say we have the same setting as above, but we now know $\mathbb{E}_p[f(x)] = 4.5$. In other words, the die is biased. The MaxEnt principle says we should chose

$$p^\star = \arg\max_{p \in \Delta(\mathcal{X})} \mathbb{H}(p) \tag{3.26}$$

$$\text{s.t. } \mathbb{E}_p[f(x)] = 4.5, \tag{3.27}$$

where $\Delta(\mathcal{X})$ is the probability simplex (i.e., the set of probability distributions) over $\mathcal{X}$. Given $\Delta(\mathcal{X})$ is a convex set, $\mathbb{H}$ is a convex function, and expectation / moment constraints are linear, this is a *convex optimization problem* (i.e., there is a unique optimal solution with an efficient algorithm to find it – the best you can hope for in machine learning!). We note in passing that maximizing entropy is equivalent to minimizing the KL divergence to the uniform distribution. One can naturally substitute this uniform distribution for a better informed prior, leading to the *principle of minimum cross-entropy* (MinRelEnt for short). We will revisit this point when we discuss RLHF.

We will explore how to solve this problem via forming the Lagrangian next time!

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 4: Maximum Entropy & FTRL

Lecturer: Drew Bagnell & Steven Wu
Scribe: Anupam Nayak & Steven Man

4.1 Recap: Entropy maximization under expectation constraint

We revisit the problem of maximizing entropy under a constraint, as discussed in the last class. The goal is to determine the probability distribution p_i for $i \in \{1, 2, \dots, 6\}$ representing a biased die. The distribution must satisfy the expectation constraint that the expected value of the top face is 4.5 while maximizing the entropy. The mathematical formulation involves finding p_i that maximizes the entropy:

$$\begin{aligned}
\text{Maximize:} \quad & H(\mathbf{p}) = -\sum_{i=1}^{6} p_i \log p_i, \\
\text{Subject to:} \quad & \sum_{i=1}^{6} p_i = 1 \quad \text{(normalization constraint)}, \\
& \mathbb{E}[f] = c \quad \text{(expectation constraint)}.
\end{aligned}$$

Where $f(i) = i$ is the value on the top face. In order to solve the problem we use the method of Lagrangian multipliers.

$$\mathcal{L}(\mathbf{p}, \gamma, \lambda) = \min_{\gamma, \lambda} \max_{\mathbf{p}} H(\mathbf{p}) - \gamma \left(\sum_{i=1}^{6} p_i - 1 \right) - \lambda \left(\mathbb{E}_{\mathbf{p}}[f] - c \right)$$

Setting $\frac{\partial \mathcal{L}}{\partial p_i} = -\gamma - \lambda f(i) - \log(p_i) - 1 = 0$ we obtain

$$p_i = \frac{e^{-\lambda f(i)}}{e^{\gamma'}} = \frac{e^{-\lambda f(i)}}{Z(\lambda)}$$

where $\gamma' = \gamma + 1 = \log\left(\sum_{i=1}^{6} e^{-\lambda f(i)}\right) = \log Z(\lambda)$. Now we are left with a variable λ. Note that setting $\lambda = 0$ we get the uniform distribution and setting $\lambda = -\infty/\infty$ we get a distribution that has all the probability mass concentrated at $i = 6/1$ respectively. The function

$Z(\lambda)$ is usually called the partition function and λ is called the temperature. Substituting p_i back into the Lagrangian we obtain the expression

$$
\begin{aligned}
&- \mathbb{E}_{\mathbf{p}}[-\lambda f(i) - \log Z(\lambda)] - \lambda\left(\mathbb{E}_{\mathbf{p}}[f] - c\right) && (4.28)\\
&= \mathbb{E}_{\mathbf{p}}[\log Z(\lambda) + \lambda c] && (4.29)
\end{aligned}
$$

Taking the gradient of the above expression wrt λ we obtain

$$
\frac{-\sum f e^{-\lambda f}}{Z(\lambda)} + c = -\mathbb{E}_{\mathbf{p}}[f] + c \tag{4.30}
$$

This is also called the Max-Ent gradient. The given distribution $p_i = \frac{e^{-\lambda f(i)}}{Z(\lambda)}$ is a member of the exponential family because it can be written in the general form $p(x|\theta) = h(x)\exp(\eta(\theta)\cdot T(x) - A(\theta))$, where $h(x) = 1$, $\eta = -\lambda$, $T(i) = f(i)$, and $A(\theta) = \log Z(\lambda)$[1].

To obtain the value of λ set the Max-Ent gradient to 0 and solve for $\mathbb{E}_{\mathbf{p}}[f] = c$. One can also note the fact that the expected value $\mathbf{E}_{\mathbf{p}}[f]$ is a monotonically decreasing function of λ which allows the use of root finding techniques for efficiently computing λ.

The principle is also widely used in statistical physics. For example, the probability of finding a molecule at height h in the Earth's atmosphere is given by $p(h) = \frac{e^{-\lambda mgh}}{Z}$, where $\lambda = 1/kT$, m is the mass of the molecule, g is the gravitational acceleration, and Z is the partition function. The expectation value of the potential energy, $\mathbb{E}[mgh]$, is constrained to a constant C, which reflects equilibrium conditions.

For a 1D ideal gas, the mean velocity is $\mathbb{E}[v] = 0$, assuming a symmetric velocity distribution. The mean kinetic energy is $\mathbb{E}[v^2] = \frac{2C}{m}$, where C relates to the temperature or average energy. The velocity distribution follows the Maxwell-Boltzmann form: $p(v) = \frac{e^{-\lambda \frac{1}{2} m v^2}}{Z}$, derived using the maximum entropy principle under the constraint of fixed mean energy.

4.2 Online Learning with Expert Advice

4.2.1 Setup

- N experts: $i = 1, \ldots, N$
- For round $t = 1, \ldots, T$:
    1. Algorithm chooses $\mathbf{p}^t = (p_1^t, \ldots, p_N^t)$
    2. Adversary chooses $\mathbf{l}^t = (l_1^t, \ldots, l_N^t)$ after observing $\mathbf{p}^t$

3. Algorithm incurs loss: $\langle \mathbf{p}^t, \mathbf{l}^t \rangle = \sum_{i=1}^N p_i^t l_i^t$
   (We will sometimes write $\ell^t(\mathbf{p^t})$ to denote the incurred loss.)

Assumption 4 (Bounded losses) *The losses l_i^t are upper bounded for all $i \in [N], t \in [T]$*

Follow-the-Leader (FTL):

A natural choice of algorithm here would be the Follow-the-Leader (FTL) algorithm which chooses the best performing expert based on losses observed until time t.

$$\mathbf{p}^t = \arg\min_{\mathbf{p}} \sum_{\tau=1}^{t-1} \ell^\tau(\mathbf{p}).$$

This strategy effectively reduces the problem to selecting a single expert, which can lead to poor performance in adversarial setups. Specifically, the adversary can design a sequence of losses that results in constant regret for the algorithm. Consider the two expert case if the adversary chooses a sequence of losses $[1,0],[0,1],[0,1],[1,0],[1,0],[0,1],[0,1],[1,0],[1,0]\cdots$ alternating between the sequences, FTL switch periodically between experts and will incur a loss that scales linearly in T.

4.2.2 Weighted Majority / Multiplicative Weights

The weighted majority algorithm, also referred to as the multiplicative weights algorithm, is a framework for decision-making with experts. At the start, the algorithm initializes the probability distribution $\mathbf{p}^1$, typically as a uniform distribution across all experts. Over the course of rounds $t = 1, 2, \ldots$, the probabilities are updated iteratively using the rule:

$$p_i^{t+1} \propto p_i^t(1 - \eta l_i^t),$$

where $\eta > 0$ is a learning rate, and l_i^t denotes the loss incurred by expert i during round t. This rule reduces the weight of experts that perform poorly while maintaining higher weights for better-performing ones. Recall that the halving algorithm discussed in lecture 2 is a special case of this algorithm under the assumption that there exists a perfect expert. Any expert that incurs a loss has its weight multiplied by 0 at each time step. $p_i^{t+1} \propto p_i^t(1 - \eta l_i^t)$, can be seen as a first-order Taylor series approximation of the standard exponential weighting update rule:

$$p_i^{t+1} \propto p_i^t \exp(-\eta l_i^t).$$

To address the issue of linear regret in FTL, we introduce another algorithm "Follow the Regularized Leader" method. This approach incorporates entropy regularization into the objective as a result of which we get a multiplicative weights update, which discourages overly confident (highly skewed) distributions and promotes diversity. The updated objective function becomes:

$$\mathbf{p}^t = \arg\min_{\mathbf{p}} \sum_{\tau=1}^{t-1} \langle \mathbf{p}, \mathbf{l}^\tau \rangle - \frac{1}{\eta} H(\mathbf{p}),$$

where $H(\mathbf{p}) = -\sum_i p_i \log p_i$ is the entropy term. This regularization ensures that the algorithm maintains a spread across multiple experts, mitigating the effects of adversarial losses. Consequently, the final update rule for the weights is given by: $p_i^t \propto \exp\left(-\eta \sum_{\tau=1}^{t-1} l_i^\tau\right)$. This formulation balances exploration (assigning nonzero probability to all experts) and exploitation (favoring experts with lower cumulative losses), achieving robustness and low regret across rounds. The optimization oracle is defined as:

$$\mathcal{O}(\ell^{1:t}) = \arg\min_{\mathbf{p}} \sum_{\tau=1}^{t} \ell^\tau(\mathbf{p}),$$

where $\ell^\tau(\mathbf{p})$ represents the loss function at time τ for a decision vector $\mathbf{p}$.

4.2.3 Analysis

To analyze the Follow-The-Regularized-Leader (FTRL) algorithm, we first introduce an auxiliary algorithm called Be-The-Leader (BTL). We begin by demonstrating that the cumulative loss incurred by the best expert in hindsight is lower-bounded by the loss of the Be-The-Leader algorithm in lemma 5. Next, we incorporate entropy regularization, denoted as l_0, which represents the loss incurred by the algorithm at time step 0. We define BTL in relation to this newly introduced cumulative loss. Finally, we establish an upper bound on the difference between the losses incurred by FTRL and BTL, thereby completing the proof.

Under the Be-The-Leader (BTL) algorithm, at each time step t, we determine the probability vector p_t by selecting the vector that minimizes the cumulative loss observed up to and including time step t.

$$p_t = \mathcal{O}(\ell^{1:t})$$

However, it is important to note that this approach is not a practical algorithm, as it relies on prior knowledge of the loss that will be incurred at time t.

The following lemma is a general result for BTL, which holds beyond linear functions.

Lemma 5 (Be the Leader) *For any sequence of loss functions $\ell^1, \ldots, \ell^T$:*

$$\sum_{t=1}^{T} \underbrace{\ell^t(\mathcal{O}(\ell^{1:t}))}_{\texttt{Be the leader}} \leq \underbrace{\sum_{t=1}^{T} \ell^t(\mathcal{O}(\ell^{1:T}))}_{\textit{Loss of the best expert in hindsight}}.$$

Here, $\mathcal{O}(\ell^{1:t})$ corresponds to the decisions made incrementally up to time t, while $\mathcal{O}(\ell^{1:T})$ represents the best expert in hindsight, based on all cumulative losses.

Now we will pretend we have another loss ℓ^0 at step zero, defined as $\ell^0 = -\frac{1}{\eta}H(\mathbf{p})$, where $H(\mathbf{p})$ denotes the entropy of the probability distribution $\mathbf{p}$. Let $l(\mathbf{p})$ denote the expected loss incurred while using the distribution $\mathbf{p}$. Using lemma 5,the cumulative regret is bounded as follows:

$$\begin{aligned}\sum_{t=0}^{T}\left(\ell^t(\mathbf{p}^t) - \ell^t(\mathbf{p}^*)\right) &\leq \sum_{t=0}^{T}\ell^t(\mathbf{p}^t) - \ell^t(\mathcal{O}(\ell^{0:t}))\\ &= \sum_{t=0}^{T}\ell^t(\mathbf{p}^t) - \ell^t(\mathbf{p}^{t+1})\end{aligned}$$

where $\mathcal{O}(\ell^{0:t})$ refers to the leader with entropy regularization. Now, incorporating the loss at step zero:

$$\text{Regret}(\mathbf{p}^{1:T}) + \ell^0(\mathbf{p}^0) - \ell^0(\mathbf{p}^*) \leq \ell^0(\mathbf{p}^0) - \ell^0(\mathbf{p}^1) + \sum_{t=1}^{T}\ell^t(\mathbf{p}^t) - \ell^t(\mathbf{p}^{t+1}).$$

Thus, the regret becomes:

$$\text{Regret}(\mathbf{p}^{1:T}) \leq \underbrace{\ell^0(\mathbf{p}^*) - \ell^0(\mathbf{p}^1)}_{\leq \frac{\log(N)}{\eta}} + \underbrace{\sum_{t=1}^{T}\ell^t(\mathbf{p}^t) - \ell^t(\mathbf{p}^{t+1})}_{\text{Stability term}}. \tag{4.31}$$

Lemma 6 (Stability)

$$\sum_{t=1}^{T}\ell^t(\mathbf{p}^t) - \ell^t(\mathbf{p}^{t+1}) \leq 2\eta\sum_{t=1}^{T}\ell^t(\mathbf{p}^t) \leq 2\eta T.$$

Proof: We have

$$\begin{aligned}\ell^t(\mathbf{p}^t) - l_t(\mathbf{p}^{t+1}) &= \langle \mathbf{l}^t, \mathbf{p}^t - \mathbf{p}^{t+1}\rangle &(4.32)\\ &= \sum_{i=1}^{N} l_i^t\left(p_i^t - p_i^t\frac{e^{-\eta l_i^t}}{\sum_{j=1}^{N}p_j^t e^{-\eta l_j^t}}\right) &(4.33)\\ &\leq \sum_{i=1}^{N} l_i^t p_i^t\left(1 - e^{-\eta l_i^t}\right) &(4.34)\\ &\leq \eta\sum_{i=1}^{N} l_i^t p_i^t = \eta\langle \mathbf{l}^t, \mathbf{p}^t\rangle \leq \eta &(4.35)\end{aligned}$$

■

Here equation (4.34) follows from the non-negativity of the loss function and equation (4.35) follows from assumption 4.

By setting $\eta = \sqrt{\frac{\log(N)}{T}}$ in (4.31) and using lemma 6, we achieve a regret bound of:

$$\text{Regret} = \mathcal{O}(\sqrt{T\log(N)}).$$

This result highlights the efficiency of the algorithm, demonstrating that the regret grows sublinearly with the number of rounds T, while also scaling logarithmically with the number of experts N.

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 5: Follow the Regularized Leader (cont.)

Lecturer: Steven Wu
Scribe: Lintang S., Alfredo G.

5.1 Introduction

- Today: This is the last lecture for online learning covering up to online convex optimization.
- Later: In future lectures we will see how these theoretical tools can be used to derive "idealized" algorithm, serving as a gateway to practical approximations.

Recall from last time the Follow-the-leader algorithm, in which the leader is greedily picking the action and expert that looks the best so far. This however can be *unstable* as the best expert/action may frequently change.

The key idea: Follow-the-regularized-leader (FTRL) is "optimization with *stability*", where the stability is induced via strong convexity.

Revisiting the problem setting, we are making decisions alongside N experts for $1 \leq t \leq T$.

At every round t the learner chooses a probability vector $x^t = (x_1^t, x_2^t, \ldots, x_n^t)$. We say that $x^t \in \Delta(N)$, or the probability simplex of experts, representing the set of probability distributions of N experts.

Meanwhile, an adversary reveals l_t incurring some loss $l_t(x^t) =< x^t, l^t >$, where $\forall t, l^t = (l_1^t, l_2^t, \ldots, l_n^t) \in [0,1]^N$.

5.1.1 Recap on algorithms

Consider the regret function of choosing expert x from time step $1 \leq t \leq T$.

$$\text{Regret}(x^{1:T}) = \sum_{t=1}^{T} l_t(x^t) - \min_{x \in \mathcal{X}} \sum_{t=1}^{T} l_t(x).$$

As the regret function measures a sum of T difference of losses, we seek to find a learner for which regret grows sub-linearly, and as a result,

$$\frac{Regret(x^{1:T})}{T} \to 0.$$

As we observed last time, there are a handful of algorithms for which we may achieve a diminishing regret.

1. Multiplicative weights (update method): In this first approach the probability distributions for the experts is chosen proportional to the losses incurred in previous time steps:

   $$x_i^t \propto \prod_{\tau=1}^{t-1}(1-\eta l_i^\tau)$$

2. Exponential weights (Hedge) Following a similar thought process as multiplicative weights:

   $$x_i^t \propto \exp\left(-\eta \sum_{t=\tau}^{t-1} l_i^\tau\right)$$

   We note that this approach is in fact equivalent to FTRL, in particular when finding the maximum entropy solution of:

   $$x^t = \arg\min_{x\in\Delta(N)} \sum_{\tau=1}^{t-1} l_t(x) + R(x).$$

   where $R(x)$ serves as an entropy regularization term:

   $$R(x) = \frac{1}{\eta}\sum_{i=1}^{N} x_i \ln\left(\frac{1}{x_i}\right).$$

   Observe $0 \leq R(x) \leq \frac{\ln(N)}{\eta}$.

5.2 Analysis of follow-the-regularized-leader (FTRL)

At its core, FTRL is implementing the idea of optimizing with stability.

We begin our analysis by first examining the *Be-the-Regularized-Leader* (BRL) approach. In this case, we assume that the initial round at $t = 0$ is dedicated to optimizing the regularization term:

$$l_0 = R(x).$$

By being the regularized leader, we "see" the loss l_t before choosing a new distribution x. In other words, we use x^{t+1} for round t. Consequently, by *using the next step to optimize the previous loss*, we perform better than any fixed choice of x. This leads to the inequality:

$$l_0(x^1) + \sum_{t=1}^{T} l_t(x^{t+1}) \leq l_0(x) + \sum_{t=1}^{T} l_t(x), \quad \forall x.$$

Referring to the definition of regret and using the previously derived inequality, we obtain:

$$\begin{aligned} \text{Regret} &= \sum_{t=1}^{T} l_t(x^t) - \sum_{t=1}^{T} l_t(x) \\ &\leq \sum_{t=1}^{T} \big(l_t(x^t) - l_t(x^{t+1})\big) + l_0(x) - l_0(x'). \end{aligned}$$

The second term, $l_0(x) - l_0(x')$, is related to how well the sequence x^{t+1} performs compared to the fixed benchmark x. Using the assumption from BRL, we note that:

$$l_0(x) - l_0(x') = R(x) - R(x') \leq \frac{\ln(N)}{\eta}.$$

For now, we will ignore this term.

Taking a closer look at the first term, $\sum_{t=1}^{T}(l_t(x^t) - l_t(x^{t+1}))$, we see that this captures the stability of the updates in the algorithm. We note that if x^t and x^{t+1} are exactly the same, this difference is zero, and we have achieved a perfectly stable algorithm. If the update rule of the algorithm is not drastic, this difference remains small. Conversely, if the updates between adjacent rounds are large, this difference will also be large.

As a consequence, we can use this observation to understand why *Follow-the-Leader* (FTL) is unstable. In FTL, rapidly swapping between different experts results in a large difference between consecutive terms, leading to instability.

5.2.1 Bounding Stability

Consider a lemma that

$$\sum_{t=1}^{T} l_t(x^t) - l_t(x^{t+1}) \leq 2\eta T$$

which implies for values up to a factor of $e^\eta \approx (1 + 2\eta)$ that

$$x_i^t \approx x_i^{t+1}$$

So, regret for FTRL:

$$\text{Regret} \leq 2\eta T + \frac{\ln(N)}{\eta}$$

By optimizing for η we find that the Regret would be minimum when $\eta = \sqrt{\frac{\ln(N)}{2T}}$. This means that regret will grown proportional to

$$Regret \leq O\left(\sqrt{T\ln(N)}\right)$$

5.3 Online convex optimization

Let's assume for $t = 1, \ldots, T$:

- Learner chooses $x^t \in \mathcal{X}$ and that this is convex.
- Adversary presents loss function $l_t \rightarrow l_t(x^t)$, where l_t is convex, differentiable, and Lipschitz continuous.

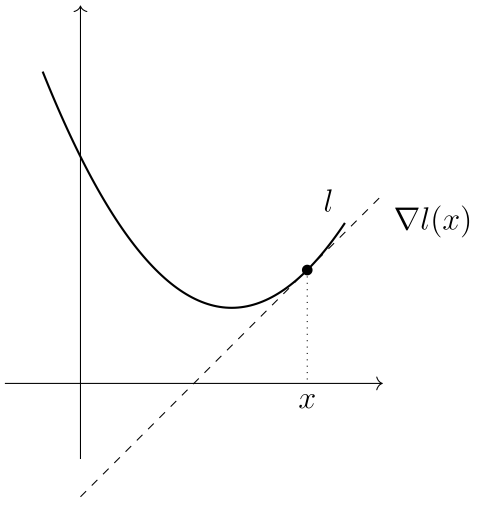


Figure 5.6: Assuming a l is a convex function, we can use gradient at point x

We can revise our FTRL algorithm by substituting the loss l_t with the gradient $\nabla l_t(x^t)$ at x_t.

$$x^t = \arg\min_{x\in\Delta(\mathcal{X})} \sum_{t=1}^{T-1} \langle \nabla l_t(x^t), x\rangle + R(x)$$

5.3.1 Special case: Online Gradient Descent

For a special case of defining $R(x)$ as the Euclidean Norm

$$R(x) = \frac{1}{2\eta}\|x\|_2^2 \quad , x \in \mathbb{R}^d$$

If we optimize w.r.t x

$$\begin{aligned} \sum_{\tau=1}^{t-1} \nabla l_t(x^t) + \frac{1}{\eta}x = 0 \\ x = -\eta \sum_{\tau=1}^{t-1} \nabla l_\tau(x^\tau) \end{aligned} \tag{5.36}$$

Which can be reformulated as gradient descent step

$$x^{t+1} = x^t - \eta \nabla l_t(x^t)$$

Part II

Game Solving

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 6: Minimax via No Regret I

Lecturer: Steven Wu
Scribe: Benji Li, Steven Man

6.1 2-Player Zero-Sum Game

We will start with the most basic two-player normal-form zero-sum game. It has the following elements:

- 2 Players: **Row**, **Col**
- Actions: R, C
- Payoff Matrix: $\mathbf{M} \in \mathbb{R}^{|R| \cdot |C|}$
  - $\mathbf{M}_{ij}$ = Amount of money **Row** wins from **Col**, if **Row** plays $i \in R$ and **Col** plays $j \in C$
- Who goes first?
  1. **Row** goes first, plays i:
     - **Col** plays "best response": $\arg\min_j \mathbf{M}_{ij}$
     - **Row** plays: $\max_i(\min_j \mathbf{M}_{ij})$
  2. **Col** goes first, plays j:
     - **Row** plays "best response": $\arg\min_i \mathbf{M}_{ij}$
     - **Col** plays: $\max_i(\min_j \mathbf{M}_{ij})$

A classical example is the infamous "Rock-Paper-Scissors," which can be described by a 3×3-payoff matrix below:

$$\mathbf{M} = \left(\begin{array}{c|ccc} & R & P & S \\ \hline R & 0 & -1 & +1 \\ P & +1 & 0 & -1 \\ S & -1 & +1 & 0 \end{array}\right)$$

Now consider a thought experiment where one of the two players has to commit to playing some action first, and then the other player can choose their action accordingly. In this case, there is a clear advantage to play second. In the Rock-Paper-Scissor game, this can be written as:

$$\max_i \min_j \mathbf{M}_{ij} = -1$$
$$\min_j \max_i \mathbf{M}_{ij} = +1$$

To read the expressions above, you can go from left to right. For example, $\max_i \min_j$ means the row player chooses to play a row indexed by i first, and then the column play gets to choose a column j later. If both players are optimizing their objective, the resulting payoff (received by the row player) is then $\max_i \min_j M_{ij}$.

One could show that everyone wants to go second in any zero-sum game.

Theorem 7 (Everybody Wants to Go Second)

$$\max_i \min_j \mathbf{M}_{ij} \leq \min_j \max_i \mathbf{M}_{ij}$$

6.2 Randomized Strategies

Now we consider randomized strategies, and see how things can change. Recall that $\Delta(S)$ denotes all probability distributions on a set S.

- **Row** plays $x \in \Delta(R)$
- **Col** plays $y \in \Delta(C)$
- Expected payoff: $\mathbb{E}_{i,j}[\mathbf{M}_{ij}] = \sum_{i,j} x_i y_j \mathbf{M}_{ij} = x^\intercal \mathbf{M} y$

Now, let us revisit the Rock-Paper-Scissor game. Suppose **Row** plays a uniform strategy $x = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. Then it is easy to see that $\min\limits_y x^\intercal \mathbf{M} y = 0$, which suggests that the column player has no real advantage for playing second.

Actually, for any normal-form zero-sum game, the second player gains no advantage from playing second, provided that the first player can randomize their strategy and the second player does not observe the realization of this randomization.

Theorem 8 (Minimax Theorem - Von Neumann '28) *There exists a value $val(\mathbf{M})$ s.t:*

$$\max_{x\in\Delta(R)} \min_{y\in\Delta(C)} x^\intercal \mathbf{M} y = \min_{x\in\Delta(R)} \max_{y\in\Delta(C)} x^\intercal \mathbf{M} y = val(\mathbf{M})$$

Proof: To simplify notations, we will write $U(x, y) = x^\intercal M y$.

Note that by the "everyone wants to go second" theorem, we have

$$\min_y \max_x U(x,y) \geq \max_x \min_y U(x,y)$$

We will proceed by proof by contradiction. Assume $\min_y \max_x U(x,y) = \max_x \min_y U(x,y) + \delta$, for some $\delta > 0$. Consider a thought experiment. We will let the two players repeatedly play the game against each other over T rounds. For each round t: the two players choose a pair of strategies (x^t, y^t) via:

- Min player (previously referred to as **Col** player) plays according to a no-regret algorithm (e.g., FTRL or multiplicative weights), using $\ell^t(y) = U(x^t, y)$ as loss function;
- Max player best responds:

$$x^t = \arg\max_x U(x, y^t)$$

Let $\text{Reg}_y = \sum_{t=1}^T U(x^t, y^t) - \min_y \sum_{t=1}^T U(x^t, y)$. Note that standard algorithms achieve $\text{Reg}_y = O(\sqrt{T})$. Let $\bar{x} = \frac{1}{T}\sum_{t=1}^T x^t$ denote the average play by the max player.

1. By no regret of min player:

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^T U(x^t, y^t) - \frac{1}{T}\text{Reg}_y &\leq \frac{1}{T}\min_y \sum_{t=1}^T U(x^t, y), \\ &= \min_y U(\bar{x}, y), \\ &\leq \max_x \min_y U(x,y)\end{aligned}$$

2. By best response of max player:

$$\begin{aligned}\frac{1}{T}\sum_{t=1}^T U(x^t, y^t) &= \frac{1}{T}\sum_{t=1}^T \max_x U(x, y^t) \\ &\geq \frac{1}{T}\sum_{t=1}^T \min_y \max_x U(x,y) \\ &= \min_y \max_x U(x,y)\end{aligned}$$

By our earlier assumption, we expect a gap of δ:

$$\min_y \max_x U = \max_x \min_y U + \delta$$

Combining 1. and 2.:

$$\min_y \max_x U(x,y) \leq \max_x \min_y U(x,y) + \frac{\text{Reg}_y}{T}$$

Note that the average regret term $\frac{\text{Reg}_y}{T}$ decreases at a rate of $\frac{1}{\sqrt{T}}$. The fact that the average regret goes to 0 contradicts our assumption. ■

Definition 13 *A pair* (x^*, y^*) *such that*

$$\begin{cases} \min_y x^{*T}\mathbf{M}y = val(M) \\ \max_x x^T\mathbf{M}y^* = val(M) \end{cases}$$

is known as a minimax equilibrium*.*

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 7: Computing Equilibria II

Lecturer: Steven Wu
Scribe: Eryn Ma, Naveen Raman, Jingwu Tang

7.1 Approximate Equilibria

7.1.1 Recap: Minimax Theorem

In this lecture, we aim to extend our prior discussion of the minimax theorem to consider approximate equilibria in situations where $\mathcal{X}$ and $\mathcal{Y}$ are convex sets. To begin, first recall the minimax theorem:

$$\max_{x\in\Delta(R)} \min_{y\in\Delta(C)} x^T My = \min_{y\in\Delta(C)} \max_{x\in\Delta(R)} x^T My = \text{Val}(M) \tag{7.37}$$

We can naturally extend the two-player games to situations where the actions x and y are chosen from convex and compact (closed + bounded) sets $\mathcal{X}$ and $\mathcal{Y}$. Now, the utility or payoff can be represented as $U(x, y)$ rather than $x^T My$. We consider the setting in which the utility function U is concave in x and convex in y. Under this scenario, the minimax theorem contines to hold:

$$\max_{x\in\mathcal{X}} \min_{y\in\mathcal{Y}} U(x, y) = \min_{y\in\mathcal{Y}} \max_{x\in\mathcal{X}} U(x, y) \tag{7.38}$$

7.1.2 ϵ-Nash Equilibria

We now aim to extend to this to scenarios that are *approximate* Nash Equilibria, or more formally, ϵ-Nash Equilibria. We first define ϵ-Nash Equilibria as follows:

ϵ-Nash Equilibrium - A pair of actions, $\hat{x}$, $\hat{y}$ is an ϵ Nash Equilibrium if

$$U(\hat{x}, \hat{y}) - \min_y U(\hat{x}, y) \le \epsilon \tag{7.39}$$

$$\max_x U(\hat{x}, y) - U(\hat{x}, \hat{y}) \le \epsilon \tag{7.40}$$

A natural way to find an ϵ-Nash Equilibrium is for two players to play a no-regret strategy. We will formally prove that statement:

Theorem 9 *Suppose two players play No Regret algorithms. That is, let* $\text{Reg}_X = \max_{x\in\mathcal{X}} \sum_{t=1}^T U(x, y^t) - U(x^t, y^t)$*, and similarly let* $\text{Reg}_Y = \min_{y\in\mathcal{Y}} \sum_{t=1}^T U(x^t, y^t) - U(x^t, y)$*. Then the strategy* $\bar{x} = \frac{1}{T}\sum_{t=1}^T x^t$ *and* $\bar{y} = \frac{1}{T}\sum_{t=1}^T y^t$ *is a* $\frac{\text{Reg}_X + \text{Reg}_Y}{T}$ *Nash Equilibrium.*

Proof:

$$\frac{\text{Reg}_X + \text{Reg}_Y}{T} \tag{7.41}$$

$$= \frac{1}{T}(\min_{y\in\mathcal{Y}} \sum_{t=1}^T U(x^t, y^t) - U(x^t, y) + \max_{x\in\mathcal{X}} \sum_{t=1}^T U(x, y^t) - U(x^t, y^t)) \tag{7.42}$$

$$= \frac{1}{T}\sum_{t=1}^T \max_{x\in\mathcal{X}} U(x, y^t) - \frac{1}{T}\sum_{t=1}^T \min_y U(x^t, y) \tag{7.43}$$

$$= \frac{1}{T}\sum_{t=1}^T \max_{x\in\mathcal{X}} U(x, y^t) - U(\bar{x}, \bar{y}) + U(\bar{x}, \bar{y}) - \frac{1}{T}\sum_{t=1}^T \min_y U(x^t, y) \tag{7.44}$$

$$\geq \max_{x\in\mathcal{X}} U(x, \bar{y}) - U(\bar{x}, \bar{y}) + U(\bar{x}, \bar{y}) - \min_y U(\bar{x}, y) \tag{7.45}$$

The last inequality is from Jensen's inequality, and this proves that we achieve a Nash Equilibrium bounded by the sum of the regrets. Therefore, playing No Regret algorithms can lead to a small ϵ Nash Equilibrium. ■

7.2 Algorithms

Note that Theorem 9 holds regardless of the specific algorithms the players follow. We now introduce two types of dynamic strategies that ensure the average play of the players converges to an ϵ-NE.

7.2.1 No-Regret vs. No-Regret (NRNR) Dynamics

The first type of dynamics is no-regret vs. no-regret (NRNR), where both players follow no-regret algorithms. A typical example is when both players employ Follow-the-Regularized-Leader (FTRL).

Max player: $x_t = \arg\min_{x\in\mathcal{X}} -\sum_{\tau=1}^{t-1} U(x, y^\tau) + R(x)$

Min player: $y_t = \arg\min_{y\in\mathcal{Y}} \sum_{\tau=1}^{t-1} U(x^\tau, y) + R(y)$

From the regret bound of FTRL, we know that both $\mathrm{Reg}_X, \mathrm{Reg}_Y = O(\sqrt{T})$. Consequently, the average play of both players converges to an approximate Nash equilibrium.

7.2.2 No-Regret vs. Best-Response (NRBR) Dynamics

The second type of dynamics is no-regret vs. best-response (NRBR), where one player follows a no-regret algorithm while the other plays the best response to the no-regret player's strategy. We consider a case where the x-player plays FTRL while the y-player plays the best response.

Max player: $x_t = \arg\min_{x \in \mathcal{X}} - \sum_{\tau=1}^{t-1} U(x, y^\tau) + R(x)$

Min player: $y^t = \arg\min_{y \in \mathcal{Y}} U(x^t, y)$

From the regret bound of FTRL, we have $\mathrm{Reg}_X = O(\sqrt{T})$. Since y_t always plays the best response, we have $\mathrm{Reg}_Y \leq 0$. Therefore, the average play of the players will converge to an approximate Nash equilibrium.

7.2.3 Revisit Max Entropy

We consider the following maximum entropy problem:

$$\begin{aligned} \min_{p \geq 0} \quad & \sum_i p(i) \ln \frac{1}{p(i)} := \mathcal{H}(p) \quad \text{(OPT)} \\ \text{s.t.} \quad & \sum_i p(i) f(i) = c \\ & \sum_i p(i) = 1 \end{aligned} \tag{7.46}$$

To solve the constraint optimization problem, we introduce the Lagrangian dual variables γ and λ, we write the Lagrangian:

$$\min_{\gamma,\lambda} \max_{p \geq 0} \mathcal{H}(p) - \gamma \left(\sum_i p(i) - 1 \right) - \lambda \left(\sum_i p(i) f(i) - c \right) \tag{7.47}$$

Here, γ and λ penalize deviations from the constraints, ensuring that:

$$\sum_i p(i) - 1 = 0, \quad \sum_i p(i) f(i) = c. \tag{7.48}$$

Now, we can view it as a minimax game, and computing the minimax equilibrium with the NRBR dynamics introduced above.

$$U(p,(\gamma,\lambda)) := \mathcal{H}(p) - \gamma\left(\sum_i p(i) - 1\right) - \lambda\left(\sum_i p(i)f(i) - c\right) \tag{7.49}$$

- The dual variables γ and λ are updated using no-regret algorithms, specifically online gradient descent.

- The primal variable p follows a best-response dynamics given by:

$$p^t(i) \propto \exp(-\lambda^t f(i)). \tag{7.50}$$

From Theorem 9, we know that $(\overline{p}, \overline{\lambda}, \overline{\gamma})$ converge to an approximate NE.

Now suppose we have an 0-NE for the minimax game. This ensures that $\min_{\gamma,\lambda} U(\overline{p},(\gamma,\lambda)) =$ OPT, which implies that both constraints are exactly satisfied.

7.2.4 Extensive-form games with incomplete information

We begin by introducing extensive-form games (EFGs) with incomplete information.

Figure 7.7 provides an example of an EFG. An EFG is represented as a graph, where blue nodes indicate that it is player 1's turn, while red nodes indicate that it is player 2's turn. In such games, players may have incomplete information, meaning they might not know their exact position in the game tree, even if they have perfect recall—that is, they remember the sequence of actions taken by all players. This uncertainty arises due to the presence of a fictitious nature player, which determines certain elements of the game. For instance, in Figure 7.7, nature decides whether the card is a King or an Ace. As a result, player 2 cannot distinguish between certain nodes when making their move. Consequently, they must play the same strategy at these nodes, which together form what is known as an *information set*. For a more detailed introduction to EFGs, we refer the reader to [2].

There are two common representations for strategies in extensive-form games.

The first representation is sequence-form strategies, which model each player's strategies as sequences walking down the tree. Using sequence-form strategies, the strategy space for both players becomes convex and compact, making game a bilinear problem. This representation shows that the minimax theorem holds in extensive-form games. A detailed introduction to sequence-form strategies can be found in [2].

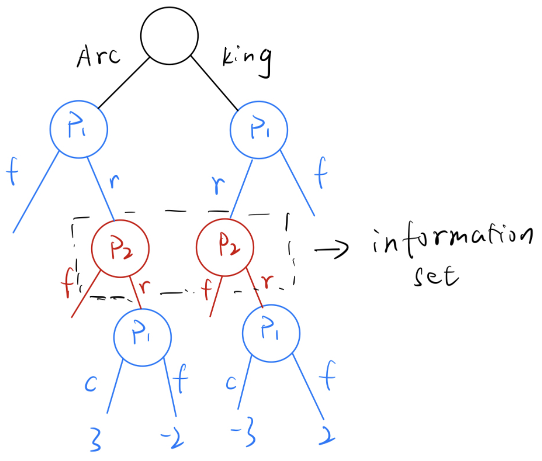


Figure 7.7: Example of EFG with imcomplete information

The second representation are behavior strategies, where a player's strategy is defined as a probability distribution over actions at each decision point. Under this representation, we are able to design no regret algorithms for EFG, such as Counterfactual Regret Minimization (CFR). CFR maintains an independent instance of a no-regret algorithm at each decision node, enabling efficient learning in extensive-form games. For a detailed discussion of CFR, we refer the reader to [3].

Part III

Sequential Decision Making

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 8: Markov Decision Processes

Lecturer: Steven Wu
Scribe: Kimberly Truong, Megan Li

8.1 Overview

We will start sequential decision making and reinforcement learning:

1. Elements of Markov Decision Processes (MDP) with a focus on finite horizons
2. Value functions: Bellman Equations/optimality
   Note: Bellman is sometimes synonymous with dynamic programming
3. Value/policy iteration

8.2 MDP Notation

Consider the following notation:

- State space S
- Action space A
- Reward function $R : S \times A \to \Delta([0, 1])$, or $r \sim R(s, a)$
- Transition operator $P : S \times A \to \Delta(S)$
- Initial state distribution $\mu_0 \in \Delta(S)$
  with initial state $s_0 \sim \mu_0$

Fact 10 *MDPs have the Markovian ("memorylessness") property where its future state(s) are independent of its history.*

8.3 Finite Horizon

Remark 11 *Discounted and Infinite Horizon problems also exist, but we will focus on problems with finite horizons in this class.*

Consider the following notation:

- Horizon H
- Trajectory $\tau = (s_0, a_0, r_0, s_1, ..., s_{H-1}, a_{H-1}, r_{H-1})$
- Policy π
- Objective

$$J(\pi) = \mathbb{E}\left[\sum_{h=0}^{H-1} r_h | S_0, a_{0:H-1} \sim \pi\right] \tag{8.51}$$

There can be as many as $|A|^{|S|H}$ *deterministic policies.*

- Value function(s)
  - State-value function

$$V_h^\pi := \mathbb{E}\left[\sum_{h'=h}^{H-1} r_{h'} | s_h = s, a_{h:H-1} \sim \pi\right] \tag{8.52}$$

  - Action-value function

$$Q_h^\pi := \mathbb{E}\left[\sum_{h'=h}^{H-1} r_{h'} | s_h = s, a_h = a, a_{h+1:H-1} \sim \pi\right] \tag{8.53}$$

8.3.1 (Non-stationary) Markov Policy

In general, a policy $\pi : \mathcal{H} \to \Delta(A)$, where $\mathcal{H}$ represents all partial history.

Definition 14 *A* ***non-stationary Markov Policy*** *compresses all partial history into the current state and time step.* π_h *denotes a policy defined at time step* h*.*

$$\pi : S \times [H] \to \Delta(A)$$

8.3.2 Bellman Equations

For any policy π, we can derive the following *Bellman equations*:

$$V_h^\pi = \mathbb{E}[r_h + V_{h+1}^\pi(s_{h+1}) | s_h = s, a_h \sim \pi] \tag{8.54}$$
$$Q_h^\pi(s, a) = \mathbb{E}[r_h + Q_{h+1}^\pi(s_{h+1}, a_{h+1}) | s_h = s, a_h = a, a_{h+1} \sim \pi] \tag{8.55}$$

Remark 12 *Considering just the first time step* $h = 1, \pi = \pi_0$ *yields:*

$$J(\pi) = \mathbb{E}_{s_0 \sim \mu_0}[V^\pi(s_0)]$$

Theorem 13 *Define* $V^* = (V_0^*, ..., V_H^*)$ *recursively as*

$$V_H^*(s) = 0 \tag{8.56}$$

$$\forall(s,h) : V_h^*(s) = \max_a \mathbb{E}_{s' \sim P(s,a)} \left[r(s,a) + V_{h+1}^*(s') \right] \tag{8.57}$$

Then, $\sup_{\pi:\mathcal{H}\to\Delta(A)} J(\pi) = \mathbb{E}_{s_0\sim\mu}[V_0^*(s_0)]$. *Now define the policy* $\pi^* := (\pi_0^*, ..., \pi_{H-1}^*)$ *as*

$$\forall(s,h) : \pi_h^*(s) = \arg\max_a \{\mathbb{E}_{s'\sim P(s,a)}[r(s,a) + V_{h+1}^*(s')]\} \tag{8.58}$$

Because π^* *achieves value* V^* *for all* (s,h)*, it achieves optimality.*

The equation (8.57) is called the Bellman optimality equation (for V). The recursive procedure (going from H to 0) defined by equations (8.56) and (8.57) is called *value iteration*.

Similarly, one can also derive the Bellman optimality equation and value iteration in terms of action value functions:

$$\forall(s,a) \in S \times A \qquad Q_H^*(s,a) = 0 \tag{8.59}$$

$$\forall(s,a,h) \in S \times A \times [H-1] \qquad Q_h^*(s,a) = \mathbb{E}\left[r(s,a) + \max_{a'} Q_{h+1}^*(s',a') \right] \tag{8.60}$$

The optimal policy can also be written as

$$\pi_h^*(s) = \arg\max_a Q_h^*(s,a)$$

8.3.3 Policy Iteration

Another algorithm for computing the optimal value function and policy is *policy iteration*. Start with $\pi^{(0)}$, and then for each increment of t, we compute $Q^{\pi^{(t-1)}}$ where $Q_H^{\pi^{(t-1)}} = 0$ and for all (s,a) pairs, $Q_h^{\pi^{(t-1)}}(s,a)$ is computed by (8.55). This is the *policy evaluation* step. It is followed by the *policy improvement* step, during which we greedily update the policy as

$$\pi_h^{(t)}(s) := \arg\max_a Q_h^{\pi^{(t-1)}}(s,a) \tag{8.61}$$

Remark 14 *This algorithm guarantees* local improvement*—that is, at every time step* h *and state* s*:*

$$\forall(s,h) : \mathbb{E}_{a\sim\pi^{(t+1)}(s)}[Q_h^{\pi^{(t)}}(s,a)] = \max_{a\in A} Q_h^{\pi^{(t)}}(s,a) \geq \mathbb{E}_{a\sim\pi^{(t)}}[Q_h^{\pi^{(t)}}(s,a)] \tag{8.62}$$

for all t*.*

Interestingly, by the seminal *Performance Difference Lemma*, we can ensure *global improvement*, provided that we can local improvement at every state s. In this context, the lemma can be stated as follows.

Lemma 15 (Performance Difference Lemma)

$$J(\pi^{(t+1)}) - J(\pi^{(t)}) = \mathbb{E}_{\tau \sim \pi^{(t+1)}} \left[\sum_{h=1}^{H-1} \mathbb{E}[Q^{\pi^{(t)}}(s_h, \pi_h^{(t+1)}(s_h))] - \mathbb{E}[Q^{\pi^{(t)}}(s_h, \pi_h^{(t)}(s_h))] \right] \tag{8.63}$$

The left-hand side of the equation is the "global improvement" in the total reward objective and the right-hand side is the sum over the expected local improvement over time steps.

17-740 Spring 2025

Algorithmic Foundations of Interactive Learning

Lecture 9: DAgger & Covariate Shift in IL

Lecturer: J. Andrew (Drew) Bagnell

Scribe: Bardienus Duisterhof, Kimberly Truong, Khush Agrawal

9.1 Overview

We will cover the basics of imitation learning, *an invitation to imitation*. Among other things, we will cover DAgger (Dataset Aggregation) and how it can improve imitation learning (IL) performance in the presence of covariate shift.

9.2 Imitation Learning (IL)

Imitation learning is when we do not have access to a reward function, and instead aim to learn from expert demonstrations. One flavor of imitation learning is *offline* behavioral cloning, where we try to mimic the actions of the expert directly. In contrast, *interactive* imitation learning algorithms (as we discuss in this lecture) instead attempt to match the outcomes of expert actions / overall expert behavior.

A natural question one might have *"how is IL different from most other ML problems?"*

- **Decisions have consequences**: errors pass through a feedback loop and compound $\rightarrow$ distribution shift, test $\neq$ train distribution. For example, a small steering error in a self-driving car can cause the vehicle to drift toward the edge of the road. If the training data only contains examples from driving in the center of the lane, the policy won't know how to correct this drift, potentially causing the car to drive off the track entirely.
- **Sequential decisions**: Actions and decisions are purposeful and sequential. They shape future states and returns, building toward long-term goals rather than just responding to the current state.
- **Non-IID data**: The sequential nature of decision-making breaks the independent and identically distributed data assumption required by traditional supervised learning methods.

9.2.1 Learning to drive by imitation

Consider an RGB camera image (state s) $\rightarrow$ policy $\pi(a|s)$ $\rightarrow$ distribution over actions a, where actions represent steering angles. As proposed by [4], we can use behavioral cloning to learn from expert demonstrations. Given a dataset $\mathcal{D} = \{(s_i, a_i)\}_{i=1}^N$ of state-action pairs demonstrated by an expert policy π_E, we solve the maximum likelihood problem:

$$\max_{\pi\in\Pi} \sum_{\xi\in\mathcal{D}} \log \mathbb{P}_\pi(\xi) = \max_{\pi\in\Pi} \sum_{\xi_E\in\mathcal{D}} \log\left(\prod_h^H \pi(a_h^E|s_h^E)\right) = \max_{\pi\in\Pi} \sum_{\xi_E\in\mathcal{D}} \sum_h^H \log \pi(a_h^E|s_h^E) \tag{9.64}$$

This objective aims to maximize the probability of taking the same actions as the expert in the states encountered by the expert and is the standard, *supervised learning* to imitation.

Now let's say, at every step h, there is an ϵ probability the policy disagrees with the expert. Each mistake at time step h can lead the learner to deviate from the expert on all subsequent steps. This results in compounding errors. After H steps, the resulting gap on $J(\pi)$ can be expressed as:

$$J(\pi) - J(\pi_E) \le \epsilon \sum_{h=1}^H (H-h) = \epsilon\frac{H(H-1)}{2} \in O(H^2\epsilon) \tag{9.65}$$

The quadratic dependence on horizon length H tells us that errors compound over time.

So then how do we mitigate this compounding error?

Core idea: use interaction. Intuitively, this is because interaction allows us to see states from the test distribution at training time, eliminating the covariate shift.

Algorithm 1: **Forward Training**. We train a sequence of policies $\pi_1, \pi_2, \dots, \pi_H$, where π_h is trained to predict the action the expert would have taken at time step h, given that we followed policies $\pi_1, \dots, \pi_{h-1}$ for the first $h-1$ steps. So, at $h = 1$, we perform vanilla behavioral cloning to learn π_1. Then, at $h = 2$, we roll out π_1 and ask the expert what they would have done, had they been in our situation, collecting a dataset of action labels. We then train π_2 via behavioral cloning on this new, on-policy dataset and repeat till $h = H$.

Assume that at each step, we make a mistake with probability ϵ and such a mistake can at most cost us u (we'll explore what u means in a bit). Then, simply by summing up over timesteps, we have

$$J(\pi_{\text{forward}}) - J(\pi_E) \le \sum_h^H p(\text{mistake}) \cdot \max\text{ cost} = \epsilon u H \tag{9.66}$$

The bound is now linear in H rather than quadratic, because each policy π_h is trained on the distribution of states that the learner actually encounters at step h. However, this approach requires training H separate policies, which becomes impractical for large H, begging the question of how we train a stationary policy interactively.

Algorithm 2: **DAgger: dataset aggregation**. DAgger [5] performs a similar procedure to learn a single, stationary policy. Algorithm 1 describes the algorithm.

Algorithm 1 DAgger: Dataset Aggregation

```
Require: Initial dataset $\mathcal{D}_0$ collected from offline data
  Train initial policy $\pi_1$ on $\mathcal{D}_0$
  for $i = 1$ to $N$ do
    Execute $\pi_i$ in the environment to collect trajectories $\xi_i$
    Query expert for action labels at all states in $\xi_i$
    Aggregate action labels into dataset: $\mathcal{D}_i \leftarrow \mathcal{D}_{i-1} \cup$ action labels
    Train new policy $\pi_{i+1}$ on aggregated dataset $\mathcal{D}_i$ via behavioral cloning
  end for
  return Best of $N$ policies on validation data.
```

As we'll prove below, with large enough N, we get:

$$J(\pi) - J(\pi_E) \leq uH\epsilon. \tag{9.67}$$

With DAgger we reduce the problem of imitation learning to that of no-regret online learning. In particular, the "dataset aggregation" procedure is implementing the follow the (regularized) leader algorithm we discussed earlier. The environment is not an adversary per se but the no-regret framework is still useful because we're not in the statistical / iid setting.

Analysis. To analyze the performance of DAgger, consider the following "zero-one" loss: [2]

$$\mathcal{L}_{\pi_E}(\pi, s) = \mathbb{E}_{a\sim\pi}\left[\mathbf{1}(a \neq \pi_E(s))\right]. \tag{9.68}$$

This is the probability that our action doesn't match the expert's. We can take the expectation over states generated by π_i to get the sequence of loss functions we will feed to our no-regret online learner (FTRL / dataset aggregation in this case):

$$\ell_i(\pi) = \mathbb{E}_{s\sim\rho_\pi}[\mathcal{L}_{\pi_E}(\pi, s)]. \tag{9.69}$$

Critically, notice that this expectation is taken over states from the learner's state distribution rather than the expert's – this is what distinguishes this loss from behavioral cloning.

[2] We're assuming the expert is deterministic here for simplicity, see [5] for the more general proof.

We can then express the optimization problem we're solving at each round of DAgger as

$$\pi_{i+1} = \min_{\pi \in \Pi} \sum_{j}^{i} \ell_j(\pi). \tag{9.70}$$

A natural question when reading the above equation might be "*why are we optimizing over the history of past visitation distributions when we care about our current policy's visitation distribution*?" Observe that if the policy we're choosing doesn't change too much from what we've seen in the past, minimizing the above equation will give us good guarantees. Intuitively, this is what the no-regret property of FT(R)L means. Perhaps the easiest way to see this is to notice that each new dataset is a vanishing fraction of the overall dataset as $N \to \infty$, which means our learning process should eventually stabilize.

Let's write out the cumulative loss accumulated by the no-regret online learner:

$$\frac{1}{N}\sum_{i=1}^{N}[l_i(\pi_i)] = \underbrace{\frac{1}{N}\sum_{i=1}^{N}[l_i(\pi_i) - l_i(\pi^\star)]}_{\text{average regret}} + \underbrace{\frac{1}{N}\sum_{i=1}^{N} l_i(\pi^\star)}_{\text{expert loss}}, \tag{9.71}$$

where $\pi^\star = \min_{\pi \in \Pi} \sum_i^N \ell_i(\pi)$. Note: $\pi^\star$ need not necessarily be π_E if we're in the *misspecified* setting (i.e. $\pi_E \notin \Pi$). If we're in the *realizable* setting (i.e. $\pi_E \in \Pi$), the second term on the RHS must be 0. We'll assume this for simplicity for the rest of the note.

Due to the no-regret property of FT(R)L / dataset aggregation, we know that the first term goes to zero on average. Via the Performance Difference Lemma, we can link the regret of our online learner to the performance of the learned policy. To do so, we'll first need to discuss the concept of *recoverability* more formally – the u from above.

Recoverability. A key challenge in imitation learning is ensuring that mistakes made by the learned policy do not cause large deviations in performance from the expert. The notion of recoverability helps formalize this. Recoverability in this context means that the maximum cost a single-step deviation from the expert could inflict is bounded. In math,

$$Q_h^{\pi_E}(s,a) - Q_h^{\pi_E}(s,\pi_E(s)) \le u,\ \forall (s,a) \in \mathcal{S} \times \mathcal{A}. \tag{9.72}$$

Consider the following two cases: (1) Flat Terrain: You're walking on a wide, flat field. If you take a slightly wrong step, you can easily recover — the cost u is small. (2) Walking Along a Cliff: Now suppose you're walking along a narrow mountain ridge. A small misstep could send you tumbling down the cliff — the cost u is very high. In both cases, ϵ could be the same (say, you make a mistake 10% of the time). But the overall performance degradation depends heavily on u. On the flat field, $T\epsilon u$ might still be small. But on the cliff, even small ϵ leads to catastrophic performance — hence, the bound becomes loose unless you can ensure very low ϵ. This highlights why recoverability matters. If the expert can't recover from mistakes (i.e., the environment is unforgiving), even DAgger can't guarantee good performance unless the learner gets very accurate.

Completing the Proof. Assume that the expected 0-1 classification loss (probability of disagreement with expert) at each time step is bounded by ϵ. This corresponds to the average regret is bounded by ϵ in the realizable setting. If the average regret is bounded by ϵ, there must exist some $i \in [N]$ that has relative loss bounded by ϵ. Furthermore, assume that the recoverability condition 9.72 holds for some constant u. Then, the performance gap between the learner and the expert can be bounded as

$$J(\pi) - J(\pi_e) \leq H\epsilon u. \tag{9.73}$$

The PDL tells us that for any policy π,

$$J(\pi) - J(\pi_E) = \sum_{h}^{H} E_{s_h \sim \pi}[Q_h^{\pi_E}(s_h, \pi(s_h)) - Q_h^{\pi_E}(s_h, \pi_E(s_h))]. \tag{9.74}$$

By Hölder's inequality, each term in the sum is bounded by ϵu for some $i \in [N]$, giving us:

$$\min_{i \in N} J(\pi_i) - J(\pi_E) \leq \sum_{h}^{H} \epsilon u = H\epsilon u. \tag{9.75}$$

QED!

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 10: Approximate Policy Iteration

Lecturer: Drew Bagnell
Scribe: Benji Li, Riku Arakawa, Vansh Kapoor

Notations In Drew's lectures, we typically think about costs instead of rewards,[3] and use T to denote horizon instead of H.

10.1 Approximate Dynamic Programming

Depending on the level of knowledge about the environment we have access to (transition dynamics, reward function, etc), as well as the amount of privilege we have in our experimental setup (such as the ability to reset to a previous state), we can use different models to sample and learn from our environment.

The following list includes different types of access models in RL.

1. Full probabilistic description of the environment: In this model, the algorithm is given full description of $\{p(s' \mid s, a), T, c(s, a)\}$. In a tabular MDP (with a moderate number of states and actions), we show in a prior lecture that value iteration provides a straightforward way to learn the optimal value functions, given the full probabilistic description of the environment.
2. Deterministic Simulative Model: In its simplest form, a deterministic simulative model provides a function that maps $(x, a) \to x'$ deterministically. More generally, even when the underlying dynamics are stochastic, we may still have access to a fixed random seed within a computer program. This allows us to perfectly recreate trajectories, including all randomness that occurred. Such access is common in computer simulations, where reproducibility is desired.
3. Generative models: In this model, we have programmatic access to state transitions, meaning we can place the system in any desired state and observe its evolution. This enables flexible exploration and controlled experimentation, making it a powerful tool for understanding and optimizing decision processes.
4. Reset models: In this model, we can execute a policy or simulate rollouts at any time,

[3]However, he may also switch midway in the lecture.

with the ability to reset the system to a known state or a predefined distribution over states. This feature makes it particularly useful in controlled settings, such as robotics experiments, where a robot can be repeatedly reset to stable configurations for consistent evaluation and testing.

5. Single Trace: This model is the most challenging, where actions are irreversible, and past states cannot be revisited. The trace model captures this fundamental constraint—the inability to ”reset” in real-world decision-making.

10.2 Approximating Value and Q Iteration

In this lecture, we will assume costs are deterministic to simplify notations. Recall the Bellman optimality equations and the Bellman equations in terms of action value functions.

$$\begin{aligned}
Q^*(s,a,t) &= c(s,a) + \mathbb{E}_{p(s'|s,a)}[\min_{a'} Q^*(s',a',t+1)] \\
Q^*(s,a,t) &= c(s,a) + \text{Total future value of acting optimally} \\
Q^\pi(s,a,t) &= c(s,a) + \text{Total future value of following policy } \pi \\
Q^\pi(s,a,t) &= c(s,a) + \mathbb{E}_{p(s'|s,a)}[Q^\pi(s,\pi(a,t+1),t+1)]
\end{aligned}$$

Note that one could derive Bellman equations in terms of state value functions as well. There are some pros and cons.

Pros of Action Value Functions Computing the optimal policy from Q^* is simpler than extracting it from V^*. With Q^*, we can obtain the optimal action using a straightforward $\arg\max$, without needing to evaluate expectations or rely on a transition model. Once we have Q^*, we don't need a transition model at all to determine the optimal policy.

Cons of Action Value Functions Action-value functions require more memory than state-value functions. While a value function only needs to store a value for each state (|States|), an action-value function must store values for every state-action pair (|States| $\times$ |Actions|), leading to a significantly larger space requirement.

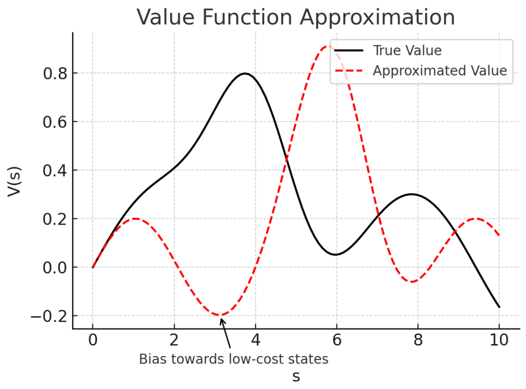
We will depart from the tabular setting in earlier lectures, where we can afford to enumerate all state-action pairs. Instead, we will introduce approximations to algorithms like value iteration and policy iteration. The first algorithm is *Fitted Q-Iteration*, an approximate dynamic programming algorithm that learns approximate action-value functions from data.

Algorithm 2 Fitted Q-Iteration

```
Require: Dataset $\{(s_i, a_i, c_i, s_i')\}_{i=1}^N$, horizon $T$
 1: Initialize: $Q(s, a, T) \leftarrow 0$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
 2: for $t \in [T-1, T-2, \dots, 0]$ do
 3:     $D^t \leftarrow \varnothing$
 4:     for $i = 1, \dots, N$ do
 5:         $input \leftarrow (s_i, a_i)$
 6:         $target \leftarrow c_i + \min_{a'} Q\big(s', a', t+1\big)$
 7:         $D^t \leftarrow D^t \cup \{(input, target)\}$
 8:     end for
 9:     $Q(\cdot, \cdot, t) \leftarrow \text{Regression}(D^t)$
10: end for
11: return $Q$
```

Challenges with Fitted Q-Iteration In class, we went through some interesting toy examples presented in [6], in which Fitted Q-Iteration fails even in settings where the true value function lives in the function class we perform regression over. Fitted Q-Iteration and its counterpart, Fitted Value Iteration suffer from bootstrapping issues and sometimes fail to converge. These methods approximate the value function inductively, propagating and even amplifying errors, leading the algorithm to favor suboptimal actions.

The core issue lies in the *minimization step* when generating target values. This step can push the policy toward states where the approximate value function underestimates the true value, making them appear deceptively attractive. This bias is especially severe in sparsely sampled regions of the state space, where poor generalization may result in policies favoring undesirable states. From a learning theory perspective, this violates the i.i.d. assumption on training and test samples.



10.3 Approximate Policy Iteration

One major problem encountered when approximating the optimal Q-function is that of overestimation, which tends to amplify errors during the update process. In addition, there is the issue of covariate shift. As the algorithm updates its policy based on the approximated Q-function, the distribution of state-action pairs encountered during training shifts away from the one initially used to learn the approximation. Consequently, when the improved policy is deployed, it may encounter states that were underrepresented (or even absent) in the training data. This mismatch between the training and deployment distributions further exacerbates the error propagation and amplification issues.

A common remedy to mitigate these issues is to shift from using the optimal $Q^\star$ directly, to instead employing a policy-dependent Q-function, denoted as Q^π. This leads naturally to the use of policy iteration rather than direct optimization of $Q^\star$. Here, there are two fundamental steps: policy evaluation and policy improvement.

10.3.1 Policy Evaluation

The algorithm is shown in Algorithm 3. Instead of $Q^\star$, we are trying to estimate Q^π.

Algorithm 3 Policy Evaluation

```
Require: Dataset $\{(s_i, a_i, c_i, s'_i)\}_{i=1}^N$, horizon $T$, discount factor $\gamma$
 1: Initialize: $Q(s, a, T) \leftarrow 0$, for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
 2: for $t = T-1, T-2, \dots, 0$ do
 3:     $D^t \leftarrow \varnothing$
 4:     for $i = 1, \dots, N$ do
 5:         $input \leftarrow (s_i, a_i)$
 6:         $target \leftarrow c_i + \gamma\, Q^\pi\big(s'_i,\, \pi(s'_i, t+1),\, t+1\big)$
 7:         $D^t \leftarrow D^t \cup \{(input, target)\}$
 8:     end for
 9:     $Q(\cdot, \cdot, t) \leftarrow \text{Regression}(D^t)$
10: end for
11: return $Q$
```

10.3.2 Policy Improvement

When using an approximated $Q^{\pi}(s,a,t)$ function to derive a policy, a straightforward approach is to select the action that maximizes the estimated value:

$$\pi^{\text{proposed}}(s,t) = \text{argmax}_a Q^{\pi}(s,a,t) \tag{10.76}$$

However, this direct maximization can still suffer from the same issues mentioned earlier (e.g., overestimation or instability due to covariate shift). To ensure a more stable update, we can interpolate between the current policy and the proposed policy. This approach is known as conservative policy iteration. The updated policy is given by:

$$\pi'(s,t) = (1-\alpha)\pi(s,t) + \alpha\pi^{\text{proposed}}(s,t), \tag{10.77}$$

where $\alpha \in (0,1]$ is a step-size parameter that controls the degree of change in the policy at each iteration.

10.4 Policy Search by Dynamic Programming

The previous algorithms discussed (Fitted Q-iteration and Conservative P.I.) rely on Bellman back-up and approximation of the action value functions. Unlike value-based methods that derive policies from value estimates, policy search by dynamic programming (PSDP) directly optimizes the policy itself by going backward in time. The intuition is that if we have already obtained the optimal (non-stationary) policies from time step $(t+1)$ and onward, then the optimization problem at time step t becomes much simpler: we can simply choose the action that maximize the total reward given that we have committed to following the policies $\{\pi_{t+1}, \dots \pi_{T-1}\}$ for later steps. By unrolling the policy at every step until termination, we could then avoid the issues of overestimation and compounding errors suffered by the previous algorithms. As we deploy the policy at every time step until termination, we incur a time complexity of $\mathcal{O}(T^2)$.

PSDP achieves the following performance bound.

Value Function Bound

$$V_{\pi}(s_0) \geq V_{\pi_{ref}}(s_0) - \sum_{t=0}^{T-1} \epsilon \left\| \frac{\partial\pi_{ref}^{t}}{\mu_t} \right\|_{\infty}$$

where $\partial\pi_{ref}^{t}(t), \mu(t)$ denote the state distributions induced by the policies π, π_{ref} at time step h, respectively, and ϵ relates to regression loss. We will give the analysis using the performance difference lemma in the next lecture.

Algorithm 4 Policy Search by Dynamic Programming

```
1: Initialize: $Q(s, a, T) \leftarrow 0$ for all $s \in \mathcal{S}$, $a \in \mathcal{A}$
2: for $t = T - 1, \ldots, 0$ do
3:     $D_t \leftarrow \varnothing$
4:     for each state $s_i \in \mathcal{S}$ do
5:         for each action $a_j \in A$ do
6:             Compute target as the sum of future rewards:
               target = sum of future rewards executing $\pi_t, \pi_{t+1}, \ldots, \pi_T$
7:             $D_t \leftarrow D_t \cup \{(s_i, a_j, \text{target}_{ij})\}$
8:         end for
9:     end for
10:    $\hat{Q}_t \leftarrow \text{Regress}(D_t)$
11:    Update policy: $\pi_t = \arg\max_a \hat{Q}_t(s, a)$
12: end for
13: return $Q$
```

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 11: Policy Gradients

Lecturer: Steven Wu
Scribe: Jiahao Zhang, Naveen Raman, Riku Arakawa

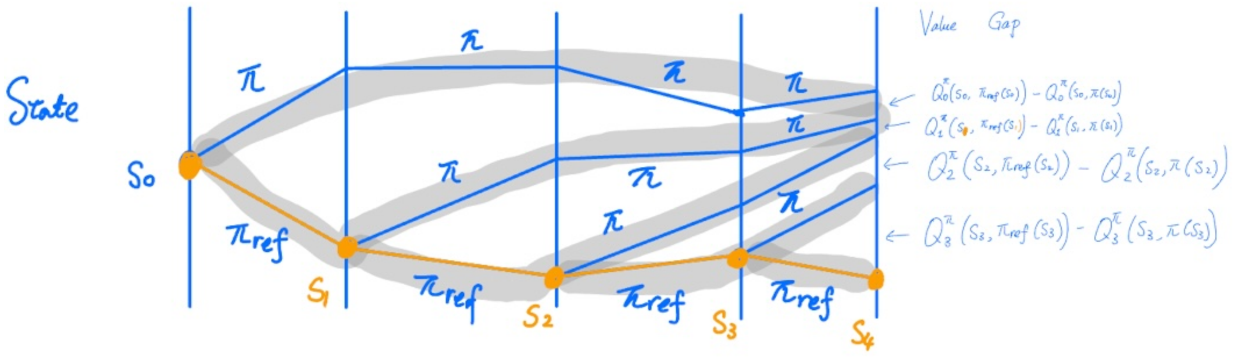

State
S_0
π
π
π
π
π
π
π
π
π
π_{ref}
S_1
π_{ref}
S_2
π_{ref}
S_3
π_{ref}
S_4
Value Gap
$Q_0^\pi(s_0, \pi_{ref}(s_0)) - Q_0^\pi(s_0, \pi(s_0))$
$Q_1^\pi(s_1, \pi_{ref}(s_1)) - Q_1^\pi(s_1, \pi(s_1))$
$Q_2^\pi(S_2, \pi_{ref}(S_2)) - Q_2^\pi(S_2, \pi(S_2))$
$Q_3^\pi(S_3, \pi_{ref}(S_3)) - Q_3^\pi(S_3, \pi(S_3))$

Figure 11.8: A visualization of the Performance Difference Lemma

11.1 Recap: PDL and PSDP

We first provide some more intuition for the *performance difference lemma* (PDL), which bounds the difference between $J(\pi_{\text{ref}})$ and $J(\pi)$. To simplify our reasoning, we will assume that both policies and transitions are deterministic. As shown in Figure 11.8, the reference policy π_{refs} will then visit a sequence of states $s_0, s_1 \ldots s_{H-1}$.

Suppose we are at time step h and at the expert's state S_h. We can compare two trajectories:

1. Taking action $\pi_{\text{ref}}(S_h)$ and then following π for the remaining steps.
2. Following π starting at this step and all future steps.

The difference in their values can be written as:

$$\delta_h = Q_h^\pi(S_h, \pi_{\text{ref}}(S_h)) - Q_h^\pi(S_h, \pi(S_h))$$

Now we do this comparison at state s_0, then δ_0 measures the value gap between the top two trajectories in Figure 11.8. Similarly, δ_1 measures the value gap between the second and third trajectories on top. Observe that the gap we are interested in $J(\pi_{\text{ref}}) - J(\pi)$ is precisely the value gap between the top trajectory (fully in blue) and bottom trajectory (fully in yellow). By telescoping, we can write

$$J(\pi_{\text{ref}}) - J(\pi) = \sum_h \delta_h$$

This is precisely the deterministic version of PDL.

Now with this intuition and picture of PDL in mind, we can also gain an intuitive understanding of the *policy search by dynamic programming* (PSDP) algorithm. The algorithm operates on the assumption that we are given a baseline distribution μ_h at every step h. PSDP essentially ensures the quantity δ_h is small over the distribution μ_h of state s_h. PSDP achieves this via backward induction: at every step h, it optimizes the policy π_h over the state distribution μ_h given the learned policies $\pi_{h+1}, \ldots, \pi_{H-1}$ at later steps. (Note that this then becomes a one-step decision-making problem, or equivalently a classification problem.) Concretely, it first computes the action value for each action a at each sampled state $s_h \sim \mu_h$:

$$Q_h^\pi(s_h, a) = r(s_h, a) + \mathbb{E}_{s' \sim P(\cdot|s_h, a)}[V_{h+1}^\pi(s')]$$

where $V_{h+1}^\pi(s')$ is the estimated value function at the next time step. Then, PSDP updates π_h to select action a that maximizes the estimated $Q_h^\pi(s_h, a)$.

Suppose the algorithm achieves ϵ error for each step over the distribution μ_h–that is,

$$\mathbb{E}_{s_h \sim \mu_h}\left[Q^\pi(s_h, \pi(s_h))\right] \geq \max_{\pi'} \mathbb{E}_{s_h \sim \mu_h}\left[Q^\pi(s_h, \pi'(s_h))\right] - \epsilon$$

By change of measure from the baseline distribution to the state distribution visited by the reference policy, we have

$$\mathbb{E}_{s_h \sim d_h^{\pi_{\text{ref}}}}\left[Q^\pi(s_h, \pi(s_h))\right] \geq \max_{\pi'} \mathbb{E}_{s_h \sim d_h^{\pi_{\text{ref}}}}\left[Q^\pi(s_h, \pi'(s_h))\right] - \epsilon \left\| \frac{d_h^{\pi_{ref}}}{\mu_h} \right\|_\infty$$

By PDL, we can bound the performance difference as

$$J(\pi_{\text{ref}}) - J(\pi) \leq \sum_h \epsilon \left\| \frac{d_h^{\pi_{ref}}}{\mu_h} \right\|_\infty$$

11.2 Policy Gradients

Another paradigm for reinforcement learning is to directly optimize the policy, known as *Policy Gradients*. In this approach, we parameterize the policy as

$$\pi_\theta(a \mid s) = \pi(a \mid s; \theta),$$

where θ denotes the policy parameters.

A trajectory (or episode) is defined as:

$$\tau = (s_0, a_0, s_1, a_1, \ldots, s_{H-1}, a_{H-1}),$$

and the performance objective is given by:

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{h=0}^{H-1} r_h\right] = \mathbb{E}_{\tau \sim \pi_\theta}\left[R(\tau)\right],$$

where $R(\tau)$ denotes the return of trajectory τ.

11.2.1 High-level Idea

The fundamental idea behind policy gradient methods is to update the policy parameters using gradient ascent. In its simplest form, the update rule is:

$$\theta_{t+1} = \theta_t + \eta\, \nabla_\theta J(\pi_{\theta_t}),$$

where η is the learning rate (step-size). In order to apply gradient ascent, it is necessary to make $J(\pi_\theta)$ differentiable with respect to θ.

There are several ways to parameterize the policy:

1. **Tabular Case:**
   When the state and action spaces are small enough to be represented in a table, the policy can be defined as:
   $$\pi_\theta(a \mid s) = \frac{\exp\left(\theta_{s,a}\right)}{\sum_{a'} \exp\left(\theta_{s,a'}\right)}.$$
2. **Log-Linear Policies:**
   In this setting, a feature vector $\phi_{s,a}$ is associated with each state-action pair (s, a). The policy is then defined as:
   $$\pi_\theta(a \mid s) = \frac{\exp\left(\langle \theta, \phi_{s,a} \rangle\right)}{\sum_{a'} \exp\left(\langle \theta, \phi_{s,a'} \rangle\right)}.$$
3. **Neural Softmax Policies:**
   For more complex scenarios, a neural network can be used to parameterize the policy:
   $$\pi_\theta(a \mid s) = \frac{\exp\left(f_\theta(s, a)\right)}{\sum_{a'} \exp\left(f_\theta(s, a')\right)},$$
   where $f_\theta(s, a)$ is a function approximated by a neural network.

11.2.2 Warm Up

Consider a simplified objective function defined as:

$$J(\theta) = \mathbb{E}_{x \sim P_\theta}\left[f(x)\right].$$

Taking the gradient with respect to θ, we have:

$$\nabla_\theta J(\theta) = \nabla_\theta \sum_x P_\theta(x) f(x) = \sum_x \nabla_\theta P_\theta(x)\, f(x).$$

Using the identity

$$\nabla_\theta P_\theta(x) = P_\theta(x) \nabla_\theta \ln P_\theta(x),$$

we obtain:

$$\nabla_\theta J(\theta) = \sum_x P_\theta(x)\, \nabla_\theta \ln P_\theta(x)\, f(x) = \mathbb{E}_{x \sim P_\theta}\left[f(x) \nabla_\theta \ln P_\theta(x)\right].$$

11.2.3 Policy Gradient Theorem

Theorem 16 (Policy Gradient Theorem)

$$\text{(REINFORCE)}\ \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim d^{\pi_\theta}}\left[\nabla_\theta \left(\sum_{h=0}^{H-1} \ln \pi_\theta(a_h \mid s_h) \cdot R(\tau)\right)\right].$$

Equivalently,

$$\text{(ADVANTAGE)}\ \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim d^{\pi_\theta}}\left[\nabla_\theta \left(\sum_{h=0}^{H-1} \ln \pi_\theta(a_h \mid s_h) \cdot A_h^{\pi_\theta}(s_h, a_h)\right)\right]$$

where $A_h^{\pi_\theta}(s_h, a_h) = Q_h^{\pi_\theta}(s_h, a_h) - V^{\pi_\theta}(s_h)$.

Proof:

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \mathbb{E}_\tau[\nabla_\theta \ln p_\theta(\tau) \cdot R(\tau)] \\
&= \mathbb{E}_\tau[\nabla_\theta(\ln \mu(s_0) + \ln \pi_\theta(a_0 \mid s_1) + \cdots + \ln \pi_\theta(a_{H-1} \mid s_{H-1}) + \ln p(s_H \mid a_{H-1}, s_{H-1})) \cdot R(\tau)] \\
&= \mathbb{E}_\tau\left[\nabla_\theta \left(\sum_{h=0}^{H-1} \ln \pi_\theta(a_h \mid s_h) \cdot R(\tau)\right)\right].
\end{aligned}$$

■

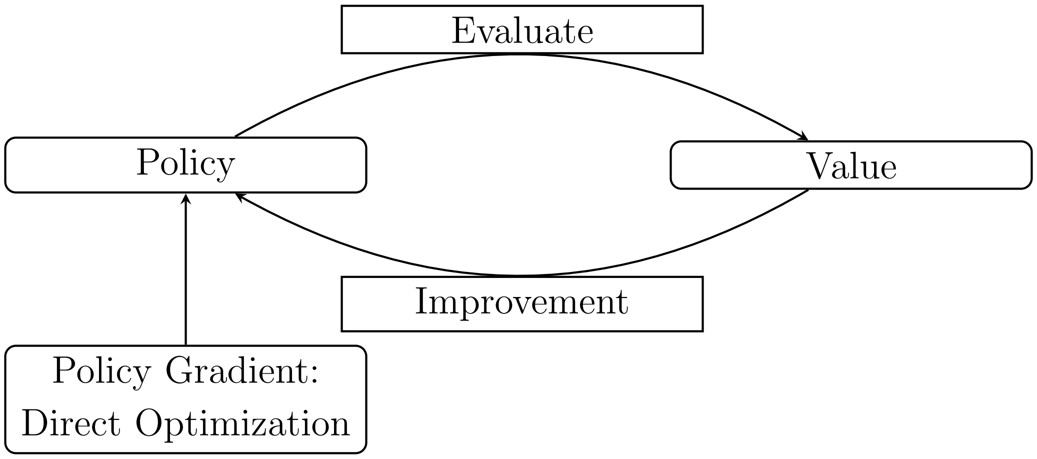
Interpretation $\sum_h \ln \pi_\theta(a_h \mid s_h)$ is a maximum likelihood estimation (MLE). Therefore, $\sum_h \ln \pi_\theta(a_h \mid s_h) A(s_h, a_h)$ can be viewed as some kind of advantage-weighted MLE.

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 12: Policy Gradient and Actor-Critic Method

Lecturer: Steven Wu
Scribe: Nuoya Xiong, Anupam Nayak, Lujing Zhang,Vansh Kapoor

12.1 Policy Gradient



In reinforcement learning, many algorithms, such as policy iteration, revolve around two key objects: policies and value functions. Policy iteration, for instance, alternates between *policy evaluation* and *policy improvement*. In contrast, policy gradient methods primarily focus on directly optimizing policy parameters. While we will momentarily step away from the interplay between value functions and policies, the second part of this lecture will reintroduce value functions as a tool for variance reduction.

To recep policy gradient methods improve $J(\pi_\theta)$ by performing policy gradient ascent update:

$$\theta = \theta + \eta \nabla_\theta J(\pi_\theta).$$

A few key points to consider are that $J(\pi_\theta)$ is almost always non-convex and must be estimated using sampled data. Methods that directly optimize the policy are particularly useful in scenarios where maintaining an explicit value function is unnecessary or impractical. This is often the case in applications to language models.

12.2 Notation

The policy parameterized by θ is represented as $\pi_\theta(a|s) = \pi(a|s,\theta)$. P_θ represents the distribution induced by π_θ over trajectories

$$\tau = (s_0, a_0, r_0, \cdots, s_{H-1}, a_{H-1}, r_{H-1}, S_H).$$

The objective function is defined by

$$J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta} \underbrace{\left[\sum_{h=0}^{H-1} r_h\right]}_{R(\tau)}.$$

12.3 Policy Gradient Theorem

The REINFORCE algorithm computes the gradient by

$$\text{(REINFORCE)} : \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim P_\theta}\left[R(\tau)\cdot\left(\sum_{h=0}^{H-1}\nabla_\theta \log(\pi(a_h|s_h))\right)\right].$$

Observations:

$$\begin{aligned}
&\sum_{h=0}^{H-1}\nabla \log\pi_\theta(a_h|s_h)\cdot\left(\sum_{h=0}^{H-1} r_h\right)\\
&\quad=\sum_{h=0}^{H-1}\nabla \log\pi_\theta(a_h|s_h)\cdot\left(\sum_{h=0}^{H-1} r_h\right)\\
&\quad=\sum_{h=0}^{H-1}\nabla \log\pi_\theta(a_h|s_h)\cdot\left(\underbrace{\sum_{h'=0}^{h-1} r_{h'}}_{A}+\underbrace{\sum_{h'=h}^{H-1} r_{h'}}_{B}\right).
\end{aligned}$$

The term A is unaffected by a_h, and the term b in expectation is $Q_h^{\pi_\theta}(s_h, a_h)$. When we take the expectation outside, the term A would be zero (as we will show later). Hence, we can derive the Q-version of REINFORCE as

$$\text{(Q-Version)} : \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim P_\theta}\left[\sum_{h=0}^{H-1} Q_h^{\pi_\theta}(s_h, a_h)\nabla_\theta\log(\pi(a_h|s_h))\right].$$

Now if $b(s)$ is unaffected by the action a, we can derive

$$\begin{aligned}\mathbb{E}_{a\sim\pi(\cdot|s)}\left[b(s)\cdot\nabla_\theta\log\pi(a|s)\right] &= \sum_a b(s)\cdot\pi(a|s)\cdot\frac{1}{\pi(a|s)}\nabla_\theta(\pi(a|s)\\ &= b(s)\sum_a \nabla_\theta\pi_\theta(a|s)\\ &= b(s)\nabla_\theta\sum_\theta \pi_\theta(a|s)\\ &= 0.\end{aligned}$$

The last equation is because $\sum_a \pi(a|s) = 1$. Hence, we can subtract a baseline $b_h(s) = V_h(s_h)$ that is independent with a_h, and get the A-version of REINFORCE as

$$\text{(A-Version)} : \nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim P_\theta}\left[\sum_{h=0}^{H-1} A_h^{\pi_\theta}(s_h,a_h)\nabla_\theta\log(\pi(a_h|s_h))\right],$$

where $A_h^{\pi_\theta}(s_h,a_h) = Q_h^{\pi_\theta}(s_h,a_h) - V_h^{\pi_\theta}(s_h)$.

12.4 Estimation of Policy Gradient

In practice, we cannot usually get the exact full gradient. Hence, we need to derive an estimator of the policy gradient.

12.4.1 Monte-Carlo Estimation

Using Monte-Carlo estimation, we first roll out N trajectories

$$\{\tau_i = (s_0^{(i)}, a_0^{(i)}, r_0^{(i)}, \cdots, s_{H-1}^{(i)}, a_{H-1}^{(i)}, r_{H-1}^{(i)}, s_H^{(i)})\}_{i\in[N]}$$

following the policy π_θ. Then, the gradient can be estimated by

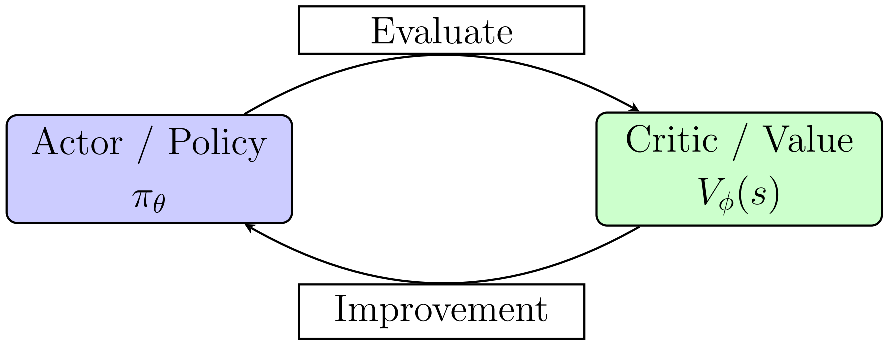
$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_{i=1}^N\sum_h \nabla_\theta\log\pi_\theta(a_h^{(i)}|s_h^{(i)})\cdot R(\tau_i).$$

You can also get the Q function by $\hat{Q}_h^{(i)} = \sum_{h'=h}^{H-1} r_{h'}^{(i)}$ and estimate the gradient by

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N}\sum_{i=1}^N\sum_h \nabla_\theta\log\pi_\theta(a_h^{(i)}|s_h^{(i)})\cdot\hat{Q}_h^{(i)}.$$

However, the Monte-Carlo sampling method often results in a gradient estimate with high variance. We will now introduce methods that reduce variance by introducing learned value functions V_ϕ as a baseline.

12.4.2 Actor-Critic Method



We define the actor/policy as π_θ, and the critic/value as $V_\phi(s)$. In practice, the actor and the critic are both modeled by neural networks.

When we have the critic/value function, we can estimate the policy gradient as

$$\nabla_\theta J(\theta) \approx \sum_{h=0}^{H-1} \nabla_\theta \log \pi(a_h|s_h) \cdot \left(\underbrace{r_h + V_\phi(s_{h+1}) - V_\phi(s_h)}_{\text{Temporal Difference Error}} \right),$$

where the **T**emporal **D**ifference (TD) error is an estimate of $A_h^{\pi_\theta}(s_h, a_h)$.

Note that the TD error is also an estimate of the *Bellman error*: $V(s) - \mathbb{E}_{a\sim\pi, s'\sim P}\left[r(s,a) + \gamma V(s')\right]$. This motivates an optimization objective for the *critic*, who tries to minimize the square loss given by

$$L(\phi) = \mathbb{E}\left[(r_h + V_\phi(s_{h+1}) - V_\phi(s_h))^2\right].$$

The critic will optimize this objective using the *semi-gradient method* which treats $V_\phi(s_{h+1})$ as a constant, ignoring its dependency when computing the gradient.

The full pseudocode is shown in the following algorithm:

Algorithm 5 Actor-Critic Method

```
1:  **Input:** Learning rate $\eta_\phi, \eta_\theta$.
2:  for episode $= 1, 2, \cdots, N$ do
3:      for $h = 1, 2, \cdots, H-1$ do
4:          Critic Update:
5:          $\delta_h = r_h + V_\phi(s_{h+1}) - V_\phi(s_h)$
6:          $\phi \leftarrow \phi + \eta_\phi \cdot \delta_h \nabla_\phi V_\phi(s_h)$.
7:          Actor Update:
8:          $\theta \leftarrow \theta + \eta_\theta \cdot \delta_h \nabla_\theta \log \pi_\theta(a_h|s_h)$.
9:      end for
10: end for
```

Suppose we execute the policy π_θ to get a trjaectory:

$$\pi_\theta \to s_0, a_0, r_0, \cdots, r_1, \cdots, r_2, \cdots, r_{H-1}, S_H.$$

At each step $h \in [H-1]$, you can receive a new reward r_h and get the (empirical) value function $V_\phi(s_h)$, and get the one-step advantage function $r_{h-1} + V_\phi(s_h) - V_\phi(s_{h-1})$.

One alternative of this one-step advantage function is called the **Multi-step** advantage function:

$$A^{(n)}(s_h, a_h) = r_h + \cdots + r_{h+n-1} + V_\phi(s_{h+n}) - V_\phi(s_h).$$

This gives us some form of interpolation between the Monte-Carlo estimation and the TD(0).

12.4.3 Generalized Advantage Estimation

We provide a more generalized advantage estimation $\hat{A}_h(\lambda)$ as

$$\hat{A}_h(\lambda) = \delta_h + \lambda\delta_{h+1} + \cdots + \lambda^{H-1-h}\delta_{H-1}.$$

When $\lambda = 0$, $\hat{A}_h(0)$ is the TD error, corresponding to the Actor-Critic Method.

When $\lambda = 1$, $\hat{A}_h(1)$ is the Monte-Carlo Estimation Approach.

With $\lambda = 0$, the estimate is a one-step TD error—highly biased due to its short-term focus but with low variance from stable updates. At $\lambda = 1$, it becomes a Monte Carlo estimator, reducing bias by considering full returns but increasing variance from accumulated stochastic noise. Hence, $\hat{A}_h(\lambda)$ is a trade-off between the high-bias estimate TD error and the high-variance Monte-Carlo estimator.

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 13: Natural Policy Gradient (NPG)

Lecturer: Steven Wu
Scribe: Zora Wang, Lindia Tjuatja, Grace Liu, Nuoya Xiong

13.1 Introduction

This lecture presents three views of how one might arrive at the Natural Policy Gradient (NPG) algorithm.

- KL Regularization for stability
- Least squares regression
- Soft Policy Iteration (Hedge at every state)

Then we will present "practical" algorithms inspired by NPG

- Trust Region Policy Optimization (TRPO)
- Proximal Policy Optimization (PPO)

13.2 Natural Policy Gradient

Recall that

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \mathbb{E}_{\tau\sim P_\theta}\left[\sum_{h=0}^{H-1} A_h^{\pi_\theta}(s_h,a_h)\nabla_\theta \log(\pi(a_h \mid s_h))\right] \\ &= H\cdot \mathbb{E}_{h\sim U[H-1], s_h\sim d_h^{\pi_\theta}, a_h\sim\pi_\theta(s_h)}[A_h^{\pi_\theta}(s_h,a_h)\nabla_\theta\log\pi_\theta(a_h\mid s_h)].\end{aligned}$$

Also recall the policy gradient update:

$$\theta_{t+1} \leftarrow \theta_t + \eta\nabla_\theta J(\pi_\theta).$$

13.2.1 KL Regularization View

The **regularization view** of the above equation is

$$\theta_{t+1} = \arg\max_{\theta} \langle \nabla_\theta J(\pi_{\theta_t}), \theta \rangle - \frac{1}{2\eta} \|\theta - \theta_t\|^2.$$

Notice that the above is a concave function. We can show that the regression view is equivalent to the policy gradient by solving this concave maximization problem, i.e. taking the derivative with respect to θ and setting it equal to zero.

We can explicitly enforce a hard limit on how far the parameters move by adding a constraint, giving us the **constrained optimization view**:

$$\max_{\theta} \langle \nabla_\theta J(\pi_{\theta_t}), \theta \rangle, \quad s.t. \quad \|\theta - \theta_t\|_2^2 \leq \delta.$$

Both the regularization view and constrained optimization view are looking at changes in parameter space. However, it can very well be the case that two sets of very different parameters lead to near-identical policies.

A more principled approach would instead constrain the differences between policies themselves. We can do this by constraining the **KL-divergence** between the old and new policies, instead of the ℓ_2 norm of their parameters:

$$\max_{\theta} J(\pi_\theta), \quad s.t. \quad \frac{1}{H} \mathrm{KL}(\mathbb{P}^{\pi_{\theta_t}} \| \mathbb{P}^{\pi_\theta}) \leq \delta,$$

By decomposing the KL divergence, we can get

$$\begin{aligned}
\frac{1}{H} \mathrm{KL}(\mathbb{P}^{\pi_{\theta_t}} \| \mathbb{P}^{\pi_\theta}) &= \frac{1}{H} \sum_{\tau} \mathbb{P}^{\theta_t}(\tau) \log \frac{\mathbb{P}^{\theta_t}(\tau)}{\mathbb{P}^{\theta}(\tau)} \\
&= \frac{1}{H} \sum_{\tau} \mathbb{P}^{\theta_t}(\tau) \sum_{h=0}^{H-1} \log \frac{\pi_{\theta_t}(a_h \mid s_h)}{\pi_\theta(a_h \mid s_h)} \\
&= \mathbb{E}_{(s_h, a_h) \sim d^{\pi_{\theta_t}}} \left[\log \frac{\pi_{\theta_t}(a_h \mid s_h)}{\pi_\theta(a_h \mid s_h)} \right] \\
&:= \ell(\theta).
\end{aligned}$$

How can we approximate $\ell(\theta)$? We can use a Taylor expansion of $\ell(\theta)$ around $\theta = \theta_t$:

$$\begin{aligned}
\ell(\theta_t) &= 0 \\
\nabla_\theta \ell(\theta) &= 0|_{\theta=\theta_t} \\
\nabla_\theta^2 \ell(\theta_t) &= \mathbb{E}\left[\nabla_\theta \log \pi_{\theta_t}(a_h \mid s_h) \nabla_\theta \log \pi_{\theta_t}(a_h \mid s_h)^\top \right]
\end{aligned}$$

The Hessian $\nabla^2_\theta \ell(\theta_t)$ is also referred to as the *Fisher Information Matrix*, denoted as $F(\theta_t)$. Hence, by the second-order approximation, we can estimate $\ell(\theta)$

$$\frac{1}{H}\mathrm{KL}(\mathbb{P}^{\pi_{\theta_t}} \| \mathbb{P}^{\pi_\theta}) \approx (\theta - \theta_t)^\top F(\theta_t)(\theta - \theta_t).$$

Thus, the update rule of *natural policy gradient* (NPG) can be written as

$$\theta_{t+1} \leftarrow \theta_t + \eta F^\dagger \nabla_\theta J(\pi_{\theta_t}),$$

where $F^\dagger$ is the Moore-Penrose Inverse.

13.2.2 Regression View

We can also view NPG as solving a weighted least squares regression problem.

Lemma 17 *Define*

$$w^* = \arg\min_w \mathbb{E}_{(s_h,a_h)\sim d^{\pi_\theta}} \left[\left(\underbrace{A_h^{\pi_\theta}(s_h,a_h)}_{\text{label}} - w^\top \underbrace{\nabla_\theta \log \pi_\theta(a_h \mid s_h)}_{\text{feature}} \right)^2 \right].$$

Then, $F^\dagger(\theta)\nabla_\theta J(\pi_\theta) = Hw^*$.

Proof. By first order condition:

$$\mathbb{E}\left[\left(A_h^{\pi_\theta}(s_h,a_h) - (w^*)^\top \nabla_\theta \log \pi_\theta(a_h \mid s_h) \right) \nabla_\theta \log \pi_\theta(a_h \mid s_h) \right] = 0$$

It is equivalent to

$$\begin{aligned} &\underbrace{\mathbb{E}\left[\left(A_h^{\pi_\theta}(s_h,a_h) \right) \nabla_\theta \log \pi_\theta(a_h \mid s_h) \right]}_{\frac{1}{H}\nabla_\theta J(\pi_\theta)} \\ &\quad = \underbrace{\mathbb{E}\left[\nabla_\theta \log \pi_\theta(a_h \mid s_h) \nabla_\theta \log \pi_\theta(a_h \mid s_h)^\intercal \right]}_{Fisher\ Matrix\ F(\theta)} w^*. \end{aligned}$$

Hence we have $F^\dagger(\theta)\nabla_\theta J(\pi_\theta) = Hw^*$.

The lemma above shows that the NPG update is indeed a regression problem under some linear transformation.

Invariance under reparameterization. The regression view of NPG also provides a way to demonstrate that NPG is invariant under affine reparameterization. An *affine reparameterization* of the policy parameters transforms θ via an invertible affine map:

$$\theta' = M\theta + b$$

where $M \in \mathbb{R}^{d\times d}$ is an *invertible* matrix, and $b \in \mathbb{R}^d$ is an arbitrary translation vector. In other words, the policy can be written as $\pi'(\theta') = \pi(M^{-1}(\theta' - b)) = \pi(\theta)$. Suppose we have $\theta'_t = M\theta_t + b$ and $\pi_{\theta'_t} = \pi_{\theta_t}$ (that is the parameterized policies are the same). By the chain rule of calculus, we have

$$\nabla_{\theta'} \log \pi'_{\theta'}(a \mid s) \mid_{\theta'=\theta'_t} = M^{-1}\nabla_\theta \log \pi_\theta(a \mid s) \mid_{\theta=\theta_t} .$$

Since least squares regression is *invariant under linear transformations* of the feature space, the transformation $\nabla_{\theta'} \log \pi'_{\theta'}(a \mid s) = M^{-1}\nabla_\theta \log \pi_\theta(a \mid s)$ does not change the optimal solution w^* when properly parameterized. In particular, the transformed solution becomes:

$$w'^*_t = M w^*_t$$

In other words, after an NPG update, we continue to have $\theta'_{t+1} = M\theta_{t+1} + b$. This invariance property is also called *covariant.*

As an exercise, you can show that policy gradient does not enjoy this invariance property.

13.2.3 Soft Policy Iteration

Assume the softmax policy parameterization in the tabular MDP setting: $\pi_\theta(a_h \mid s_h) \propto \exp(\theta_{s_h,a_h})$. Then, we have the following lemma showing the relationship between NPG and SPI.

Lemma 18 *The update rule $\theta_{t+1} \leftarrow \theta_t + \eta H A^t$ is equivalent to*

$$\pi^{t+1}(a_h \mid s_h) \propto \pi^t(a_h \mid s_h) \cdot \exp(\eta H A^t_h(s_h, a_h)).$$

Thus, we can view NPG as running a copy of the hedge algorithm at every state s_h.

To show this result, we can look at the regression view of the NPG algorithm and find out that the advantage function is a solution to the least squares regression problem.

13.3 "Practical" Algorithms

13.3.1 Trust Region Policy Optimization (TRPO)

TRPO is the precursor to PPO, motivated by the NPG update. TRPO solves a constrained optimization problem that explicitly bounds the KL divergence between the updated policy and the current policy:

$$\begin{aligned} \max_{\theta} \quad & \mathbb{E}_{(s,a)\sim d^{\pi_{\theta_t}}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_t}(a|s)} A^{\pi_{\theta_t}}(s,a)\right] \\ \text{s.t.} \quad & \mathbb{D}_{\mathrm{KL}}(\pi_{\theta_t} || \pi_\theta) \le \delta \end{aligned}$$

The objective function is called surrogate advantage objective, which was also used in *conservative policy iteration* covered in a previous lecture. One way to make sense of the objective function is to start with the performance difference lemma:

$$J(\pi_\theta) - J(\pi_{\theta_t}) = H\mathbb{E}_{s\sim d^{\pi_\theta}}\left[\sum_a \left(\pi_\theta(a \mid s) - \pi_{\theta_t}(a \mid s)\right) Q^{\pi_{\theta_t}}(s,a)\right]$$

To ensure policy improvement, we want the performance difference above to be positive. An immediate difficulty for optimizing this quantity is that the expectation is over the state distribution d^{π_θ} induced by the new policy π_θ. Since TRPO ensures that the KL divergence between the new and old policies is small, one strategy is to replace the state distribution by $d^{\pi_{\theta_t}}$. This gives us

$$\begin{aligned} J(\pi_\theta) - J(\pi_{\theta_t}) &= H\mathbb{E}_{s\sim d^{\pi_{\theta_t}}}\left[\sum_a \left(\pi_\theta(a \mid s) - \pi_{\theta_t}(a \mid s)\right) Q^{\pi_{\theta_t}}(s,a)\right] \\ &= H\mathbb{E}_{s\sim d^{\pi_{\theta_t}}, a\sim \pi_{\theta_t}(s)}\left[\left(\frac{\pi_\theta(a \mid s) - \pi_{\theta_t}(a \mid s)}{\pi_{\theta_t}(a \mid s)}\right) Q^{\pi_{\theta_t}}(s,a)\right] \\ &= H\mathbb{E}_{s\sim d^{\pi_{\theta_t}}, a\sim \pi_{\theta_t}(s)}\left[\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_t}(a \mid s)}\right) A^{\pi_{\theta_t}}(s,a)\right] \end{aligned}$$

which is exactly the objective function in TRPO (up to a factor of H). Similar to NPG, TRPO will also approximate the KL constraint with the Fisher information matrix, which leads to a quadratic constraint

$$(\theta - \theta_t)^\top F(\theta_t)(\theta - \theta_t) \le \delta$$

Then TRPO proceeds to solve the quadratically constrained problem by solving the Lagrangian (which we saw in the information theory lecture). Similar to NPG, this step requires computing $F^{\dagger}\nabla_{\theta}J(\pi_{\theta_t})$, which can be costly due to the computation of a matrix inverse. TRPO instead solves the associated linear system:

$$Fx = \nabla_{\theta}J(\pi_{\theta_t})$$

using the conjugate gradient method, which iteratively finds x without inverting F.

13.3.2 Proximal Policy Optimization (PPO)

PPO is a somewhat "hacky" approximation of TRPO. The motivation is to prevent the policy from changing too much without doing explicitly KL-constrained optimization. Instead, PPO introduces *clipping*:

$$\hat{R}_{\mathbf{clip}} = \mathbb{E}_{(s,a)\sim d^{\pi_{\theta_t}}}\left[\underbrace{\text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_t}(a|s)}\right)A^{\pi_{\theta_t}}(s,a)}_{L(\theta,s,a)}\right]$$

where **clip** projects the value into the interval $[1-\epsilon, 1+\epsilon]$, and ϵ is a small positive hyperparameter (e.g., 0.1 or 0.2).

The PPO objective then becomes:

$$\hat{R}_{\mathbf{PPO}} = \mathbb{E}_{(s,a)\sim d^{\pi_{\theta_t}}}\left[\min\left\{\frac{\pi_\theta(a|s)}{\pi_{\theta_t}(a|s)}A^{\pi_{\theta_t}}(s,a), \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_t}(a|s)}\right)A^{\pi_{\theta_t}}(s,a)\right\}\right]$$

We analyze the effect of the min operation separately for positive and negative advantage values. Let the probability ratio be defined as:

$$r_\theta(s,a) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\theta_t}}(a|s)},$$

Case 1: Positive Advantage ($A(s,a) > 0$) If the advantage is positive, we prefer to increase the probability of the chosen action. The objective $L(\theta, s, a)$ simplifies to:

$$A(s,a)\min\left(r_\theta(s,a),\, 1+\varepsilon\right).$$

Analyzing the behavior based on the value of r_θ:

$$L(\theta,s,a) = \begin{cases} r_\theta(s,a)A(s,a), & \text{if } r_\theta(s,a) \le 1+\varepsilon \\ (1+\varepsilon)A(s,a), & \text{if } r_\theta(s,a) > 1+\varepsilon \end{cases}$$

Interpretation:

- If the probability ratio is within the threshold ($\leq 1 + \varepsilon$), the objective increases proportionally as we increase r_θ.
- If r_θ surpasses $1+\varepsilon$, the objective saturates at $(1+\varepsilon)A(s, a)$. Thus, there is no incentive for the policy to further increase this action's probability.

Case 2: Negative Advantage ($A(s, a) < 0$) If the advantage is negative, ideally we want to decrease the probability of the chosen action. The objective is $L(\theta, s, a)$ simplifies to:

$$A(s, a) \max\left(r_\theta(s, a),\, 1 - \varepsilon\right) \quad \text{(since } A(s, a) < 0 \text{ reverses the inequality).}$$

We analyze based on the value of r_θ:

$$L(\theta, s, a) = \begin{cases} r_\theta(s, a)A(s, a), & \text{if } r_\theta(s, a) \geq 1 - \varepsilon \\ (1 - \varepsilon)A(s, a), & \text{if } r_\theta(s, a) < 1 - \varepsilon \end{cases}$$

Interpretation:

- If the probability ratio is within the lower bound ($\geq 1 - \varepsilon$), the objective improves as we reduce r_θ, since this makes $r_\theta A(s, a)$ less negative (larger).
- If r_θ falls below $1 - \varepsilon$, the objective hits the floor $(1 - \varepsilon)A(s, a)$. Thus, reducing the probability further yields no additional improvement.

The min operation together with the clip operation ensures stable and conservative updates by limiting the magnitude of policy changes in each optimization step.

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 15: Model-Based RL

Lecturer: Gokul Swamy
Scribe: Yuemin Mao, Miaosi Dong, Khush Agrawal, Eryn Ma

Model-based RL is a subclass of reinforcement learning where the agent learns a dynamics model of the environment. This model can be used to simulate future states, calculate rewards, and optimize action selection accordingly without ever interacting with the actual environment.

15.1 What's a "model"?

Recall a Markov Decision Process (MDP) that can be defined as:

$$\begin{aligned} M = \{ & S \quad \text{(state space)} \\ & A \quad \text{(action space)} \\ & r \quad \text{(reward function)} \\ & \mathcal{T} \quad \text{(transition dynamics)} \\ & H \quad \text{(horizon)} \\ & \rho_0 \quad \text{(initial state distribution)}\} \end{aligned}$$

The goal of model-based RL is to learn the transition dynamics ($\mathcal{T}$) of the environment. This model can be used to evaluate trajectories without interacting with the actual environment.

1. An MDP includes a *transition function / dynamics* $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, which has $|S|^2|A|$ elements to learn. This is a lot more than learning a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$, which only has $|S||A|$ elements, increasing the data required.
2. *Generative model* access to an MDP allows an agent to query the "model" at any preferred state-action pair to get predictions of the next state (i.e. samples from the next state distribution). Learning a dynamics model gives us such access to the underlying MDP we're trying to solve.
3. Model-based RL (MBRL) is performing RL in an MDP M' is M but with ground truth T replaced with learned T', which is approximated. Alternatively, it could also involve

test time planning (e.g., model predictive control) for finding actions that optimize an objective function inside M' – we will discuss this in detail in a future lecture.

15.2 What makes a good model?

A perfect model would be the one returning predictions exactly the same as the ground truth. However, in the real-world, learning problems are prone to errors. Since we cannot model everything accurately, it is better to give more importance to the regions of the state and action spaces visited by the policy we're evaluating. To summarize, a "good" model is one that allows us to accurately evaluate the performance of a policy: *a policy looks performant in a good model if and only if it is performant in the real world.* We can make this intuition more precise via the following lemma, which tells us that if we've learned a model that can accurately predict state transitions everywhere [4], we will achieve the preceding desiderata.

Lemma 19 (Simulation Lemma, Kearns & Singh, 2002) *Suppose that for all (s, a) in state action spaces, the total variation distance between the learned model and the ground truth is less than ϵ, $\sum_{s'} |T'(s'|s,a) - T(s'|s,a)| \leq \epsilon$. Then for any policy $\pi : \mathcal{S} \to \mathcal{A}$, we have*

$$|J(\pi, M) - J(\pi, M')| \leq \frac{H(H-1)}{2}\epsilon \tag{15.78}$$

Proof: If the following inequality holds for all states, then it also holds in expectation over the initial state distribution, since expectation preserves inequalities. Therefore, in the proof, we bound the value difference pointwise over states, which allows us to derive a bound on the performance difference by averaging these pointwise bounds over the initial states:

$$|V_h^\pi(s, M) - V_h^\pi(s, M')| \leq (H-h)\epsilon + \left|V_{h+1}^\pi(s, M) - V_{h+1}^\pi(s, M')\right| \tag{15.79}$$

We prove Eqn. 15.79 using backwards induction. The following lemmas are used in the proof:

Lemma 20 (Triangle Inequality) $|a - b| = |a - c + c - b| \leq |a - c| + |c - b|$

Lemma 21 (Holder's Inequality) $\sum_x p(x)g(x) =< p, g >\leq \|p\|_1 \cdot \|g\|_\infty$

Base Case:

$$h = H \Rightarrow LHS = 0 \leq 0 = RHS \tag{15.80}$$

Inductive Hypothesis: Assume for $h + 1$, we have:

$$|V_{h+1}^\pi(s, M) - V_{h+1}^\pi(s, M')| \leq (H - h - 1)\epsilon. \tag{15.81}$$

[4]One can easily modify this proof to add an expectation over any desired state distribution.

Inductive Step: Expanding the value functions via Bellman equation:

$$
\begin{aligned}
|V_h^\pi(s,M) - V_h^\pi(s,M')| = \big|&\mathbb{E}_{a\sim\pi(s),s'\sim T(s,a)}[r(s,a) + V_{h+1}^\pi(s',M)] && (15.82)\\
&-\mathbb{E}_{a\sim\pi(s),s'\sim T'(s,a)}[r(s,a) + V_{h+1}^\pi(s',M')]\big| && (15.83)
\end{aligned}
$$

Next, we observe that the reward function does not depend on the next states, so the expectation of $r(s,a)$ in both terms are the same.

$$
\begin{aligned}
= \big|&\mathbb{E}_{a\sim\pi(s),s'\sim T(s,a)}[\cancel{r(s,a)} + V_{h+1}^\pi(s',M)] && (15.84)\\
&-\mathbb{E}_{a\sim\pi(s),s'\sim T'(s,a)}[\cancel{r(s,a)} + V_{h+1}^\pi(s',M')]\big|\,. && (15.85)
\end{aligned}
$$

For simplicity, we upper bound the expectation with the maximum over actions:

$$
\leq \max_a \left|\sum_{s'} T(s'|s,a)V_{h+1}^\pi(s',M) - T'(s'|s,a)V_{h+1}^\pi(s',M')\right|. \qquad (15.86)
$$

We now proceed by applying Lemma 20. Let $a = \sum_{s'} T(s'|s,a)V_{h+1}^\pi(s',M)$, $b = \sum_{s'} T'(s'|s,a)V_{h+1}^\pi(s',M')$, $c = \sum_{s'} T'(s'|s,a)V_{h+1}^\pi(s',M)$. Then, we have:

$$
\begin{aligned}
\leq \max_a &\left|\sum_{s'} T(s'|s,a)V_{h+1}^\pi(s',M) - T'(s'|s,a)V_{h+1}^\pi(s',M)\right| && (15.87)\\
&+\left|\sum_{s'} T'(s'|s,a)V_{h+1}^\pi(s',M) - T'(s'|s,a)V_{h+1}^\pi(s',M')\right| && (15.88)\\
& && (15.89)
\end{aligned}
$$

Next, we apply Lemma 21. Define $p = T(s'|s,a) - T'(s'|s,a)$, $q = V_{h+1}^\pi(s',M)$. Then, $|p| \leq \epsilon$ by assumption, and the maximum value of the value function, $|q|$, is at most $H-h$, hence the first term is upper bounded by $\epsilon(H-h)$. By assumption, the 2nd term is upper bounded by the last step. Intuitively, this is because if we have a bound on the probability of a difference and the maximum amount we can pay per difference, we can in the worst case pay the product of those two quantities. Thus:

$$
\begin{aligned}
\leq \max_a &\left|\underbrace{\sum_{s'} T(s'|s,a)V_{h+1}^\pi(s',M) - T'(s'|s,a)V_{h+1}^\pi(s',M)}_{\leq\epsilon(H-h)}\right| && (15.90)\\
&+\left|\underbrace{\sum_{s'} T'(s'|s,a)V_{h+1}^\pi(s',M) - T'(s'|s,a)V_{h+1}^\pi(s',M')}_{\leq\text{value of the last step}}\right| && (15.91)
\end{aligned}
$$

We can now expand out the recursion over h to complete the proof:

$$|J(\pi, M) - J(\pi, M')| \leq \sum_{h}^{H} (H - h)\epsilon = \frac{H(H-1)}{2}\epsilon. \tag{15.92}$$

■

15.3 How do we fit a good model?

Prima facie, one might like the model to be accurate on the current policy's state distribution. However, just doing this is not enough. Once we learn a model, we also run RL steps that optimize the policy using this approximated model. When the model is inaccurate, the RL step might exploit such inaccuracies and lead to poor performance in the real world.

This can work extremely poorly. Similar to approximate policy iteration, and the follow the leader counter-example, the model might switch back and forth repeatedly, with the RL step never finding a good policy. We now sketch such an example on a tree-structured MDP.

$M^\star$ in Figure 15.9 represents the real world. The optimal policy π_E is highlighted in green. Assume the initial policy we start off with goes right twice, as shown by the highlighted red trajectory in M_0 (Figure 15.10). We'll likely be accurate on states seen in the training distribution, causing us to correctly predict the green label of 0 for going right twice. However, we can be arbitrarily bad on states we didn't train on – for example, we can predict a reward of 1 for going right and then left, as we do in M_0. The optimal policy inside M_0 would then exhibit this behavior, effectively exploiting the inaccuracies in the learned model that happen in OOD states. Then, based on the new policy, we could learn some new M_1 (Figure 15.10). Again, we're correct in-distribution, but could be overly optimistic on the value we'd receive of going right twice. Observe that the optimal policy in M_1 could induce M_0, so we could just oscillate between these two models and their corresponding optimal policies without learning anything useful. Intuitively, this is because a model trained on a single round of data can be too *optimistic* on unseen states.

Instead, we could aggregate all the past datasets we have observed. This is using a no-regret algorithm over models (specifically, Follow the Regularized) Leader) to deal with the "best response" over policies. This is not explicitly game-solving, but the no-regret principle can help us deal with the distribution shift caused by RL exploiting the model. While $M_{no-regret}$ will accurately predict 0 for the two paths it has seen (visualized in Figure 15.11), since it has never observed any data of optimal policy π_E, it could still underestimate the value of going left twice. Intuitively, without seeing data from the expert, we can be overly *pessimistic* about good OOD trajectories. So, while no-regret model fitting guarantees us correct policy

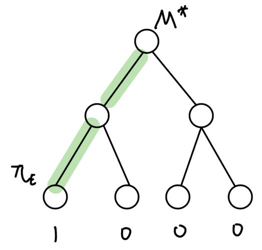


Figure 15.9: The optimal policy π_E goes left twice and recieves reward 1.

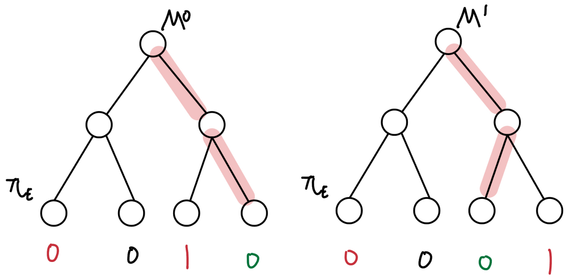


Figure 15.10: Since doing RL exploits the model and can cause it to be queried on states outside of its training distribution, only training on the latest data might lead to a situation where we oscillate between inaccurate models and suboptimal policies. We use a red highlight to visualize the policy used to generate the training data and a green leaf node label to denote we got this reward correct. We use red to denote OOD leaf nodes where we incorrectly predict the reward value. Observe that the optimal policy in M^0 induces M^1 and vice-versa.

evaluation (i.e. we correctly predict both policies get zero reward), it doesn't guarantee we'll compute a policy via RL in the model that is anywhere close to the quality of π_E

Now say if we have samples from π_E is that, how could we use them? Even if we fit our model perfectly on the expert data, if our policy distribution is different (i.e., taking the right path instead of left from the top node), our model will still behave poorly on unseen paths. One way to resolve this tension is to combine expert data with our policy distribution, which is essentially DAgger for MBRL, with extra expert data to reduce the exploration burden on the learner. Fitting the model on the *hybrid* distribution leads to Agnostic SysID:

Algorithm 6 No-Regret MBRL

```
1: Initialize some $\pi_0, \hat{T}_0$
2: for $t = 1, ..., T$ do
3:     (1) Roll out $\pi_{t-1}$ in $T^*$ to collect $D_t$
4:     (2) Fit $\hat{T}_t$ on $\bigcup_{\tau=1}^{t} D_\tau$
5:     (3) Run RL inside $\hat{T}_t$ to compute $\pi_t$
6:         $\rightarrow \pi_t = \arg\max_{\pi \in \Pi} J(\pi, M_t)$        "best response"
7: end for
```

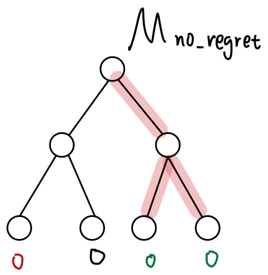


Figure 15.11: A data aggregation data method on the other hand will assign the correct reward values to both the states and not demonstrate an oscillating behavior.

Algorithm 7 Agnostic System Identification (Ross and Bagnell 2012)

```
1: Initialize some $\pi_0, \hat{T}_0$
2: for $t = 1, ..., T$ do
3:     (1) Roll out $\pi_{t-1}$ in $T^*$ to collect $D_t$
4:     (2) Fit $\hat{T}_t$ on $(\bigcup_{\tau=1}^{t} D_\tau)\left(\frac{1}{2}\right) + \frac{1}{2} D_E$
5:     (3) Run RL inside $\hat{T}_t$ to compute $\pi_t$
6:         $\rightarrow \pi_t = \arg\max_{\pi \in \Pi} J(\pi, M_t)$        "best response"
7: end for
```

Intuitively, *we need to be able to accurately predict the consequences of our actions (both good and bad) to learn a good policy.* Fitting the model on the mixture of expert and learner data helps balance optimism and pessimism without computational complexity. We will further explore this idea of *hybrid* RL in the model-free setting during the following guest lecture.

We can provide some more intuition for why this hybrid model-fitting procedure ensures that learn a model such that *policies look good in the model if and only if they are good in*

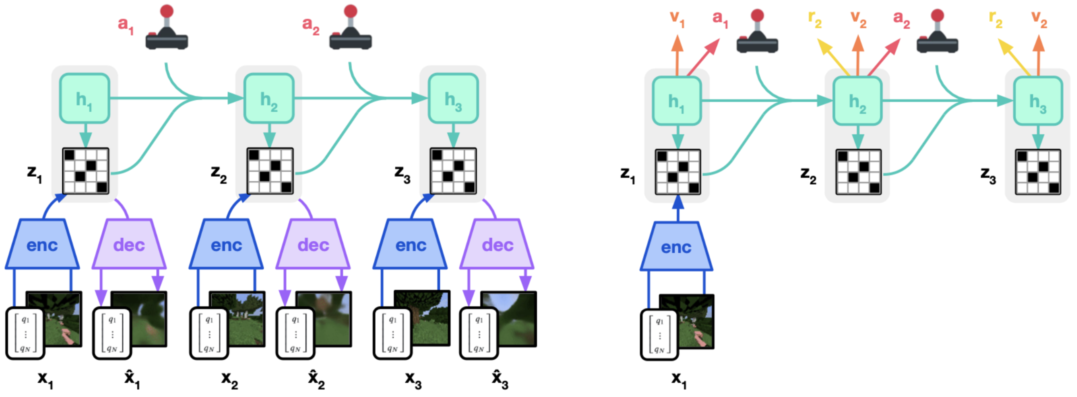


Figure 15.12: Training Process of Dreamer

the real world via the following three-term decomposition:

$$\begin{aligned} J(\pi_E, M) - J(\pi, M) = &(J(\pi_E, M) - J(\pi_E, M')) && \rightarrow \text{inaccuracy on } \pi_E \\ &+ (J(\pi, M) - J(\pi, M')) && \rightarrow \text{inaccuracy on } \pi \\ &+ (J(\pi_E, M') - J(\pi, M')) && \rightarrow \text{planning error} \end{aligned}$$

If the RL solver of our model is perfect, $(J(\pi_E, M') - J(\pi, M') \leq 0$ and the inner term will be non-negative. If we also fit the model well on learner and expert state distributions, $(J(\pi_E, M), J(\pi_E, M')) + ((J(\pi, M) - J(\pi, M'))$ will also be small via the Simulation Lemma. Thus, this idea is sometimes affectionately referred to as the *double simulation lemma*. In the Agnostic SysID paper, Ross & Bagnell use an upper bound on the Simulation Lemma to derive loss functions for the no-regret model fitting procedure on hybrid data (off-policy expert data + on-policy learner data), which boil down to standard MLE on the hybrid data.

15.4 How do we scale this idea?

To address the key challenge that modeling pixels is hard, we can do things in latent space by using sequential VAEs like Dreamer [7]. The training process of Dreamer is illustrated as Figure 15.12. First, an encoder maps sensory inputs x_t to stochastic representations z_t. Then, a sequence model with recurrent state h_t predicts the sequence of these representations given past actions a_{t-1}. The concatenation of h_t and z_t forms the model state from which we predict rewards r_t and episode continuation flags $c_t \in \{0, 1\}$ and reconstruct the inputs to ensure informative representations:

Sequence Model	$h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1})$
Encoder	$z_t \sim q_\phi(z_t \| x_t, h_t)$
Dynamic Prediction	$\hat{z}_t \sim p_\phi(\hat{z}_t \| h_t)$
Reward predictor:	$\hat{r}_t \sim p_\phi(\hat{r}_t \| h_t, z_t)$
Continue predictor:	$\hat{c}_t \sim p_\phi(\hat{c}_t \| h_t, z_t)$
Decoder	$\hat{x}_t \sim p_\phi(\hat{x}_t \| z_t, h_t)$
Dynamic Prediction	$\hat{z}_t \sim p_\phi(\hat{z}_t \| h_t)$

Question 1: Why we need a decoder?

- Without a decoder, the learned latent representations are at risk of collapsing to trivial or degenerate solutions—such as constant vectors—since there is no explicit constraint enforcing the retention of meaningful information from the original observations. In such a scenario, the latent state may become uninformative. Essentially, the decoder acts as a *grounding* mechanism, ensuring that the latent representation remains coupled with the actual observations.

Question 2: Why do we need alternative objectives beyond a decoder?

- While decoders are commonly used to encourage latent representations to retain input information, they can be inefficient for decision-making tasks. For example, reconstructing the entire input image may force the model to encode irrelevant details—such as leaves blowing in the wind—that have no impact on the task, such as driving. This leads to representations that are perceptually rich but behaviorally redundant.
- To obtain minimal sufficient representations for reinforcement learning, alternative objectives are needed. One effective approach is to replace the decoder with an inverse dynamics model, which predicts the action taken between two latent states [8]. This encourages the model to retain only information that is influenced by agent behavior, naturally filtering out task-irrelevant factors, leading to more minimal representations.

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 16: Hybrid RL

Lecturer: Yuda Song
Scribe: Apurva Gandhi, Sangyun Lee, Ryan Schuerkamp, Grace Liu

16.1 Introduction

Offline data

Some examples of offline data are YouTube videos and teleoperation data for robotics.

We sample $s, a \sim \mu$, $r = R(s, a)$ (assume deterministic reward), $s' \sim P(\cdot|s, a)$

Unlike imitation learning, we do not place any guarantee on the quality of the distribution μ and the samples – they can be meaningfully suboptimal. However, we do see rewards.

Value learning

Notation. We consider finite horizon Markov Decision Process $M = \{S, A, H, R, P, d_0\}$. We define a policy π where $\pi_h : S \mapsto \Delta(A)$ and let d^π denotes the visitation distribution induced by π at step h. Let $V^\pi(s) = \mathbb{E}[\sum_{\tau=0}^{H-1} r_\tau|\pi, s_h = s]$ and $Q_h^\pi(s, a) = \mathbb{E}[\sum_{\tau=0}^{H-1} r_\tau|\pi, s_h = s, a_h = a]$ be value functions and let $Q^\star$ and $V^\star$ denote the optimal value functions. We will use the following pieces of notation repeatedly:

Definition 15 *Let π_Q to be the greedy policy w.r.t. a state-action value function Q.*

$$\pi^Q(s) = \arg\max_{a\in\mathcal{A}} Q(s, a) \tag{16.93}$$

Definition 16 *We define the Bellman operator $\mathcal{T}$ such that for any $f : S \times A \mapsto \mathbb{R}$,*

$$\mathcal{T}f(s, a) = \mathbb{E}[r(s, a)] + \mathbb{E}_{s'\sim P(s,a)} \max_{a'} f(s', a'). \tag{16.94}$$

Recall that $\mathcal{T}Q^\star = Q^\star$ – the optimal Q function is a *fixed point* of the Bellman Operator. Thus, one way to learn a good Q function is to minimize the **Bellman error** – the difference between the two sides of the fixed point condition. More formally,

Definition 17 *Bellman Error is defined as $f - \mathcal{T}f$.*

Algorithm 8 Q-Value Iteration

```
Require: MDP $M = \{S, A, H, R, P, d_0\}$
  Initialize $Q_H(s, a) = 0$ for all $(s, a) \in S \times A$
  for $h = H-1, H-2, \dots, 0$ do
    for each $(s, a) \in S \times A$ do
      $Q_h(s, a) \leftarrow \mathcal{T} Q_{h-1}$
    end for
  end for
  Return $\{Q_0, Q_1, \dots, Q_{H-1}\}$
```

Per the above, the Bellman Error of the optimal Q function is $Q^\star - \mathcal{T} Q^\star = 0$.

Recall the Q-Value iteration algorithm (Alg. 8). Observe that this algorithm is attempting to minimize the Bellman Error by iteratively applying the Bellman Operator. Once we have near optimal Q-value estimates ($\hat{Q} \simeq Q^\star$), we can get a near optimal policy via a simple action-level argmax: $\pi^{Q^\star} = \pi^\star$. However, these sorts of tabular algorithms assume we have access to the transition dynamics ($P(\cdot|s, a)$) *everywhere*. If we have to learn the dynamics from data, we have to answer the question of where it is most important to do so to ensure value learning leads to a strong policy.

16.2 Where should Bellman error be minimized?

16.2.1 Naive case, assuming we have data everywhere

If we have data for every state-action pair, we do not need to explore online.

Assumption 22 (Full Coverage) *A distribution $\mu(s, a)$ is said to satisfy* full coverage *if*

$$\frac{1}{\mu(s, a)} \leq C \quad \textit{for all } (s, a).$$

This condition implies that the density ratio $\frac{d^\pi(s,a)}{\mu(s,a)}$ is bounded for any policy π, where $d^\pi(s, a)$ is the *visitation distribution* defined as the probability of encountering the state-action pair (s, a) when following policy π starting from the initial state distribution.

Under full coverage, our dataset is sufficiently rich so that we do not need to explore online. Instead, we can learn a model, plan using it, and extract a greedy policy. This approach is known as **certainty-equivalent model-based RL**. Given a fixed dataset, the method computes the policy as follows:

$$\hat{P}(s'|s,a) = \frac{\text{count}(s,a,s')}{\text{count}(s,a)}$$
$$\hat{\pi}(s) = \arg\max_a \hat{Q}^{\hat{P}}(s,a) \text{ , where } Q^{\hat{P}} \text{ is obtained via Q-value iteration inside } \hat{P} \text{ (MBRL)}$$

A standard uniform convergence argument tells us that with enough samples from μ, the learned dynamics/transitions are close to the true ones. Moreover, the more data you have, the lower the error will be. More formally, a Hoeffding + union bound (out of scope) tells us that

$$||\hat{P}(\cdot|s,a) - P(\cdot|s,a)||_{\text{TV}} \leq \epsilon := |S|\sqrt{\frac{C}{h}} \quad \forall s,a.$$

Recall that under the assumption that we have a model that is good everywhere, i.e.

$$\|\hat{P}(\cdot|s,a) - P(\cdot|s,a)\|_{\text{TV}} \leq \epsilon \quad \forall\,(s,a),$$

the Simulation Lemma implies that for any state-action pair,

$$\left|Q(s,a) - \hat{Q}(s,a)\right| \leq H^2\epsilon.$$

Proof: For any state s, let $\pi^\star$ be the optimal policy for the true MDP M and let $\hat{\pi}$ be the greedy policy with respect to $\hat{Q}$. Then, via an add and subtract trick,

$$\begin{aligned} Q(s,\pi^*(s)) - Q(s,\hat{\pi}(s)) &= \underbrace{Q(s,\pi^*(s)) - \hat{Q}(s,\pi^*(s))}_{\leq H^2\epsilon} + \underbrace{\hat{Q}(s,\pi^*(s)) - Q(s,\hat{\pi}(s))}_{\leq H^2\epsilon} \\ &\leq H^2\epsilon + H^2\epsilon \\ &= 2H^2\epsilon. \end{aligned}$$

Applying the simulation lemma to both terms yields the final bound.

■

Note: in translating between finite horizon and discounted infinite horizon setting, $\frac{1}{1-\gamma} \simeq H$. We will interchange throughout the lecture for convenience of analysis.

16.2.2 All-π concentrability

All-π concentrability: $\forall\pi$, $\max_{s,a} \frac{d^\pi(s,a)}{\mu(s,a)} \leq C$. In other words, instead of needing to cover every state-action pair, we only need to cover state and actions where any policies could visit.

However, there may be policies that visit every state-action pair, making this condition not that much weaker than full coverage.

Let us now introduce function approximation – i.e., going beyond the tabular setting. We'll search over a class of candidate value functions $\mathcal{F}$ for some $f \in \mathcal{F}$ such that $f \simeq Q^\star$. This function class is defined as $\mathcal{F} \subseteq S \times A \mapsto [0, V_{\max}]$. The natural offline algorithm for this setting is called Fitted Q-Iteration (FQI), and it minimizes the squared Bellman error over your offline dataset (Algorithm 3).

Algorithm 9 Fitted Q-iteration (FQI)

```
for $h : H, ..., 0$ do
    $f_h \leftarrow \arg\min_{f \in \mathcal{F}} \hat{\mathbb{E}}_{s,a}[(f(s,a) - \mathcal{T} f_{h+1}(s,a))^2]$
end for
```

To prove guarantees for the above algorithm, we'll need the **Bellman Completeness Assumption**: $\forall f \in \mathcal{F}$, $\mathcal{T} f \in \mathcal{F}$ – i.e. the Bellman backup of an arbitrary function in $\mathcal{F}$ is contained in $\mathcal{F}$. This is a very strong assumption, which is satisfied by limited settings such as the tabular setting (the setting considered in this course). For instance, this doesn't hold for the deep neural net policies. The reason this is strong is that because adding a function can break the assumption, unlike expert realizability in imitation learning where expanding the policy class can only help. Given the strength of these assumptions, we will now consider a weaker set of assumptions and a corresponding algorithm that works under them.

16.2.3 Single-Policy Coverage (SPC)

Definition 18 (Single-Policy Coverage) *For some policy π, a distribution $\mu(s,a)$ satisfies* single-policy coverage *if*

$$\max_{s,a} \frac{d^\pi(s,a)}{\mu(s,a)} \leq C^\pi < \infty$$

where $d^\pi(s,a)$ is the visitation distribution under policy π and C^π is the coverage coefficient.

Moving from all-π concentratability to SPC is similar to moving from for all to there exists. Intuitively, we should be able to compete against policies where we've seen what they'll do in our dataset.

There are lower bounds showing that even in the realizable setting with Bellman completeness, SPC is insufficient to guarantee strong performance in the offline setting—online algorithms can surpass these lower bounds. In practice, pure offline RL methods often face severe distribution shift, making it challenging to determine when to stop training or which checkpoint to deploy. Furthermore, model selection in offline RL often relies on some form of online testing (e.g., testing the performance of different checkpoints online and then choosing the best one), so directly incorporating online interactions may not be too much of a leap. The hybrid RL framework addresses these issues by using limited online data to correct for distribution mismatch, thereby enabling more reliable policy improvement—if we have access to an online environment, then SPC can be made to work.

Hybrid RL - You are given offline data but also access to online environment access. Note that Agnostic System Identification [9] which we covered in the model-based RL lecture (Lecture 15) was the first example of this setting. Here we introduce a model-free counterpart.

Algorithm 10 Hybrid Q-iteration (HyQ) [10]

```
We have offline dataset $\mathcal{D}_{\text{off}}$, initialize $f^0$
for $t = 1, \ldots, T$ do
    $\pi_h^t(s) \leftarrow \arg\max_a f_h^{t-1}(s,a);\ \forall h \in [1, ..., H]$
    $\mathcal{D}_{\text{on}} \leftarrow \mathcal{D}_{\text{on}} \cup (s,a,r,s') \sim \pi^t$   (online data)
    $f^t \leftarrow \text{FQI}(\mathcal{D}_{\text{off}} \cup \mathcal{D}_{\text{on}})$
end for
```

Note, that h above indexes the timesteps of the MDP and the inner FQI loop, while t indexes the iterations of the outer loop. In each outer-loop step, we learn a sequence of H policies in the finite horizon setting. Observe that we aggregate online data like in DAgger.

In Hybrid RL, the goal is to compete against the best π with $C^\pi \leq \infty$, meaning we aim to compete against any policy covered by the offline dataset. This is the same fundamental objective as offline RL, but the ability to interact with the environment in Hybrid RL allows us to circumvent the lower bounds that restrict purely offline methods. This goal is also quite reasonable—if a policy is represented in the dataset, we should be able to imitate or even surpass it. While we are typically limited to competing with the best policy contained in the offline dataset, there are variations of this setting that allow us to get closer to the optimal policy $\pi^\star$, often by incorporating an exploration bonus to guide the online interactions.

Why Should HyQ Work?

Intuitively, running FQI on the hybrid data guarantees low Bellman Error on states from π^e and the policy induced by HyQ π^f. We now argue this is sufficient to guarantee we learn a strong policy. We begin with a lemma analogous to the Performance Difference Lemma.

Lemma 23 *Given any comparator policy π^e, for any $f \in \mathcal{F}$ and corresponding greedy policy*

π^f, we have

$$\mathbb{E}_{s_0\sim d_0}\Big[V_0^{\pi^e}(s_0) - V_0^{\pi^f}(s_0)\Big] \leq \underbrace{\sum_{h=1}^{H}\mathbb{E}_{(s_h,a_h)\sim d_h^{\pi^e}}\Big[\mathcal{T}f_{h+1}(s_h,a_h) - f_h(s_h,a_h)\Big]}_{\textit{offline error}}$$
$$+ \underbrace{\sum_{h=1}^{H}\mathbb{E}_{(s_h,a_h)\sim d_h^{\pi^f}}\Big[f_h(s_h,a_h) - \mathcal{T}f_{h+1}(s_h,a_h)\Big]}_{\textit{online error}}.$$

Proof: We can consider the following decomposition:

$$\mathbb{E}_{s_0\sim d_0}\Big[V_0^{\pi^e}(s_0) - V_0^{\pi^f}(s_0)\Big] = \mathbb{E}_{s_0\sim d_0}\Big[V_0^{\pi^e}(s_0) - \max_a f_0(s_0,a) + \max_a f_0(s_0,a) - V_0^{\pi^f}(s_0)\Big].$$

We bound the second difference using a variant of the Performance Difference Lemma:

$$\begin{aligned}
\mathbb{E}_{s\sim d_0}\Big[\max_a f_0(s,a) - V^{\pi^f}(s)\Big] &\overset{(1)}{=} \mathbb{E}_{s\sim d_0}\Big[\mathbb{E}_{a\sim\pi_0^f(s)}\big(f_0(s,a) - V_0^{\pi^f}(s)\big)\Big] \\
&\overset{(2)}{=} \mathbb{E}_{s\sim d_0}\Big[\mathbb{E}_{a\sim\pi_0^f(s)}\big(f_0(s,a) - \mathcal{T}f_1(s,a)\big)\Big] \\
&\quad + \mathbb{E}_{s\sim d_0}\Big[\mathbb{E}_{a\sim\pi_0^f(s)}\big(\mathcal{T}f_1(s,a) - V_0^{\pi^f}(s)\big)\Big] \\
&\overset{(3)}{=} \mathbb{E}_{(s,a)\sim d_0^{\pi^f}}\Big[f_0(s,a) - \mathcal{T}f_1(s,a)\Big] \\
&\quad + \mathbb{E}_{s\sim d_0}\Bigg[\mathbb{E}_{a\sim\pi_0^f(s)}\Bigg(R(s,a) + \gamma\,\mathbb{E}_{s'\sim P(s,a)}\max_{a'} f_1(s',a') \\
&\qquad\qquad - R(s,a) + \mathbb{E}_{s'\sim P(s,a)}V_1^{\pi^f}(s')\Bigg)\Bigg] \\
&\overset{(4)}{=} \mathbb{E}_{(s,a)\sim d_0^{\pi^f}}\Big[f_0(s,a) - \mathcal{T}f_1(s,a)\Big] + \mathbb{E}_{s\sim d_1^{\pi^f}}\Big[\max_a f_1(s,a) - V_1^{\pi^f}(s)\Big].
\end{aligned}$$

(1) follows from the definition of the greedy policy. (2) follows from an add and subtract trick. (3) follows from expanding out the Bellman operator and usimg the definition of the value function. Lastly, (4) follows from canceling terms, leaving us with what we started with except one step further in the future. Then, we we can solve the recurrence relation for the overall bound. We can similarly bound the first term, which is done in [10].

■

Completing the Proof

To complete the performance bound, note that offline error term can be further bounded by a change of measure. Specifically, noting that

$$\mathbb{E}_{(s,a)\sim d^{\pi_e}}[h(s,a)] = \mathbb{E}_{(s,a)\sim\mu}\left[\frac{d^{\pi_e}(s,a)}{\mu(s,a)}\,h(s,a)\right],$$

with $h(s,a) = \mathcal{T} f_{h+1}(s,a) - f_h(s,a)$ (i.e. the Bellman Error at (s,a)), we have

$$\sum_{h=1}^{H}\mathbb{E}_{(s_h,a_h)\sim d_h^{\pi_e}}\Big[\mathcal{T} f_{h+1}(s_h,a_h) - f_h(s_h,a_h)\Big] = \sum_{h=1}^{H}\mathbb{E}_{(s_h,a_h)\sim\mu}\left[\frac{d^{\pi_e}(s_h,a_h)}{\mu(s_h,a_h)}\Big(\mathcal{T} f_{h+1}(s_h,a_h) - f_h(s_h,a_h)\Big)\right]$$

Next, we can apply the Cauchy-Shwartz inequality:

$$\begin{aligned}
&\le \sum_{h=1}^{H}\sqrt{\mathbb{E}_{(s_h,a_h)\sim\mu}\left[\left(\frac{d^{\pi_e}(s_h,a_h)}{\mu(s_h,a_h)}\right)^2\right]\cdot\mathbb{E}_{(s_h,a_h)\sim\mu}\left[\Big(\mathcal{T} f_{h+1}(s_h,a_h) - f_h(s_h,a_h)\Big)^2\right]}\\
&\le \sum_{h=1}^{H}\sqrt{C\cdot\mathbb{E}_{(s_h,a_h)\sim\mu}\left[\Big(\mathcal{T} f_{h+1}(s_h,a_h) - f_h(s_h,a_h)\Big)^2\right]},
\end{aligned}$$

where the final inequality uses the SPC assumption w.r.t. π_e, i.e. that

$$\frac{d^{\pi_e}(s,a)}{\mu(s,a)} \le C \quad \text{for all } (s,a).$$

This gives us an overall bound of

$$\mathbb{E}_{s_0\sim d_0}\Big[V_0^{\pi^e}(s_0) - V_0^{\pi^f}(s_0)\Big] \le \sum_{h=1}^{H}\sqrt{C\cdot\mathbb{E}_{(s_h,a_h)\sim\mu}\left[\Big(\mathcal{T} f_{h+1}(s_h,a_h) - f_h(s_h,a_h)\Big)^2\right]} \tag{16.95}$$

$$+\sum_{h=1}^{H}\mathbb{E}_{(s_h,a_h)\sim d_h^{\pi^f}}\left[f_h(s_h,a_h) - \mathcal{T} f_{h+1}(s_h,a_h)\right]. \tag{16.96}$$

Observe that minimizing squared Bellman Error on samples from μ makes the first term small. Under certain assumptions we don't discuss here, an analogous bound can be proved for the second term in terms of squared Bellman Error – see [10] for the full proof. The key take-away from the above proof is that hybrid RL allows us to compete against a policy with just SPC, which purely offline algorithms provably cannot achieve. Furthermore, hybrid RL is efficient, in the sense we can avoid explicit optimism / pessimism procedures.

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 17: Model Predictive Control & Test-Time Scaling

Lecturer: Gokul Swamy
Scribe: Michael, Megan Li, Yuemin Mao, Jehan Yang

17.1 What is Model Predictive Control (MPC)?

Model Predictive Control (MPC) is a fundamental building block of most modern robotics systems. Intuitively, rather than solving a planning problem at all states one could see in an *offline* fashion, we simply solve a truncated horizon planning problem at the states we do see during *online* rollouts in MPC. In this sense, MPC is a *lazy* algorithm, in the sense it only does computation at the last moment before it is required.

A bit more formally, for each $h \in [H]$:

1. Plan a sequence of k actions starting from s_h in $T^\star$ (perfect model) or $\hat{T}$ (imperfect model): $\tilde{a}_{h+h+k}$.
2. Execute the first of these actions $a_h = \tilde{a_h}$ in $T^\star$.

In other words, we solve

$$\tilde{a}_{h:h+k} = \arg\max_{a_{h:h+k}} \mathbb{E}_T \left[\sum_{\tau=h}^{h+k} r(s_\tau) + V(s_{h+k}) | a_{h:h+k} \right], \tag{17.97}$$

before executing the first of these actions in the real world and then re-planning. Re-planning is only valuable when our estimated s'_{h+1} (what we expect would happen) differs from the true s_{h+1}, i.e. when the dynamics are stochastic. This is true for many robotics problems but not for auto-regressive language generation. Thus, the variants of MPC that are used for language models (e.g. best-of-N) usually don't involve a re-planning procedure.

17.2 How much does lookahead improve performance?

A natural question at this point might be what the value of MPC / lookahead search is for performance. As we will discuss in greater detail below, a k-step lookahead search can be considered performing k iterations of the policy improvement procedure.

Consider some policy π with value function V^π. Single-step policy improvement looks like

$$\pi_1^+(s) = \arg\max_{a \in \mathcal{A}} r(s) + \mathbb{E}_{s' \sim T(s,a)}[V^\pi(s')]]. \tag{17.98}$$

Observe that this is just MPC with $k = 1$. Now, using the Performance Difference Lemma, we can prove that this *local* improvement guarantees a better policy, *globally* speaking:

$$\begin{aligned}
J(\pi_1^+) - J(\pi) &= \mathbb{E}_{\zeta \sim \pi_1^+}\left[\sum_h^H Q^\pi(s_h, a_h) - \mathbb{E}_{a' \sim \pi(s_h)}\left[Q^\pi(s_h, a')\right]\right] && (17.99)\\
&= \mathbb{E}_{\zeta \sim \pi_1^+}\left[\sum_h^H \max_a Q^\pi(s_h, a) - \mathbb{E}_{a' \sim \pi(s_h)}\left[Q^\pi(s_h, a')\right]\right] && (17.100)\\
&\geq 0, && (17.101)
\end{aligned}$$

where the last follows from the fact that each term inside the expectation is non-negative.

Let us now consider MPC with k-step lookahead. For simplicity, we will assume access to a perfect model but a more general version of the following results can be proved via an appeal to the simulation lemma we discussed previously. Define the k-step lookahead policy as

$$\pi_k^+(s_h) = \arg\max_{a_{h:h+k}} \mathbb{E}_{T^\star}\left[\sum_{\tau=h}^{h+k} \gamma^{\tau-h} r(s_\tau) + \gamma^k V^\pi(s_{h+k}) \mid s_h, a_{h:h+k}\right]. \tag{17.102}$$

Define ϵ as the largest value difference between our current policy and the optimal policy over the state space: $\epsilon = \max_{s \in S} \frac{V^*(s) - V^\pi(s)}{H}$. We define ϵ this way to make things horizon-independent. We will now show that

$$J(\pi^*) - J(\pi_k^+) \leq O\left(\frac{\epsilon \gamma^k H(1-\gamma^H)}{1-\gamma^k}\right), \tag{17.103}$$

i.e. that lookahead search allows us to decrease the performance difference as we scale up k.

Proof: Similar to the simulation lemma, the proof proceeds via adding / subtracting a "cross-term" and then canceling terms to end up with time-shifted terms. For notational convenience, we will drop the $s_1 = s$ conditioning from all of the preceding equations.

$$\begin{aligned}
V^\star(s) - V_+^k(s) &= \underbrace{\mathbb{E}\left[\sum_h^k \gamma^h r(s_h) + \gamma^k V^\star(s_k) \mid \pi^\star\right]}_{a} - \underbrace{\mathbb{E}\left[\sum_h^k \gamma^h r(s_h) + \gamma^k V_+^k(s_k) \mid \pi_k^+\right]}_{d}\\
&= a + \underbrace{\mathbb{E}\left[\sum_h^k \gamma^h r(s_h) + \gamma^k V^\star(s_k) \mid \pi_k^+\right]}_{b} - \underbrace{\mathbb{E}\left[\sum_h^k \gamma^h r(s_h) + \gamma^k V^\star(s_k) \mid \pi_k^+\right]}_{c} + d.
\end{aligned}$$

Observe that the the discounted sum of rewards in b and d cancel out as the expectations are taken over actions chosen by the same policy. After canceling and collecting like terms, we are left with

$$= a - \mathbb{E}\left[\sum_{h}^{k} \gamma^h r(s_h) + \gamma^k V^\star(s_k) \mid \pi_k^+\right] + \gamma^k \mathbb{E}\left[V^*(s_h) - V_k^+(s_h) \mid \pi_k^+\right]. \tag{17.104}$$

Now, given we don't a-priori know that $V^\star$ is, we're going to replace it with the value function of the policy V^π, which by assumption differs by at most ϵH at any state. We'll do this in both terms a and b, which gives us

$$\begin{aligned}
&\le \mathbb{E}\left[\sum_{h}^{k} \gamma^h r(s_h) + \gamma^k V^\pi(s_k) \mid \pi^\star\right] - \mathbb{E}\left[\sum_{h}^{k} \gamma^k r(s_h) + \gamma^k V^\pi(s_h) \mid \pi_k^+\right] \\
&+ \gamma^k \mathbb{E}\left[V^\star(s_k) - V_k^+(s_h) \mid \pi_k^+\right] + 2\epsilon H \gamma^k. &(17.105)
\end{aligned}$$

Next, we note that π_+^k is the policy that by definition maximizes the first term so we can upper bound the expectation under $\pi^\star$ and cancel like terms:

$$\begin{aligned}
&\le \mathbb{E}\left[\sum_{h}^{k} \gamma^h r(s_h) + \gamma^k V^\pi(s_k) \mid \pi_+^k\right] - \mathbb{E}\left[\sum_{h}^{k} \gamma^h r(s_h) + \gamma^k V^\pi(s_k) \mid \pi_k^+\right] \\
&+ \gamma^k \mathbb{E}\left[V^*(s_k) - V_k^+(s_h) \mid \pi_k^+\right] + 2\epsilon H \gamma^k &(17.106) \\
&= \gamma^k \mathbb{E}\left[V^*(s_k) - V_k^+(s_h) \mid \pi_k^+\right] + 2\epsilon H \gamma^k. &(17.107)
\end{aligned}$$

Observe this is the same term we started off with, just shifted k steps into the future. We can therefore compute the sum of the geometric series analytically to arrive at

$$\le 2\epsilon H \gamma^k \sum_{i=1}^{H/k} (\gamma^k)^i = \frac{2\epsilon\gamma^k H(1-\gamma^H)}{1-\gamma^k}. \tag{17.108}$$

For the infinite horizon version, one would instead solve a recurrence relation. ■

17.3 What are the different variants of planning?

Given we now understand the performance benefits of MPC, let us consider several standard algorithms which fall under the template we sketched above. Once again, our optimization problem will be to solve

$$\tilde{a}_{h:h+k} = \underset{a_{h:h+k}}{\arg\max}\, \mathbb{E}_T\left[\sum_{\tau=h}^{h+k} r(s_\tau) + V(s_{h+k}) \mid a_{h:h+k}, s_h\right]. \tag{17.109}$$

Let us consider each part of the above in sequence. Consider for a moment a deterministic problem. A *"shooting" method* expands the above objective via recursion:

$$r(s_0) + r(T(s_0, a_0)) + r(T(T(s_0, a_0), a_1) + \dots \tag{17.110}$$

Observe that the repeated applications of T can lead to an ill-conditioned optimization problem, similar to the vanishing / exploding gradient issue when training a recurrent neural network. This can make it hard to use first-order methods (which rely on gradients) unless there's a more nicely behaved $\hat{T}$ that approximates the ground-truth dynamics well (e.g. in iLQR algorithms). Thus, we'll mostly focus on zeroth-order algorithms for this lecture.

Solving the argmax over k actions can often be expensive, so it is common to have some "base policy" one samples m k-step plans from before selecting from this set of m options.

The expectation over the next k steps is often done inside a (learned) world model.

The value function / terminal heuristic V is left underspecified in the above general formulation. Ideally it would be $V^\star$. In practice, we might hope to have access to some expert value function V^E, in which case the above is a lookahead version of DAgger / AggraVaTe.

Let us now discuss four popular algorithms that fit under the above template.

17.3.1 Best-of-N (BoN)

Under the assumption of having tree-structured, deterministic, perfectly known dynamics (i.e. auto-regressive language generation), we can simplify the above formulation to $a_{1:H} = \arg\max_{a^1_{1:H} \dots a^N_{1:H}} r(s_H)$. No re-planning is required because we always append the token we expect to. A terminal cost is sufficient because s_H allows us to uniquely decode all actions / tokens – it is merely their concatenation.

17.3.2 Cross Entropy Method (CEM)

Sample $a^1_{1:k} \dots a^N_{1:k} \sim \pi_t$. Then, pick the top (both in terms of the k rewards and the terminal cost) $\approx 10\%$ of these and compute their mean μ_{t+1}. Finally, set $\pi_{t+1}(s_h) = \mathcal{N}(\mu_{t+1}, \sigma^2)$. Commonly used robotics planning algorithms like Model Predictive Path Integral Control (MPPI) are essentially fancier versions of the above idea.

17.3.3 Sparse Sampling (Kearns, Mansour, Ng)

The algorithm proceeds as follows: start with some state s. Try each action a_i some number (c) of times and track where each action led. Next, from each of these resulting states, repeat the process. Do this up to a maximum depth of H. Critically, we don't do this for all possible next states – only for the next states seen during the sampling process. Given these empirical frequencies, we can just divide the number of times taking action a in state s lead to state s' by c to build an approximate dynamics model / MDP $\hat{M}$. One can then run an arbitrary RL algorithm inside $\hat{M}$ to compute $\hat{\pi}^*$. It is possible to set c and H such that we have strong policy performance guarantees independent of the size of the state space $|\mathcal{S}|$. This contrasts with the standard policy / value iteration algorithms, where the amount of computation you have to do scales linearly with the size of the state space. We won't delve into it in this lecture but this is, more formally, what the "laziness" of MPC means.

17.3.4 Monte Carlo Tree Search

Rather than sampling each action c times, we can use a more efficient exploration strategy. Intuitively, if we consistently see that some state/action pair leads to a bad outcome, we stop exploring from it. More formally, to trade off exploration and exploitation, we use

$$\pi_{UCT}(s) = \arg\max_a \hat{Q}(s,a) + c\sqrt{\frac{\ln(n(s))}{n(s,a)}}. \tag{17.111}$$

The second term comes from an upper confidence / Hoeffding bound. This can be seen as the natural sequential generalization of a bandit algorithm.

17.4 How can we iterate this process?

Once we've ran a k-step look-ahead search, it often makes sense to not just throw away that computation and instead use it to update the base policy so that we can have a better starting point for the next round of policy improvement. This is often called *expert / dual policy iteration* and is fundamental to how systems like AlphaGo work. Intuitively, one uses MPC as a *local* expert and updates the policy via DAgger / distillation. A bit more formally:

For each round of the algorithm $t \in [T]$:

1. Do a k-step local search / MPC around π_t to produce an improved η_t.
2. Use DAgger with η_t as the expert to produce $\pi_{t+1} = \arg\max_{\pi\in\Pi} \mathbb{E}_{s_h\sim\pi_t, a_h\sim\eta_t}[\log \pi(a_h|s_h)]$.

Part IV

Imitation Learning

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 18: Imitation Learning as Game-Solving

Lecturer: Gokul Swamy
Scribe: Jim Wang, Jason Wei, Sangyun Lee, Yang Zhou

18.1 Outline

In this lecture, we will focus on three key questions:

1. Why do we need interaction in imitation learning?
   A: to be able to tell that we've made a mistake that compounds. (necessity)
2. What else do we need to tell which mistakes matter?
   A: information about the set of rewards we could be judged on. (sufficiency)
3. How do we learn a policy that recovers from mistakes that matter if we don't know what the reward function is?
   A: find the policy that is the least distinguishable from the expert's under ***any*** *reward function in the moment set $\mathcal{R}$.*

18.2 Why do we need interaction in IL?

At a high level, it is to be able to tell that we've made a mistake that compounds, i.e. one that compounds over the horizon.

18.2.1 The Pitfalls of Behavioral Cloning

Behavioral cloning is an offline / supervised learning approach to imitation learning. Given some set of expert demonstrations, the BC optimization problem is to solve

$$\arg\min_{\pi\in\Pi} \mathbb{E}_{\xi\sim\pi_E}\left[-\log\left(\prod_h^H \pi(a_h|s_h)\right)\right] = \arg\min_{\pi\in\Pi} \sum_h^H \mathbb{E}_{s_h,a_h\sim\pi_E}[-\log \pi(a_h|s_h)].$$

As we discussed in the DAgger lecture, an ϵ probability of making a mistake can often lead to a performance that scales *quadratically* with the horizon, rather than the expected linear scaling. This phenomenon is known as *compounding errors*.

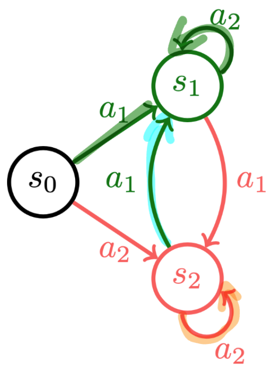
18.2.2 What went wrong?

At train time, offline algorithms like BC look at

$$\ell_{BC}(\pi) = \mathbb{E}_{s_h, a_h \sim \pi_E}[-\log \pi(a_h | s_h)].$$

However, at test time, the learner actually sees $\mathbb{E}_{s_h \sim \pi} \neq \mathbb{E}_{s_h \sim \pi_E}$. Thus, $p_{\text{test}}(x) \neq p_{\text{train}}(x)$, which is called *covariate shift* in the machine learning literature.

Not being able to avoid compounding errors is a fundamental property of *all* offline algorithms, as we now sketch via a lower bound example.



$$r(s,a) = \mathbf{1}[s = s_1]$$
$$\pi_1(s_0) = \pi_2(s_0) = [1,0]$$
$$\pi_1(s_1) = \pi_2(s_1) = [\varepsilon, 1-\varepsilon]$$
$$\boxed{\pi_1(s_2)} = \overset{\text{recovers}}{[1,0]} \quad \boxed{\pi_2(s_2)} = \overset{\text{doesn't recover}}{[0,1]}$$

$$\ell_{BC}(\pi_1) = \ell_{BC}(\pi_2)$$
$$J(\pi_E, r) - J(\pi_1, r) = \varepsilon \cdot H$$
$$J(\pi_E, r) - J(\pi_2, r) = \varepsilon \cdot H^2$$

Figure 18.13: Suppose we have an expert policy π_E that at $h = 1$, takes action a_1 to go from state s_0 to s_1, and then afterwards takes action a_2 to stay in state s_1 forever. Now suppose we have two policies (π_1, π_2), both of which take a_1 at $h = 1$ and with probability ε take action a_1 to end up in s_2 while at state s_1. However, these two policies differ in terms of their behavior at s_2, which is outside of the support of the expert. Specifically, π_1 moves immediately back to s_1 via a_1 when it gets to s_2, but π_2 stays in s_2 indefinitely by taking a_2. In this MDP, only state s_1 gives a reward of 1 per timestep ($r(s,a) = \mathbf{1}[s = s_1]$).

Observe **no offline IL algorithm can tell the difference between** π_1 **and** π_2 because they only differ on states outside of the support of the demonstrations. Mathematically, we have $J(\pi_E, r) - J(\pi_1, r) = \varepsilon H$, and $J(\pi_E, r) - J(\pi_2, r) = \varepsilon H^2$ (quadratic), despite $\ell_{BC}(\pi_1) = \ell_{BC}(\pi_2)$. In short, offline IL fundamentally can't differentiate between policies that recover and those that don't, which means it can't avoid compounding errors.

Where does the ε come from?

A natural question after going through the above example is where ε comes from. Roughly speaking, there are three sources, each of which might matter for a particular problem:

1. Finite-sample error: limited number of expert demos.
   A: Get more data.
2. Optimization error: imperfect search over policy class.
   A: Use more compute (e.g. use a better optimizer).
3. Misspecification error: irreducible error from $\pi_e \notin \Pi$.
   A: Use an interactive algorithm.

We'll focus mostly on the third case for this lecture as it is the most fundamental / hard to solve via scaling data or compute. As we'll discuss further, interaction effectively generates samples from the test distribution, allowing us to handle covariate shift.

18.3 What else do we need to tell which mistakes matter?

At the highest level, we need some sort of *information about the* ***set*** *of rewards we could be judged on.* We'll use $\mathcal{R}$ to denote this set for the rest of the lecture.

18.3.1 Not All Mistakes are Made Equal

We now sketch a simple example of why some knowledge of $\mathcal{R}$ is necessary to avoid compounding errors.

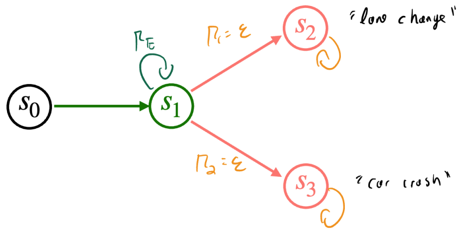


Figure 18.14: Assume π_E goes from s_0 to s_1 and stays there indefinitely. Now at s_1, π_1 has probability ε to go to s_2 and stay there indefinitely, and π_1 has probability ε to go to s_3 and stay there indefinitely.

To be able to pick between π_1 and π_2, **we need to be able to tell which mistakes cost us performance**, which is what $\mathcal{R}$ represents. For example, if we know s_2 corresponds to a "lane change" but s_3 corresponds to a "car crash," it becomes much easier to select between π_1 and π_2.

18.3.2 Moments in Imitation Learning

For example, in autonomous driving, the set of *moments* (basis rewards) could include:

$$\begin{aligned}\mathcal{R} = \{&\text{distance to nearest car, distance to center of lane, distance from edge of road,}\\ &\text{distance from nearest person, speed/speed limit, ...}\}\end{aligned}$$

Note that knowing $\mathcal{R}$ is often *much* easier than knowing r. Concretely, rather than needing to know the precise trade-offs between factors (e.g. how much better it is to arrive five minutes earlier by being 1 inch closer to the nearest vehicle), we simply need to know the set of criteria we could be judged under.

For the rest of this lecture, we'll assume *reward realizability*, i.e. $r \in \mathcal{R}$. However, given we don't know precisely which $f \in \mathcal{R}$ is R, we will attempt to **be good under *all*** $f \in \mathcal{R}$!

18.4 How do we learn a policy that recovers from mistakes that matter if we don't know what the reward function is?

More formally, we can attempt to have a bounded performance difference from the expert under all reward functions in $\mathcal{R}$, which naturally leads to the form of a zero sum game:

$$\max_{\pi \in \Pi} \min_{f \in \mathcal{R}} J(\pi, f) - J(\pi_E, f), \quad \text{where} \quad J(\pi, f) = \mathbb{E}_{\xi \sim \pi}\left[\sum_{h}^{H} f(s_h, a_h)\right] \tag{18.112}$$

Some notes on the above objective:

- If we dropped the first term and just tried to optimize $\max_{\pi \in \Pi} \min_{f \in \mathcal{R}} -J(\pi_E, f)$, we could just pick a reward function that is a constant everywhere. This is because the expert (as well as any other policy) is optimal under this reward function. This is what people mean when they talk about the ill-posedness of an inverse problem.
- This maximin objective is linear in f but *not* convex in π, but we'll ignore the non-convexity for now. Also note that the range of this payoff is $[-H, H]$.

18.4.1 Game-Solving Searches the Pareto Frontier

Before we discuss the performance bounds of inverse RL, we now provide some intuition about what the solution (i.e. Nash equilibrium) to the above game looks like.

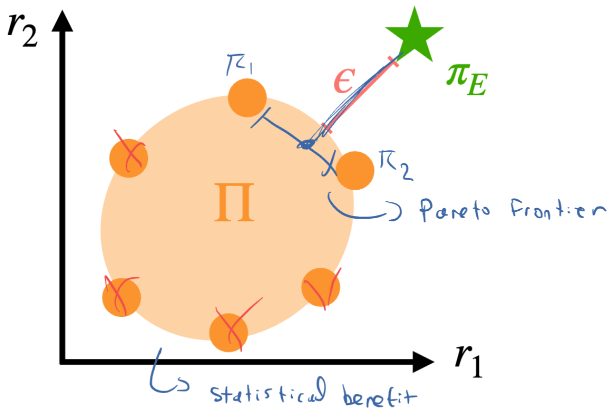


Figure 18.15: For each candidate policy in Π, we can look at its performance on each possible reward function $r \in \mathcal{R} = \{r_1, r_2\}$. The Pareto frontier is then the set of policies that cannot increase its performance on one reward function without decreasing the performance on some other reward function $(\alpha\pi_1 + (1-\alpha)\pi_2)$. A solution to the game must live on this Pareto frontier, effectively reducing the search space we look over, providing a *statistical* benefit (i.e. fewer samples needed to find a good policy than behavioral cloning).

We will now do the simplest proof in this entire course:

Lemma 24 *Assume $\hat{\pi}$ is an ε-approximate equilibria for the IRL game and for simplicity assume $\pi_E \in \Pi$ and $r \in \mathcal{R}$. Then,*

$$J(\pi_E, r) - J(\pi, r) \leq \mathcal{O}(\varepsilon H). \tag{18.113}$$

Proof:

$$\min_{f \in \mathcal{R}} \frac{J(\pi_E, f) - J(\hat{\pi}, f)}{H} \leq \varepsilon \Rightarrow J(\pi_E, r) - J(\hat{\pi}, r) \leq \mathcal{O}(\varepsilon H). \tag{18.114}$$

■

Notice that the performance bound is linear in the horizon, which means we provably avoid compounding errors. Note that we don't need a queryable expert like DAgger.

To summarize, there are three key facts to remember about inverse RL:

1. Inverse RL lets avoid compounding errors without needing access to extra expert interaction.
2. Inverse RL reduces the search space of policies to just those that are on the Pareto frontier.
3. Inverse RL isn't merely picking a reward that makes the expert look optimal—it is fundamentally game-theoretic.

18.5 How do we solve the IRL Game?

We'll use BR to mean "Best Response" and NR to mean "No Regret". As we discussed in the game-solving lecture, we can solve a game by running a no-regret player against a best-response player or against another no-regret player. We have special names for these two flavors of game solving in IRL: *primal* and *dual.* In greater detail:

	Dual	Primal
Policy Update	**BR**: RL	**NR**: GD (PG Step)
Reward Update	**NR**: OGD	**NR**: OGD

A popular dual algorithm is Maximum Entropy Inverse RL (MaxEnt IRL) and a popular primal algorithm is Generative Adversarial Imitation Learning (GAIL). We will now discuss the former in greater detail as it is usually not presented from a game-theoretic lens.

18.6 MaxEnt Inverse RL

We seek the policy with maximum entropy that matches the expert's moments:

$$\begin{aligned} &\max_{\pi} \mathbb{E}_{\xi\sim\pi}\left[\sum_{h}^{H} -\log \pi(a_h|s_h)\right] \quad \text{(maximize causal entropy)} \\ &\text{s.t.} \forall f \in \mathcal{R},\ \mathbb{E}_{\xi\sim\pi}\left[\sum_{h}^{H} f(s_h, a_h)\right] = \mathbb{E}_{\xi\sim\pi_E}\left[\sum_{h}^{H} f(s_h, a_h)\right] \quad \text{(match expert moments)} \end{aligned}$$

It may seem a bit daunting to solve a trajectory-level maximum entropy problem (rather than a single-step problem). However, as we will now derive, the solution follows a beautiful form. First, forming the Lagrangian, we have

$$\max_{\lambda\in\mathbb{R}^{|\mathcal{R}|}} \min_{\pi} \mathbb{E}_{\xi\sim\pi}\left[\sum_{h}^{H} -\log \pi(a_h|s_h)\right] + \sum_{f\in\mathcal{R}} \lambda^f (J(\pi, f) - J(\pi_E, f)). \tag{18.115}$$

Observe that this is just an entropy-regularized variant of our above inverse RL game. Another way to derive the game is via this constraint satisfaction perspective. [5]

[5] In the misspecified setting, there might be no policy that satisfies the constraints. Thus, it is common to regularize the Lagrange multipliers with a quadratic penalty $\sum_{f\in\mathcal{R}} \lambda_f^2$, sometimes called an *augmented Lagrangian* or AuLa for short.

Let us now solve this game via a dual strategy (i.e. with best responses over policies). For some fixed λ_t, we can write the best-response over π as

$$\min_{\pi} \mathbb{E}_{\xi\sim\pi}\left[\sum_{h}^{H} \log \pi(a_h|s_h)\right] + J\left(\pi, \sum_{f\in\mathcal{R}} \lambda_t^f f\right). \tag{18.116}$$

This is because (1) we can drop the $J(\pi_E, f)$ as it is constant with respect to π, (2) J is a linear function with respect to the second argument. Next, observing that both J and causal entropy are expectations over trajectories sampled from π, we have

$$\pi_t = \arg\min_{\pi\in\Pi} \mathbb{E}_{\xi\sim\pi}\left[\sum_{h}^{H}\left(-\log\pi(a_h|s_h) + \sum_{f\in\mathcal{R}} \lambda_t^f f(s_h, a_h)\right)\right]. \tag{18.117}$$

Observe that what remains is just a standard RL problem with a somewhat special reward:

$$r_t(s_h, a_h) \triangleq \log\pi(a_h|s_h) + \sum_{f\in\mathcal{R}} \lambda_t^f f(s_h, a_h). \tag{18.118}$$

We can therefore apply our standard RL tools like value iteration to solve this problem. Let us proceed backwards in time, starting with $h = H$. Because there is nothing left to do,

$$V_t^\star(s_H) = 0. \tag{18.119}$$

For the inductive step (i.e. $h \in [0, H-1]$), value iteration tells us to solve

$$\pi_t^\star(\cdot|s_h) = \min_{p\in\Delta(\mathcal{A})} \mathbb{E}_p\left[\log p(a) + \sum_{f\in\mathcal{R}} \lambda_t^f f(s_h, a) + \mathbb{E}_{T(s_h,a)}[V_t^\star(s_{h+1})]\right] \tag{18.120}$$

Observe that this is a single-step maximum entropy problem – i.e. the kind we previously covered how to solve! Recall that for MaxEnt problems of the form

$$\min_{p\in\Delta(\mathcal{X})} -\mathbb{H}(p) + \mathbb{E}_p[m(x)] \tag{18.121}$$

have solutions of the form

$$p^\star(x) = \frac{\exp(m(x))}{\sum_{x'\in\mathcal{X}} \exp(m(x'))}. \tag{18.122}$$

Here, we have

$$m(a) = \sum_{f\in\mathcal{R}} \lambda_t^f f(s_h, a) + \mathbb{E}_{T(s_h,a)}[V_t^\star(s_{h+1})]. \tag{18.123}$$

Matching terms, we now know the solution to the above in closed form:

$$\pi_t^\star(a_h|s_h) = \frac{\exp\left(\sum_{f\in\mathcal{R}} \lambda_t^f f(s_h, a_h) + \mathbb{E}_{T(s_h,a_h)}[V_t^\star(s_{h+1})]\right)}{\sum_{a\in\mathcal{A}} \exp\left(\sum_{f\in\mathcal{R}} \lambda_t^f f(s_h, a) + \mathbb{E}_{T(s_h,a)}[V_t^\star(s_{h+1})]\right)}. \tag{18.124}$$

The fact that $\pi_t^\star$ takes the form of a softmax is what leads to the above procedure being referred to as *soft value iteration*. More generally, *soft RL* refers to entropy-regularized RL problems. Recall that this softmax form is also what we derived when discussing Hedge / Multiplicative Weights and the Natural Policy Gradient. Thus, even though we took a completely different route, we ended up with a strikingly similar solution!

We can now back up another timestep to get the next value function:

$$V_t^\star(s_h) = \mathbb{E}_{a_h \sim \pi_t^\star(s_h)}[\log \pi_t^\star(a_h|s_h) + \lambda_t^f f(s_h, a_h) + \mathbb{E}_{T(s_h,a_h)}[V_t^\star(s_{h+1})]. \tag{18.125}$$

To close, recall that Bellman's principle of optimality says that an optimal policy acts optimally at the current step and then acts optimally in the future. Intuitively, the above derivation is telling us that the maximum entropy policy is maximally random at the current step and then maximally random in the future.

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 19: Efficient Algorithms for Inverse RL

Lecturer: Gokul Swamy
Scribe: Lujing Zhang, Jim Wang, Ryan Schuerkamp

19.1 Outline

We will cover the following topics today:

1. What makes inverse RL sample-inefficient?
   A: Repeatedly solving a global exploration (RL) problem.
2. Are best responses required for solving the IRL game?
   A: Actually, competing with the expert is "all you need".
3. What algorithms can we use in our new reduction?
   A: A wide variety of sample-efficient "local search" algorithms.

19.2 Recap: IRL as Game-Solving

Recall that we can frame *inverse reinforcement learning* (IRL) as solving a zero sum game:

$$\max_{\pi \in \Pi} \min_{f \in \mathcal{R}} J(\pi, f) - J(\pi_E, f), \tag{19.126}$$

where $J(\pi, f) = \mathbb{E}_{\xi \sim \pi}\left[\sum_h^H f(s_h, a_h)\right]$. Intuitively, when solving the above game, we're trying to find a policy π that isn't too distinguishable from π_E under any reward function $f \in \mathcal{R}$. More formally, an ϵ-approximate Nash Equilibrium of the above game guarantees that $J(\pi, r) - J(\pi_E, r) \leq \mathcal{O}(\epsilon H)$ if $r \in \mathcal{R}$. We discussed two strategies for solving this game:

	Dual	Primal
Policy Update	**BR**: RL	**NR**: GD
Reward Update	**NR**: GD	**NR**: GD

An example of a *dual* algorithm is MaxEnt IRL, while an example of a *primal* algorithm is GAIL. Observe that in computing the *best response* over policies, dual algorithms need to solve the global exploration problem inherent in RL. It is more subtle but the same is true for primal algorithms as well. This is because the adversary could keep playing the same f and to satisfy the no-regret property, the policy player would need to compute π^* eventually. More formally, *any* no-regret algorithm can be used to compute a best response. And, as we discussed in earlier lectures, algorithms like *follow the regularized leader* use a best response oracle to implement a no-regret algorithm. Thus, NR and BR aren't that different.

19.3 What makes inverse RL sample-inefficient?

Repeatedly needing to solve a global exploration problem over the entire state-space is what makes inverse RL inefficient. We ground this point in a tree-structured MDP:

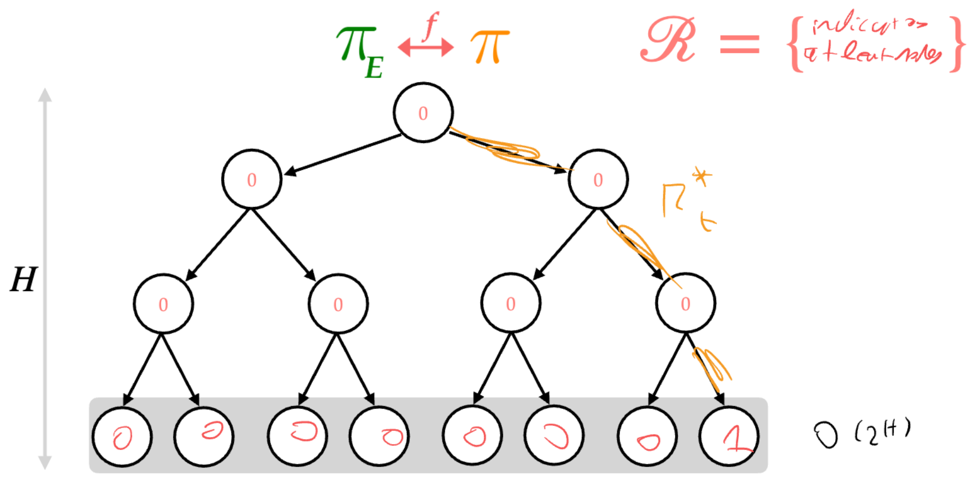
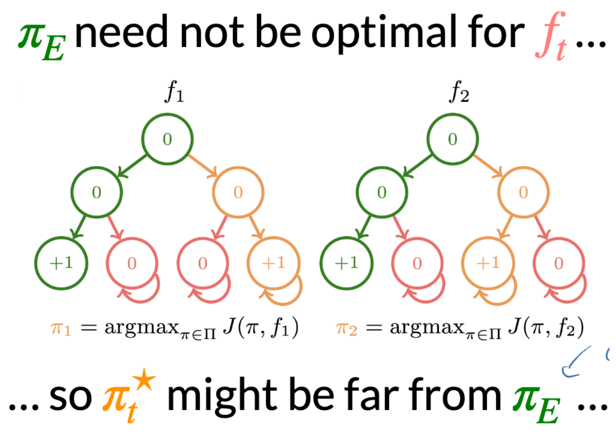

$\pi_E \overset{f}{\longleftrightarrow} \pi$
$\mathcal{R} = \{$indicators at leaf nodes$\}$
H
0
1
π^*_t
$O(2^H)$

Figure 19.16: Consider a binary tree structured MDP of depth H where the reward function class $\mathcal{R}$ is indicator functions at "leaf nodes" at time H (1 node out of 2^H nodes at time H has reward 1, all other nodes 0). Thus, at each round of IRL, after the adversary picks some f, we might have to explore all leaf nodes before we find any reward, an amount of interaction that grows *exponentially* with the task horizon – $\mathsf{exp}(H)$. Then, at the next iteration, we need to solve a similar problem again after the adversary shifts the reward. In some sense, *we're using an RL hammer in a game of adversarial whack-a-mole.*

Beyond being a worst-case constructions, applications like auto-regressive language generation have this tree-structured property. Intuitively, this is strange because *we've reduced the "easier" problem of IL to repeatedly solving the "harder" problem of RL.*

19.4 Are BRs required for IRL?

Intuitively, we waste interaction in IRL by searching over policies that are dissimilar to the expert's. We sketch an example of this below:



π_E need not be optimal for f_t ...
f_1
f_2
0
+1
$\pi_1 = \text{argmax}_{\pi \in \Pi} J(\pi, f_1)$
$\pi_2 = \text{argmax}_{\pi \in \Pi} J(\pi, f_2)$
... so $\pi_t^\star$ might be far from π_E ...

Figure 19.17: Consider some adversarially chosen reward f_t. The optimal policy for this reward, π_t^*, might be far from π_E. This means it can't be the policy we want, as we're just trying to compete with the expert for inverse RL.

This leads to our *key idea: saving interaction by competing with π_E, not π_t^*.* This allows us to prune large parts of the state / policy space. We now make this intuition more formal.

19.4.1 Reducing IRL to Expert-Competitive RL

We now derive a more informed reduction for IRL that allows us to eliminate the complexity of global exploration. First, we introduce the notion of an *expert-relative regret oracle,* i.e. an algorithm that merely competes with the expert on average:

Definition 19 (ERROr$\{\text{Reg}_\pi(T)\}$) *A policy-selection algorithm $\mathbb{A}_\pi$ satisfies the $Reg_\pi(T)$ expert-relative regret guarantee if given any sequence of reward functions $f_{1:T}$, it produces a sequence of policies $\pi_{t+1} = \mathbb{A}_\pi(f_{1:t})$ such that*

$$\sum_{t=1}^{T} J(\pi_E, f_t) - J(\pi_t, f_t) \leq Reg_\pi(T). \tag{19.127}$$

We never need to compute a best response to an f_t – doing so is sufficient but not necessary!

For completeness, we also define a no-regret reward selection algorithm.

Definition 20 *$\mathbb{A}_f$ is a no-regret reward selection algorithm if when given a sequence of policies $\pi_{1:t}$, it produces iterates $f_{1:t} = \mathbb{A}_f(\pi_{1:t})$ such that*

$$\sum_{t=1}^{T} J(\pi_t, f_t) - J(\pi_E, f_t) - \min_{f^* \in \mathcal{R}} \sum_{t=1}^{T} J(\pi_t, f^*) - J(\pi_E, f^*) \leq H Reg_f(T),$$

with $\lim_{T\to\infty} \frac{Reg_f(T)}{T} = 0$.

We now show that with just these two oracles, we can solve the IRL game:

Proof: Consider using an ERROr algorithm to select policies and a no-regret reward selection algorithm to select discriminators. Then, we can expand the average performance difference over T rounds of the algorithm as

$$\begin{aligned}
J(\pi_E, r) - J(\bar{\pi}, r) &= \frac{1}{T}\sum_{t=1}^{T} J(\pi_E, r) - J(\pi_t, r) \quad \text{(Linearity of expectation)} \\
&\leq \max_{f^* \in \mathcal{F}_r} \frac{1}{T}\sum_{t=1}^{T} J(\pi_E, f^*) - J(\pi_t, f^*) \quad \text{(Reward realizability)} \\
&\leq \frac{1}{T}\sum_{t=1}^{T} J(\pi_E, f_t) - J(\pi_t, f_t) + \frac{\text{Reg}_f(T)}{T} H \quad \text{(Defn. of regret)} \\
&\leq \frac{\text{Reg}_\pi(T)}{T} + \frac{\text{Reg}_f(T)}{T} H \quad \text{(Defn. of ERROr)}
\end{aligned}$$

■

In short, the above proof tells us that *competing with the expert is all you need for IRL!*

19.5 What algorithms can we use in our new reduction

A natural question at this point might be as to what an efficient algorithm that actually satisfies the ERROr property is. It turns out we've covered quite a few over the semester: PSDP, HyQ, Agnostic SysID. Basically, anything derived from a variation of the performance difference lemma works! We now step through the use of PSDP in IRL in detail.

19.5.1 Expert-Competitive RL via PSDP

Key Idea: Reset to states from the demonstrations to reduce unnecessary exploration in IRL!

In words, we're reducing interaction complexity by performing a *local search*. For longer horizon problems where we'd like a single, stationary policy, we can instead pick a random

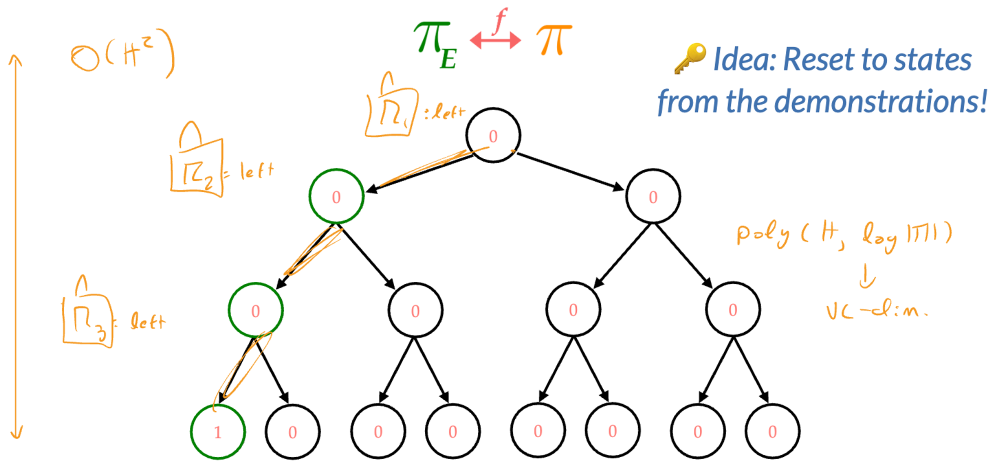


Figure 19.18: Let us consider our tree-structured MDP again with the additional information that the expert takes the leftward, green path. We proceed via backwards in time induction. First, at $h = H-1$, we have to figure out whether we go left or right from the expert's s_{H-1}. We would pick to go left to get reward 1, which we then lock in to compute π_3. We then back up a timestep to $h = H-2$ and again pick a single-step decision conditioned on the future policies we've computed. If we go left, we'd go left again and get reward 1. If we go right, we'd then go left and get reward 0. So, we'd choose to go left, giving us π_2. We can do this all the way up the tree. Observe that we only need $\mathcal{O}(H)$ interaction at each step of the algorithm and there are H steps, which gives us an overall interaction complexity of $\mathcal{O}(H^2)$, an *exponential* speedup.

timestep rather than doing backwards-in-time induction and achieve similar guarantees. Practically, this algorithm corresponds to doing resets to states from the expert demonstrations during the inner loop of IRL, rather than to the true start-state distribution. Of course, it is hard to do resets in the real world, so one can instead learn a model (fitting it on a *hybrid* distribution of learner and expert data) and do resets inside the model. Intuitively, fitting a model on just expert data can make it too optimistic, while fitting a model on just learner data can make it too pessimistic. Fitting on a hybrid mixture provably balances optimism and pessimism without computational intractability.

19.5.2 PSDP Performance Guarantees

For completeness, we now state the formal performance guarantee for PSDP:

Lemma 25 (PSDP Performance Guarantee) *Assume that at each time-step $h \in [H]$, we perform policy optimization up to ε-optimality on some baseline distribution μ:*

$$\mathbb{E}_{s_h\sim\mu}\left[\mathbb{E}_{a\sim\pi'_h(s_h)}\left[Q_t^{\pi_{h+1:H}}(s_h,a)\right]-\mathbb{E}_{a\sim\pi_h(s_h)}\left[Q_t^{\pi_{h+1:H}}(s_h,a)\right]\right]\leq\varepsilon H$$

Then, the performance of the learned policy satisfies:

$$J(\pi'_{1:H},f_t)-J(\pi_{1:H},f_t)\leq\mathcal{O}((\varepsilon+\mathbb{D}_{TV}(\mu,\rho_{\pi'})\cdot H^2)$$

where μ is the reset distribution used for PSDP, $\rho_{\pi'}$ is the occupancy measure of the reference policy π', and $\mathrm{TV}(\mu,\rho_{\pi'})$ is the total variation distance between the two distributions.

Proof: We proceed via the Performance Difference Lemma (PDL):

$$\begin{aligned}J(\pi'_{1:H},f_t)-J(\pi_{1:H},f_t)&=\sum_h^H\mathbb{E}_{s_h,a_h\sim\pi'_{1:h}}\left[Q^{\pi_{h+1:H}}(s_h,a_h)-\mathbb{E}_{a\sim\pi_h(s_h)}\left[Q^{\pi_{h+1:H}}(s_h,a_h)\right]\right]\\&\leq\sum_h^H\mathbb{E}_{s_h,a_h\sim\mu}\left[Q^{\pi_{h+1:H}}(s_h,a_h)-\mathbb{E}_{a\sim\pi_h(s_h)}Q^{\pi_{h+1:H}}(s_h,a_h)\right]\\&\quad+H\cdot\mathbb{D}_{TV}(\mu_h,\rho_h^{\pi'})\quad\text{(by Hölder's inequality)}\\&\leq\sum_h^H(\varepsilon+\mathbb{D}_{TV}(\mu_h,\rho_h^{\pi'}))\cdot 2(H-h)\\&\leq(\varepsilon+\mathbb{D}_{TV}(\mu,\rho_{\pi'}))\cdot H^2\end{aligned}$$

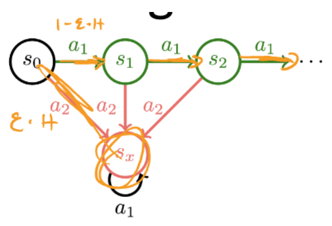
■

In the language we used above, this result means that $Reg_\pi(T)\leq(\varepsilon+\mathbb{D}_{TV}(\mu,\rho_{\pi'}))\cdot H^2T$. While we don't discuss this in detail in the lecture, observe that we can use reset distributions other than the expert demonstrations and still have strong performance guarantees so long as μ (the reset distribution) is close to ρ_E (the expert's state distribution). This is a natural opportunity to use plentiful suboptimal data to speed up exploration – see the linked paper on the course website for more information.

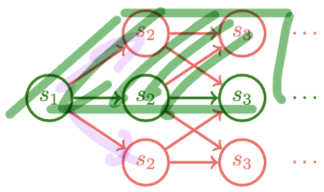
19.5.3 Compounding Errors in PSDP

A natural question after seeing the $\mathcal{O}(H^2)$ term in the above bound might be whether the above bound is tight. Put differently, does PSDP suffer from compounding errors just like behavior cloning does in some situations? Recall that the whole reason we're using an interactive algorithm for imitation is to avoid compounding errors.

It turns out that in the worst case (e.g. cliff-walking, irrecoverable problems), this is true. However, on less harsh problems, PDSP can recover from some errors, since it knows some information about what it's going to do in the future. We sketch an example of both below.



Unavoidable in general on cliff-like (irrecoverable) problems.



Interaction can still help us figure out which mistakes compound.

Figure 19.19: **Top:** We consider a cliff-walking problem where a single mistake can doom the learner for the rest of the horizon. Assume the policies for timesteps $[1, H]$ have been computed and stay on the top of the cliff. At $h = 0$, the learner picks a policy that makes a mistake with probability ϵH. While they have an average error of ϵ, they have a performance difference from the expert that scales with $\mathcal{O}(\epsilon H^2)$. Furthermore, an interactive algorithm like PSDP wouldn't be able to fix such issues any better than behavioral cloning as during the computation of policies $[1, H]$, we pretend the learner is going to start from the top of the cliff rather than having already fallen off. **Bottom:** We now consider a recoverable problem, where the expert takes the middle path but either of the top two rows give accessible reward. Similar to before, assume we perfectly imitate the expert for steps $[1, H]$ by going straight. At $h = 0$, we give the learner the ability to choose between going up or down. Observe that PSDP would know to choose to go up because it observes the consequences of its future actions, while BC would be unable to tie-break correctly.

Thus, while in the worst case, resetting to expert states can introduce compounding errors, it is less likely to do so on problems with recoverability. See papers linked on course website for a more formal statement of the above. In practice, one can also start by only looking at expert state distributions, but then gradually interpolate with or anneal towards the ground-truth initial state distribution ρ_0 to reduce compounding errors.

19.5.4 Additional Algorithms

While we do not discuss this idea in detail, one can also run HyQ as part of the above reduction and have similar guarantees. In practice, this corresponds to running some off-policy RL algorithm (e.g., SAC) with a hybrid replay buffer (e.g., RLPD). Intuitively, this helps speed up policy search as we're picking a reward function that makes the expert look good, so training on expert data is likely to provide positive signal.

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 20: Imitation Learning In Real Life

Lecturer: Sanjiban Choudhury
Scribe: Yang Zhou, Miaosi Dong, Pranjal, Harshita Diddee

20.1 Why Imitation Learning?

There are two main reasons why imitation learning (IL) is used in real-world setups:

20.1.1 The Alignment Problem

It is intractable to enumerate all the rules and rewards necessary to completely and correctly incentivize the right actions to an agent. Hand-designing rewards often relies on implicit "common sense" or shared values that the agent doesn't possess.

- **For example,** When tasked with making a cup of coffee, a reward that awards a cup of coffee may need to be accompanied with an explicit penalty for procuring the coffee through other means than actually making the coffee (Stealing someone else's coffee!).
- **Another example:** Trying to encode driving rules from an official handbook would fail in complex situations. (e.g., in gridlock intersections)

The above example also demonstrates *explicitly* programming reward is tedious, error prone, hackable. Instead, robots should infer rewards from natural human feedback. That's why imitation learning is needed. Finally, a rule-based approach to maximizing rewards is also unlikely to be sustainable in the noisy, real world where humans and other agents may not comply with the rules expected during the agent's policy training.

Summary: Given this brittleness and impracticality of hand engineering rewards, it is much more desirable to design a way of learning from the ideal behavior demonstrated by humans. This learning should help the agent infer the rewards from demonstrations provided by the human and also allow the agent to actively query the human to seek demonstrations in scenarios where it isn't able to infer a reward. Imitation Learning is a step in this direction.

In nutshell, alignment problem can be defined as:

Definition 21 (Alignment Problem) *We have two entities: humans and robots. Hu-*

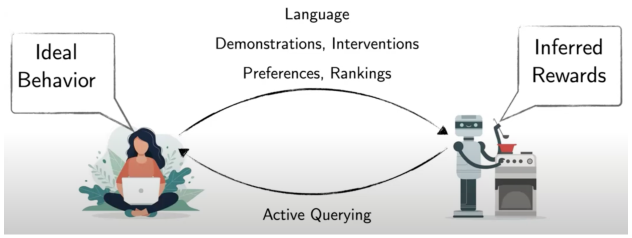


Figure 20.20: Alignment Problem

mans have some ideal behavior in mind, which is difficult to express in functional form, but can be conveyed naturally through language, demonstrations, preferences, or rankings. The goal of the Robot is to actively query the human to gather more information to infer the reward.

(as Figure 20.20 illustrates.)

20.1.2 The Collaboration Problem

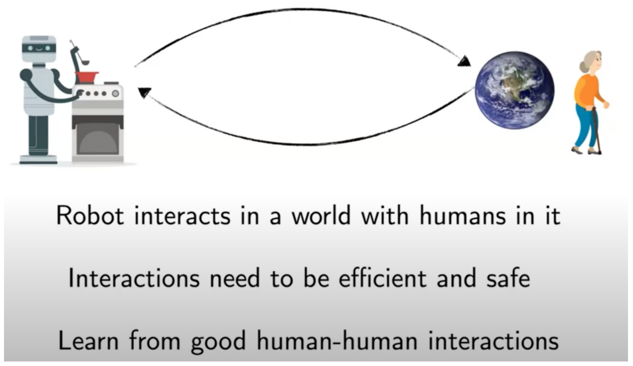
We want to consider the human in the environment as first-class citizens.

- **Example of the MOSAIC system (A Modular System for Assistive and Interactive Cooking)**: A human user interacts with a team of robots assisting them in cooking a recipe. The recipe exists in the user's mind and is not explicitly known to the robots. There is a high degree of personalization—such as different ways of cooking—and the robot's goal is to help the human achieve this personalized outcome. The robots must infer what the human wants to do, and equally what they would prefer not to do because it is too boring or repetitive, allowing the human to focus on tasks that are creative and engaging. The robot observes the human through cameras, predicts their intent, and collaborates accordingly.

Based on this example, the definition of the collaboration problem is as follows:

Definition 22 (Collaboration Problem) *Robots interact in a world with humans in it. Hence, all their interactions must be efficient and safe. This makes it impossible for them to explore with the human in the loop, as is necessitated by traditional reinforcement learning. We must be able to learn from good human-human interaction.*

(as Figure 20.21 illustrates.)



Robot interacts in a world with humans in it

Interactions need to be efficient and safe

Learn from good human-human interactions

Figure 20.21: Alignment Problem

20.1.3 Key Challenge in IL

The key challenge in IL is jointly optimizing the performance and efficiency of both the (a) agent-human interaction and (b) agent-real-world interaction.

$$\begin{tikzcd}
& \text{Robot} \arrow[dl, shift left] \arrow[dr, shift left] & \\
\text{Human} \arrow[ur, shift left] & & \text{Real World} \arrow[ul, shift left]
\end{tikzcd}$$

20.2 Challenge 1: Agent-Human Interactions

Setup: We are attempting to train a policy for a self-driving robotic race car [11].

- **Data Collection:** Collect human demonstration data $(s_1^*, a_1^*), (s_2^*, a_2^*), ...$
  - **State Space(s):** Occupancy Map (an observation about what parts of its environment are occupied/obstacles, while which are not).
  - **Action Space(a):** Joystick Directions.
- **Train a policy(classifier)** $\pi : \Phi \to \Delta(\mathcal{A})$ (Select one out of a set of possible trajectories that closely match what the human was trying to show)

We review the space of IL approaches for training such a policy.

20.2.1 Behavior Cloning (BC)

The first method is behavior cloning [12] (i.e. directly regressing from states to actions). This is the supervised learning / maximum likelihood estimation approach to IL.

Result: Empirically, BC often achieves a reasonable validation error, but during rollouts, the car crashes quite frequently.

Reason: When the policy makes a small mistake and enters a state not seen in the expert data (e.g., heading towards a wall), it lacks information on how to recover. It often defaults to the most common action seen in demonstrations (e.g., "drive straight"), leading to a crash. This is because the policy is only evaluated on the expert's state distribution, not its own induced state distribution. More formally,

$$\text{Training Loss: } \sum_{t=0}^{T-1} \mathbb{E}_{s_t \sim d_t^{\pi_E}} \left[\ell(s_t, \pi(s_t))\right] \neq \text{Test Loss: } \sum_{t=0}^{T-1} \mathbb{E}_{s_t \sim d_t^{\pi}} \left[\ell(s_t, \pi(s_t))\right] \tag{20.128}$$

20.2.2 Dataset Aggregation (DAgger)

To mitigate this, we naturally gravitate towards a system that increases the coverage of the learned policy to observe how the expert will respond in states that the learned policy is likely to encounter. This involves querying the expert in all the states where the car visits and then aggregating the expert's labels for these instances with the original data.

Training Algorithm: DAgger

1. Roll out the current learner policy to sample robot states.
2. Collect post-hoc expert action labels at ***all robot states***.
3. Aggregate the (learner state, expert action) data with that from previous rounds.
4. Update policy on aggregate data $\mathcal{D}$:

$$\min_{\pi} \mathbb{E}_{s,a \sim \mathcal{D}}[\mathbb{I}(\pi(s) \neq a_E)]. \tag{20.129}$$

By training our policy on the states it actually visits rather than just the ideal ones visited by the expert, we can eliminate compounding errors.

$$\begin{aligned} \text{Naive BC:} \quad & J(\pi) - J(\pi_E) \leq \mathcal{O}(\varepsilon T^2), \\ \text{DAgger:} \quad & J(\pi) - J(\pi_E) \leq \mathcal{O}(\varepsilon T). \end{aligned}$$

Practical Concern: While DAgger provides a nice performance bound, querying the expert for every action that the learning policy visits can be expensive, risky (e.g., driving in the real world), and hard to scale (as the environment becomes more complex).

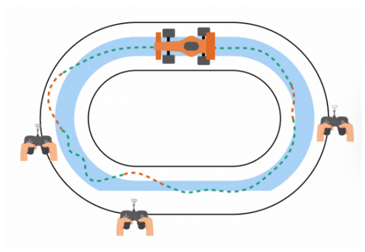


Figure 20.22: HG-DAgger: Interactive Imitation Learning with Human Experts

20.2.3 Human-Gated DAgger (HG-DAgger) [13]

Instead of querying the expert *everywhere* the learner goes, we can instead only have the expert intervene when necessary, as in Figure 20.22.

Training Algorithm: HG-DAgger (i.e. *learning form interventions*):

1. Roll out the current learner policy to sample robot states.
2. Collect action labels at states where the expert intervened.
3. Aggregate the (learner state, expert action) data with that from previous rounds.
4. Update policy on aggregate data $\mathcal{D}$:

$$\min_{\pi} \mathbb{E}_{s,a\sim\mathcal{D}}[\mathbb{I}(\pi(s) \neq a_E)]. \tag{20.130}$$

Concern: Though more efficient in terms of querying the expert, the agent would learn to make a mistake and then recover from it, rather than avoiding the mistake in the first place.

20.2.4 Expert Intervention Learning (EIL)

An alterative perspective on expert interventions is that they tell us information about the expert's *latent Q-value function*. Specifically, we can model the expert as intervening at the last moment before even they couldn't fix the agent's behavior. We could then try and infer this Q_E-function from data of when and how they intervene. More formally,

$$\begin{aligned} \min_{Q_E\in\mathcal{Q}} \quad & \mathbb{E}_{(s,a_E)\sim P_{\text{expert}}}\left[\ell(Q_E(s), a_E)\right] \\ \text{subject to:} \quad & Q_E(s,a) \leq \delta_{\text{good}} \quad \forall (s,a) \in \text{(I) (before expert intervenes)} \\ & Q_E(s,a) \geq \delta_{\text{good}} \quad \forall (s,a) \in \text{(II)} \quad \text{(after expert intervenes)} \\ & Q_E(s,a) \leq \min_{a'} Q_E(s,a') \quad \forall (s,a) \in \text{(III) (expert intervenes optimally)} \end{aligned}$$

Note we're using costs rather than rewards above – a low Q_E is a good thing. By inferring such a Q_E, we also know what states we should avoid – those where $\min_a Q_E(s,a)$ is big. We can then augment the HG-DAgger loss (i.e. match the intervention action) with an additional term to avoid the states where interventions happened – see paper for more.

20.3 Scaling to the Real World

When training a real self-driving car, one of the most important differences from our above toy setup is the architecture used, which borrows from developments in language modeling. These days, SDC policies are often transformers [14], as Figure 20.23 illustrates:

1. A Scene Encoder might be pretrained on vast amounts of (suboptimal) driving data. We want to be able to forecast how *all* actors act, not just those who act optimally.
2. A Decoder is fine-tuned to predict future actions on *good* driving data.

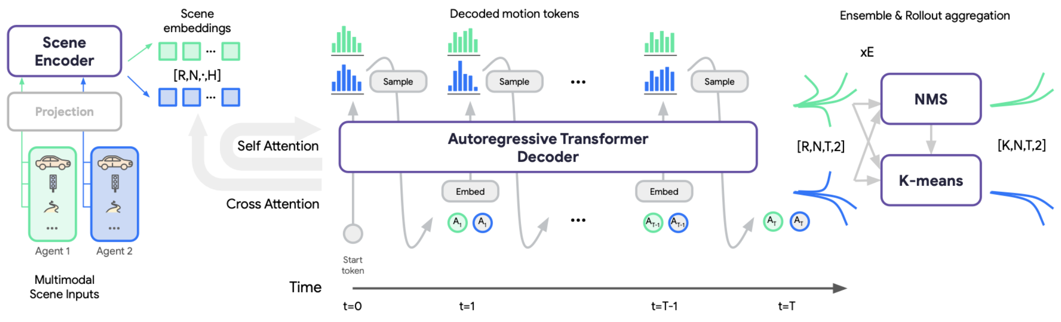


Figure 20.23: MotionLM Architecture

However, demonstrations alone are often missing some information we'd like our SDC to be trained on. First, demonstrations only tell you what to do – they don't tell you what *not* to do. Second, demonstrations don't show recovery behavior from mistakes the expert did not make themselves. We will now discuss how we can use ideas like EIL to address this.

When deploying a self-driving car in the real world, a human operator will sometimes need to intervene and correct the behavior of the system. We can then triage such events and use them to generate tests that specify what right / wrong behavior is in such situations. More generally, we can think of interventions as *automatic preference generation*: we prefer not to enter intervention states and we prefer what the expert did to what the robot was going to do. We can then use this preference data to back out a reward function to apply RL to, similar to the RL form Human Feedback pipeline we will discuss soon.

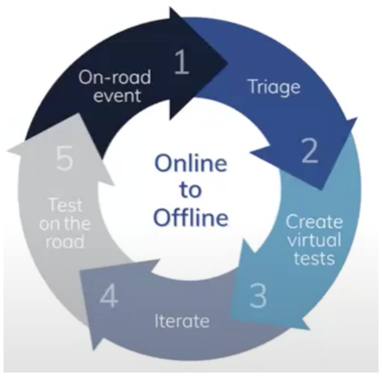


Figure 20.24: The real-world analog of EIL is using interventions to define unit tests.

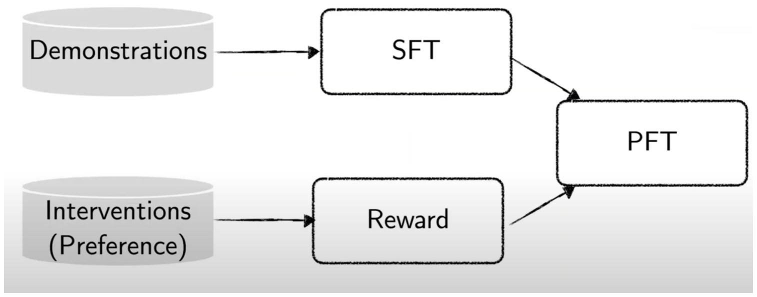


Figure 20.25: An RLHF Pipeline for Self-Driving.

Part V

RL from Human Feedback

17-740 Spring 2025

Algorithmic Foundations of Interactive Learning

Lecture 21: The Information Geometry of RLHF

Lecturer: Gokul Swamy

Scribe: Zichun Y., Yiming Z., Lintang S., Alfredo G.

21.1 Outline

The lecture addresses three main points that characterizes Reinforcement Learning from Human Feedback (RLHF) for language models (LM):

1. What is the fine-tuning problem?
   A: Regularized maximum likelihood estimation (MLE).
2. End-to-end, what is the process of RLHF doing?
   A: A two-stage process consisting of MLE over reward models followed by MaxEnt (Entropy-Regularized RL) over policies.
3. What are direct alignment algorithms?
   A: Algorithms that directly maximize likelihood over the policy space Π *without explicitly passing through the reward model space* $\mathcal{R}$*.*

21.2 Motivation: The Era of Fine-Tuning

Pretrained language models such as GPT-3/4 [15, 16] learn a wide range of capabilities and statistical patterns from vast amounts of textual data mainly sourced from unstructured web data. However, their behavior might not align perfectly with human intentions or desired interaction styles (e.g., instruction following, helpfulness, harmlessness).

Fine-tuning, particularly using methods like RLHF [17], aims to "steer" these base models towards desired behaviors. This alignment process engineers the model's outputs to better match human preferences, leading to models like InstructGPT [17] or ChatGPT, which exhibit significantly improved instruction-following and conversational abilities compared to their base counterparts. The core problem we are trying to solve during this fine-tuning or alignment phase can be framed as regularized maximum likelihood estimation. However, it is really hard to write an effective reward function by hand.

21.3 The Standard Three Steps of RLHF

The standard RLHF pipeline typically involves three stages:

1. **Supervised Fine-Tuning (SFT)**: (Also referred to as Imitation Learning - IL). A pre-trained language model is fine-tuned on a dataset of high-quality human demonstrations or instructions. For various prompts, human labelers provide the desired outputs. The model learns to mimic these expert responses (behavior cloning). Let the policy after this stage be π_{SFT} or π_{ref}.
2. **Reward Model (RM) Training**: Reward models are specific models that can asses the quality of a model response or in this case how preferable a response is to a human. While in theory, RMs can be any function that provides a score given a response, in the case of RLHF, it is often initialized from the SFT model. This involves collecting comparison data: for a given prompt, multiple outputs are generated (e.g., from the SFT model), and human labelers rank these outputs or choose the best one (pairwise comparison). The reward model learns a scalar function $r_\phi(\text{prompt}, \text{completion})$ that assigns higher scores to preferred responses. This is essentially a classification or regression problem aiming to model the human preference distribution.
3. **Reinforcement Learning (RL) Optimization**: The language model (initialized from the SFT model) is further optimized using RL. The goal is to maximize the expected reward predicted by the trained reward model r_ϕ. To prevent the policy from deviating too much from the SFT model (which has good generative capabilities and prevents reward hacking), a KL-divergence penalty term is added to the objective. Common RL algorithms used include PPO (Proximal Policy Optimization)[18], but other policy gradient methods (like REINFORCE[19], REBEL[20], GRPO[21]) can also work effectively with proper tuning.

This process iteratively refines the language model to align better with human preferences while retaining its core language capabilities.

21.4 Language Modeling as a Markov Decision Process (MDP)

A language model predicts the probability of the next token w_t given the preceding tokens $w_t \sim P(w_t | w_{t-1} w_{t-2} \cdots w_1)$. We can think of language modeling as a special MDP.

- **Prompts as Initial States (s_0)** We can regard prompts s_0 as an initial state sampled from a distribution $s_0 \sim p_0$. LMs are typically provided instructions as input which initializes the language model to provide a sequence of continuation text.

- **Next Token Predictions as Actions (a_t)**: Provided an initial state s_0, a model predicts the next token from a vocabulary / *action space*. We denote this as $a_t \in \mathcal{A}$.
- **Generated Tokens as State (s_t)**: At each successive step t, the sequence of tokens thus far can be regarded as the current state on which a model is conditioned on to predict the next token. This way, we can write any arbitrary state s_t as a concatenation of the initial state and the sequence of actions (or tokens) predicted up to time step t, $s_t = [s_0, a_1, a_2, \ldots, a_t]$. Thus, our state space is $\mathcal{S} = \mathcal{A}^H$, where H is the horizon.
- **The LM as the Policy (π)**: If we define generated tokens s_{t-1} as state and the next token a_t as action, the language model itself becomes the policy $\pi(a_t|s_{t-1})$ that provides the probability distribution over the next token given the preceding sequence.
- **Transitions (T)**: The transitions are deterministic and known. Given state s and action (token) a, the next state s' is simply the concatenation $s' = [s, a]$.

$$T(s' \mid s, a) = \begin{cases} 1 & \text{if } s' = [s, a] \\ 0 & \text{otherwise} \end{cases}$$

  This structure forms a tree rooted at the prompt s_0.
- **Horizon (H)**: A maximum generation length H is typically set which limits the generated sequence to $s_H = [s_0, a_1, a_2, \ldots, a_H]$. Generation can end earlier if an end-of-sequence token is produced.
- **Reward (r)**: In the RLHF context, the reward is typically assigned only at the end of the generation (at horizon H or when an end-of-sequence token is generated). This is because it is often hard to break down the quality of some text into a sum of per-word scores. The reward function $r(s_H)$ or $r(\xi)$ (where $\xi = (a_1, \ldots, a_H)$ is the full completion) is given by the output of the trained reward model from Step 2.

Remark 26 (Special Characteristics of the Language MDP)

1. Dynamics are deterministic, known, and tree-structured.
2. Resets are easy: starting a new generation from any prefix (state) is trivial.
3. The reward function $r(s_H)$ is non-Markovian with respect to the token-level states s_t (it depends on the entire sequence) and doesn't naturally decompose per token.

Remark 27 *Since the reward is only given at the end based on the complete trajectory (completion), this MDP can be analyzed as a contextual bandit problem, where the context is the prompt s_0 and the "action" is the entire generated sequence ξ.*

Remark 28 (Fixed Horizon H) *While language generations vary in length, using a fixed horizon H is often practical. Models can generate a special "$\langle eos \rangle$" token. Tokens generated after "$\langle eos \rangle$" can be considered padding and ignored, effectively handling variable lengths within the fixed horizon framework.*

21.5 Preference Fine-Tuning: The Data and Goal

Setup: The dataset $\mathcal{D}$ consists of tuples (s_0, ξ^+, ξ^-), where:

- $s_0 \sim p_0$ is a prompt.
- ξ^+ and ξ^- are two completions generated for the prompt s_0, often sampled from the SFT policy π_{ref} (i.e., $\xi^+, \xi^- \sim \pi_{\text{ref}}(\cdot|s_0)$).
- A human label indicates that ξ^+ is preferred over ξ^- (denoted $\xi^+ \succ \xi^-$).

Optimization Problem: The ultimate goal of RLHF is to find a policy π that generates completions aligning with human preferences, while staying "close" to the initial SFT policy π_{ref}. This "closeness" is enforced by a (reverse) KL penalty. The fine-tuning problem can be seen as finding a policy $\pi^\star$ that minimizes some loss on the preference data $\mathcal{D}$, regularized by its distance to π_{ref}:

$$\pi^\star = \operatorname*{argmin}_{\pi \in \Pi} \mathbb{D}_{\text{KL}}(\mathcal{D}||\pi) + \beta \mathbb{D}_{\text{KL}}(\pi||\pi_{\text{ref}}) \tag{21.131}$$

where $\mathcal{L}_{\mathcal{D}}(\pi)$ measures how well π explains the preferences in $\mathcal{D}$ (i.e., data likelihood), and β controls the strength of prior regularization. Intuitively, the reason we need regularization to a prior is that $\mathcal{D}$ often doesn't cover all possible generations, which means we don't want our policy to start generating text that we haven't received human feedback on.

Pro vs SFT: Collecting preference data is often easier and cheaper than asking experts to write high-quality demonstrations.

Con vs. SFT: Each data point provides only relative information (e.g., 1 bit for pairwise comparison), which is much less informative than a full demonstration.

21.6 Two-Stage RLHF: Information Geometry View

Let's analyze the standard two-stage process (RM training + RL optimization) using concepts from the field of *information geometry*.

21.6.1 Stage 1: Reward Modeling as MLE (FKL Projection)

We typically model human preferences using the Bradley-Terry (BT) model. It assumes an underlying latent reward function r shared across the entire population such that the probability of preferring ξ_1 over ξ_2 given prompt s_0 is:

$$\mathbb{P}_r(\xi_1 \succ \xi_2 | s_0) = \sigma(r(\xi_1) - r(\xi_2)) \tag{21.132}$$

where $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function.

Remark 29 *This model assumes consistent preferences across the population, which frequently does not hold in practice. In such situations, intransitivity might occur from preference aggregation, which means that* ***no*** *reward function can explain the observed preferences. We will discuss how to deal with this issue in a later lecture.*

Also, let $\mathbb{P}_\mathcal{D}(\xi_1 \succ \xi_2 | s_0)$ denote the empirical probability (frequency) in the dataset $\mathcal{D}$ that ξ_1 was preferred over ξ_2 for prompt s_0 by our pool of raters..

A reward model is nothing but a classifier. Training the reward model r_ϕ (parameterized by ϕ, often within a space $\mathcal{R}$) is performed by maximizing the likelihood of the observed human preferences in $\mathcal{D}$. This is equivalent to minimizing the forward KL divergence from the empirical preference distribution $\mathbb{P}_\mathcal{D}$ to the model's predicted preference distribution $\mathbb{P}_{r_\phi}$:

$$\begin{aligned}
\hat{r}_{\text{mle}} = r_\phi^* &= \underset{r_\phi \in \mathcal{R}}{\text{argmin}} \mathbb{E}_{s_0 \sim \mathcal{D}} \left[\mathbb{D}_{\text{KL}}(\mathbb{P}_\mathcal{D}(\cdot \succ \cdot | s_0) || \mathbb{P}_{r_\phi}(\cdot \succ \cdot | s_0)) \right] &(21.133)\\
&= \underset{r_\phi \in \mathcal{R}}{\text{argmax}} \mathbb{E}_{(s_0, \xi^+, \xi^-) \sim \mathcal{D}} \left[\log \mathbb{P}_{r_\phi}(\xi^+ \succ \xi^- | s_0) \right] &(21.134)\\
&= \underset{r_\phi \in \mathcal{R}}{\text{argmax}} \mathbb{E}_{(s_0, \xi^+, \xi^-) \sim \mathcal{D}} \left[\log \sigma(r_\phi(\xi^+) - r_\phi(\xi^-)) \right] &(21.135)
\end{aligned}$$

Observe that this is precisely the objective function for training a classifier using logistic regression on the preference pairs. From an information geometry perspective, this MLE step is performing a **Forward KL projection** (FKL) of the empirical preference distribution $\mathbb{P}_\mathcal{D}$ onto the space of realizable preference distributions induced by reward models in $\mathcal{R}$.

21.6.2 Stage 2: RL Optimization as MaxEnt RL (RKL Projection)

In the second stage, we use the learned reward model $\hat{r}_{\text{mle}}$ to optimize the policy π (parameterized by θ, within a space Π). The objective is to maximize the expected reward under the policy π, while regularizing with the reverse KL divergence to the reference policy π_{ref} (usually the SFT policy) to prevent *reward hacking* caused by limited $\mathcal{D}$ coverage:

$$\hat{\pi}_{\text{rlhf}} = \underset{\pi \in \Pi}{\text{argmax}} \mathbb{E}_{\xi \sim \pi(\cdot | s_0)} \left[\hat{r}_{\text{mle}}(\xi) \right] - \beta \mathbb{D}_{\text{KL}}(\pi(\cdot | s_0) || \pi_{\text{ref}}(\cdot | s_0)) \tag{21.136}$$

Here, β is a hyperparameter controlling the regularization strength. The KL divergence term is calculated over entire sequences ξ:

$$\mathbb{D}_{\mathrm{KL}}(\pi||\pi_{\mathrm{ref}}) = \mathbb{E}_{\xi\sim\pi}\left[\log\frac{\pi(\xi|s_0)}{\pi_{\mathrm{ref}}(\xi|s_0)}\right] = \mathbb{E}_{\xi\sim\pi}\left[\sum_{h=1}^{H}\log\frac{\pi(a_h|s_{h-1})}{\pi_{\mathrm{ref}}(a_h|s_{h-1})}\right] \tag{21.137}$$

This is a standard objective in entropy-regularized RL or "Soft RL". Recall from a past lecture that the optimal policy $\pi^\star$ for this objective has the form:

$$\mathbb{P}^\star_{\hat{r}_{\mathrm{mle}}}(\xi|s_0) = \frac{1}{Z(s_0)}\mathbb{P}_{\mathrm{ref}}(\xi|s_0)\exp\left(\frac{1}{\beta}\hat{r}_{\mathrm{mle}}(\xi)\right) = \prod_{h}^{H}\pi^\star_{\hat{r}_{\mathrm{mle}}}(a_h|s_h), \tag{21.138}$$

where $Z(s_0)$ is the partition function ensuring the distribution sums to 1 over all possible sequences ξ starting from s_0, and the RHS equality uses the fact that the dynamics are deterministic. We don't discuss the proof in lecture but it can be shown that solving this soft RL problem over some policy class Π is equivalent to projecting $\mathbb{P}^\star_{\hat{r}_{\mathrm{mle}}}$ onto the space of trajectory distributions $\mathbb{P}_\pi$ induced by $\pi \in \Pi$ under the reverse KL metric:

$$\hat{\pi}_{\mathrm{rlhf}} = \underset{\pi\in\Pi}{\operatorname{argmin}}\,\mathbb{D}_{\mathrm{KL}}(\mathbb{P}_\pi||\mathbb{P}^\star_{\hat{r}_{\mathrm{mle}}}) \tag{21.139}$$

End-to-End Summary: The standard two-stage RLHF process first performs an FKL projection from the data $\mathcal{D}$ to the reward space $\mathcal{R}$ (MLE for $\hat{r}_{\mathrm{mle}}$), and then an RKL projection from the reward-induced target distribution $\mathbb{P}^\star_{\hat{r}_{\mathrm{mle}}}$ to the policy space Π (MaxEnt RL for $\hat{\pi}_{\mathrm{rlhf}}$). We visualize this process in Figure 21.26.

21.7 Direct Alignment Algorithms (e.g., DPO)

Direct Alignment Algorithms aim to bypass the explicit reward modeling step (Stage 1) and directly optimize the policy π using the preference data $\mathcal{D}$ via offline maximum likelihood estimation. Direct Preference Optimization (DPO) is a prominent example.

The core idea of DPO is to re-express the reward function in terms of the optimal policy and the reference policy. Recall the form of the soft optimal policy $\pi^\star$ for a given reward r:

$$\mathbb{P}^\star_r(\xi|s_0) = \frac{1}{Z(s_0)}\mathbb{P}_{\mathrm{ref}}(\xi|s_0)\exp\left(\frac{1}{\beta}r(\xi)\right) \tag{21.140}$$

We can take a log on both sides and and rearrange to solve for $r(\xi)$:

$$\log\mathbb{P}^\star_r(\xi|s_0) = \log\mathbb{P}_{\mathrm{ref}}(\xi|s_0) + \frac{1}{\beta}r(\xi) - \log Z(s_0) \tag{21.141}$$

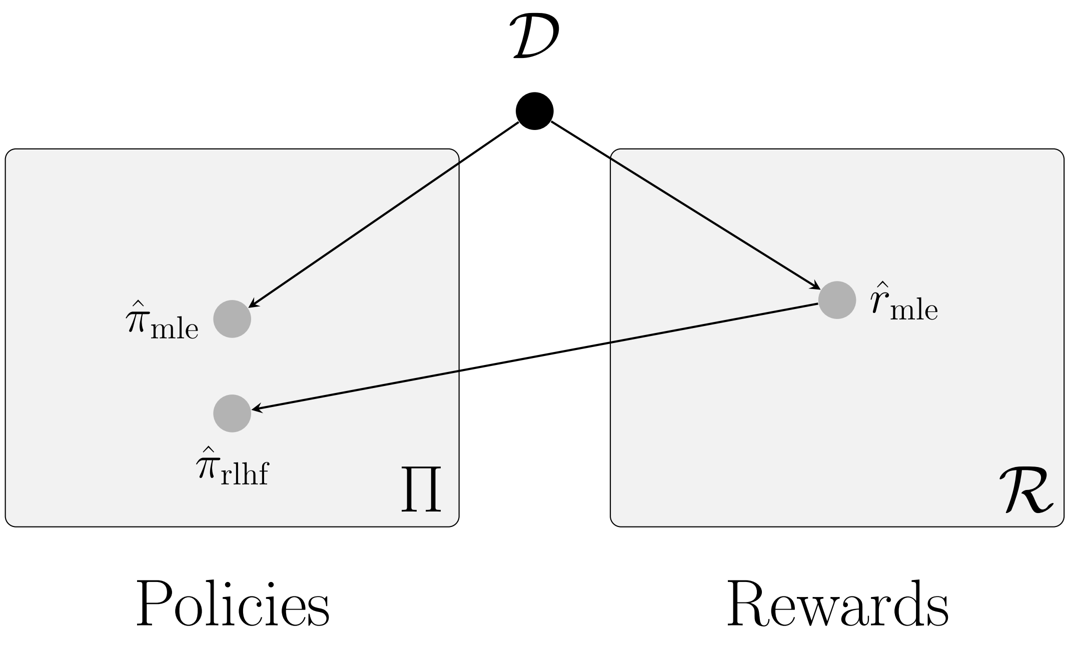


Figure 21.26: Geometric Interpretation of RLHF and DPO (Adapted from [22]).

$$r(\xi) = \beta \left(\log \frac{\mathbb{P}_r^\star(\xi|s_0)}{\mathbb{P}_{\text{ref}}(\xi|s_0)} \right) + \beta \log Z(s_0) \tag{21.142}$$

Since $\mathbb{P}_r^\star(\xi|s_0) = \prod_h \pi^\star(a_h|s_{h-1})$ and $\mathbb{P}_{\text{ref}}(\xi|s_0) = \prod_h \pi_{\text{ref}}(a_h|s_{h-1})$ due to the deterministic dynamics, we can expand to write terms using token-level probabilities:

$$r(\xi) = \beta \sum_{h=1}^{H} (\log \pi^\star(a_h|s_{h-1}) - \log \pi_{\text{ref}}(a_h|s_{h-1})) + \beta \log Z(s_0) \triangleq r_\pi(\xi). \tag{21.143}$$

Thus, we can express the reward function that makes a policy soft optimal in terms of said policy by inverting the MaxEnt RL equations. So, while soft value iteration lets us go from r to $\pi^\star$, DPO tells us that logistic regression lets us go in the reverse direction.

Remark 30 (Your Language Model is Secretly a Reward Model) *One can show that the optimal policy for the entropy-regularized RL objective using this implicit reward r_π is* *precisely the policy π itself.*

$$\begin{aligned}
\mathbb{P}^{\star}_{r_\pi}(\xi) &\propto \exp(r_\pi(\xi)) && (21.144)\\
&\propto \exp\left(\sum_h^H \log \pi(a_h|s_h) + \log Z(s_0)\right) && (21.145)\\
&\propto \exp\left(\sum_h^H \log \pi(a_h|s_h)\right) && (21.146)\\
&\propto \prod_h^H \pi(a_h|s_h). && (21.147)
\end{aligned}$$

Thus, if we optimize over r_π, we get the corresponding soft-optimal policy π "for-free".

The DPO Objective: DPO substitutes this implicit reward function r_π directly into the Bradley-Terry likelihood objective used for reward modeling (i.e. Stage 1), cancelling out the partition function as both trajectories share the same prompt:

$$\begin{aligned}
\hat{\pi}_{\text{dpo}} &= \underset{\pi\in\Pi}{\text{argmax}}\mathbb{E}_{(s_0,\xi^+,\xi^-)\sim\mathcal{D}}\left[\log\sigma(r_\pi(\xi^+) - r_\pi(\xi^-))\right]\\
&= \underset{\pi\in\Pi}{\text{argmax}}\mathbb{E}_{(s_0,\xi^+,\xi^-)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi(\xi^+|s_0)}{\pi_{\text{ref}}(\xi^+|s_0)} - \beta\log\frac{\pi(\xi^-|s_0)}{\pi_{\text{ref}}(\xi^-|s_0)}\right)\right]
\end{aligned}$$

This results in a single-stage optimization problem where we directly maximize the likelihood of the preference data under the policy π without any RL / on-policy sampling.

Information Geometry View of DPO: DPO performs a forward KL projection (FKL) directly from the empirical preference distribution $\mathbb{P}_{\mathcal{D}}$ onto the policy space Π. It bypasses the intermediate reward model space $\mathcal{R}$.

- Standard RLHF: $\mathcal{D} \xrightarrow{\text{FKL (MLE)}} \mathcal{R} \xrightarrow{\text{RKL (MaxEnt RL)}} \Pi$
- DPO: $\mathcal{D} \xrightarrow{\text{FKL (MLE)}} \Pi$

Key Point: While both methods aim to align the policy with preferences, the resulting policies $\hat{\pi}_{\text{rlhf}}$ and $\hat{\pi}_{\text{dpo}}$ are not necessarily identical, as we discuss in subsequent lectures.

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 22: The Value of RL in Fine-Tuning

Lecturer: Gokul Swamy
Scribe: Harshita D., Michael C., Zichun Y., Yiming Z.

22.1 Outline

1. What assumption on the preference dataset did we make in the DPO derivation and what happens when it breaks?
   A: Full coverage of $\mathcal{D}$. Without it, we can't control the RKL.
2. When are two-stage RLHF and DPO equivalent?
   A: When Π and $\mathcal{R}$ are isomorphic and all projections are exact.
3. Why does two-stage RLHF work much better in practice?
   A: RLHF only has to search over policies (generators) that are optimal for simple rewards (verifiers) rather than over all of Π, which requires less data.

22.2 Recap

Recall the formulas for *forward* and *reverse* KL divergence.

- FKL: $\min_p \mathbb{D}_{KL}(q||p) = \min_p \sum_x q(x) \log\left(\frac{q(x)}{p(x)}\right)$ (MLE)
- RKL: $\min_p \mathbb{D}_{KL}(p||q) = \min_p \sum_x p(x) \log\left(\frac{p(x)}{q(x)}\right)$ (Soft RL)

FKL is mode-covering while RKL is mode-seeking. To see why, imagine we tried fitting a single Gaussian distribution to a bimodal sum of Gaussians. FKL doesn't want $\frac{q(x)}{p(x)}$ to get too big, so it ensures $p(x)$ is never too small when $q(x)$ is large. This results in it roughly fitting an "average" of the sum of Gaussians. However, RKL doesn't want $\frac{p(x)}{q(x)}$ to get too big, so it ensures $p(x)$ is never too large when $q(x)$ is close to zero. This results in it just trying to fit to one of the modes. Note that both of these extremes (average of the two modes from FKL vs. just one mode from RKL) result in a very high divergence in the other direction.

In RLHF, we often care about mode selection (mode-seeking) more than mode covering (i.e. we'd prefer to do one thing well rather than two things poorly).

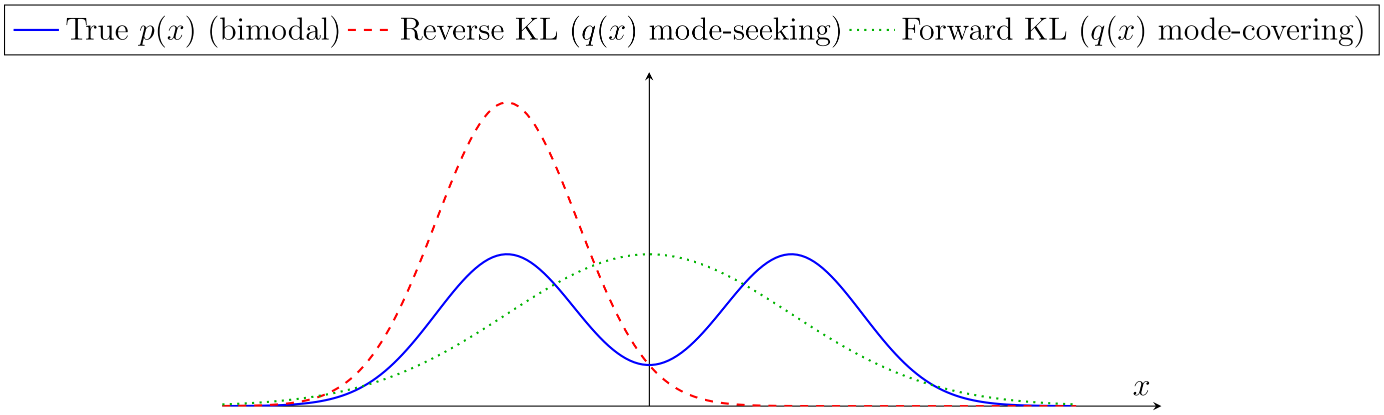


Figure 22.27: Forward KL covers both modes while reverse KL selects one when trying to fit a bimodal mixture of Gaussians with a single Gaussian.

Last lecture, we described two methods for learning from preference feedback:

1. Two-stage RLHF: learn a reward model $\hat{r}_{\text{mle}}$ via logistic regression over $\mathcal{D}$. MLE corresponds to minimizing the FKL. Then, learn a policy via soft RL on the RM:

$$\hat{\pi}_{\text{rlhf}} = \arg\max_{\pi \in \Pi} \mathbb{E}_{\xi \sim \pi} [\hat{r}_{\text{mle}}(\xi)] + \mathbb{D}_{KL}(\pi || \pi_{\text{ref}}). \tag{22.148}$$

   As discussed last time, soft RL corresponds to minimizing the RKL.

2. One-step Direct Alignment (e.g. DPO): directly learn a policy over $\mathcal{D}$ via MLE ($\approx$ logistic regression over the space of policies).

In practice, most if not all frontier models use the two-stage procedure. However, it is significantly more computationally intensive as it requires sampling from the policy and performing finicky policy gradient updates. The key question we will focus on for the rest of this lecture is exploring when this extra complexity in implementation is beneficial.

22.3 DPO Breaks under Partial Coverage

As we will now discuss, DPO implicitly assumes full coverage of the preference dataset $\mathcal{D}$, an assumption that is rarely if ever true in practice. Without this assumption, DPO cannot control the RKL. Intuitively, this is because RKL involves an *on-policy* expectation, which can only be estimated if $\mathcal{D}$ covers all places the learner might visit.

More explicitly, suppose we had three possible completions ξ_1, ξ_2, ξ_3 with probabilities $0.5, 0.5, 0$ under π_{ref}. Furthermore, assume that our human labeler always prefer $\xi_1 \succ \xi_2$.

First, let's consider the two-stage procedure. When fitting $\hat{r}_{\text{mle}}$ on $\mathcal{D}$, we would want to

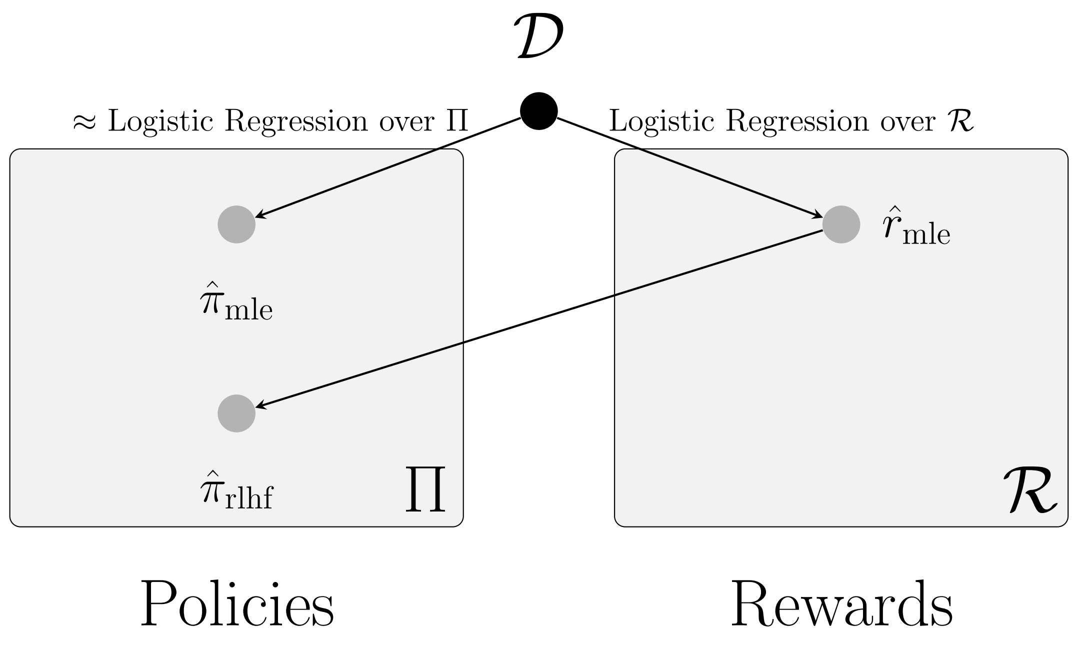

$\mathcal{D}$
$\approx$ Logistic Regression over Π
Logistic Regression over $\mathcal{R}$
$\hat{\pi}_{\text{mle}}$
$\hat{r}_{\text{mle}}$
$\hat{\pi}_{\text{rlhf}}$
Π
$\mathcal{R}$
Policies
Rewards

Figure 22.28: We visualize the two-stage and one-stage approaches to PFT.

assign a much higher score to ξ_1 than to ξ_2. However, because we never see any preference data with ξ_3, we could erroneously assign it a high reward. For example, $\hat{r}_{\text{mle}} = [1, 0, 10]$ has a high Bradley-Terry likelihood. However, the RKL term blows up if we place any probability mass on ξ_3, so $\hat{\pi}_{\text{rlhf}}$ wouldn't generate ξ_3 and therefore awards *reward hacking*.

Now, let us consider applying DPO to this problem. Because π_{ref} has the same probability of generating either ξ_1 or ξ_2, we can drop the reference probabilities from the DPO loss and end up with a vanilla logistic regression objective:

$$\arg\max_{\pi\in\Pi} \mathbb{E}_{\mathcal{D}}\left[\log\sigma\left(\sum_h^H \log\frac{\pi(a_h^+|s_h^+)}{\pi_{\mathsf{ref}}(a_h^+|s_h+)} - \log\frac{\pi(a_h^-|s_h^-)}{\pi_{\mathsf{ref}}(a_h^-|s_h^-)}\right)\right] \tag{22.149}$$

$$= \arg\max_{\pi\in\Pi} \mathbb{E}_{\mathcal{D}}\left[\log\sigma\left(\sum_h^H \log\pi(a_h^+|s_h^+) - \log\pi(a_h^-|s_h^-)\right)\right] \tag{22.150}$$

$$= \arg\max_{\pi\in\Pi} \log\sigma\left(\sum_h^H \log\pi(a_h^1|s_h^1) - \log\pi(a_h^2|s_h^2)\right). \tag{22.151}$$

Observe this loss function does not constrain the probability of generating ξ_3 in any way and there is no on-policy RKL correction to fix it. In summary, DPO doesn't regularize to π_{ref} properly in the partial coverage setting, and can therefore produce OOD responses.

22.4 When are two-stage RLHF and DPO equivalent?

As we will now argue, when the space of policies Π is isomorphic to the space of reward models $\mathcal{R}$ and all projections are exact, DPO and RLHF will produce the same policy (i.e. $\hat{\pi}_{\text{mle}} = \hat{\pi}_{\text{rlhf}}$). For simplicity, we will assume there is no prior regularization, but a more general version of the following claim can be proved under slightly stronger assumptions.

Proof: First, we observe that because we are minimizing the same functional over isomorphic classes, we know that $\hat{\pi}_{\text{mle}}$ and $\hat{r}_{\text{mle}}$ must achieve the same loss value:

$$\mathbb{E}_{\mathcal{D}}\left[\log\sigma\left(\sum_{h}^{H}\left(\log\hat{\pi}_{\text{mle}}(a_h^+ \mid s_h^+) - \log\hat{\pi}_{\text{mle}}(a_h^- \mid s_h^-)\right)\right)\right] = \mathbb{E}_{\mathcal{D}}\left[\log\sigma\left(\hat{r}_{\text{mle}}(\xi^+) - \hat{r}_{\text{mle}}(\xi^-)\right)\right]. \tag{22.152}$$

Assuming uniqueness of minimizers for simplicity, we then know that

$$\forall\,\xi \in \Xi, r_{\hat{\pi}_{\text{mle}}}(\xi) = \sum_{h}^{H}\log\hat{\pi}_{\text{mle}}(a_h \mid s_h) = \hat{r}_{\text{mle}}(\xi). \tag{22.153}$$

Next, we recall that soft RL can be written as an RKL projection onto Π and substitute $\hat{r}_{\text{mle}}(\xi)$ for $r_{\hat{\pi}_{\text{mle}}}(\xi)$:

$$\begin{aligned}\hat{\pi}_{\text{rlhf}} &= \arg\min_{\pi\in\Pi}\mathbb{D}_{KL}\left(\mathbb{P}_{\pi} \,\|\, \mathbb{P}^{\star}_{\hat{r}}\right)\\ &= \arg\min_{\pi\in\Pi}\mathbb{D}_{KL}\left(\mathbb{P}_{\pi} \,\|\, \mathbb{P}_{r_{\hat{\pi}}}\right)\\ &= \hat{\pi}_{\text{mle}}\end{aligned}$$

The last step above uses the DPO isomorphism we discussed last lecture. ■

If you've taken a statistics class before, you might have heard this stated more formally as *MLE is invariant to reparameterization*. Interestingly enough, we're roughly in the isomorphic classes setting for practical applications of RLHF to LLMs. This is because both the policy and the reward model are fine-tuned from the same SFT checkpoint. Specifically, to get an RM, one removes the final softmax from the SFT policy and then adds a linear layer.

The above result should be surprising to you – we've proved theoretically that there should be no benefit to RL in fine-tuning but practically, we see everyone using RL-based methods. We will now focus on reconciling this theory-practice gap via the use of controlled experiments.

22.5 Why is RLHF > DPO in Practice?

We will focus on the task of summarization of Reddit posts, using models from the Pythia family pre-trained on the Pile. To eliminate confounders, we will try to equalize as many things as possible between offline and online PFT:

1. We will use the **same** dataset to train both policies and reward models.
2. We will start from the **same** SFT checkpoint to train both.
3. We will use the **same** optimizer (DPO) for both online and offline PFT with the **same** hyperparameters.

By online DPO, we mean sampling from the policy, ranking the samples with the reward model, and using the top and bottom of the list as a new preference pair for a DPO procedure. Thus, the *only* difference between the offline and online PFT procedures is the data passed to them – the offline algorithm learns directly from the human preferences, while the online procedure uses the inputed preferences generated by reward model rankings.

Key Takeaways While maintaining the same experimental configuration (same initial checkpoint, training data, objective function, same gradient steps) online PFT robustly out-performs offline PFT. See [22] for full results.

22.5.1 Hypotheses for the Online-Offline Gap

We now explore and refute several hypotheses for the online-offline performance gap – see [22] for the complete list and more comprehensive experiments.

1. **Hypothesis 1: Intrinsic Value of On-Policy Feedback.** If the RM was able to give us new information during on-policy sampling, we might hope to out-perform the offline algorithm. This is the heart of the classic separations between online and offline RL. However, we do not observe any ground-truth reward labels during on-policy sampling – the labels are merely imputed by an RM trained on the same data as the offline policy. Furthermore, due to the data processing inequality, we cannot create any new information (i.e. bona fide new human preferences) via on-policy sampling.
2. **Hypothesis 2: Failure of Offline Regularization.** As we discussed above, offline algorithms like DPO need strong assumptions to properly regularize to the prior. While this is true in general, we used DPO as our optimizer for both the online and offline experiments. Thus, this fact cannot explain the preceding results.
3. **Hypothesis 5: RMs generalize better OOD than Policies.** Intuitively, if our RM generalizes better OOD than the *implicit* RM r_π we train in DPO-like approaches,

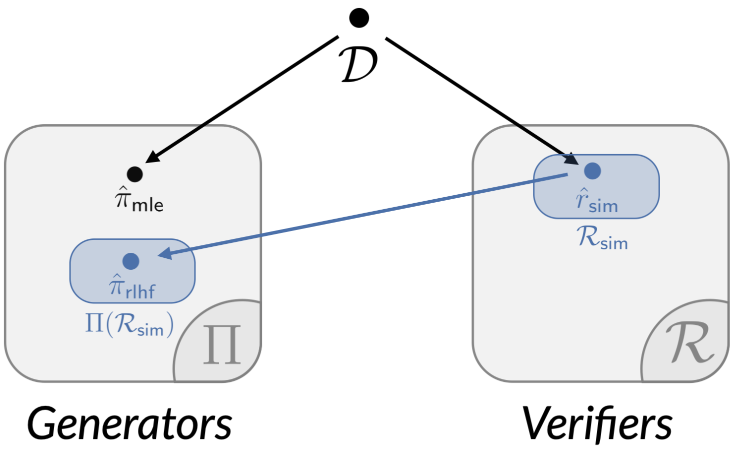

$\mathcal{D}$
$\hat{\pi}_{\mathsf{mle}}$
$\hat{\pi}_{\mathsf{rlhf}}$
$\Pi(\mathcal{R}_{\mathsf{sim}})$
Π
$\hat{r}_{\mathsf{sim}}$
$\mathcal{R}_{\mathsf{sim}}$
$\mathcal{R}$
Generators
Verifiers

Figure 22.29: On problems with a *generation-verification gap*, the first stage of RLHF can be thought of picking a relatively simple reward model $\hat{r}_{\text{sim}} \in \mathcal{R}_{\text{sim}}$. The secondary RL step will pick a policy that is soft-optimal for this relatively simple verifier. Thus, end to end, the two-stage RLHF procedure only needs to look at policies that are soft-optimal for relatively simple verifiers $\Pi(\mathcal{R}_{\text{sim}})$, rather than across all of Π like vanilla maximum likelihood estimation. This provides a *statistical* benefit (i.e., fewer samples required for learning).

we might expect online PFT perform better. This is true (as measured in terms of Best-of-N performance), but is best explained by the fact that RMs also generalize better *in-distribution* (i.e. they have better validation BT likelihoods). Intuitively, having better margins in distribution is often correlated with better OOD generalization, similar to the arguments for maximizing the margin of an SVM. However, this just kicks the can down the road – we haven't explained *why* RMs should have better BT validation likelihoods ID than implicit RMs. It isn't obvious they should, given we're minimizing the same loss function over similar function classes using the same dataset.

22.5.2 Generation-Verification Gaps in RLHF

We now discuss an alternative hypothesis grounded in the phenomenon of *generation-verification gaps*: that for many problems, it is easier to *check* that one has the right answer rather than to generate it in the first place. For example, checking if a Sudoku puzzle is correctly solved is much easier (just check every row, column and square) rather than generating a complete solution (requires an exponentially large search). A bit more formally, this is what problems in NP but not P are like. In RL, policies are *generators* while reward models are *verifiers*. If we believe that the underlying reward function for a problem is simpler than the corresponding soft-optimal policy, this means that it should take less data to learn the former. Then, the second RL stage of RLHF only picks policies that soft-optimal for the relatively simple learned verifier. Thus, end to end, two-stage RLHF only has to look at the *subset* of policies that are soft-optimal for relatively simple verifiers, rather than across *all* policies, which can require much more preference data to search across. This hypothesis is saying there is a *statistical* separation between online and offline PFT. We note in passing that this hypothesis is analogous to the statistical benefit of inverse RL we discussed where the learner has a reduction in search space to just those policies that are on the Pareto frontier. We visualize this reduction in effective hypothesis class size in Figure 22.29. Observe that this hypothesis explains why there should still be an online-offline performance gap in the isomorphic classes setting. Furthermore, it explains why RMs generalize better ID than implicit RMs, as one is attempting to learn a simpler function in the former case and therefore should require less data to do so (via one of many arguments from statistical learning theory).

[22] provide evidence that the summarization problem we consider has a GV-gap by noting relatively small reward models can accurately rank samples from relatively large policy. Furthermore, they observe that using an RM that is much larger than the generating policy does not improve upon the performance of an RM of the same size as the generating policy. They also discuss how this hypothesis explains all other experimental results in their paper. To further stress-test the hypothesis, they then perform experiments where they eliminate the GV-gap and see that offline and online PFT collapse to the same performance, as predicted by the information-theoretic picture we started off the lecture with.

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 23: RL via Regressing Relative Rewards

Lecturer: Wen Sun
Scribe: Pranjal Aggarwal

23.1 Recap

Recall that in RLHF, we're trying to solve the following KL-regularized RL problem:

$$J(\pi) = \mathbb{E}_{x\sim\mathcal{D}}\left[\mathbb{E}_{\tau\sim\pi(\cdot|x)} r(x,\tau) - \beta \mathrm{KL}(\pi(\cdot|x) \parallel \pi_{\text{ref}}(\cdot|x))\right], \tag{23.154}$$

where

- x is the input context (prompt),
- τ is the sequence of actions / tokens or response generated by the policy π,
- $r(x,\tau)$ is the reward associated with the prompt-response pair (x,τ),
- β is a scalar factor controlling KL penalty strength,
- π_{ref} is the reference policy.

As we discussed previously, the optimal policy $\pi^\star$ has the following closed form:

$$\hat{\pi}(\tau|x) \propto \pi_{\text{ref}}(\tau|x) \cdot \exp\left(\frac{r(x,\tau)}{\beta}\right). \tag{23.155}$$

23.1.1 Recap: DPO

DPO takes the closed-form solution above and reparameterizes the reward using the policy. It models the reward difference in terms of the policy. The DPO training objective is:

$$\arg\max_\theta \sum_{x,\tau,\tau',z} \ln \frac{1}{1+\exp\left(-\beta\left(\ln\frac{\pi_\theta(\tau|x)}{\pi_{\text{ref}}(\tau|x)} - \ln\frac{\pi_\theta(\tau'|x)}{\pi_{\text{ref}}(\tau'|x)}\right)\right)}, \tag{23.156}$$

where

- x is a prompt,
- τ, τ' are responses to x,

- $z \in \{-1, 0, 1\}$ labels preference: $z = 1$ if τ is preferred to τ',
- $\pi_\theta(\tau|x)$ is the probability of generating τ given x under policy π_θ,
- $\pi_{\theta_{\text{ref}}}(\tau|x)$ is the probability under the reference policy π_{ref}.

DPO's performance is often weaker than Reward Model (RM) + PPO due to the *generation-verification gap*: evaluating a response is often easier than generating one. Furthermore, online RL approaches allow one to use state-of-the-art RMs (e.g., from RewardBench) trained by the community and train the policy on prompts unseen in the preference data.

Main Question: PPO is computationally expensive, requiring storing four large models in memory: π, π_{ref}, the reward model r, and the critic / value function V. Can we develop a more efficient and potentially more effective RL algorithm?

23.2 Mirror Descent

Mirror Descent is a well-studied no-regret algorithm that most, if not all, popular RL algorithms can be thought of as approximating. [6] We will begin by discussing this idealized algorithm before describing how best to approximate it. Given some reward function $r(x, \tau)$, RL focuses on finding some policy π that maximizes expected reward:

$$\max_{\pi} \mathbb{E}_{x,\tau\sim\pi(.|x)}[r(x, \tau)]. \tag{23.157}$$

At each iteration, t, *mirror descent* attempts to solve the above via the following update:

$$\pi_{t+1} = \arg\max_{\pi} \mathbb{E}_{x,\tau\sim\pi_t(.|x)} \left[r(x, \tau) - \beta \text{KL}(\pi(\cdot|x) \parallel \pi_t(\cdot|x))\right]. \tag{23.158}$$

Observe that this is nothing but the KL-regularized / MaxEnt RL problem we've been studying, but with $\pi_{\text{ref}} = \pi_t$ (i.e. we regularize to the last policy we learned). Pattern matching from the above, we know that the closed-form solution to the above problem is

$$\pi_{t+1}(\tau|x) = \frac{\pi_t(\tau|x) \exp(r(x, \tau)/\beta)}{Z(x)}, \tag{23.159}$$

where $Z(x)$ is the normalization constant. We can expand out this recursion over T iterations:

$$\pi_{t+T}(\tau|x) \propto \pi_t(\tau|x) \exp(\sum_{i=1}^{T} r(x, \tau)/\beta) \propto \pi_{\text{ref}}(\tau|x) \exp(\sum_{i=1}^{t+T} r(x, \tau)/\beta) \tag{23.160}$$

[6]In fact, the Follow The Regularized Leader algorithm we discussed extensively in earlier lectures is equivalent to mirror descent for a particular choice of the regularizer.

This update should be intuitive: we're steadily up-weighting high-reward trajectories and down-weighting the prior π_{ref}. Observe that this down-weighting implies that we may eventually drift away from the reference policy, but this is acceptable if we trust our reward model. We don't prove in lecture but exact mirror descent has a *fast* rate of $\mathcal{O}(1/T)$.

While conceptually elegant, exact mirror descent is difficult to implement if π and r are neural networks (transformers) – exactly computing the above product for *all* τ (i.e. all prompt completions) is computationally infeasible. Thus, we will now discuss how to approximate this idealized update via a simple, square-loss regression.

23.3 Reparameterization Trick and REBEL

Taking a log on both sides of the above expression and re-arranging terms, we get:

$$r(x,\tau) = \beta \left(\ln \frac{\pi_{t+1}(\tau|x)}{\pi_t(\tau|x)} \right) + \ln Z(x) \tag{23.161}$$

where $Z(x) = \mathbb{E}_{\tau\sim\pi_t(.|x)}[\exp(r(x,\tau)/\beta)]$. Akin to DPO, to eliminate the partition function, we can consider the *relative reward* between two completions to the same prompt x:

$$r(x,\tau) - r(x,\tau') = \beta \left(\ln \frac{\pi_{t+1}(\tau|x)}{\pi_t(\tau|x)} - \ln \frac{\pi_{t+1}(\tau'|x)}{\pi_t(\tau'|x)} \right). \tag{23.162}$$

For a small, tabular problem, we could attempt to solve the above equation exactly. Unfortunately, this is infeasible at the scale of language models. Thus, we will minimize the squared difference between the LHS and RHS of the above expression, giving us the following:

$$\pi_{t+1} = \arg\min_{\pi} \mathbb{E}_{x,\tau,\tau'\sim\pi_t(.|x)} \left[\underbrace{\beta \left(\ln \frac{\pi(\tau|x)}{\pi_t(\tau|x)} - \ln \frac{\pi(\tau'|x)}{\pi_t(\tau'|x)} \right)}_{\text{Regressor}} - \underbrace{(r(x,\tau) - r(x,\tau'))}_{\text{Relative reward}} \right]^2, \tag{23.163}$$

where τ, τ' are sampled independently from the latest policy $\pi_t(\cdot|x)$. [7] We call this the **REBEL** (REgression to RElative REward Based RL) update. Observe that if we were able to perfectly minimize this objective to zero and π_t has full support for all x, we would recover the exact mirror descent update. One can still prove strong guarantees with *approximate* minimization via reduction to supervised learning, as is discussed further in the paper.

[7] One can instead compare the current policy to samples from some *baseline* distribution, as discussed in the full paper. This could be suboptimal data from the internet or SFT data.

23.4 Differences between REBEL, DPO, and PPO

It is worth pausing for a moment to consider the differences between the three RLHF algorithms we've discussed in this course.

1. **DPO:** Uses finite, offline preference data. Can't take advantage of reward models or generation-verification gaps. Simple, supervised learning-based update.
2. **PPO:** Can use an arbitrary RM and optimize on unseen prompts. Complex update, requiring attention to detail in terms of implementation (e.g. clipping, baselines, GAE) and having four separate models in memory.
3. **REBEL:** Achieves the best of both worlds: simple, regression based update akin to DPO. Doesn't require critic network like PPO. Can take advantage of arbitrary RMs.

Intuitively, one can think of REBEL as DPO but with a reward model (that may or may not be learned from human preference data).

23.5 Connecting REBEL to Other Algorithms

To recap, REBEL minimizes the following least squares loss problem at each iteration:

$$\ell_t(\theta) = \mathbb{E}_{x,\tau,\tau' \sim \pi_{\theta_t}(\cdot|x)} \left[\beta \left(\ln \frac{\pi_\theta(\tau|x)}{\pi_{\theta_t}(\tau|x)} - \ln \frac{\pi_\theta(\tau'|x)}{\pi_{\theta_t}(\tau'|x)} \right) - (r(x,\tau) - r(x,\tau')) \right]^2 \tag{23.164}$$

Let us consider what happens when we *approximately* minimize the above loss function.

23.5.1 One Step of Gradient Descent

Consider taking a single step of gradient descent in the above objective, i.e.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta l_t|_{\theta_t} \tag{23.165}$$

For some fixed (x, τ, τ'), the gradient at $\theta = \theta_t$ is

$$\nabla_\theta l(\theta_t) = \beta \left(\nabla_\theta \ln \pi_{\theta_t}(\tau|x) - \nabla_\theta \ln \pi_{\theta_t}(\tau'|x) \right) \left(r(x,\tau) - r(x,\tau') \right). \tag{23.166}$$

Observe that this recovers the REINFORCE Leave One Out (RLOO) update (i.e. REINFORCE, but using the reward of the other sample in the batch as a baseline).

23.5.2 One Step of Gauss-Newton

To take a Gauss-Newton (GN) step, we first approximate the non-linear part inside the square via a first-order Taylor expansion at θ_t:

$$\ln \frac{\pi_\theta(\tau|x)}{\pi_{\theta_t}(\tau|x)} - \ln \frac{\pi_\theta(\tau'|x)}{\pi_{\theta_t}(\tau'|x)} \approx \left(\nabla \ln \pi_{\theta_t}(\tau|x) - \nabla \ln \pi_{\theta_t}(\tau'|x)\right)^T (\theta - \theta_t). \tag{23.167}$$

We then plug this linear approximation back into the least squares problem:

$$\theta_{t+1} = \arg\min_\theta \mathbb{E}_{x,(\tau,\tau')\sim\pi_{\theta_t}(\cdot|x)} \left[\Big(\left(\nabla \ln \pi_{\theta_t}(\tau|x) - \nabla \ln \pi_{\theta_t}(\tau'|x)\right)^T (\theta - \theta_t) - (r(x,\tau) - r(x,\tau')) \Big)^2 \right]$$

This is precisely the "regression view" of the Natural Policy Gradient (NPG) we discussed many lectures ago! Thus, NPG can be seen as an approximation of the REBEL update.

23.5.3 Other Domains

REBEL can be applied to other RL problems, such as continuous control or fine-tuning image diffusion models. While we assumed deterministic dynamics in the above derivation, if we expand the probability of a trajectory as

$$\pi(\zeta|s_0) = \prod_{h=0}^{H-1} \pi(a_h|s_h) P(s_{h+1}|s_h, a_h), \tag{23.168}$$

observe that

$$\ln \frac{\pi(\zeta|s_0)}{\pi_{\text{old}}(\zeta|s_0)} = \sum_{h=0}^{H-1} \ln \frac{\pi(a_h|s_h)}{\pi_{\text{old}}(a_h|s_h)}, \tag{23.169}$$

i.e. that the dynamics cancel out as they are evaluated along the same trajectory. This could be useful in a multi-turn dialog setting, where the person's responses to what the agent says are stochastic. For such a problem, each action may more naturally correspond to an entire response, rather than a token. See the follow-up paper, REFUEL, for more information.

The REBEL algorithm can also be adapted to use a critic Q (rather than a reward model), which leads to a similar update to Soft Actor Critic (SAC):

$$\min_\pi \mathbb{E}_{s,a\sim\pi_t(\cdot|s)} \left[\left(\beta \left(\ln \frac{\pi(a|s)}{\pi_t(a|s)} - \ln \frac{\pi(a'|s)}{\pi_t(a'|s)} \right) - (Q(s,a) - Q(s,a')) \right)^2 \right], \tag{23.170}$$

where a and a' are sampled independently from the current policy $\pi_t(\cdot|s)$. This can help for longer horizon problems for which more precise credit assignment is required. If one doesn't want to explicitly train a critic, one can instead simply roll out the policy from the state s after taking some action a and observe the sum of rewards accumulated over the rest of the episode. This is an unbiased, Monte-Carlo estimate of the desired $Q^\pi(s, a)$ value. Recall that these sorts of "resets" are free in text problems as they are just generating from a prefix.

17-740 Spring 2025
Algorithmic Foundations of Interactive Learning

Lecture 24: RLHF as Game Solving

Lecturer: Gokul Swamy
Scribe: Apurva Gandhi, Jason Wei, Jehan Yang

24.1 Outline

We will cover two points in today's lecture:

1. When is the Bradley-Terry assumption inaccurate and what happens to online/offline PFT as a result?
   A: BT is violated when a reward function can't explain (aggregate) preferences, leading to mode collapse in RLHF.
2. What is a more robust criterion for preference aggregation and how can we efficiently optimize it?
   A: The minimax winner doesn't assume transitivity of preferences. We can use a self-play algorithm to compute it.

24.2 When is Bradley-Terry assumption inaccurate?

24.2.1 Preference Matrix

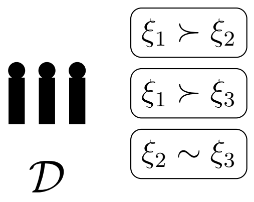
Given some set of human preferences recorded in a dataset $\mathcal{D}$, we can transform them into a **preference matrix** $\mathcal{P}$ where each entry $\mathcal{P}(\xi_i \succ \xi_j)$ is the empirical probability that generation ξ_i is preferred over ξ_j:

$$\mathcal{P}(\xi_i \succ \xi_j) \triangleq \mathbb{P}_{\mathcal{D}}(\xi_i \succ \xi_j). \tag{24.171}$$

From this definition, we can back out a few key properties of $\mathcal{P}$:

1. All elements on the diagonal must be 0.5.
2. Elements across the diagonal must sum to 1 (as an event and its compliment have a probability that sums to 1). This property will be key later.

Above, we give an example of the translation from dataset $\mathcal{D}$ to matrix $\mathcal{P}$. For this particular preference matrix, we can back out a reward function that *rationalizes* it (i.e. that the raters



$\mathcal{P}$	ξ_1	ξ_2	ξ_3
ξ_1	0.5	1	1
ξ_2	0	0.5	0.5
ξ_3	0	0.5	0.5

Figure 24.30: Observe how, without loss of generality, we can translate human preferences (left) into a matrix (right). The same is not true for reward functions.

could have had in their heads to guide the choices they made). Assuming our raters make choices according to the Bradley-Terry model of preferences, we know that the underlying r must satisfy the following set of constraints:

$$r(\xi_1) > r(\xi_2) \quad \& \quad r(\xi_1) > r(\xi_3) \quad \& \quad r(\xi_2) = r(\xi_3). \tag{24.172}$$

One such reward function is $r = [1, 0, 0]$. Unfortunately, it is not always possible to collapse a preference *matrix* into a reward *vector*. For example, if we instead had the following three constraints, there is no scalar reward function that simultaneously satisfies them all:

$$r(\xi_1) > r(\xi_2) \quad \& \quad r(\xi_1) < r(\xi_3) \quad \& \quad r(\xi_2) > r(\xi_3). \tag{24.173}$$

This is known as *intransitivity* – a lack of a global ordering over completions. We will now discuss where intransitivity comes from in practice and how to handle it.

24.2.2 Violations of Bradley-Terry

Recall that the Bradley-Terry model states that the probability of preferring ξ_i over ξ_j is:

$$P(\xi_i \succ \xi_j) = \frac{\exp(r(\xi_i))}{\exp(r(\xi_i)) + \exp(r(\xi_j))} \tag{24.174}$$

BT assumes that population-level preferences can be explained by a single reward, shared function. However, this assumption breaks when preferences are **intransitive** (e.g. $\xi_1 \succ \xi_2 \succ \xi_3 \succ \xi_1$). This happens mainly due to the following reasons:

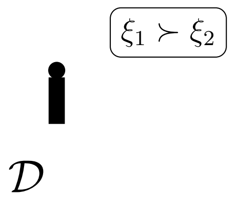
- **Intransitivity in Individual Preferences:** An individual may judge different pairs of samples on different features/criteria (e.g. we compare apples based on color but apples and oranges based on sweetness).
- **Intransitivity from Preference Aggregation:** Even if each individual has consistent preferences, aggregating preferences across a group of individuals can lead to intransitivity (e.g. rock-paper-scissors-like scenarios).

For the rest of the lecture, we will focus on the second case, where the intransitivity arises from the aggregation of preferences across multiple individuals.

24.2.3 Intransitivity from Preference Aggregation

In real-world scenarios, when collecting a preference dataset, for any single individual, we usually only have preferences over a small subset of pairs of prompt completions.

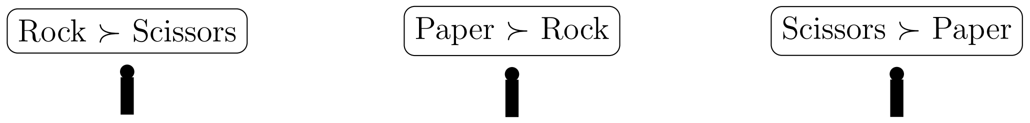
We can turn this data into a ***partial*** *preference matrix* $\mathcal{P}$, which has unknown entries:



$\mathcal{P}$	ξ_1	ξ_2	ξ_3
ξ_1	0.5	1	?
ξ_2	0	0.5	?
ξ_3	?	?	0.5

Figure 24.31: A limited number of comparisons from a single user can lead to a partially completed preference matrix. The question marks indicate pairs for which we have no data.

Even if the known entries are consistent with a reward function (e.g., $r(\xi_1) > r(\xi_2)$), the full matrix might still exhibit intransitivity once the missing entries are filled in with data from other users. We will walk through an example of this now, inspired by Rock Paper Scissors.



$\mathcal{P}_1$	R	P	S
R	0.5	?	1
P	?	0.5	?
S	0	?	0.5

$\mathcal{P}_2$	R	P	S
R	0.5	0	?
P	1	0.5	?
S	?	?	0.5

$\mathcal{P}_3$	R	P	S
R	0.5	?	?
P	?	0.5	0
S	?	1	0.5

Figure 24.32: Consider aggregating the preferences of three raters, each of which only compares two of the three options for us. This will lead to the appearance of intransitivity.

Aggregating these preferences can result in the following matrix:

$\mathcal{P}_{agg}$	Rock	Paper	Scissors
Rock	0.5	0	1
Paper	1	0.5	0
Scissors	0	1	0.5

$\mathcal{P}_{agg}$ exhibits intransitivity: Rock $\succ$ Scissors, Scissors $\succ$ Paper, and Paper $\succ$ Rock. Thus, there can be no reward function $r^\star$ that can explain these preferences simultaneously.

24.2.4 No $r^\star$ Leads to Mode Collapse in RLHF

When the underlying aggregated preferences $\mathcal{P}$ are intransitive, no single reward function $r^\star$ can perfectly capture them. Standard RLHF methods attempt to find the Maximum Likelihood Estimate (MLE) reward function $\hat{r}_{\text{mle}}$ under the Bradley-Terry model. However, this $\hat{r}_{\text{mle}}$ will be a poor fit for the true population preferences, leading to mode collapse.

Consider a slightly different intransitive preference matrix (rows preferred over columns):

$\mathcal{P}$	Rock	Paper	Scissors
Rock	0.5	0.7	0
Paper	0.3	0.5	1
Scissors	1	0	0.5

Figure 24.33: Another set of intransitive preferences. Row sums are $1.2, 1.8, 1.5$.

Assume we have a uniform reference policy – i.e. $\hat{\pi}_{\text{ref}} = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$. In such a situation, loosely speaking, we would expect fitting a BT reward model via MLE would lead to higher values on the choices corresponding to rows with higher sums (i.e. the choices that have higher probabilities of being preferred by the raters on average). For simplicity, let's assume $\hat{r}_{\text{mle}} = [1.2, 1.8, 1.5]$. Next, recall that the soft RL policy has the form

$$\hat{\pi}_{\text{RLHF}} \propto \pi_{\text{ref}} \cdot \exp\left(\frac{1}{\beta}\hat{r}_{\text{mle}}\right). \tag{24.175}$$

This means that even if the reward model puts a slightly higher value on something (e.g. paper), the resulting soft-optimal policy will choose paper significantly more often. This leads to **mode collapse**: the RLHF policy the flawed $\hat{r}_{mle}$ and primarily generates outputs corresponding to the highest estimated reward (Paper), neglecting other valid or even superior options in certain contexts (like Rock, which beats Scissors, or Scissors, which beats Paper). The policy fails to capture the cyclic nature of the true preferences. In practice, this can correspond to ignoring minority preferences and instead only generating completions preferred by the majority, which can be undesirable in a variety of situations.

We conclude by noting that beyond the toy instance above, this problem becomes increasingly severe as the size of the output space Ξ grows larger, as it becomes increasingly unlikely that there is a global ordering over all natural language prompt completions all raters agree with.

24.3 Beyond Bradley-Terry in RLHF

Given *any* preference matrix, e.g.:

$\mathcal{P}$	Rock	Paper	Scissors
Rock	0.5	0.7	0
Paper	0.3	0.5	1
Scissors	1	0	0.5

We can transform this matrix into an *anti-symmetric matrix* (i.e. one where elements across the diagonal are negations) by multiplying each element by 2 and subtracting by 1:

$\mathcal{P}$	Rock	Paper	Scissors
Rock	0	0.4	-1
Paper	-0.4	0	1
Scissors	1	-1	0

Observe that this is now the *payoff matrix* for a two-player zero sum game. Furthermore, it is a *symmetric* game, which means swapping strategies with your opponent leads to the opposite outcome for both players. We can now consider solving the corresponding game:

$$\pi_1^\star, \pi_2^\star = \arg\max_{\pi_1 \in \Pi} \arg\min_{\pi_2 \in \Pi} \mathbb{E}_{\xi_1 \sim \pi_1, \xi_2 \sim \pi_2}[2\mathcal{P}(\xi_1 \succ \xi_2) - 1]. \tag{24.176}$$

the solutions to which are known as *Von Neumann / Minimax winners*. Before we discuss how to solve this game, we first note that this solution concept has appealing properties:

- Does not assume we can collapse the preference matrix into a reward function and therefore doesn't assume transitivity / a shared reward function.
- Learns a strategy that is robust against the worst-case comparator. More formally, MWs are preferred to any other policy with probability at least $\frac{1}{2}$.

For the preceding game the MW is $[5/12, 5/12, 1/6]$, which does not suffer from mode collapse. This reflects the fact that equilibrium strategies are often randomized. Also observe that the above strategy is not merely a scaled version of the policy we'd get out of soft RL – we're instead learning something qualitatively different and more robust.

24.4 SPO: Self-Play Preference Optimization

Unfortunately, doing adversarial training at the scale of LLMs can be challenging in practice. In response, we will now describe how a simple, *self-play* strategy can be used instead to compute an approximate equilibria, leveraging the symmetry of the above game.

Proof: First, let us consider solving the above game by running two no-regret algorithms against eachother. As usual, we can define a sequence of losses for each player:

$$\ell_t^1(\pi) = \mathbb{E}_{\xi \sim \pi, \xi' \sim \pi_2^t}[2\mathcal{P}(\xi \succ \xi') - 1], \quad \ell_t^2(\pi) = \mathbb{E}_{\xi \sim \pi_t^1, \xi' \sim \pi}[-(2\mathcal{P}(\xi \succ \xi') - 1)]. \tag{24.177}$$

Without loss of generality, assume that $\pi_1^0 = \pi_2^0$. Then, we have

$$\begin{aligned}
\ell_0^1(\pi) &= \mathbb{E}_{\xi\sim\pi,\xi'\sim\pi_2^0}[2\mathcal{P}(\xi \succ \xi') - 1] && (24.178)\\
&= \mathbb{E}_{\xi\sim\pi,\xi'\sim\pi_1^0}[2\mathcal{P}(\xi \succ \xi') - 1] && (24.179)\\
&= \mathbb{E}_{\xi\sim\pi_1^0,\xi'\sim\pi}[-(2\mathcal{P}(\xi \succ \xi') - 1)] && (24.180)\\
&= \ell_0^2(\pi), && (24.181)
\end{aligned}$$

where we use the anti-symmetry of the payoff matrix in the second to last equality. If we feed two (deterministic) no-regret algorithms the same sequence of loss functions and start them from the same initial strategy, they will produce identical iterates. Thus, an implication of the above is that $\forall t \in [T], \pi_1^t = \pi_2^t$. ■

Practically, this means that rather than needing to have two separate policy networks and memory and dealing with the instabilities of adversarial training, we can instead merely treat a second sample from the same policy as a sample from a fictitious second player.

To implement SPO, one first trains a pairwise *preference* model $\widehat{\mathcal{P}}$ that maps from pairs of completions to the probability the former is preferred to the latter. This can be done via standard maximum likelihood estimation (MLE) but doesn't assume transitivity like a reward model. Observe that this is just using a function approximator to model the ground-truth preference matrix, $\mathcal{P}$. One then samples multiple times from the policy, queries the preference model on each pair, and computes a *win rate* for each sample – i.e., an empirical estimate of the above loss function. One can then use these win-rates as rewards for a downstream RM-based policy optimization procedure. In short, the changes from standard RLHF are (1) training a preference rather than a reward model and (2) sampling multiple completions per prompt (which one often does anyway in practice). We visualize the full process below. One can also use another LLM as the preference model, often referred to as LLM-as-a-judge. Because such models are trained on internet data, they often exhibit intransitivities in their judgments due to the implicit preference aggregation caused by scale.

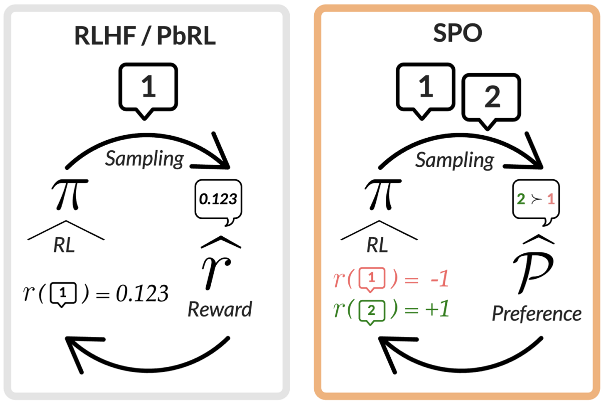


Figure 24.34: In SPO (Self-Play Preference Optimization), one compares a policy to itself via the use of a learned preference model or "judge" LLM. In particular, one samples multiple completions from the policy, asks the preference model to pick a winner for each pairwise comparison, and then uses the win rate of each completion as the reward for an RL procedure.

Bibliography

[1] Exponential family — wikipedia, the free encyclopedia, 2025.

[2] 15-888 lecture 2: Representation of strategies in tree-form decision spaces, 2021.

[3] 15-888 lecture 5: Regret circuits and the counterfactual regret minimization (cfr) paradigm, 2021.

[4] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

[5] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning, 2011.

[6] Justin Boyan and Andrew Moore. Generalization in reinforcement learning: Safely approximating the value function. *Advances in neural information processing systems*, 7, 1994.

[7] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[8] Alex Lamb, Riashat Islam, Yonathan Efroni, Aniket Didolkar, Dipendra Misra, Dylan Foster, Lekan Molu, Rajan Chari, Akshay Krishnamurthy, and John Langford. Guaranteed discovery of control-endogenous latent states with multi-step inverse models, 2022.

[9] Stephane Ross and J Andrew Bagnell. Agnostic system identification for model-based reinforcement learning. *arXiv preprint arXiv:1203.1007*, 2012.

[10] Yuda Song, Yifei Zhou, Ayush Sekhari, Drew Bagnell, Akshay Krishnamurthy, and Wen Sun. Hybrid rl: Using both offline and online data can make rl efficient. In *The Eleventh International Conference on Learning Representations*.

[11] Siddhartha S. Srinivasa, Patrick Lancaster, Johan Michalove, Matt Schmittle, Colin Summers, Matthew Rockett, Rosario Scalise, Joshua R. Smith, Sanjiban Choudhury,

Christoforos Mavrogiannis, and Fereshteh Sadeghi. Mushr: A low-cost, open-source robotic racecar for education and research, 2023.

[12] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3(1):88–97, 1991.

[13] Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. Hg-dagger: Interactive imitation learning with human experts, 2019.

[14] Ari Seff, Brian Cera, Dian Chen, Mason Ng, Aurick Zhou, Nigamaa Nayakanti, Khaled S Refaat, Rami Al-Rfou, and Benjamin Sapp. Motionlm: Multi-agent motion forecasting as language modeling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8579–8590, 2023.

[15] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[16] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser,

Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. Gpt-4 technical report, 2024.

[17] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022.

[18] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[19] Junzi Zhang, Jongho Kim, Brendan O'Donoghue, and Stephen Boyd. Sample efficient reinforcement learning with reinforce, 2020.

[20] Zhaolin Gao, Jonathan D. Chang, Wenhao Zhan, Owen Oertell, Gokul Swamy, Kianté Brantley, Thorsten Joachims, J. Andrew Bagnell, Jason D. Lee, and Wen Sun. Rebel: Reinforcement learning via regressing relative rewards, 2024.

[21] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024.

[22] Gokul Swamy, Sanjiban Choudhury, Wen Sun, Zhiwei Steven Wu, and J Andrew Bagnell. All roads lead to likelihood: The value of reinforcement learning in fine-tuning. *arXiv preprint arXiv:2503.01067*, 2025.