Learning for Downstream Decision Making

Outline

- Motivation: Learning for downstream decision makers
- Supervised learning and the single-loss paradigm
- Omniprediction: learning predictors for many losses
- Reward-free reinforcement learning
- Conclusion and open questions

Motivating Examples

Weather forecasting

A weather model predicts the probability of rain.

Different decision makers use the forecast differently:

- individuals deciding whether to carry umbrellas
- airlines scheduling flights
- city governments planning flood responses

Motivating Examples

Weather forecasting

A weather model predicts the probability of rain.

Different decision makers use the forecast differently:

- individuals deciding whether to carry umbrellas
- airlines scheduling flights
- city governments planning flood responses

Medical risk prediction

A model predicts risk of heart disease.

- doctors decide treatments
- hospitals allocate resources
- insurance companies price policies

Motivating Examples (continued)

Navigation systems

Google Maps predicts travel times.

Different users have different preferences:

- shortest time
- avoiding highways
- minimizing tolls
- scenic routes

Motivating Examples (continued)

Navigation systems
Google Maps predicts travel times.
Different users have different preferences:

- shortest time
- avoiding highways
- minimizing tolls
- scenic routes

Question

How should we learn models that serve many decision makers?

Prediction and Decisions

Prediction model

$$p(x) \approx \mathbb{E}[Y|X = x]$$

Downstream user applies a decision rule

$$a = k(p(x))$$

Prediction and Decisions

Prediction model

$$p(x) \approx \mathbb{E}[Y|X = x]$$

Downstream user applies a decision rule

$$a = k(p(x))$$

Example

$$p(x) = P(\text{rain}|x)$$

Decision maker optimizes

$$\min_a \mathbb{E}[\ell(a, Y)]$$

Prediction and Decisions

Prediction model

$$p(x) \approx \mathbb{E}[Y|X = x]$$

Downstream user applies a decision rule

$$a = k(p(x))$$

Example

$$p(x) = P(\text{rain}|x)$$

Decision maker optimizes

$$\min_a \mathbb{E}[\ell(a, Y)]$$

Can one predictor support many downstream losses?

Supervised Learning Setup

We observe data

$$(x, y) \sim D$$

$$x \in \mathcal{X} \quad \text{(features)}, \qquad y \in \mathcal{Y} \quad \text{(label)}$$

Examples: weather $\rightarrow$ rain, patient data $\rightarrow$ disease, traffic $\rightarrow$ travel time

Supervised Learning Setup

We observe data

$$(x, y) \sim D$$

$$x \in \mathcal{X} \quad \text{(features)}, \qquad y \in \mathcal{Y} \quad \text{(label)}$$

Examples: weather $\rightarrow$ rain, patient data $\rightarrow$ disease, traffic $\rightarrow$ travel time

Hypothesis class

$$\mathcal{H} \subseteq \{h : \mathcal{X} \rightarrow \mathcal{Y}\}$$

Examples: linear models $h(x) = w^\top x$, decision trees, neural networks

Supervised Learning Setup

We observe data

$$(x, y) \sim D$$

$$x \in \mathcal{X} \quad \text{(features)}, \qquad y \in \mathcal{Y} \quad \text{(label)}$$

Examples: weather $\rightarrow$ rain, patient data $\rightarrow$ disease, traffic $\rightarrow$ travel time

Hypothesis class

$$\mathcal{H} \subseteq \{h : \mathcal{X} \rightarrow \mathcal{Y}\}$$

Examples: linear models $h(x) = w^\top x$, decision trees, neural networks

Loss function

$$\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_+$$

Examples: 0-1 loss $\mathbf{1}[\hat{y} \neq y]$, squared loss $(\hat{y} - y)^2$, absolute loss $|\hat{y} - y|$

Supervised Learning Setup

We observe data

$$(x, y) \sim D$$

$$x \in \mathcal{X} \quad \text{(features)}, \qquad y \in \mathcal{Y} \quad \text{(label)}$$

Examples: weather $\rightarrow$ rain, patient data $\rightarrow$ disease, traffic $\rightarrow$ travel time

Hypothesis class

$$\mathcal{H} \subseteq \{h : \mathcal{X} \rightarrow \mathcal{Y}\}$$

Examples: linear models $h(x) = w^\top x$, decision trees, neural networks

Loss function

$$\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_+$$

Examples: 0-1 loss $\mathbf{1}[\hat{y} \neq y]$, squared loss $(\hat{y} - y)^2$, absolute loss $|\hat{y} - y|$

Learning objective

$$\min_{h \in \mathcal{H}} \mathbb{E}_{(x,y) \sim D}[\ell(h(x), y)]$$

PAC Learning Framework

Given a hypothesis class $\mathcal{H}$ and i.i.d. samples

$$S = \{(x_1, y_1), \ldots, (x_n, y_n)\},$$

the goal is to output $h \in \mathcal{H}$ such that

$$L(h) \leq \inf_{h' \in \mathcal{H}} L(h') + \epsilon$$

with probability at least $1 - \delta$.

PAC Learning Framework

Given a hypothesis class $\mathcal{H}$ and i.i.d. samples

$$S = \{(x_1, y_1), \ldots, (x_n, y_n)\},$$

the goal is to output $h \in \mathcal{H}$ such that

$$L(h) \leq \inf_{h' \in \mathcal{H}} L(h') + \epsilon$$

with probability at least $1 - \delta$.
This framework is very well studied.

PAC Learning Framework

Given a hypothesis class $\mathcal{H}$ and i.i.d. samples

$$S = \{(x_1, y_1), \ldots, (x_n, y_n)\},$$

the goal is to output $h \in \mathcal{H}$ such that

$$L(h) \leq \inf_{h' \in \mathcal{H}} L(h') + \epsilon$$

with probability at least $1 - \delta$.
This framework is very well studied.
Two key questions:

- **Computation:** how do we find a good hypothesis efficiently?
- **Sample complexity:** how many examples are needed?

Data + Empirical Risk Minimization(ERM)

The basic recipe is simple: collect data run $\Longrightarrow$ ERM on the data

Data + Empirical Risk Minimization(ERM)

The basic recipe is simple: collect data run $\Longrightarrow$ ERM on the data

Given samples $S = \{(x_i, y_i)\}_{i=1}^n$, define the empirical loss

$$\widehat{L}_S(h) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i), y_i),$$

and compute

$$\widehat{h} \in \arg\min_{h \in \mathcal{H}} \widehat{L}_S(h).$$

Data + Empirical Risk Minimization(ERM)

The basic recipe is simple: collect data run $\Longrightarrow$ ERM on the data

Given samples $S = \{(x_i, y_i)\}_{i=1}^n$, define the empirical loss

$$\widehat{L}_S(h) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i), y_i),$$

and compute

$$\widehat{h} \in \arg\min_{h \in \mathcal{H}} \widehat{L}_S(h).$$

How we solve ERM depends on the class: linear models $\rightarrow$ convex optimization / gradient descent, decision trees $\rightarrow$ combinatorial search / heuristics, neural networks $\rightarrow$ gradient-based methods

Data + Empirical Risk Minimization(ERM)

The basic recipe is simple: collect data run $\Longrightarrow$ ERM on the data

Given samples $S = \{(x_i, y_i)\}_{i=1}^n$, define the empirical loss

$$\widehat{L}_S(h) = \frac{1}{n} \sum_{i=1}^{n} \ell(h(x_i), y_i),$$

and compute

$$\widehat{h} \in \arg\min_{h \in \mathcal{H}} \widehat{L}_S(h).$$

How we solve ERM depends on the class: linear models $\rightarrow$ convex optimization / gradient descent, decision trees $\rightarrow$ combinatorial search / heuristics, neural networks $\rightarrow$ gradient-based methods

The number of samples needed scales with the complexity of the class:

$$n \asymp \frac{\text{complexity}(\ell \circ \mathcal{H})}{\epsilon^2}.$$

For example: VC dimension for 0-1 classification, fat-shattering dimension for real-valued prediction.

Limitation of Training with a Single Loss

Standard learning solves

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)]$$

Limitation of Training with a Single Loss

Standard learning solves

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)]$$

But different losses correspond to different optimal predictions.

Limitation of Training with a Single Loss

Standard learning solves

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)]$$

But different losses correspond to different optimal predictions.

Binary example

Let

$$p^*(x) = P(Y = 1 | X = x)$$

If $p^*(x) = 0.7$,

$$\arg \min_{a \in [0,1]} \mathbb{E}[(a - Y)^2 \mid X = x] = 0.7$$

$$\arg \min_{a \in [0,1]} \mathbb{E}[|a - Y| \mid X = x] = 1$$

Limitation of Training with a Single Loss

Standard learning solves

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)]$$

Limitation of Training with a Single Loss

Standard learning solves

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)]$$

But different losses correspond to different optimal predictions.

Regression example

Optimal predictions satisfy

$$\arg\min_{a} \mathbb{E}[(a - Y)^2 \mid X] = \mathbb{E}[Y|X]$$

$$\arg\min_{a} \mathbb{E}[|a - Y| \mid X] = \text{median}(Y|X)$$

$$\arg\min_{a} \mathbb{E}[\ell_\tau(a, Y) \mid X] = \tau\text{-quantile}(Y|X)$$

Different losses elicit different statistics of $P(Y|X)$

Omniprediction

Single predictor, simultaneous loss minimizer for many losses

(Gopalan, Kalai, Reingold, Sharan, Wieder; ITCS'22)

Omniprediction

Single predictor, simultaneous loss minimizer for many losses

(Gopalan, Kalai, Reingold, Sharan, Wieder; ITCS'22)

Assume

$$Y \in \{0, 1\}.$$

Given a loss class $\mathcal{L}$, a hypothesis class $\mathcal{H}$, and $\epsilon > 0$, we want to find a predictor $p : \mathcal{X} \to [0, 1]$ such that for every loss $\ell \in \mathcal{L}$,

$$\mathbb{E}\big[\ell(k_\ell(p(x)), y)\big] \;\leq\; \inf_{h \in \mathcal{H}} \mathbb{E}\big[\ell(h(x), y)\big] + \epsilon.$$

Here

$$k_\ell(v) \in \arg\min_{a \in [0,1]} \; \mathbb{E}_{Y \sim \mathrm{Ber}(v)}[\ell(a, Y)]$$

is the loss-specific post-processing. E.g for ℓ_1, $k_\ell(p(x)) = \mathbf{1}(p(x) \geq 0.5)$

Omniprediction

Single predictor, simultaneous loss minimizer for many losses

(Gopalan, Kalai, Reingold, Sharan, Wieder; ITCS'22)

Assume

$$Y \in \{0, 1\}.$$

Given a loss class $\mathcal{L}$, a hypothesis class $\mathcal{H}$, and $\epsilon > 0$, we want to find a predictor $p : \mathcal{X} \to [0, 1]$ such that for every loss $\ell \in \mathcal{L}$,

$$\mathbb{E}\big[\ell(k_\ell(p(x)), y)\big] \;\leq\; \inf_{h \in \mathcal{H}} \mathbb{E}\big[\ell(h(x), y)\big] + \epsilon.$$

Here

$$k_\ell(v) \in \arg\min_{a \in [0,1]} \; \mathbb{E}_{Y \sim \mathrm{Ber}(v)}[\ell(a, Y)]$$

is the loss-specific post-processing. E.g for ℓ_1, $k_\ell(p(x)) = \mathbf{1}(p(x) \geq 0.5)$
Key points: one predictor p, no retraining for each new loss, recovers approximate loss minimization simultaneously for all $\ell \in \mathcal{L}$

Ground Truth

Define the Bayes-optimal predictor

$$p^*(x) = P(Y = 1|X = x)$$

Ground Truth

Define the Bayes-optimal predictor

$$p^*(x) = P(Y = 1|X = x)$$

is an omnipredictor for the class of all losses. That is, for any decision maker

$$k_\ell(p^*(x)) = \arg \min_{a \in [0,1]} \mathbb{E}_{Y \sim \text{Ber}(p^*(x))}[\ell(a, Y)]$$

gives optimal decisions

Ground Truth

Define the Bayes-optimal predictor

$$p^*(x) = P(Y = 1|X = x)$$

is an omnipredictor for the class of all losses. That is, for any decision maker

$$k_\ell(p^*(x)) = \arg \min_{a \in [0,1]} \mathbb{E}_{Y \sim \text{Ber}(p^*(x))}[\ell(a, Y)]$$

gives optimal decisions

Why Finding Omnipredictors is Hard

In PAC learning we solve

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)]$$

Why Finding Omnipredictors is Hard

In PAC learning we solve

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)]$$

But omniprediction requires

$$p \quad \text{works for many losses } \ell$$

Why Finding Omnipredictors is Hard

In PAC learning we solve

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)]$$

But omniprediction requires

$$p \quad \text{works for many losses } \ell$$

Challenge:

- Bayes predictor p^* (or even any omnipredictor) may not lie in hypothesis class
- What works for one loss might not work for another
- How much more samples do we need?

Why Finding Omnipredictors is Hard

In PAC learning we solve

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)]$$

But omniprediction requires

$$p \quad \text{works for many losses } \ell$$

Challenge:

- Bayes predictor p^* (or even any omnipredictor) may not lie in hypothesis class
- What works for one loss might not work for another
- How much more samples do we need?

Thus finding omnipredictors is **not a simple search problem** unlike PAC learning.

Main Result

(Gopalan, Kalai, Reingold, Sharan, Wieder; ITCS'22), (Gopalan, Hu, Kim, Reingold, Wieder, ITCS'22)

Given a loss class $\mathcal{L}$, hypothesis class $\mathcal{H}$, and $\epsilon > 0$, there exists an algorithm that learns an ϵ-omnipredictor using

$$n = \text{poly}(1/\epsilon) \cdot \text{complexity}(\mathcal{L} \circ \mathcal{H})$$

samples.

Main Result

(Gopalan, Kalai, Reingold, Sharan, Wieder; ITCS'22), (Gopalan, Hu, Kim, Reingold, Wieder, ITCS'22)

Given a loss class $\mathcal{L}$, hypothesis class $\mathcal{H}$, and $\epsilon > 0$, there exists an algorithm that learns an ϵ-omnipredictor using

$$n = \text{poly}(1/\epsilon) \cdot \text{complexity}(\mathcal{L} \circ \mathcal{H})$$

samples.
Moreover, the algorithm runs in time

- polynomial in $1/\epsilon$
- assuming access to a weak learning (or ERM) oracle for $\mathcal{L} \circ \mathcal{H}$

Main Result

(Gopalan, Kalai, Reingold, Sharan, Wieder; ITCS'22), (Gopalan, Hu, Kim, Reingold, Wieder, ITCS'22)

Given a loss class $\mathcal{L}$, hypothesis class $\mathcal{H}$, and $\epsilon > 0$, there exists an algorithm that learns an ϵ-omnipredictor using

$$n = \text{poly}(1/\epsilon) \cdot \text{complexity}(\mathcal{L} \circ \mathcal{H})$$

samples.
Moreover, the algorithm runs in time

- polynomial in $1/\epsilon$
- assuming access to a weak learning (or ERM) oracle for $\mathcal{L} \circ \mathcal{H}$

Thus omnipredictors can be learned efficiently for any loss class $\mathcal{L}$ and hypothesis class $\mathcal{H}$.
Problem: the polynomial factors are large $O(1/\epsilon^{10})$ vs $O(1/\epsilon^2)$ from PAC learning.

Main Result

(Okoroafor, Kleinberg, Kim FOCS'25)

Given a loss class $\mathcal{L}$, hypothesis class $\mathcal{H}$, and $\epsilon > 0$, there exists an algorithm that learns an ϵ-omnipredictor using

$$n = O(1/\epsilon^2) \cdot \text{complexity}(\mathcal{L} \circ \mathcal{H})$$

samples.

Main Result

(Okoroafor, Kleinberg, Kim FOCS'25)

Given a loss class $\mathcal{L}$, hypothesis class $\mathcal{H}$, and $\epsilon > 0$, there exists an algorithm that learns an ϵ-omnipredictor using

$$n = O(1/\epsilon^2) \cdot \text{complexity}(\mathcal{L} \circ \mathcal{H})$$

samples.

Moreover, the algorithm runs in time

- polynomial in $1/\epsilon$
- assuming access to a weak learning (or ERM) oracle for $\mathcal{L} \circ \mathcal{H}$

Main Result

(Okoroafor, Kleinberg, Kim FOCS'25)

Given a loss class $\mathcal{L}$, hypothesis class $\mathcal{H}$, and $\epsilon > 0$, there exists an algorithm that learns an ϵ-omnipredictor using

$$n = O(1/\epsilon^2) \cdot \text{complexity}(\mathcal{L} \circ \mathcal{H})$$

samples.

Moreover, the algorithm runs in time

- polynomial in $1/\epsilon$
- assuming access to a weak learning (or ERM) oracle for $\mathcal{L} \circ \mathcal{H}$

Thus omnipredictors can be learned efficiently for any loss class $\mathcal{L}$ and hypothesis class $\mathcal{H}$.

Loss–OI Decomposition

Let $(x, y) \sim D$ and predictor p.
For any loss ℓ and hypothesis $h \in \mathcal{H}$,

$$\ell(k_\ell(p(x)), y) - \ell(h(x), y)$$

can be written as

$$\ell(k_\ell(p(x)), y) - \mathbb{E}_{\tilde{y} \sim \mathrm{Ber}(p(x))}[\ell(k_\ell(p(x)), \tilde{y})]$$

$$+\mathbb{E}_{\tilde{y} \sim \mathrm{Ber}(p(x))}[\ell(h(x), \tilde{y}) - \ell(h(x), y)].$$

Loss–OI Decomposition

Let $(x, y) \sim D$ and predictor p.
For any loss ℓ and hypothesis $h \in \mathcal{H}$,

$$\ell(k_\ell(p(x)), y) - \ell(h(x), y)$$

can be written as

$$\ell(k_\ell(p(x)), y) - \mathbb{E}_{\tilde{y} \sim \mathrm{Ber}(p(x))}[\ell(k_\ell(p(x)), \tilde{y})]$$

$$+\mathbb{E}_{\tilde{y} \sim \mathrm{Ber}(p(x))}[\ell(h(x), \tilde{y}) - \ell(h(x), y)].$$

Taking expectations,

$$\mathbb{E}[\ell(k_\ell(p(x)), y)] - \mathbb{E}[\ell(h(x), y)]$$

is bounded by the sum of these two terms.

From Loss Differences to Residual Correlations

Key identity for binary outcomes

$$\ell(p, y) = y(\ell(p, 1) - \ell(p, 0)) + \ell(p, 0).$$

Define $\Delta\ell(p) = \ell(p, 1) - \ell(p, 0)$.

From Loss Differences to Residual Correlations

Key identity for binary outcomes

$$\ell(p, y) = y(\ell(p, 1) - \ell(p, 0)) + \ell(p, 0).$$

Define $\Delta\ell(p) = \ell(p, 1) - \ell(p, 0)$.
Then

$$\ell(k_\ell(p(x)), y) - \mathbb{E}_{\tilde{y}\sim \mathrm{Ber}(p(x))}[\ell(k_\ell(p(x)), \tilde{y})] = (y - p(x))\,\Delta\ell(k_\ell(p(x))).$$

From Loss Differences to Residual Correlations

Key identity for binary outcomes

$$\ell(p, y) = y(\ell(p, 1) - \ell(p, 0)) + \ell(p, 0).$$

Define $\Delta\ell(p) = \ell(p, 1) - \ell(p, 0)$.
Then

$$\ell(k_\ell(p(x)), y) - \mathbb{E}_{\tilde{y}\sim\mathrm{Ber}(p(x))}[\ell(k_\ell(p(x)), \tilde{y})] = (y - p(x))\,\Delta\ell(k_\ell(p(x))).$$

Similarly

$$\mathbb{E}_{\tilde{y}\sim\mathrm{Ber}(p(x))}[\ell(h(x), \tilde{y}) - \ell(h(x), y)] = (p(x) - y)\,\Delta\ell(h(x)).$$

From Loss Differences to Residual Correlations

Key identity for binary outcomes

$$\ell(p, y) = y(\ell(p, 1) - \ell(p, 0)) + \ell(p, 0).$$

Define $\Delta\ell(p) = \ell(p, 1) - \ell(p, 0)$.
Then

$$\ell(k_\ell(p(x)), y) - \mathbb{E}_{\tilde{y}\sim\mathrm{Ber}(p(x))}[\ell(k_\ell(p(x)), \tilde{y})] = (y - p(x))\,\Delta\ell(k_\ell(p(x))).$$

Similarly

$$\mathbb{E}_{\tilde{y}\sim\mathrm{Ber}(p(x))}[\ell(h(x), \tilde{y}) - \ell(h(x), y)] = (p(x) - y)\,\Delta\ell(h(x)).$$

Thus omniprediction error reduces to controlling

$$\mathbb{E}[(y - p(x))\,c(x)]$$

for functions $c(x) \in \{\Delta\ell(k_\ell(p(x))), \Delta\ell(h(x))\}$.

Omnipredictors via Residual Correlations

Calibration: For all functions $w : [0, 1] \to [-1, 1]$

$$\mathbb{E}\Big[w(p(x)) \cdot (y - p(x))\Big] \leq \epsilon$$

Multiaccuracy: For all $\ell \in \mathcal{L}$ and $h \in \mathcal{H}$

$$\mathbb{E}\Big[\Delta\ell(h(x)) \cdot (y - p(x))\Big] \leq \epsilon$$

Omnipredictors via Residual Correlations

Calibration: For all functions $w : [0, 1] \to [-1, 1]$

$$\mathbb{E}\big[w(p(x)) \cdot (y - p(x))\big] \leq \epsilon$$

Multiaccuracy: For all $\ell \in \mathcal{L}$ and $h \in \mathcal{H}$

$$\mathbb{E}\big[\Delta\ell(h(x)) \cdot (y - p(x))\big] \leq \epsilon$$

Theorem (Gopalan, Hu, Kalai, Reingold, Wieder; ITCS'23)

$$\text{Multiaccuracy} + \text{Calibration} \implies \text{Omniprediction}$$

Boosting Style Algorithm

Initialize

$$p_0(x) = 1/2$$

Boosting Style Algorithm

Initialize

$$p_0(x) = 1/2$$

Iteratively update

$$p_{t+1}(x) = p_t(x) + \eta c_t(x)$$

where

$$c_t = \arg\max_c \mathbb{E}[c(x)(y - p_t(x))]$$

Boosting Style Algorithm

Initialize

$$p_0(x) = 1/2$$

Iteratively update

$$p_{t+1}(x) = p_t(x) + \eta c_t(x)$$

where

$$c_t = \arg\max_c \mathbb{E}[c(x)(y - p_t(x))]$$

Terminate when residual correlations are small.

Sample Complexity of Boosting Approach

A predictor satisfy residual constraints specified by C can be learned after $O(1/\epsilon^2)$ iterations using

$$O\left(\frac{\text{complexity}(\mathcal{C})}{\epsilon^2}\right)$$

samples per iteration leading to total runtime $\tilde{O}(1/\epsilon^4)$

Sample Complexity of Boosting Approach

A predictor satisfy residual constraints specified by C can be learned after $O(1/\epsilon^2)$ iterations using

$$O\left(\frac{\text{complexity}(\mathcal{C})}{\epsilon^2}\right)$$

samples per iteration leading to total runtime $\tilde{O}(1/\epsilon^4)$

Algorithm in (Okoroafor, Kleinberg, Kim FOCS'25)

- Gives a tighter characterization of the residual constraints
- Goes through a multi-objective online learning tool called Blackwell's Approachability Theorem
- Does an Online to Batch Conversion

Regression Setting

Now consider $Y \in \mathbb{R}$.
Different losses require different statistics.
Examples

$$\ell_2 \rightarrow \mathbb{E}[Y|X]$$

$$\ell_1 \rightarrow \text{median}(Y|X)$$

$$\text{quantile loss} \rightarrow \text{quantiles}$$

Question: What should you predict about the distribution $Y|X$

Sufficient Statistics for Classes of Loss Functions

(Gopalan, Okoroafor, Raghavendra, Shetty, Singhal; COLT'22)

Given a class of loss functions $\mathcal{L}$.

Sufficient Statistics for Classes of Loss Functions

(Gopalan, Okoroafor, Raghavendra, Shetty, Singhal; COLT'22)

Given a class of loss functions $\mathcal{L}$.

Definition

A d-dimensional function

$$s(Y) : \mathcal{Y} \to [-1, 1]^d$$

is a *sufficient statistic* for $\mathcal{L}$ if it captures all information needed to evaluate every loss in $\mathcal{L}$.

Sufficient Statistics for Classes of Loss Functions

(Gopalan, Okoroafor, Raghavendra, Shetty, Singhal; COLT'22)

Given a class of loss functions $\mathcal{L}$.

Definition

A d-dimensional function

$$s(Y) : \mathcal{Y} \to [-1, 1]^d$$

is a *sufficient statistic* for $\mathcal{L}$ if it captures all information needed to evaluate every loss in $\mathcal{L}$.

Formally, for any distribution $\mathcal{D}$ on $\mathcal{Y}$, any $\ell \in \mathcal{L}$, and any decision t,

$$\mathbb{E}_{\mathcal{D}}[\ell(Y, t)] = \langle r^{\ell}(t), \mathbb{E}_{\mathcal{D}}[s(Y)] \rangle \pm \epsilon.$$

Sufficient Statistics for Classes of Loss Functions

(Gopalan, Okoroafor, Raghavendra, Shetty, Singhal; COLT'22)

Given a class of loss functions $\mathcal{L}$.

Definition

A d-dimensional function

$$s(Y) : \mathcal{Y} \to [-1, 1]^d$$

is a *sufficient statistic* for $\mathcal{L}$ if it captures all information needed to evaluate every loss in $\mathcal{L}$.

Formally, for any distribution $\mathcal{D}$ on $\mathcal{Y}$, any $\ell \in \mathcal{L}$, and any decision t,

$$\mathbb{E}_{\mathcal{D}}[\ell(Y, t)] = \langle r^{\ell}(t), \mathbb{E}_{\mathcal{D}}[s(Y)] \rangle \pm \epsilon.$$

Interpretation:

- predictions only need to estimate $\mathbb{E}[s(Y) \mid X]$
- downstream users compute optimal decisions from this statistic

Sufficient Statistics for Classes of Loss Functions

(Gopalan, Okoroafor, Raghavendra, Shetty, Singhal; COLT'22)

Given a class of loss functions $\mathcal{L}$.

Definition

A d-dimensional function

$$s(Y) : \mathcal{Y} \to [-1, 1]^d$$

is a *sufficient statistic* for $\mathcal{L}$ if it captures all information needed to evaluate every loss in $\mathcal{L}$.

Formally, for any distribution $\mathcal{D}$ on $\mathcal{Y}$, any $\ell \in \mathcal{L}$, and any decision t,

$$\mathbb{E}_{\mathcal{D}}[\ell(Y, t)] = \langle r^{\ell}(t), \mathbb{E}_{\mathcal{D}}[s(Y)] \rangle \pm \epsilon.$$

Interpretation:

- predictions only need to estimate $\mathbb{E}[s(Y) \mid X]$
- downstream users compute optimal decisions from this statistic

Key question: for a given loss class $\mathcal{L}$, how large must d be?

Loss Families and Omniprediction

(Gopalan, Okoroafor, Raghavendra, Shetty, Singhal; COLT'22)

Result: Characterize sufficient statistics for common loss families such as Generalized Linear Losses, 1-Lipschitz losses, convex 1-Lipschitz losses, low degree losses

Loss Families and Omniprediction

(Gopalan, Okoroafor, Raghavendra, Shetty, Singhal; COLT'22)

Result: Characterize sufficient statistics for common loss families such as Generalized Linear Losses, 1-Lipschitz losses, convex 1-Lipschitz losses, low degree losses

Algorithmic implication

Using the boosting-style omniprediction algorithm:

- we obtain omnipredictors for the regression setting
- runtime and sample complexity scale as

$$\exp(1/\epsilon)$$

Loss Families and Omniprediction

(Gopalan, Okoroafor, Raghavendra, Shetty, Singhal; COLT'22)

Result: Characterize sufficient statistics for common loss families such as Generalized Linear Losses, 1-Lipschitz losses, convex 1-Lipschitz losses, low degree losses

Algorithmic implication

Using the boosting-style omniprediction algorithm:

- we obtain omnipredictors for the regression setting
- runtime and sample complexity scale as

$$\exp(1/\epsilon)$$

Key open question: can we achieve **poly**$(1/\epsilon)$ algorithms?

Motivation (RL for downstream decision making)

Agent interacts with an environment

$$s_1 \to s_2 \to s_3 \to \dots$$

Performance is measured by reward

$$R(\tau) = \sum_{h=1}^{H} r(s_h, a_h)$$

Motivation (RL for downstream decision making)

Agent interacts with an environment

$$s_1 \to s_2 \to s_3 \to \dots$$

Performance is measured by reward

$$R(\tau) = \sum_{h=1}^{H} r(s_h, a_h)$$

Example: navigation - different users may have different priorities

- minimize travel time
- avoid highways
- scenic route

Each corresponds to a different reward function r.

Motivation (RL for downstream decision making)

Agent interacts with an environment

$$s_1 \to s_2 \to s_3 \to \dots$$

Performance is measured by reward

$$R(\tau) = \sum_{h=1}^{H} r(s_h, a_h)$$

Example: navigation - different users may have different priorities

- minimize travel time
- avoid highways
- scenic route

Each corresponds to a different reward function r.

How do we avoid running a separate RL algorithm for each user?

Motivation

Agent interacts with an environment

$$s_1 \to s_2 \to s_3 \to \dots$$

Performance is measured by reward

$$R(\tau) = \sum_{h=1}^{H} r(s_h, a_h)$$

Example: navigation - different users may have different priorities

- minimize travel time
- avoid highways
- scenic route

Each corresponds to a different reward function r.

Explore once $\Rightarrow$ optimize many rewards

Reward-Free Reinforcement Learning

Unknown MDP $(\mathcal{S}, \mathcal{A}, H, P)$

Reward-Free Reinforcement Learning

Unknown MDP $(\mathcal{S}, \mathcal{A}, H, P)$

Exploration Phase (Task-Agnostic)

Interact with the environment for K episodes. Collect dataset

$$D = \{(s_h^{(k)}, a_h^{(k)}, s_{h+1}^{(k)})\}_{k \in [K], h \in [H]}$$

Goal: learn about the transition dynamics P.

Reward-Free Reinforcement Learning

Unknown MDP $(\mathcal{S}, \mathcal{A}, H, P)$

Exploration Phase (Task-Agnostic)

Interact with the environment for K episodes. Collect dataset

$$D = \{(s_h^{(k)}, a_h^{(k)}, s_{h+1}^{(k)})\}_{k \in [K], h \in [H]}$$

Goal: learn about the transition dynamics P.

Planning Phase (Task-Specific)

A user provides a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Compute policy using only the dataset D.

$$\pi_r = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{h=1}^{H} r(s_h, a_h) \right]$$

Reward-Free Reinforcement Learning

Unknown MDP $(\mathcal{S}, \mathcal{A}, H, P)$

Exploration Phase (Task-Agnostic)

Interact with the environment for K episodes. Collect dataset

$$D = \{(s_h^{(k)}, a_h^{(k)}, s_{h+1}^{(k)})\}_{k \in [K], h \in [H]}$$

Goal: learn about the transition dynamics P.

Planning Phase (Task-Specific)

A user provides a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. Compute policy using only the dataset D.

$$\pi_r = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{h=1}^{H} r(s_h, a_h) \right]$$

Explore once $\Rightarrow$ solve many downstream tasks

Key Questions

Reward-free RL separates exploration and planning.

Two main questions arise.

Exploration

- How should we explore the MDP so that the collected dataset is useful?
- how large must K (the number of episodes) be so that the dataset $D = \{(s_h^{(k)}, a_h^{(k)}, s_{h+1}^{(k)})\}$ is sufficient for planning?

Key Questions

Reward-free RL separates exploration and planning.

Two main questions arise.

Exploration

- How should we explore the MDP so that the collected dataset is useful?
- how large must K (the number of episodes) be so that the dataset $D = \{(s_h^{(k)}, a_h^{(k)}, s_{h+1}^{(k)})\}$ is sufficient for planning?

Planning

Given dataset D and reward r, compute

$$\pi_r = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{h=1}^{H} r(s_h, a_h) \right]$$

Key Questions

Reward-free RL separates exploration and planning.
Two main questions arise.

Exploration

- How should we explore the MDP so that the collected dataset is useful?
- how large must K (the number of episodes) be so that the dataset $D = \{(s_h^{(k)}, a_h^{(k)}, s_{h+1}^{(k)})\}$ is sufficient for planning?

Planning

Given dataset D and reward r, compute

$$\pi_r = \arg\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{h=1}^{H} r(s_h, a_h) \right]$$

This part is standard once the model is known (value iteration or dynamic programming).

Sample Complexity: First Intuition

What should we expect for the amount of exploration needed?

Sample Complexity: First Intuition

What should we expect for the amount of exploration needed?

Generative model setting

Suppose we can query transitions $(s, a) \to s'$ arbitrarily. Then we can estimate the transition dynamics $P(s'|s, a)$ using roughly

$$O\left(H^3|S|^2|A|\log(|S||A|)\right)$$

samples.

Sample Complexity: First Intuition

What should we expect for the amount of exploration needed?

Generative model setting

Suppose we can query transitions $(s, a) \to s'$ arbitrarily. Then we can estimate the transition dynamics $P(s'|s, a)$ using roughly

$$O\left(H^3|S|^2|A|\log(|S||A|)\right)$$

samples.

This suggests the amount of data needed to solve arbitrary reward functions.

Why Reward-Free RL is Harder

In reward-free RL we do **not** have a generative model.
Instead we collect data from trajectories

$$s_1, a_1, s_2, a_2, \ldots$$

generated by a policy.

Why Reward-Free RL is Harder

In reward-free RL we do **not** have a generative model.
Instead we collect data from trajectories

$$s_1, a_1, s_2, a_2, \ldots$$

generated by a policy.
Two challenges:

- Some states may be difficult to reach.
- Exploration must cover states that could matter for *any* reward function.

Why Reward-Free RL is Harder

In reward-free RL we do **not** have a generative model.
Instead we collect data from trajectories

$$s_1, a_1, s_2, a_2, \ldots$$

generated by a policy.
Two challenges:

- Some states may be difficult to reach.
- Exploration must cover states that could matter for *any* reward function.

Additionally, standard RL exploration techniques do not apply.

Why Reward-Free RL is Harder

In reward-free RL we do **not** have a generative model.
Instead we collect data from trajectories

$$s_1, a_1, s_2, a_2, \ldots$$

generated by a policy.
Two challenges:

- Some states may be difficult to reach.
- Exploration must cover states that could matter for *any* reward function.

Additionally, standard RL exploration techniques do not apply.
Bonus-driven exploration

$$r(s, a) + \text{bonus}(s, a)$$

cannot be used because the reward function is unknown.

Main Result

(Jin, Krishnamurthy, Simchowitz, Yu 2020)

Main Result

(Jin, Krishnamurthy, Simchowitz, Yu 2020)

There exists an exploration algorithm such that

$$K = \tilde{O}\left(\frac{H^5|S|^2|A|}{\epsilon^2}\right)$$

episodes suffice.

Main Result

(Jin, Krishnamurthy, Simchowitz, Yu 2020)

There exists an exploration algorithm such that

$$K = \tilde{O}\left(\frac{H^5|S|^2|A|}{\epsilon^2}\right)$$

episodes suffice.
After exploration, for *any* reward function

$$r : \mathcal{S} \times \mathcal{A} \to [0, 1]$$

we can compute an ϵ-optimal policy using only the dataset.

Main Result

(Jin, Krishnamurthy, Simchowitz, Yu 2020)

There exists an exploration algorithm such that

$$K = \tilde{O}\left(\frac{H^5|S|^2|A|}{\epsilon^2}\right)$$

episodes suffice.
After exploration, for *any* reward function

$$r : \mathcal{S} \times \mathcal{A} \to [0, 1]$$

we can compute an ϵ-optimal policy using only the dataset.
Importantly:

- reward functions can be chosen **after** exploration
- many reward queries can be answered

Algorithm Overview

(Jin, Krishnamurthy, Simchowitz, Yu 2020)

Key idea: explore states that could matter for some reward.

Algorithm Overview

(Jin, Krishnamurthy, Simchowitz, Yu 2020)

Key idea: explore states that could matter for some reward.
For each state-action-round triple (s, a, h) define reward

$$r_{s,a,h}(s', a', h') = \begin{cases} 1 & (s', a', h') = (s, a, h) \\ 0 & \text{otherwise} \end{cases}$$

Algorithm Overview

(Jin, Krishnamurthy, Simchowitz, Yu 2020)

Key idea: explore states that could matter for some reward.
For each state-action-round triple (s, a, h) define reward

$$r_{s,a,h}(s', a', h') = \begin{cases} 1 & (s', a', h') = (s, a, h) \\ 0 & \text{otherwise} \end{cases}$$

Interpretation:

- this reward encourages visiting (s, a) in round h
- optimal policy optimizes for trajectories that reach (s, a) in round h

Algorithm Overview

(Jin, Krishnamurthy, Simchowitz, Yu 2020)

Key idea: explore states that could matter for some reward.
For each state-action-round triple (s, a, h) define reward

$$r_{s,a,h}(s', a', h') = \begin{cases} 1 & (s', a', h') = (s, a, h) \\ 0 & \text{otherwise} \end{cases}$$

Interpretation:

- this reward encourages visiting (s, a) in round h
- optimal policy optimizes for trajectories that reach (s, a) in round h

The Algorithm finds all such optimal policies and collects their trajectories

Algorithm Overview

(Jin, Krishnamurthy, Simchowitz, Yu 2020)

Key idea: explore states that could matter for some reward.
For each state-action-round triple (s, a, h) define reward

$$r_{s,a,h}(s', a', h') = \begin{cases} 1 & (s', a', h') = (s, a, h) \\ 0 & \text{otherwise} \end{cases}$$

Interpretation:

- this reward encourages visiting (s, a) in round h
- optimal policy optimizes for trajectories that reach (s, a) in round h

The Algorithm finds all such optimal policies and collects their trajectories
This exploration visits on **significant states** (states that can be visited with non-negligible probability).

Conclusion

Much of machine learning optimizes a single objective.

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)] \qquad \max_{\pi} \mathbb{E}\left[\sum r(s_t, a_t)\right]$$

Conclusion

Much of machine learning optimizes a single objective.

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)] \qquad \max_{\pi} \mathbb{E}\left[\sum r(s_t, a_t)\right]$$

However real systems serve many downstream users.

Conclusion

Much of machine learning optimizes a single objective.

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)] \qquad \max_{\pi} \mathbb{E}\left[\sum r(s_t, a_t)\right]$$

However real systems serve many downstream users.
Two emerging approaches:

- **Omniprediction**: one predictor supports many loss functions
- **Reward-free RL**: explore once, optimize many reward functions

Conclusion

Much of machine learning optimizes a single objective.

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)] \qquad \max_{\pi} \mathbb{E}\left[\sum r(s_t, a_t)\right]$$

However real systems serve many downstream users.
Two emerging approaches:

- **Omniprediction**: one predictor supports many loss functions
- **Reward-free RL**: explore once, optimize many reward functions

Many open questions remain:

- computational efficiency
- optimal sample complexity
- scalable algorithms for large models

Conclusion

Much of machine learning optimizes a single objective.

$$\min_{h \in \mathcal{H}} \mathbb{E}[\ell(h(x), y)] \qquad \max_{\pi} \mathbb{E}\left[\sum r(s_t, a_t)\right]$$

However real systems serve many downstream users.
Two emerging approaches:

- **Omniprediction**: one predictor supports many loss functions
- **Reward-free RL**: explore once, optimize many reward functions

Many open questions remain:

- computational efficiency
- optimal sample complexity
- scalable algorithms for large models

Learn once $\Rightarrow$ serve many downstream tasks