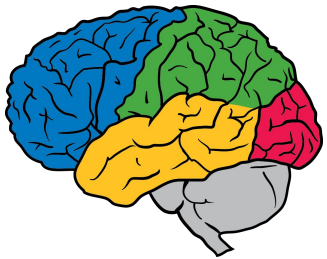
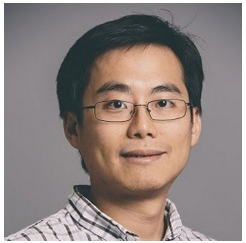
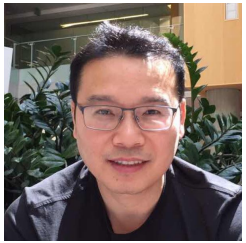
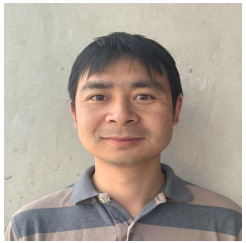
Decision Making with Many Actions and Few Online Experiments

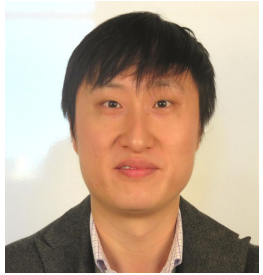
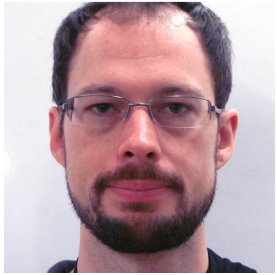
Ali Mousavi



Google Research

Collaborators


Decision Making in Real World

Decision Making Challenges

- **Dealing with a large output space:**
  - Having to choose from many class labels or possible actions.

- **Online experiments may not be feasible:**
  - A/B testing new policies might be expensive or even impossible.

- **Existence of confounders:**
  - We might not observe how they affect decision making process.

Decision Making Challenges

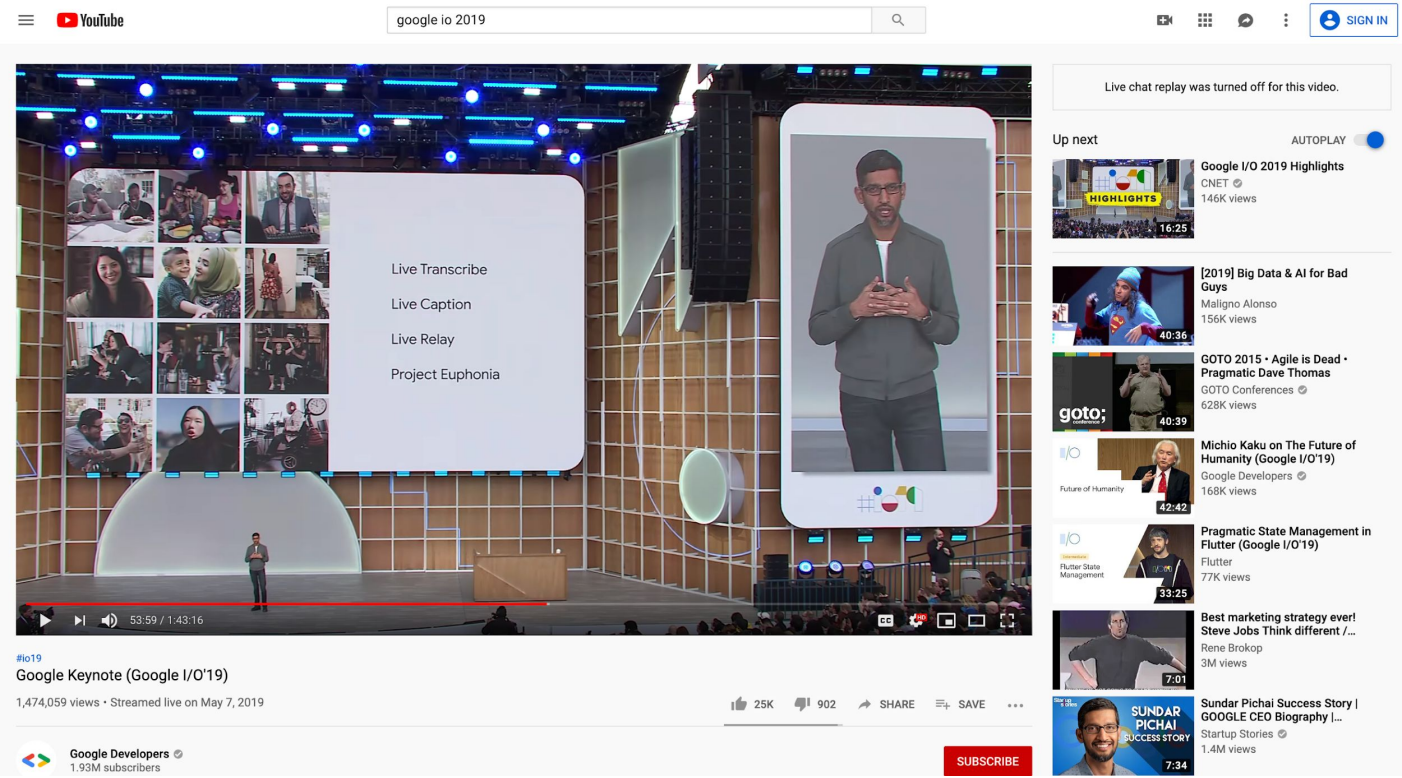
- **Dealing with a large output space:**
  - Having to choose from many class labels or possible actions.

(NeurIPS 2019)

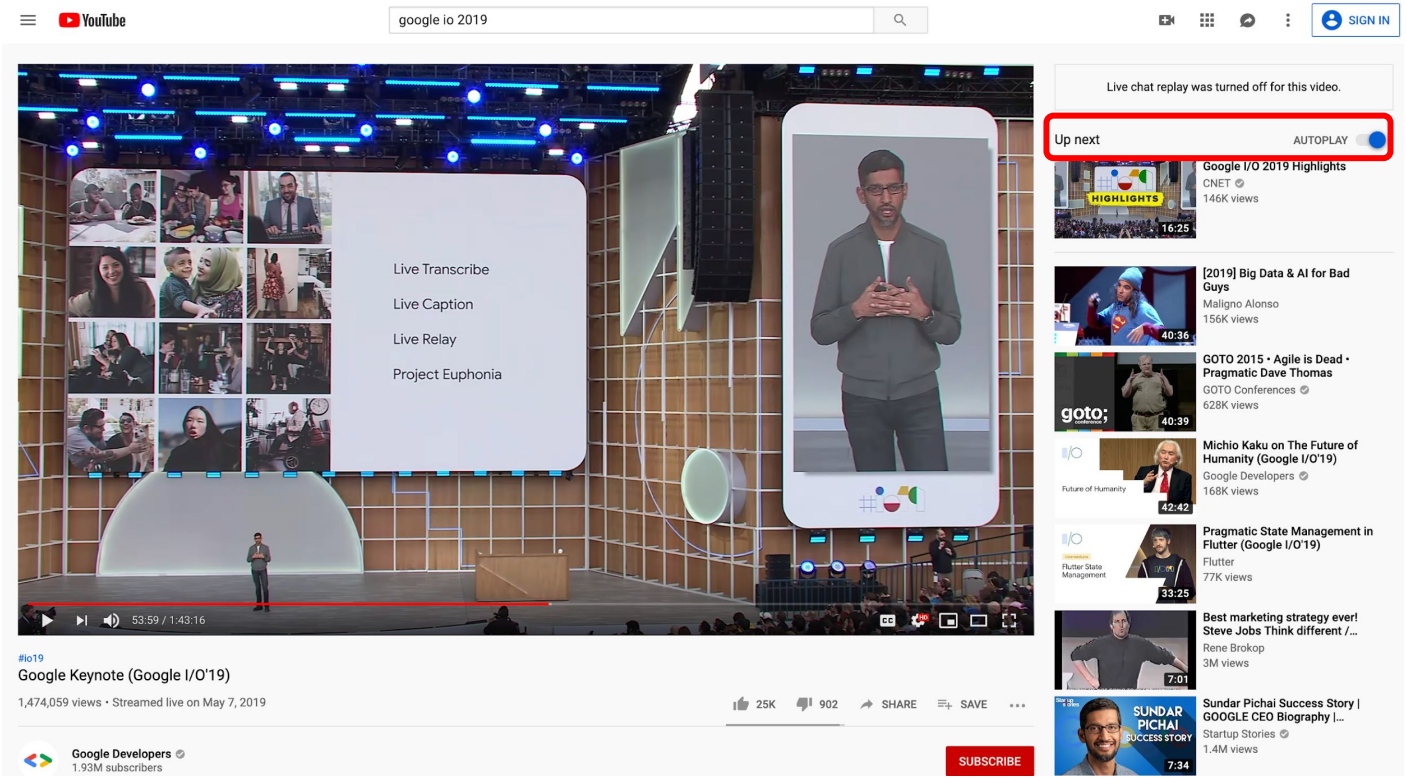
- **Online experiments may not be feasible:**
  - A/B testing new policies might be expensive or even impossible.

- **Existence of confounders:**
  - We might not observe how they affect decision making process.

Motivation

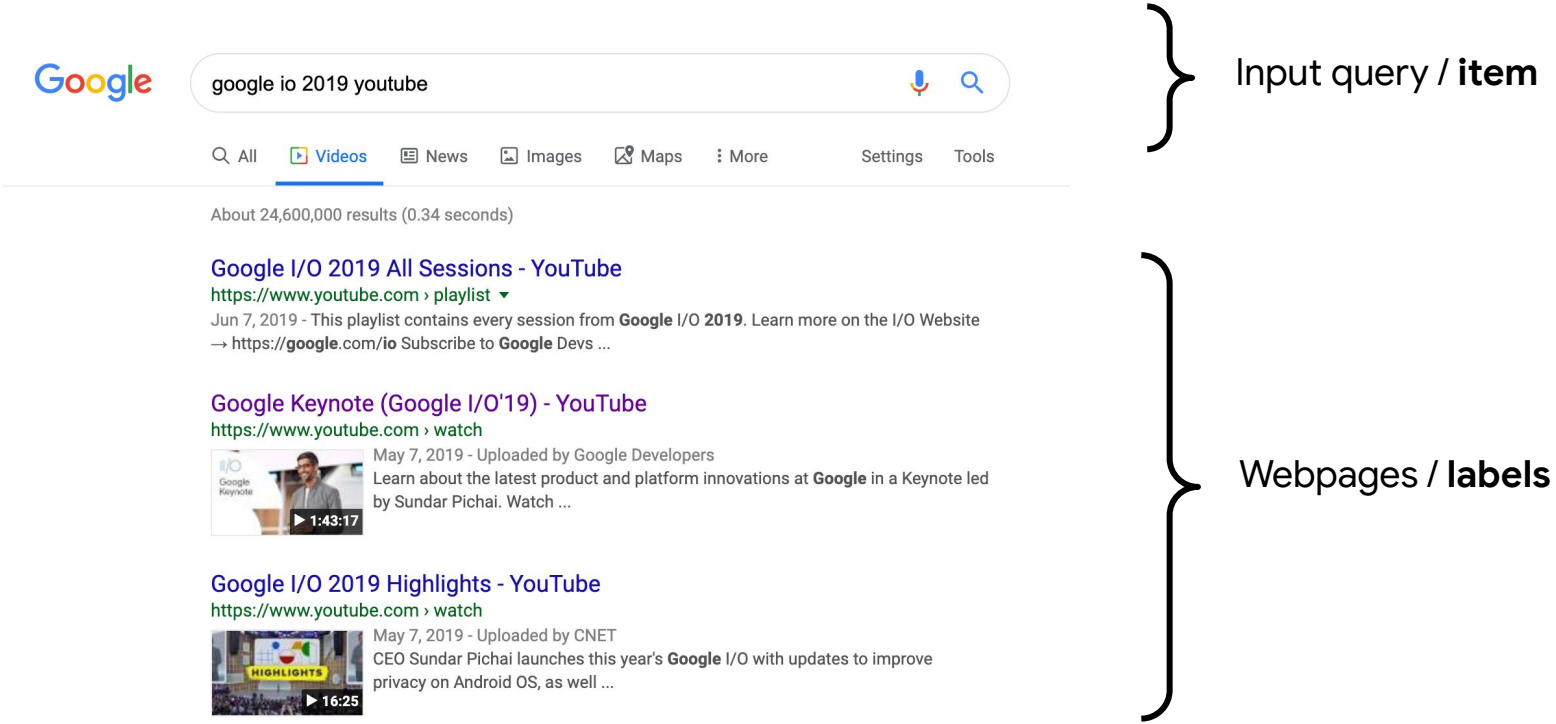

YouTube
google io 2019
SIGN IN
Live Transcribe
Live Caption
Live Relay
Project Euphonia
53:59 / 1:43:16
#io19
Google Keynote (Google I/O'19)
1,474,059 views • Streamed live on May 7, 2019
25K
902
SHARE
SAVE
Google Developers
1.93M subscribers
SUBSCRIBE
Live chat replay was turned off for this video.
Up next
AUTOPLAY
Google I/O 2019 Highlights
CNET
146K views
16:25
[2019] Big Data & AI for Bad Guys
Maligno Alonso
156K views
40:36
GOTO 2015 • Agile is Dead • Pragmatic Dave Thomas
GOTO Conferences
628K views
40:39
Michio Kaku on The Future of Humanity (Google I/O'19)
Google Developers
168K views
42:42
Pragmatic State Management in Flutter (Google I/O'19)
Flutter
77K views
33:25
Best marketing strategy ever! Steve Jobs Think different /...
Rene Brokop
3M views
7:01
Sundar Pichai Success Story | GOOGLE CEO Biography |...
Startup Stories
1.4M views
7:34

Motivation


YouTube
google io 2019
SIGN IN
Live Transcribe
Live Caption
Live Relay
Project Euphonia
53:59 / 1:43:16
#io19
Google Keynote (Google I/O'19)
1,474,059 views • Streamed live on May 7, 2019
25K
902
SHARE
SAVE
Google Developers
1.93M subscribers
SUBSCRIBE
Live chat replay was turned off for this video.
Up next
AUTOPLAY
Google I/O 2019 Highlights
CNET
146K views
16:25
[2019] Big Data & AI for Bad Guys
Maligno Alonso
156K views
40:36
GOTO 2015 • Agile is Dead • Pragmatic Dave Thomas
GOTO Conferences
628K views
40:39
Michio Kaku on The Future of Humanity (Google I/O'19)
Google Developers
168K views
42:42
Pragmatic State Management in Flutter (Google I/O'19)
Flutter
77K views
33:25
Best marketing strategy ever! Steve Jobs Think different /...
Rene Brokop
3M views
7:01
Sundar Pichai Success Story | GOOGLE CEO Biography |...
Startup Stories
1.4M views
7:34

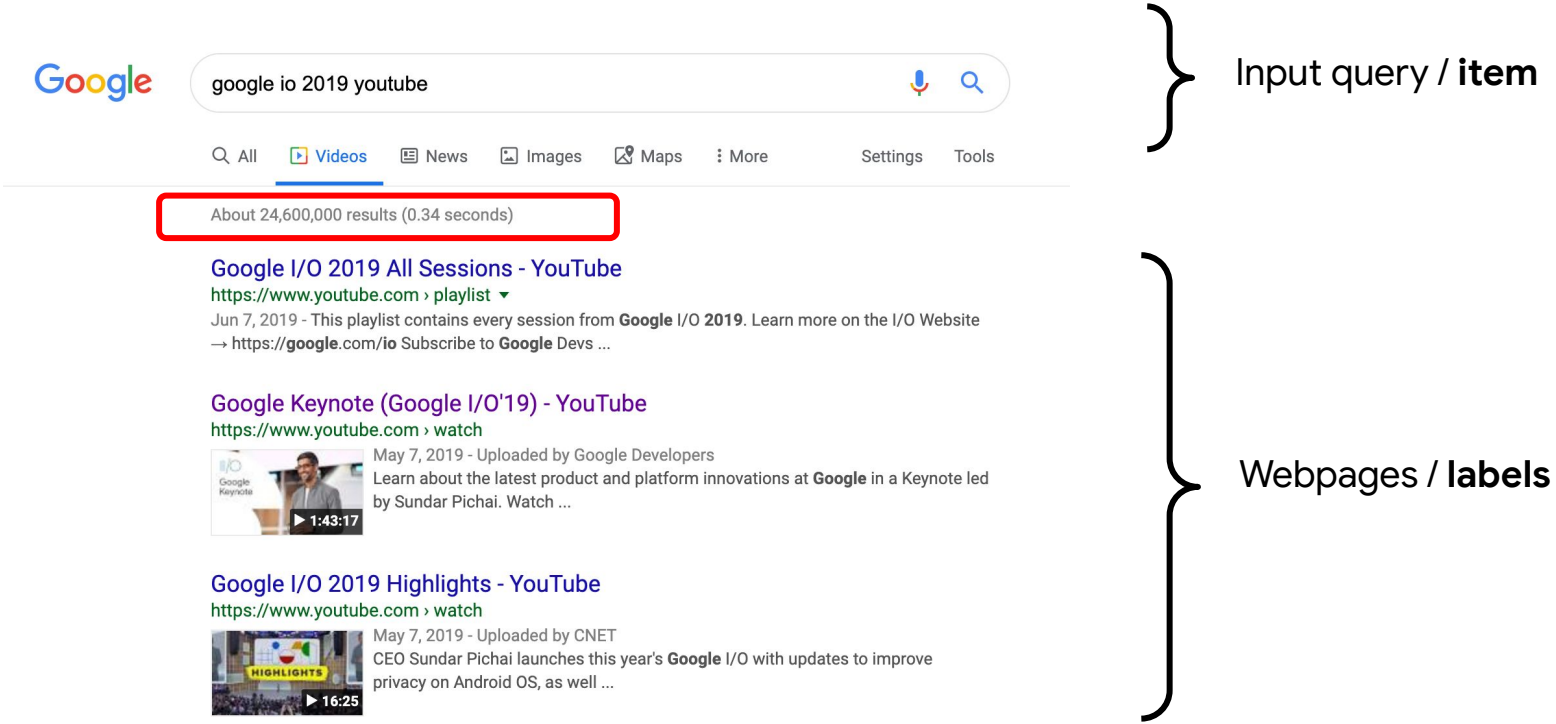
Motivation

- Search as a classification problem.



Motivation

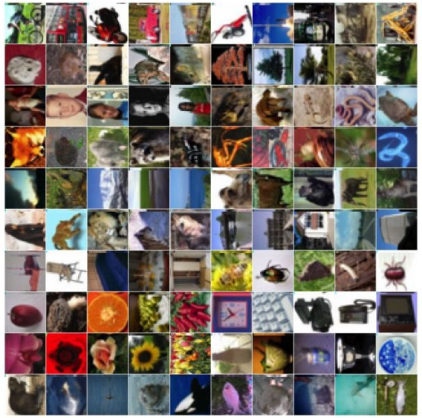
- Search as a classification problem.



Traditional Space vs. Large Output Space

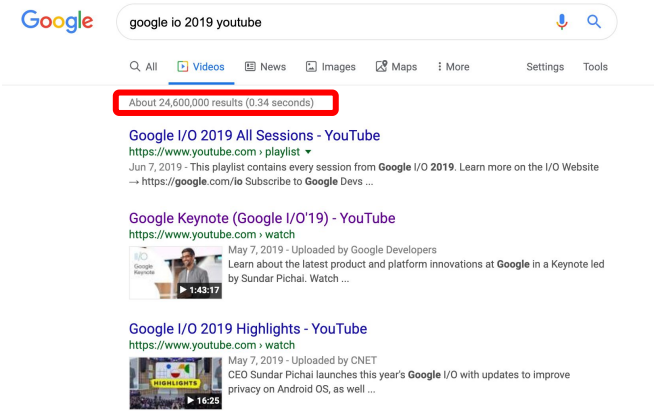
MNIST

10 Classes



CIFAR-100

100 Classes

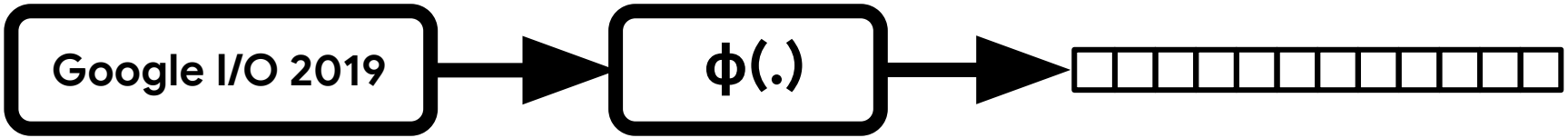


Google I/O 2019

~25 Million Classes

Problem Statement

- Input query: **Google I/O 2019**



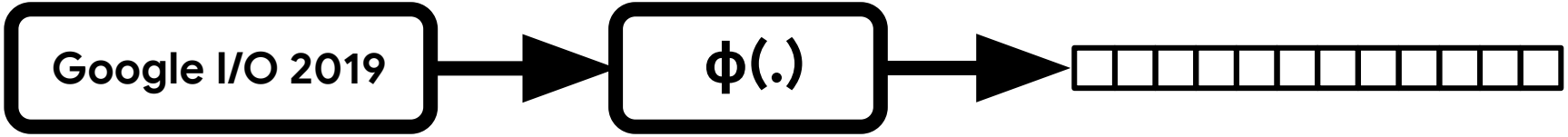
Input Vector
$\mathbf{x} \in \mathbb{R}^D$

Embedding Function
$\phi : \mathbb{R}^D \rightarrow \mathbb{R}^d$

Embedded Vector
$\phi(\mathbf{x}) \in \mathbb{R}^d$

Problem Statement

- Input query: **Google I/O 2019**



Input Vector
$\mathbf{x} \in \mathbb{R}^D$

Embedding Function
$\phi : \mathbb{R}^D \rightarrow \mathbb{R}^d$

Embedded Vector
$\phi(\mathbf{x}) \in \mathbb{R}^d$

- Classification:

Embedded Vector
$\phi(\mathbf{x}) \in \mathbb{R}^d$

Classifier
$f : \mathbb{R}^d \rightarrow \{0,1\}^K$

Output Labels Vector
$f(\phi(\mathbf{x})) \in \{0,1\}^K$

Problem Statement

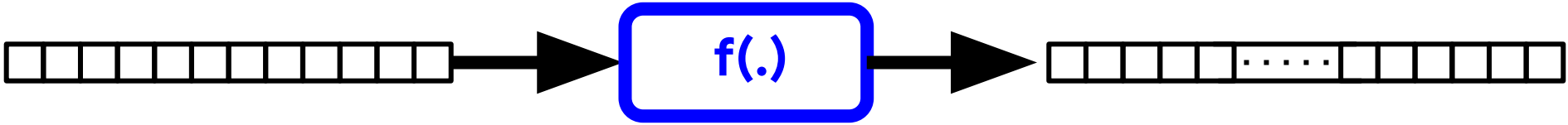
- Input query: **Google I/O 2019**

Input Vector
$\mathbf{x} \in \mathbb{R}^D$

Embedding Function
$\phi : \mathbb{R}^D \rightarrow \mathbb{R}^d$

Embedded Vector
$\phi(\mathbf{x}) \in \mathbb{R}^d$

- Classification:



Embedded Vector
$\phi(\mathbf{x}) \in \mathbb{R}^d$

Classifier
$f : \mathbb{R}^d \rightarrow \{0,1\}^K$

Output Labels Vector
$f(\phi(\mathbf{x})) \in \{0,1\}^K$

Previous Work

- Previous methods:
  - Tree-based methods.
  - Sparse methods.
  - Embedding-based methods.
- Widespread belief:
  - Embedding-based methods are not good classifiers for large output space.

Input $\mathbf{x} \in \mathbb{R}^D$

Embedded Input $\phi(\mathbf{x}) \in \mathbb{R}^d$

$$d \ll D$$

- Our work:
  - **The problem with embedding-based methods is not their representation power!**

Embedding-Based Methods

- Input: $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^D$
- Label vector of an input: $\mathbf{y} \in \mathcal{Y} \subset \{0, 1\}^K$

Embedding-Based Methods

- Input: $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^D$
- Label vector of an input: $\mathbf{y} \in \mathcal{Y} \subset \{0,1\}^K$
- The goal of an embedding-based method: learn $f(\phi(\mathbf{x}))$

$$f(\phi(\mathbf{x})) : \mathcal{X} \to \{0,1\}^K$$

$$\phi(\mathbf{x}) \in \mathbb{R}^d \text{ and } d \ll D$$

$$f : \mathbb{R}^d \to \{0,1\}^K$$

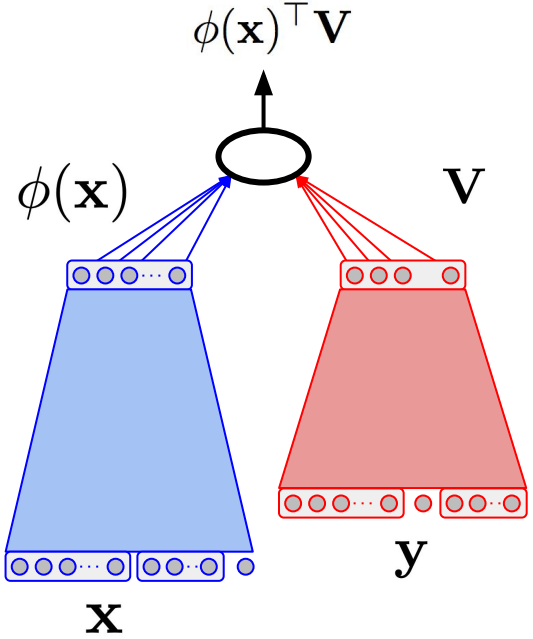
Embedding-Based Methods

- Input: $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^D$
- Label vector of an input: $\mathbf{y} \in \mathcal{Y} \subset \{0,1\}^K$
- The goal of an embedding-based method: learn $f(\phi(\mathbf{x}))$

  $f(\phi(\mathbf{x})) : \mathcal{X} \to \{0,1\}^K$

  $\phi(\mathbf{x}) \in \mathbb{R}^d$ and $d \ll D$

  $f : \mathbb{R}^d \to \{0,1\}^K$
- Label embedding matrix: $\mathbf{V} \in \mathbb{R}^{d \times K}$
- Linear Classifier: $\phi(\mathbf{x})^\top \mathbf{V}$



Embedding-Based Methods

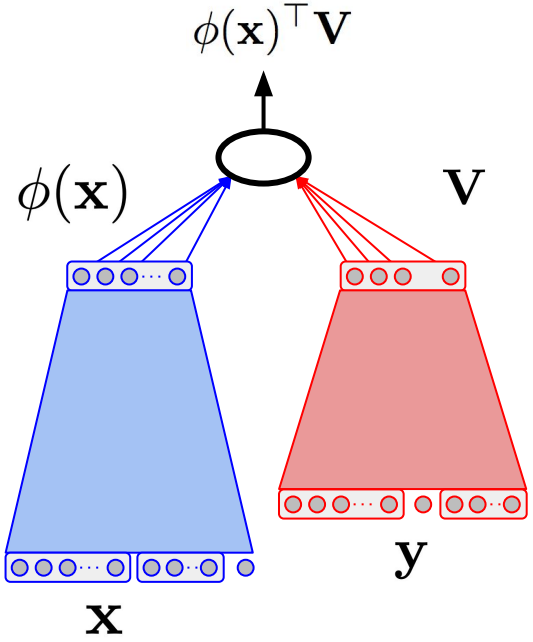
- Input: $\mathbf{x} \in \mathcal{X} \subset \mathbb{R}^D$
- Label vector of an input: $\mathbf{y} \in \mathcal{Y} \subset \{0,1\}^K$
- The goal of an embedding-based method: learn $f(\phi(\mathbf{x}))$

  $f(\phi(\mathbf{x})) : \mathcal{X} \to \{0,1\}^K$

  $\phi(\mathbf{x}) \in \mathbb{R}^d$ and $d \ll D$

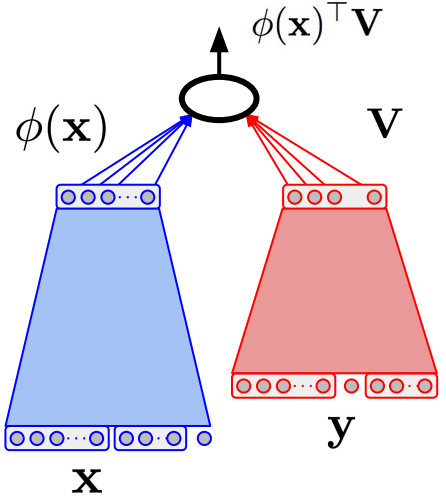
  $f : \mathbb{R}^d \to \{0,1\}^K$
- Label embedding matrix: $\mathbf{V} \in \mathbb{R}^{d \times K}$
- Linear Classifier: $\phi(\mathbf{x})^\top \mathbf{V}$
- Predicting labels:

  1-Thresholding: $\{y : \phi(\mathbf{x})^\top \mathbf{v}_y \geq \tau\}$ 2- Top-m: $\text{Top}(\phi(\mathbf{x})^\top \mathbf{V}, m)$



Existence of Perfect Low-Dimensional Embedding Classifier

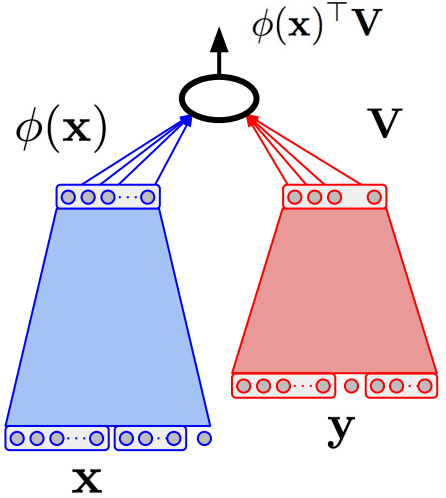
Theorem: in the limit of infinite expressivity of the embedding map, there exists a low-dimensional embedding-based linear classifier that thresholds at 1/2 and has perfect accuracy.



Existence of Perfect Low-Dimensional Embedding Classifier

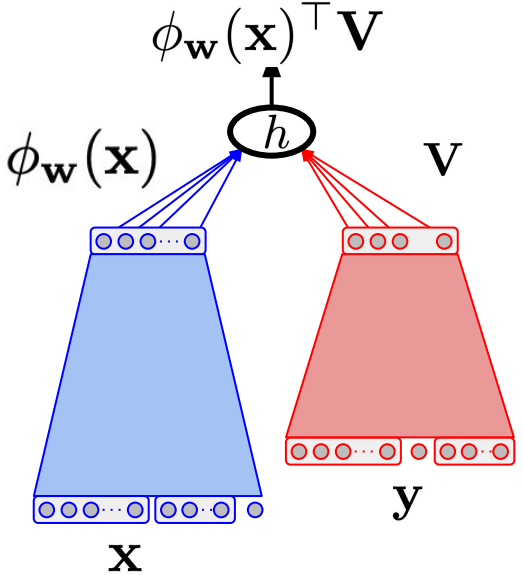
Theorem: in the limit of infinite expressivity of the embedding map, there exists a low-dimensional embedding-based linear classifier that thresholds at 1/2 and has perfect accuracy.

- Deep neural networks have excellent function approximation capabilities.
- Use neural networks to compute the embedding map.



Algorithm

- Input embedding function: $\phi_{\mathbf{w}}(\mathbf{x}) \in \mathbb{R}^d$
- Label embedding matrix: $\mathbf{V} \in \mathbb{R}^{d \times K}$
- Scoring function: $h : \mathcal{X} \to \mathbb{R}^K \quad h(\mathbf{x}) = \phi_{\mathbf{w}}(\mathbf{x})^\top \mathbf{V}$

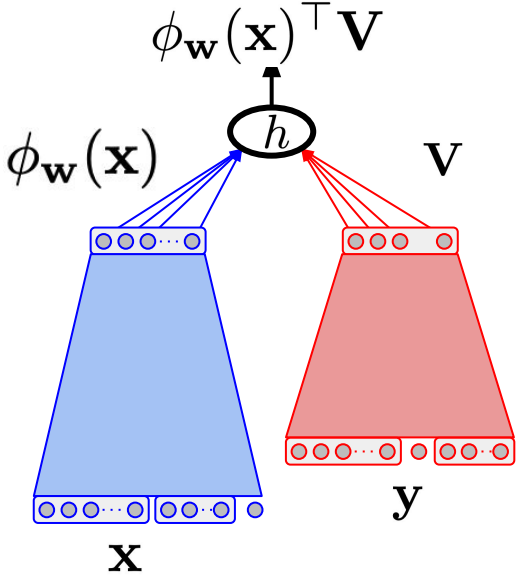


Algorithm

- Input embedding function: $\phi_{\mathbf{w}}(\mathbf{x}) \in \mathbb{R}^d$
- Label embedding matrix: $\mathbf{V} \in \mathbb{R}^{d \times K}$
- Scoring function: $h : \mathcal{X} \to \mathbb{R}^K \quad h(\mathbf{x}) = \phi_{\mathbf{w}}(\mathbf{x})^\top \mathbf{V}$

- Optimization:
  - **Margin Loss**

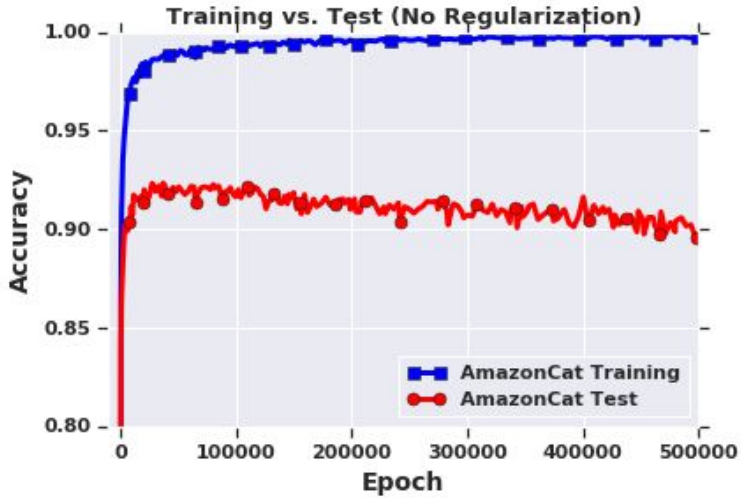
$$\ell(h(\mathbf{x}), \mathbf{y}) = \sum_{y \in L_{\mathbf{y}}} \sum_{y' \notin L_{\mathbf{y}}} [h(\mathbf{x})_{y'} - h(\mathbf{x})_y + c]_+ ,$$
$$:= \sum_{y \in L_{\mathbf{y}}} \ell(h(\mathbf{x}), y).$$



Algorithm

Algorithm 1 Training the basic embedding model

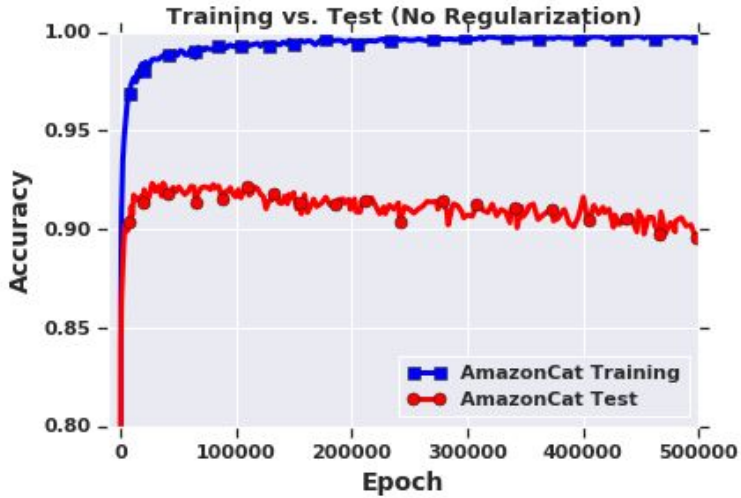
Input: Dataset $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$
  Feature embedding model $\phi_{\mathbf{w}} : \mathcal{X} \to \mathbb{R}^d$
  Label embedding matrix $\mathbf{V} \in \mathbb{R}^{d \times K}$
  Loss function $\ell : \mathbb{R}^K \times [K] \to \mathbb{R}$
  Learning rates $\eta_{\mathbf{w}}, \eta_{\mathbf{V}}$
Initialize $\mathbf{w}$, $\mathbf{V}$
repeat
  Sample a batch $\mathbf{x}_1, \ldots, \mathbf{x}_B$
  Sample indices $k_1, \ldots, k_B$ uniformly from non-zero indices of $\mathbf{y}_1, \ldots, \mathbf{y}_B$
  Compute loss $L \leftarrow \frac{1}{B} \sum_{i=1}^{B} \ell(\phi_{\mathbf{w}}(\mathbf{x}_i)^\top \mathbf{V}, k_i)$
  Compute gradients $\frac{dL}{d\mathbf{w}}$ and $\frac{dL}{d\mathbf{V}}$ via backpropagation
  Update $\mathbf{w} \leftarrow \mathbf{w} - \eta_{\mathbf{w}} \frac{dL}{d\mathbf{w}}$, $\mathbf{V} \leftarrow \mathbf{V} - \eta_{\mathbf{V}} \frac{dL}{d\mathbf{V}}$
until convergence



Algorithm

Algorithm 1 Training the basic embedding model

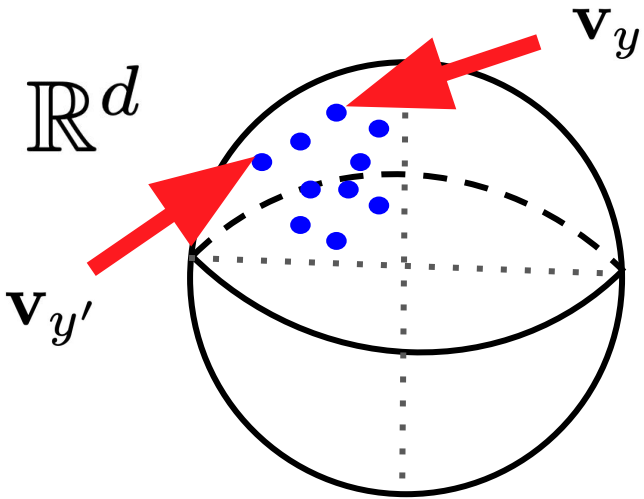
Input: Dataset $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$
  Feature embedding model $\phi_{\mathbf{w}} : \mathcal{X} \to \mathbb{R}^d$
  Label embedding matrix $\mathbf{V} \in \mathbb{R}^{d \times K}$
  Loss function $\ell : \mathbb{R}^K \times [K] \to \mathbb{R}$
  Learning rates $\eta_{\mathbf{w}}, \eta_{\mathbf{V}}$
Initialize $\mathbf{w}$, $\mathbf{V}$
repeat
  Sample a batch $\mathbf{x}_1, \ldots, \mathbf{x}_B$
  Sample indices $k_1, \ldots, k_B$ uniformly from non-zero indices of $\mathbf{y}_1, \ldots, \mathbf{y}_B$
  Compute loss $L \leftarrow \frac{1}{B} \sum_{i=1}^{B} \ell(\phi_{\mathbf{w}}(\mathbf{x}_i)^\top \mathbf{V}, k_i)$
  Compute gradients $\frac{dL}{d\mathbf{w}}$ and $\frac{dL}{d\mathbf{V}}$ via backpropagation
  Update $\mathbf{w} \leftarrow \mathbf{w} - \eta_{\mathbf{w}} \frac{dL}{d\mathbf{w}}$, $\mathbf{V} \leftarrow \mathbf{V} - \eta_{\mathbf{V}} \frac{dL}{d\mathbf{V}}$
until convergence



Overfitting!!!

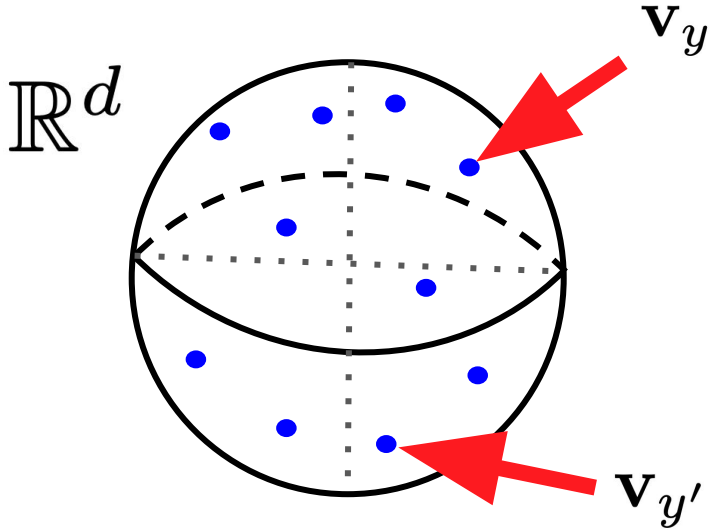
Regularization

- Spread-out embedding: $\ell_{\text{spreadout}} = \frac{1}{K^2} \sum_{y=1}^{K} \sum_{y'=1}^{K} (\mathbf{v}_y^\top \mathbf{v}_{y'})^2$



Not Spreaded-out embeddings

$\mathbf{v}_y^\top \mathbf{v}_{y'} \approx 1$

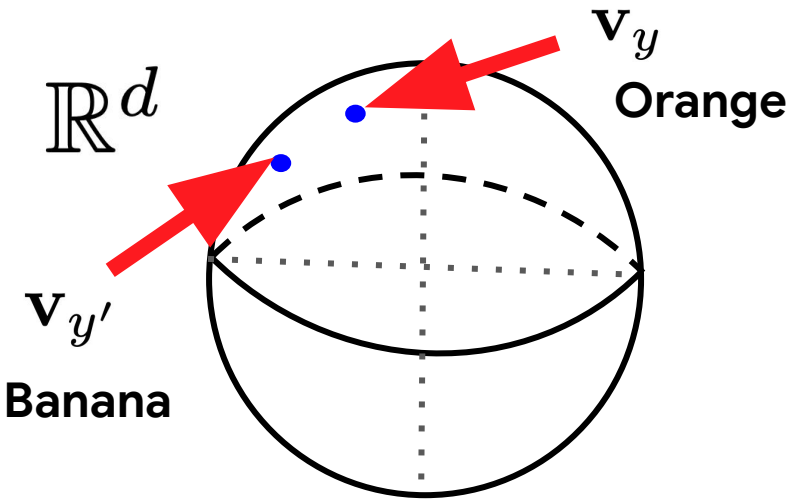


Spreaded-out embeddings

$\mathbf{v}_y^\top \mathbf{v}_{y'} \approx 0$

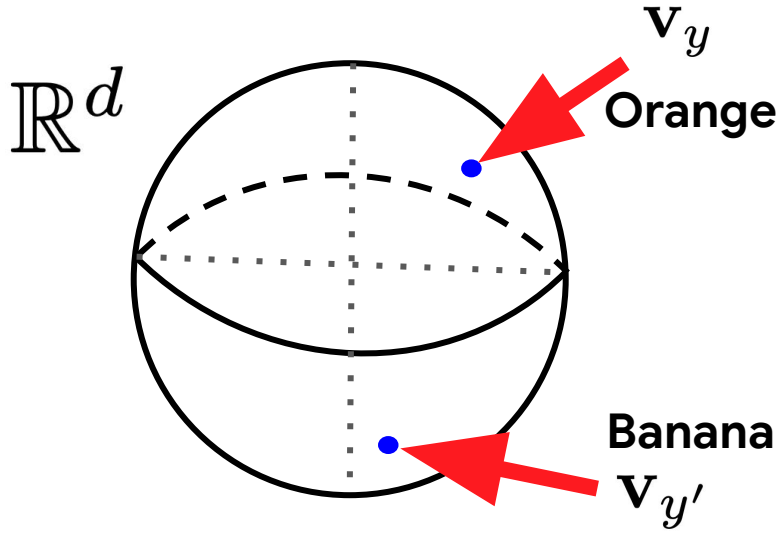
Regularization

- Laplacian-based regularizer: $\ell_{\text{Laplacian}} = \frac{1}{K^2} \sum_{y=1}^{K} \sum_{y'=1}^{K} \|\mathbf{v}_y - \mathbf{v}_{y'}\|_2^2 u_{yy'}$



Not overpenalized!

$\mathbf{v}_y^\top \mathbf{v}_{y'} \approx 1$



Spreaded-out but overpenalized

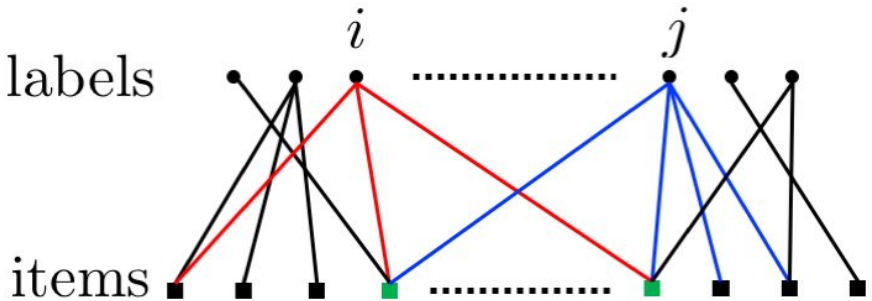
$\mathbf{v}_y^\top \mathbf{v}_{y'} \approx 0$

GLaS Regularization

- Spread-out embedding: $\ell_{\text{spreadout}} = \frac{1}{K^2} \sum_{y=1}^{K} \sum_{y'=1}^{K} (\mathbf{v}_y^\top \mathbf{v}_{y'})^2$
- Laplacian-based regularizer: $\ell_{\text{Laplacian}} = \frac{1}{K^2} \sum_{y=1}^{K} \sum_{y'=1}^{K} \|\mathbf{v}_y - \mathbf{v}_{y'}\|_2^2 u_{yy'}$
- **GLaS regularizer:** $\ell_{\text{Laplacian}} + \ell_{\text{spreadout}} = \frac{1}{K^2} \|\mathbf{V}^\top \mathbf{V} - U\|_F^2$

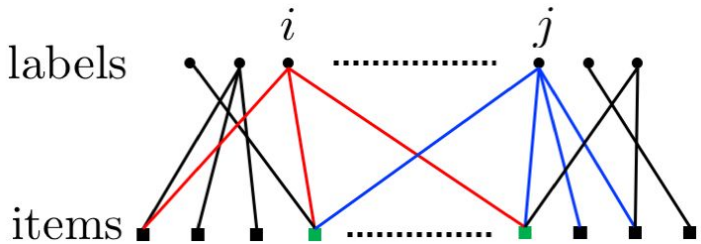
GLaS Regularization

- Spread-out embedding: $\ell_{\text{spreadout}} = \frac{1}{K^2} \sum_{y=1}^{K} \sum_{y'=1}^{K} (\mathbf{v}_y^\top \mathbf{v}_{y'})^2$
- Laplacian-based regularizer: $\ell_{\text{Laplacian}} = \frac{1}{K^2} \sum_{y=1}^{K} \sum_{y'=1}^{K} \|\mathbf{v}_y - \mathbf{v}_{y'}\|_2^2 u_{yy'}$
- **GLaS regularizer:** $\ell_{\text{Laplacian}} + \ell_{\text{spreadout}} = \frac{1}{K^2} \|\mathbf{V}^\top \mathbf{V} - U\|_F^2$



GLaS Regularization

- Label matrix: $Y \in \{0,1\}^{n \times K}$
- Co-occurrence matrix: $A = Y^\top Y$
- Degree Matrix: $Z = \mathrm{diag}(A) \in \mathbb{R}^{K \times K}$



- Conditional frequency:

$$(AZ^{-1})_{y,y'} = \frac{A_{y,y'}}{A_{y',y'}} = \frac{\text{number of times } y \text{ and } y' \text{ co-occur}}{\text{number of times } y' \text{ occurs}} =: F(y|y')$$

- Conditional frequency:

$$\ell_{\text{GLaS}} = \frac{1}{K^2} \sum_{y=1}^{K} \sum_{y'=1}^{K} \left(\mathbf{v}_y^\top \mathbf{v}_{y'} - \frac{1}{2}[F(y|y') + F(y'|y)] \right)^2$$

$$= \frac{1}{K^2} \left\| \mathbf{V}^\top \mathbf{V} - \frac{1}{2}(AZ^{-1} + Z^{-1}A) \right\|_F^2$$

Algorithm

- Regularizations:
  - Embedding normalization.
  - GLaS regularization.
  - Input dropout.

Algorithm 2 Training with regularization

Input: Dataset $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)\}$
Feature embedding model $\phi_{\mathbf{w}} : \mathcal{X} \to \mathbb{R}^d$
Label embedding matrix $\mathbf{V} \in \mathbb{R}^{d \times K}$
Loss function $\ell : \mathbb{R}^K \times \mathcal{Y} \to \mathbb{R}$
GLaS loss $\ell_{\text{GLaS}} : \mathbb{R}^{B \times B} \times \mathbb{R}^{B \times B} \to \mathbb{R}$
Regularization weight λ
Dropout probability $\rho \in [0, 1]$
Learning rates $\eta_{\mathbf{w}}, \eta_{\mathbf{V}}$
Initialize $\mathbf{w}$, $\mathbf{V}$
repeat
  Sample a batch $\mathbf{x}_1, \ldots, \mathbf{x}_B$
  Sample labels $y_1, \ldots, y_B$ uniformly from non-zero indices of $\mathbf{y}_1, \ldots, \mathbf{y}_B$
  Apply input dropout $\mathbf{x}_i \leftarrow \mathbf{x}_i \odot \text{Bernoulli}(\rho, D)$
  Compute loss $L \leftarrow \frac{1}{B} \sum_{i=1}^{B} \ell(\phi_{\mathbf{w}}(\mathbf{x}_i)^\top \mathbf{V}, y_i)$
  $Y \leftarrow [\mathbf{y}_1 | \cdots | \mathbf{y}_B]$
  $U \leftarrow B \times B$ submatrix of Equation (5) corresponding to indices $y_1, \ldots, y_B$
  $\mathbf{V} \leftarrow [\mathbf{v}_{y_1} | \cdots | \mathbf{v}_{y_B}] \in \mathbb{R}^{B \times B}$
  Regularize $L \leftarrow L + \lambda \ell_{\text{GLaS}}(\mathbf{V}^\top \mathbf{V}, U)$
  Compute gradients $\frac{dL}{d\mathbf{w}}$ and $\frac{dL}{d\mathbf{V}}$ via backpropagation
  Update $\mathbf{w} \leftarrow \mathbf{w} - \eta_{\mathbf{w}} \frac{dL}{d\mathbf{w}}$, $\mathbf{V} \leftarrow \mathbf{V} - \eta_{\mathbf{V}} \frac{dL}{d\mathbf{V}}$
until convergence

Public Datasets

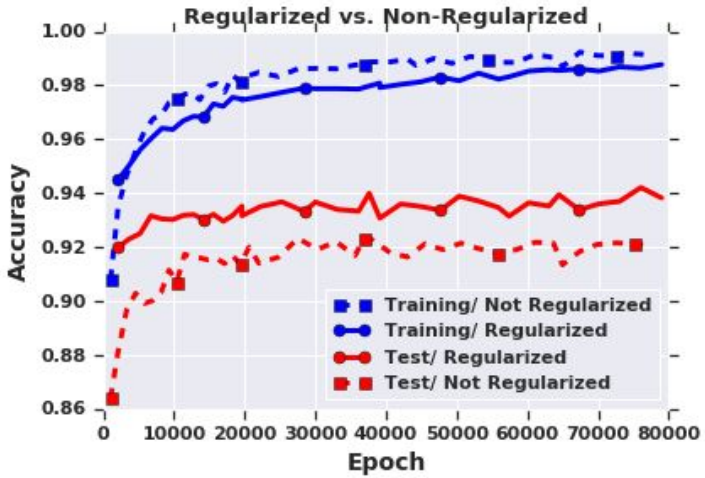
Dataset	Feature Dimensionality	Label Dimensionality	Number of Train Points	Number of Test Points	Avg. Points Per Label	Avg. Labels Per Point
AMAZONCAT-13K	203,882	13,330	1,186,239	306,782	448.57	5.04
AMAZON-670K	135,909	670,091	490,449	153,025	3.99	5.45
WIKILSHTC	1,617,899	325,056	1,778,351	587,084	17.46	3.19
DELICIOUS-200K	782,585	205,443	196,606	100,095	72.29	75.54
EURLEX-4K	5,000	3,993	12,920	3,185	25.73	5.31
WIKIPEDIA-500K	2,381,304	501,070	1,813,391	783,743	24.75	4.77

Hyperparameter Tuning

Variable	Parameters			
Regularizer	None	GLaS	Spread-out	Gravity
	92.34	**94.21**	93.34	93.42
Regularization Weight	$\lambda = 1$	$\lambda = 10$	$\lambda = 100$	
	93.68	**94.21**	93.75	
Input Dropout	$\rho = 1.0$	$\rho = 0.8$	$\rho = 0.6$	
	93.39	**94.21**	94.08	
Batch Size	1024	2048	4096	
	94.04	93.98	**94.21**	
Embedding Size	$d = 256$	$d = 512$	$d = 1024$	
	93.24	93.82	**94.21**	
Embedding Type	Linear	Nonlinear (ReLU)		
	91.77	**94.21**		

Generalization Gap

Dataset	Regularization	Train Acc.	Test Acc.	Gen. Gap
AMAZONCAT	GLaS	98.77	94.21	**4.56**
	None	99.23	92.34	6.89
AMAZON-670K	GLaS	96.10	46.32	**49.78**
	None	98.21	44.53	53.68



Comparison with Other methods

		Embedding-Based					Other Methods					
Dataset	**P@k**	Ours	SLEEC [5]	LEML [34]	RobustXML [30]	XML-CNN [18]	PfastreXML [13]	FastXML [23]	Parabel [22]	DiSMEC [3]	PD-Sparse [33]	PPD-Sparse [32]
AmazonCat-13K	P@1	***94.21***	90.53	-	88.4	-	91.75	93.11	93.03	93.40	90.60	-
	P@3	***79.70***	76.33	-	74.6	-	77.97	78.2	79.16	79.10	75.14	-
	P@5	***64.84***	61.52	-	60.6	-	63.68	63.41	64.52	64.10	60.69	-
WikiLSHTC-325K	P@1	***65.46***	54.83	19.82	53.5	-	56.05	49.75	65.04	64.40	61.26	64.08
	P@3	***45.44***	33.42	11.43	31.8	-	36.79	33.10	43.23	42.50	39.48	41.26
	P@5	***34.51***	23.85	8.39	29.9	-	27.09	24.45	32.05	31.50	28.79	30.12
Amazon-670K	P@1	***46.38***	35.05	8.13	31.0	35.39	39.46	36.99	44.89	44.70	-	45.32
	P@3	***42.09***	31.25	6.83	28.0	31.93	35.81	33.28	39.80	39.70	-	40.37
	P@5	***38.56***	28.56	6.03	24.0	29.32	33.05	30.53	36.00	36.10	-	36.92
Delicious-200K	P@1	46.4	***47.85***	40.73	45.0	-	41.72	43.07	46.97	45.50	34.37	-
	P@3	40.49	***42.21***	37.71	40.0	-	37.83	38.66	40.08	38.70	29.48	-
	P@5	38.1	***39.43***	35.84	38.0	-	35.58	36.19	36.63	35.50	27.04	-
EURLex-4K	P@1	77.5	**79.26**	63.4	-	76.38	75.45	71.36	81.73	82.4	76.43	*83.83*
	P@3	**65.01**	64.3	50.35	-	62.81	62.7	59.9	68.78	68.5	60.37	*70.72*
	P@5	**54.37**	52.33	41.28	-	51.41	52.51	50.39	57.44	57.7	49.72	*59.21*
Wikipedia-500K	P@1	**69.91**	48.2	41.3	-	59.85	59.52	54.1	66.73	*70.2*	-	70.16
	P@3	**49.08**	29.4	30.1	-	39.28	40.24	35.5	47.48	*50.6*	-	50.57
	P@5	**38.35**	21.2	19.8	-	29.81	30.72	26.2	36.78	*39.7*	-	39.66

Comparison with Other methods

		Embedding-Based			Other Methods					
Dataset	PSP@k	Ours	SLEEC [6]	LEML [36]	PfastreXML [14]	FastXML [24]	Parabel [23]	DiSMEC [3]	PD-Sparse [34]	PPD-Sparse [33]
AMAZONCAT-13K	PSP@1	**47.53**	46.75	-	*69.52*	48.31	50.93	59.10	49.58	-
	PSP@3	**62.74**	58.46	-	*73.22*	60.26	64.00	67.10	61.63	-
	PSP@5	**71.66**	65.96	-	*75.48*	69.30	72.08	71.20	68.23	-
WIKILSHTC-325K	PSP@1	***46.22***	20.27	3.48	30.66	16.35	26.76	29.1	28.34	27.47
	PSP@3	***46.15***	23.18	3.79	31.55	20.99	33.27	35.6	33.50	33.00
	PSP@5	***47.28***	25.08	4.27	33.12	23.56	37.36	39.5	36.62	36.29
AMAZON-670K	PSP@1	***38.94***	20.62	2.07	29.30	19.37	25.43	27.8	-	26.64
	PSP@3	***39.72***	23.32	2.26	30.80	23.26	29.43	30.6	-	30.65
	PSP@5	***41.24***	25.98	2.47	32.43	26.85	32.85	34.2	-	34.65
DELICIOUS-200K	PSP@1	***28.68***	7.17	6.06	3.15	6.48	7.25	6.5	5.29	-
	PSP@3	***24.93***	8.16	7.24	3.87	7.52	7.94	7.6	5.80	-
	PSP@5	***23.87***	8.96	8.10	4.43	8.31	8.52	8.4	6.24	-
EURLEX-4K	PSP@1	***49.77***	34.25	24.10	43.86	26.62	36.36	41.20	36.28	-
	PSP@3	***51.05***	38.35	26.37	45.23	32.07	41.95	44.30	40.96	-
	PSP@5	***53.82***	40.30	27.62	46.03	35.23	44.78	46.90	42.84	-

Summary

- Embedding-based methods can have competitive performance in classification with large output space.
- Neural network models could suffer from overfitting instead of low-dimensional embedding bottleneck when applied to classification with large output space.
- Proper regularization and data-augmentation techniques for embedding-based classifiers can significantly reduce overfitting.

Decision Making Challenges

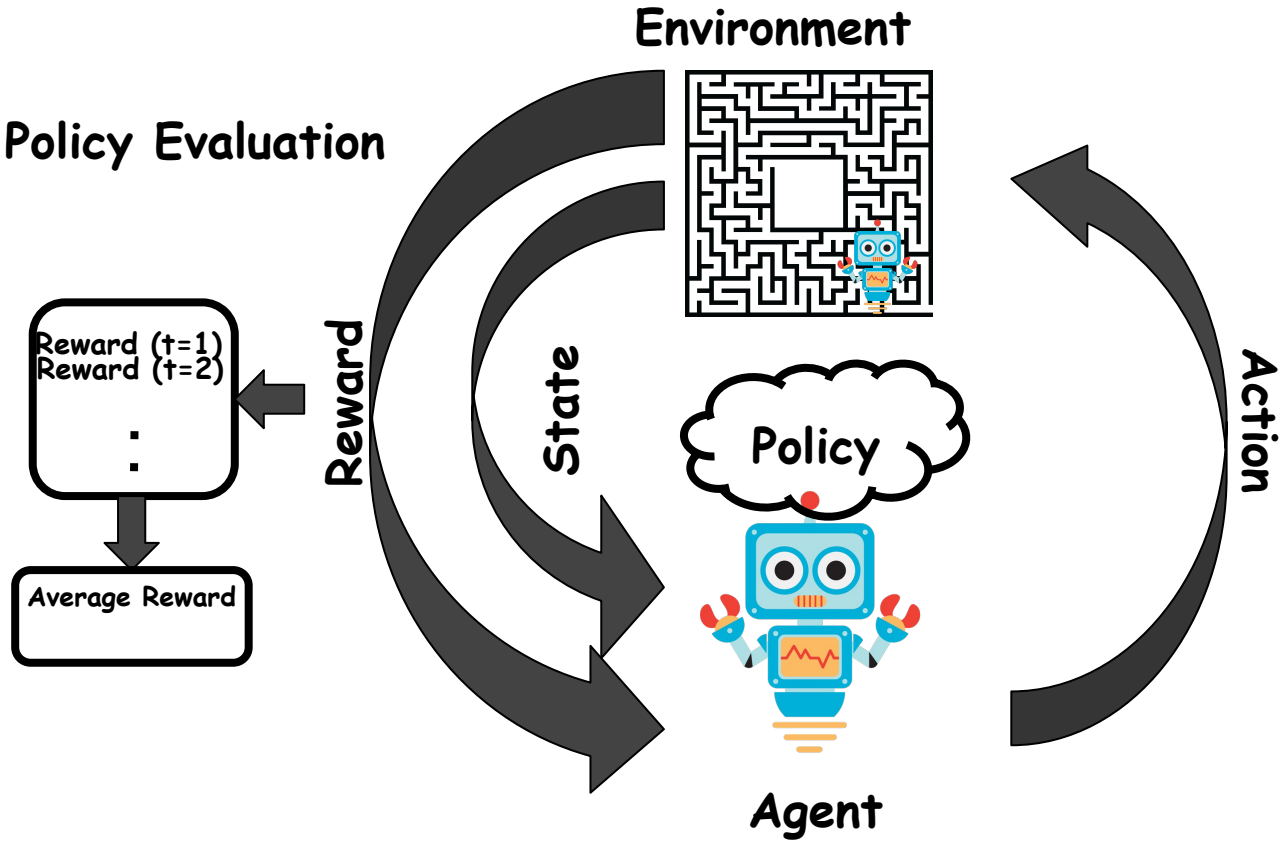
- **Dealing with a large output space:**
  - Having to choose from many class labels or possible actions.

- **Online experiments may not be feasible:**
  - A/B testing new policies might be expensive or even impossible.

(ICLR 2020)

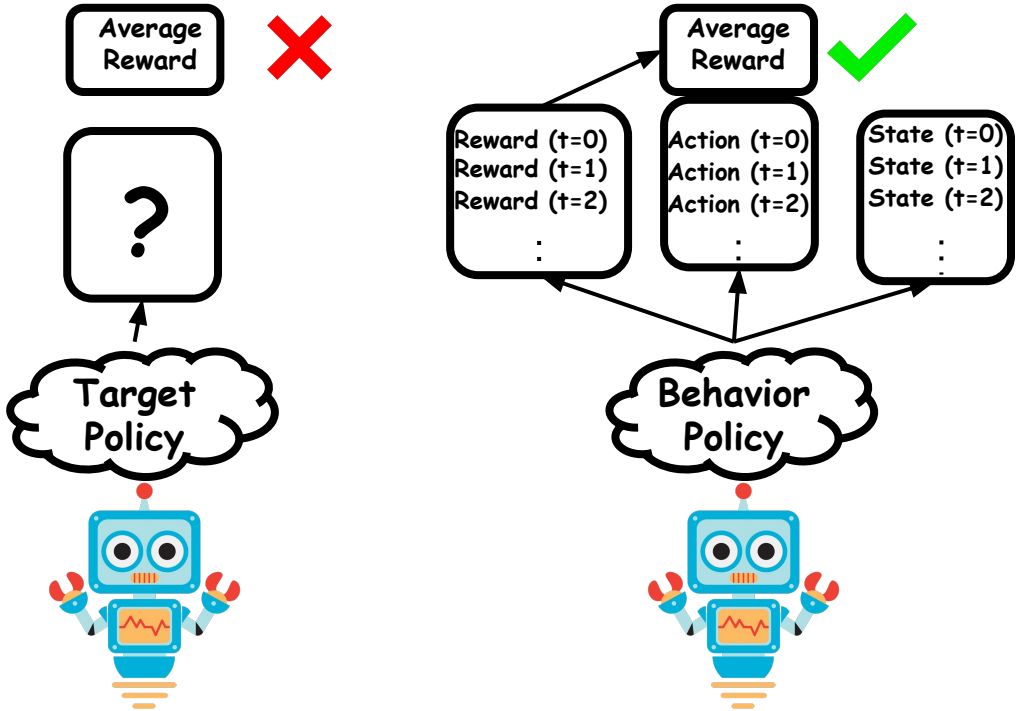
- **Existence of confounders:**
  - We might not observe how they affect decision making process.

Reinforcement Learning



Off-Policy Estimation

- **Goal:** Estimating the average reward of a target policy, given data from behavior policies.



Off-Policy Estimation

- **Applications:** In general, where high-fidelity simulators may not be available and on-policy evaluation is expensive or impossible.
  - Healthcare
  - Robotics
  - Recommendation Systems

Off-Policy Estimation

- **Applications:** In general, where high-fidelity simulators may not be available and on-policy evaluation is expensive or impossible.
  - Healthcare
  - Robotics
  - Recommendation Systems
- **Our work:** A new approach that:
  - *avoids the curse of horizon problem.*
    - Unlike traditional methods that have variance explosion in the long horizon.
  - *does not need the knowledge of the behavior policy.*
    - Unlike “Breaking the curse of Horizon” paper in NeurIPS 2018.
  - *does not need the behavior policy data to reach stationary distribution.*
    - Unlike “Breaking the curse of Horizon” paper in NeurIPS 2018.
  - *works for both discounted and undiscounted reward criteria.*
    - Unlike “DualDICE” paper in NeurIPS 2019.

Problem Statement

- Standard MDP formulation: $M = \langle \mathcal{S}, \mathcal{A}, P, R, p_0, \gamma \rangle$
- Data trajectory: $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$

$$s_0 \sim p_0(\cdot) \qquad a_t \sim \pi(\cdot|s_t) \qquad r_t = R(s_t, a_t) \qquad s_{t+1} \sim P(\cdot|s_t, a_t)$$

- Reward of a policy:

$$\rho_\pi := \lim_{T \to \infty} \mathbb{E}_\pi \left[\frac{1}{T} \sum_{t=1}^{T} r_t \right] = \mathbb{E}_{(s,a) \sim d_\pi}[r]$$

- **Problem**: estimate the reward of a policy given data from another policy

$$\mathcal{D} := \{(s_i, a_i, r_i, s_i')\}_{1 \leq i \leq n}$$

Key Element for Black-box Estimation

- **Backward Operator:**
  - Defined on functions over *states and actions*.
  - The discrete version

$$\mathcal{B}_\pi d(s,a) := \pi(a|s) \sum_{\xi \in \mathcal{S}, \alpha \in \mathcal{A}} P(s|\xi,\alpha) d(\xi,\alpha)$$

  - The continuous version

$$\mathcal{B}_\pi d(s,a) = \pi(a|s) \int_{\xi,\alpha} \mathrm{d}P(s|\xi,\alpha) d(\xi,\alpha)$$

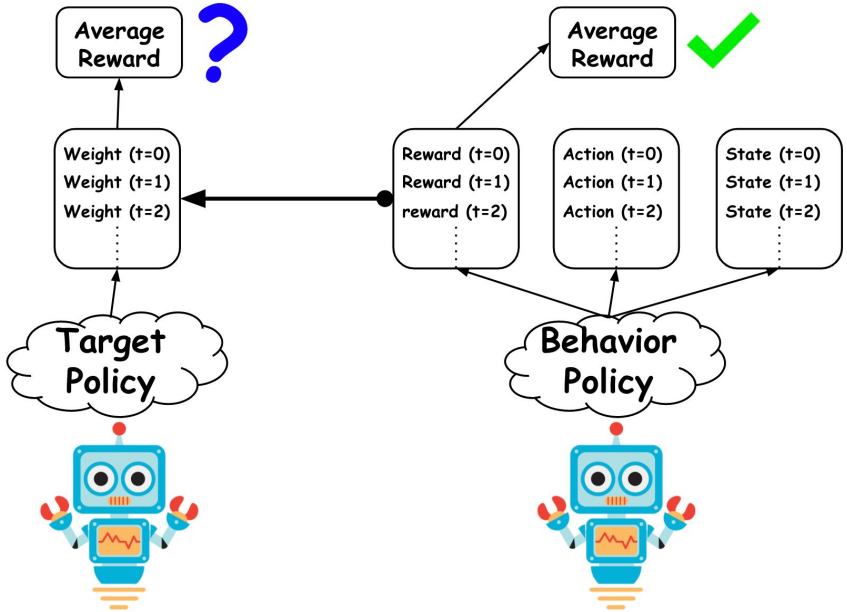
  - **Fixed Point Property**: The stationary distribution of the policy is the unique fixed point of the Backward Operator:

$$d_\pi = \mathcal{B}_\pi d_\pi$$

Black-box Estimator

- Off-policy reward as a weighted average:

$$\hat{\rho}_\pi = \sum_{i=1}^{n} w_i r_i$$



Black-box Estimator

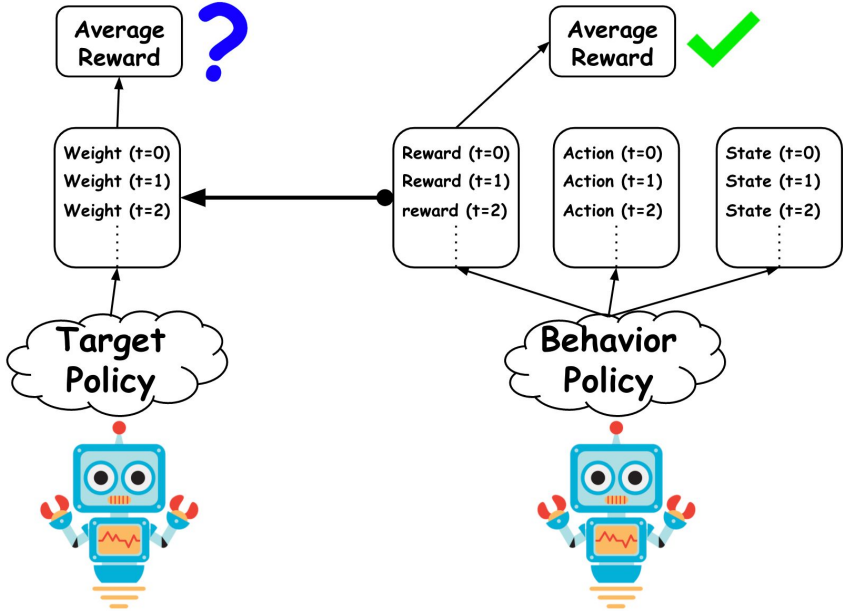
- Off-policy reward as a weighted average:

$$\hat{\rho}_\pi = \sum_{i=1}^{n} w_i r_i$$

- Empirical Distribution:

$$d_w(s,a) := \sum_{i=1}^{n} w_i \mathbf{1}\{s_i = s, a_i = a\}$$

$$\begin{cases} w_i = \tilde{w}_i / \sum_l \tilde{w}_l \\ \tilde{w}_i := W(s_i, a_i; \omega) \geq 0 \end{cases}$$



Black-box Estimator

- Empirical vs. True Distribution:

$$\hat{\rho}_\pi = \sum_{i=1}^{n} w_i r_i$$

$$d_w(s,a) := \sum_{i=1}^{n} w_i \mathbf{1}\{s_i = s, a_i = a\}$$

$$\hat{\rho}_\pi = \mathbb{E}_{(s,a)\sim d_w}[r]$$

$$\pi : \mathcal{S} \mapsto \Delta(\mathcal{A})$$

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$$

$$\rho_\pi := \lim_{T\to\infty} \mathbb{E}_\pi \left[\frac{1}{T}\sum_{t=1}^{T} r_t\right] = \mathbb{E}_{(s,a)\sim d_\pi}[r]$$

Black-box Estimator

- Empirical vs. True Distribution:

$$\hat{\rho}_\pi = \sum_{i=1}^{n} w_i r_i$$

$$d_w(s,a) := \sum_{i=1}^{n} w_i \mathbf{1}\{s_i = s, a_i = a\}$$

$$\hat{\rho}_\pi = \mathbb{E}_{(s,a)\sim d_w}[r]$$

$$\pi : \mathcal{S} \mapsto \Delta(\mathcal{A})$$

$$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$$

$$\rho_\pi := \lim_{T\to\infty} \mathbb{E}_\pi \left[\frac{1}{T}\sum_{t=1}^{T} r_t\right] = \mathbb{E}_{(s,a)\sim d_\pi}[r]$$

- Optimization Problem:

$$\min_{\omega\in\Omega} D(d_w \parallel \mathcal{B}_\pi d_w)$$

Maximum Mean Discrepancy (MMD)

- Strictly integrally positive definite kernel function:

$$\mathbf{k}\left[f;\ f\right] := \sum_{(s,a)\in\mathcal{S}\times\mathcal{A},(\bar{s},\bar{a})\in\mathcal{S}\times\mathcal{A}} f(s,a)k((s,a),(\bar{s},\bar{a}))f(\bar{s},\bar{a}) > 0$$

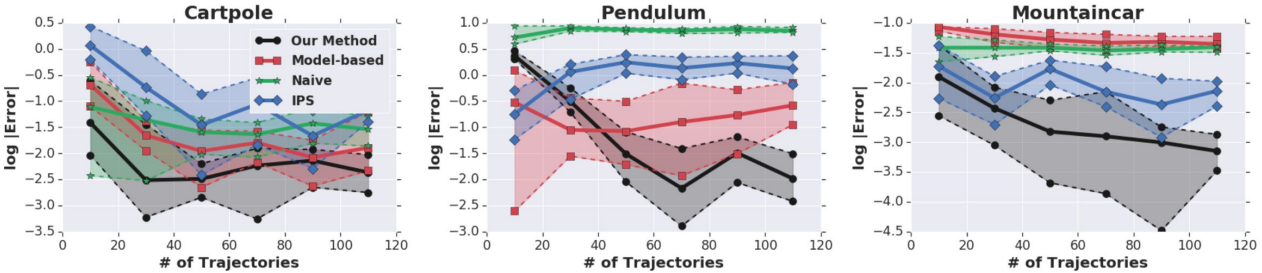
- MMD avoids density estimation and uses **mean** directly:

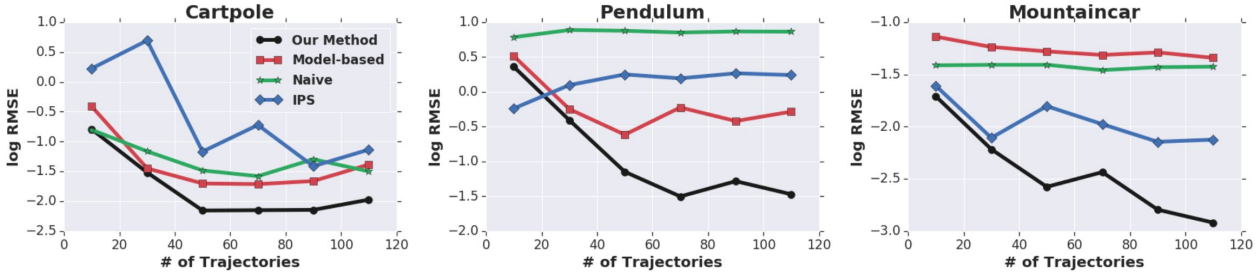
$$\begin{aligned} D(d_w \parallel \mathcal{B}_\pi d_w) &= \sup_{f\in\mathcal{H}} \{\mathbb{E}_{d_w}[f] - \mathbb{E}_{\mathcal{B}_\pi d_w}[f] \quad s.t. \quad \|f\|_{\mathcal{H}} \leq 1\} \\ &= \mathbf{k}\left[d_w;\ d_w\right] - 2\mathbf{k}\left[d_w;\ \mathcal{B}_\pi d_w\right] + \mathbf{k}\left[\mathcal{B}_\pi d_w;\ \mathcal{B}_\pi d_w\right] \end{aligned}$$

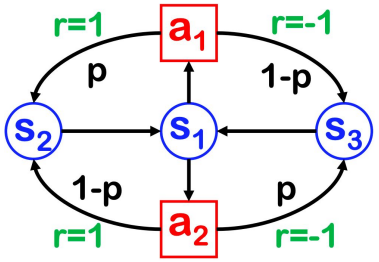
- Simplified loss function:

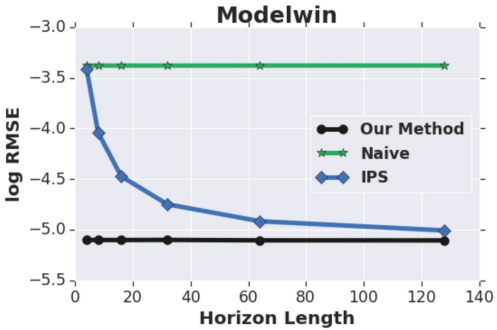
$$\ell(\omega) = \sum_{i,j} W(s_i, a_i; \omega) W(s_j, a_j; \omega) K_{i,j}$$

Experimental Results

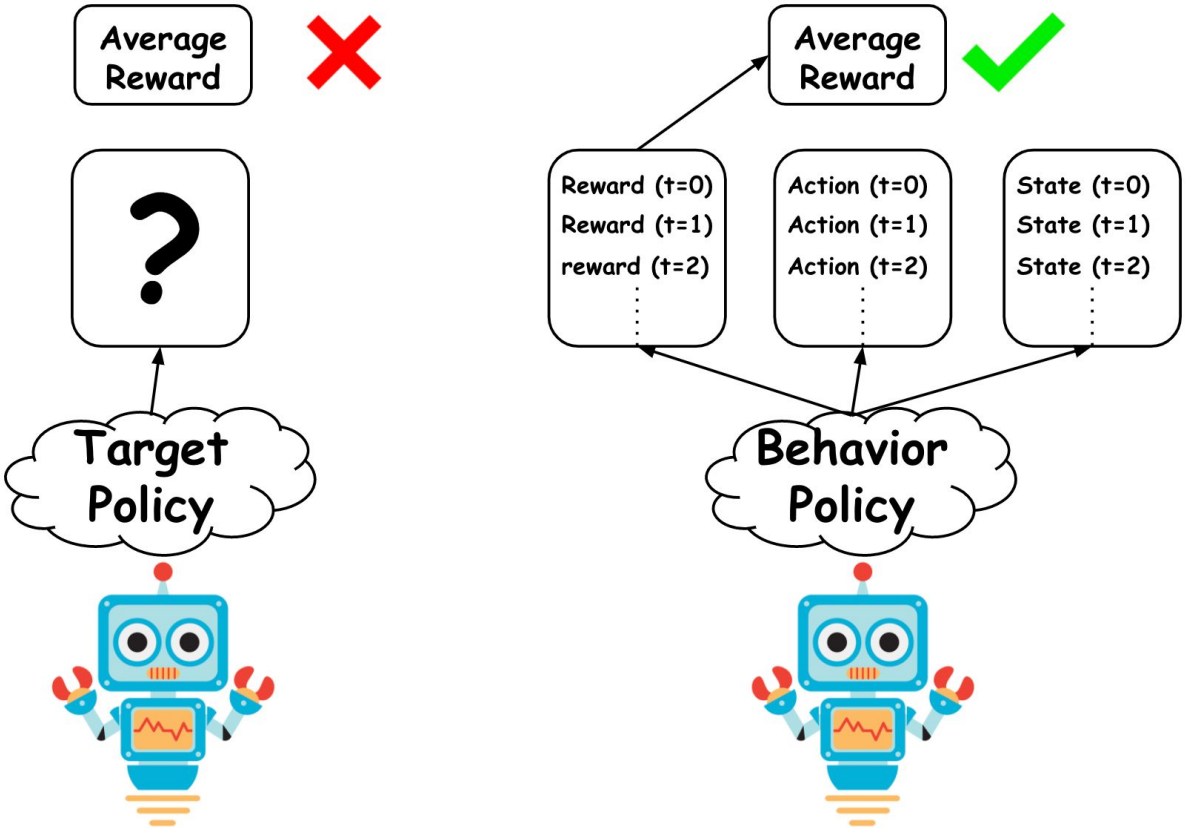




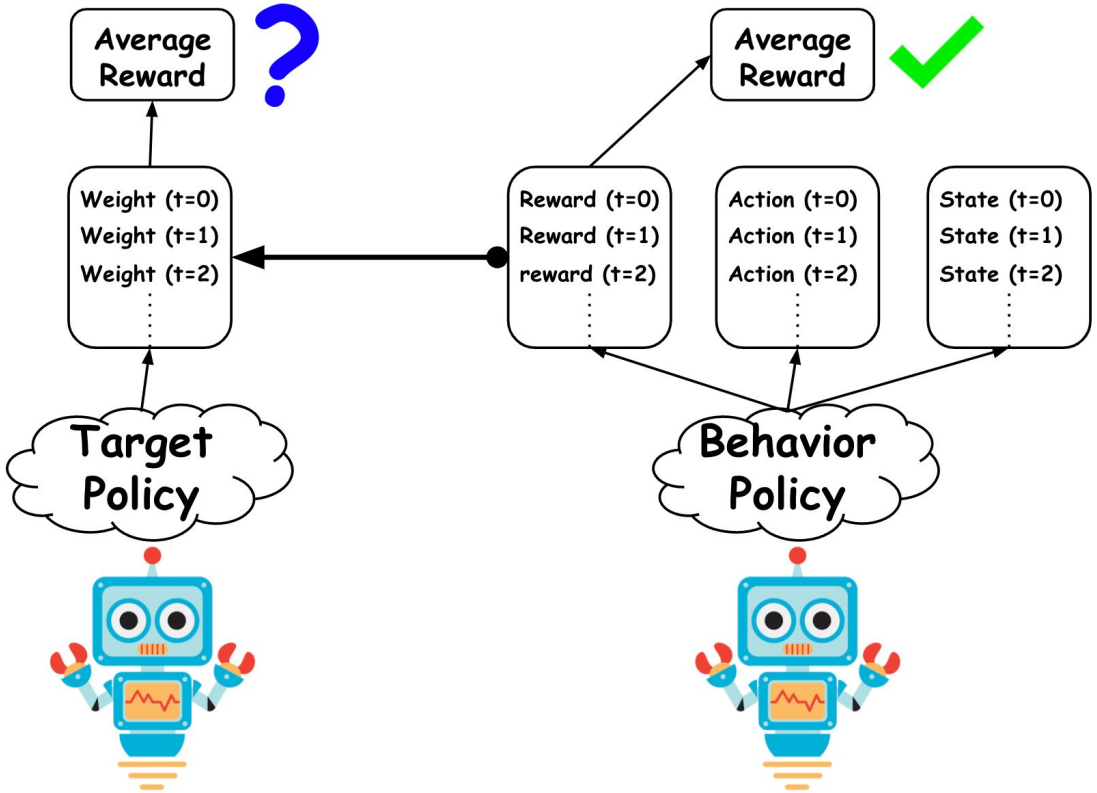




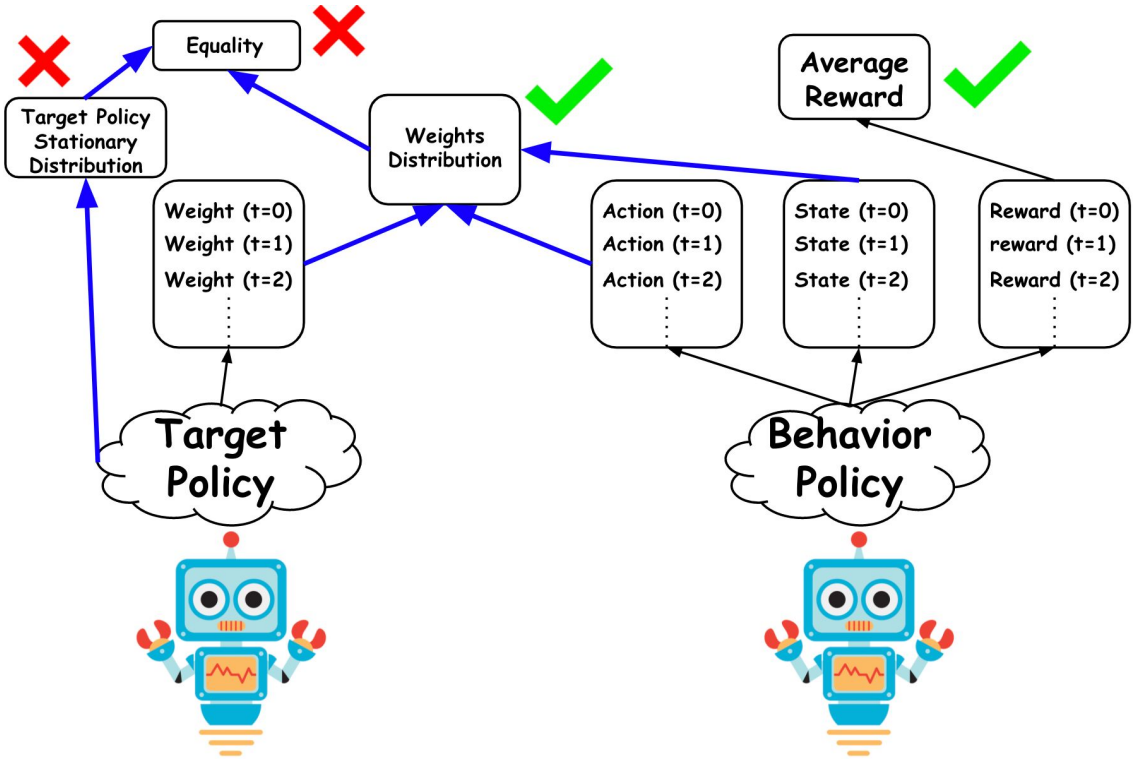
Summary


Average Reward
?
Target Policy
Average Reward
Reward (t=0)
Reward (t=1)
reward (t=2)
Action (t=0)
Action (t=1)
Action (t=2)
State (t=0)
State (t=1)
State (t=2)
Behavior Policy

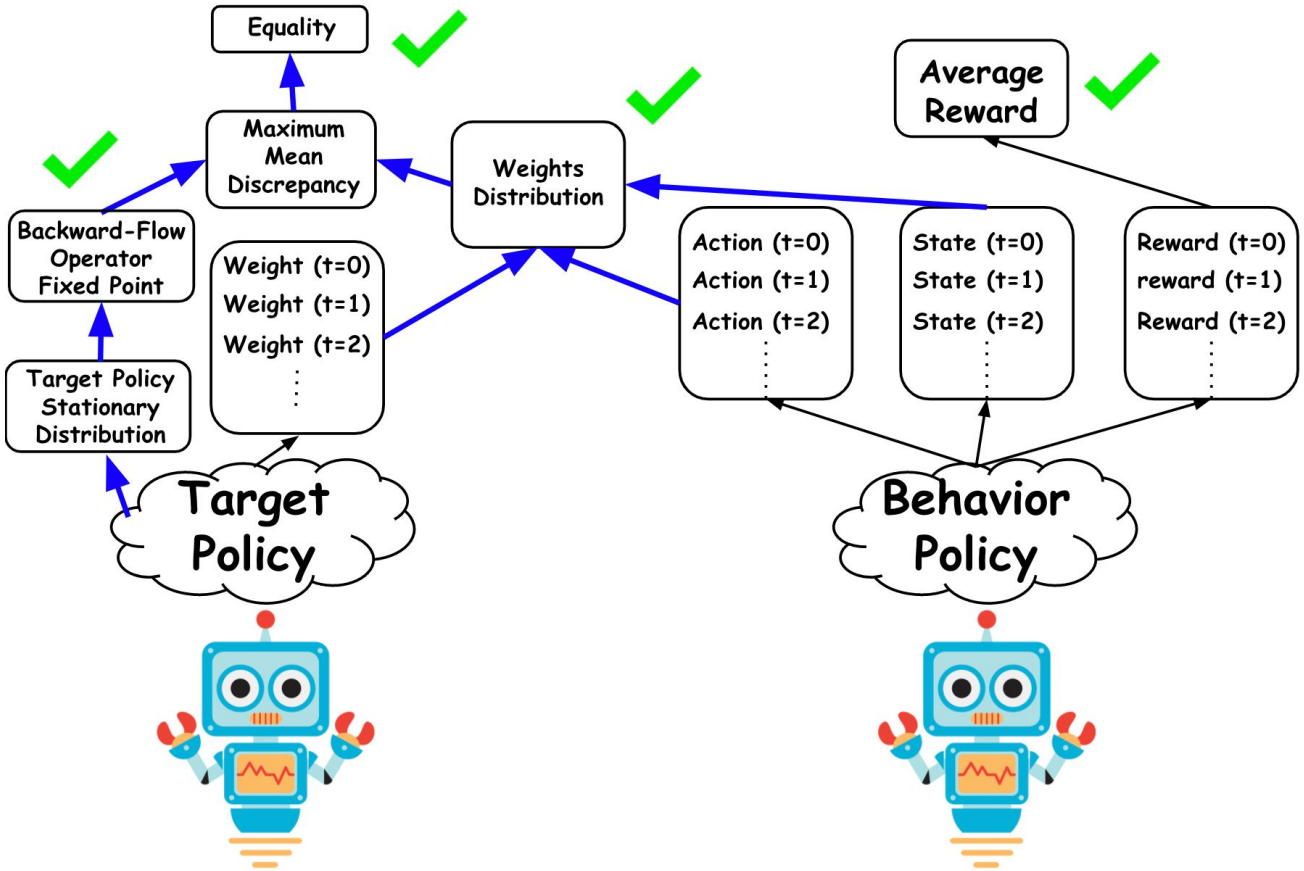
Summary


Average Reward
Average Reward
Weight (t=0)
Weight (t=1)
Weight (t=2)
Reward (t=0)
Reward (t=1)
reward (t=2)
Action (t=0)
Action (t=1)
Action (t=2)
State (t=0)
State (t=1)
State (t=2)
Target Policy
Behavior Policy

Summary


Equality
Target Policy
Stationary
Distribution
Weights
Distribution
Average
Reward
Weight (t=0)
Weight (t=1)
Weight (t=2)
Action (t=0)
Action (t=1)
Action (t=2)
State (t=0)
State (t=1)
State (t=2)
Reward (t=0)
reward (t=1)
Reward (t=2)
Target
Policy
Behavior
Policy

Summary


Equality
Maximum Mean Discrepancy
Backward-Flow Operator Fixed Point
Target Policy Stationary Distribution
Weight (t=0)
Weight (t=1)
Weight (t=2)
Weights Distribution
Average Reward
Action (t=0)
Action (t=1)
Action (t=2)
State (t=0)
State (t=1)
State (t=2)
Reward (t=0)
reward (t=1)
Reward (t=2)
Target Policy
Behavior Policy

Decision Making Challenges

- **Dealing with a large output space:**
  - Having to choose from many class labels or possible actions.

- **Online experiments may not be feasible:**
  - A/B testing new policies might be expensive or even impossible.

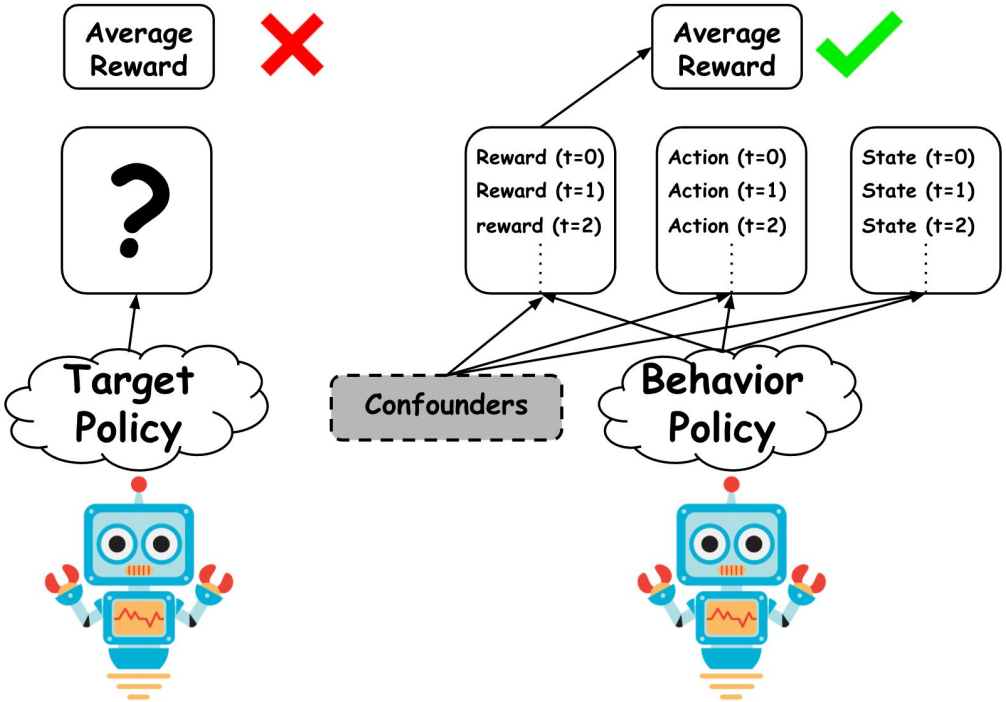
- **Existence of confounders:**
  - We might not observe how they affect decision making process.

(ICLR 2020, Causal Learning for Decision Making Workshop)

(Submitted to NeurIPS 2020)

Off-Policy Estimation

- **Goal:** Estimating the average reward of a target policy, given data from behavior policies in the presence of confounders.

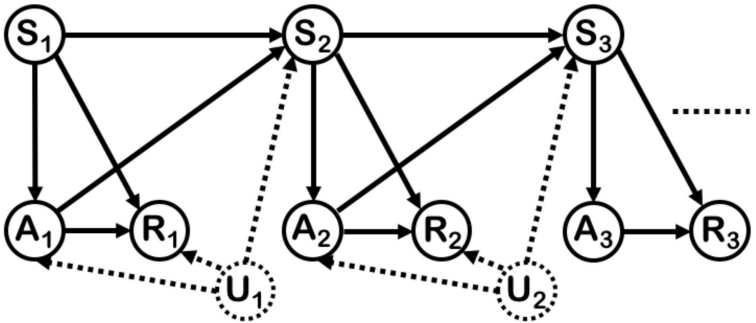


Model

- **Model:** MDP with unobserved confounders affecting action selection, state transition, and reward values.

- **Assumption:** Confounders are IID.

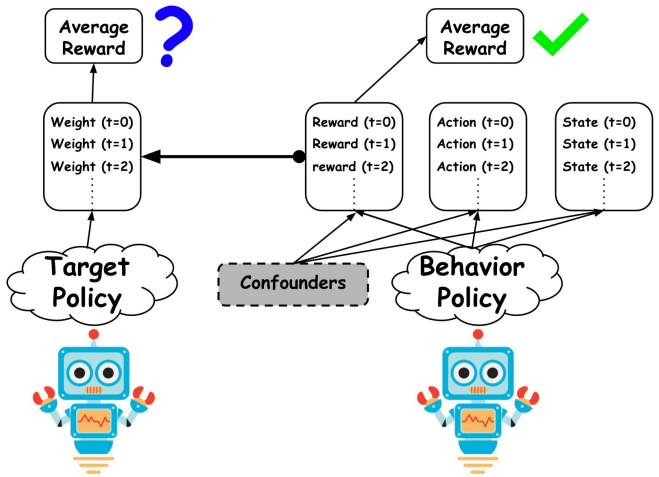
- **Assumption:** We have a posterior oracle to sample confounders.
  - Example: *"Causal Effect Inference with Deep Latent-Variable Models"* NeurIPS 2017



Balancing Estimator

- Off-policy reward as a weighted average:

$$\hat{\rho}_\pi = \sum_{i=1}^{n} w_i r_i$$

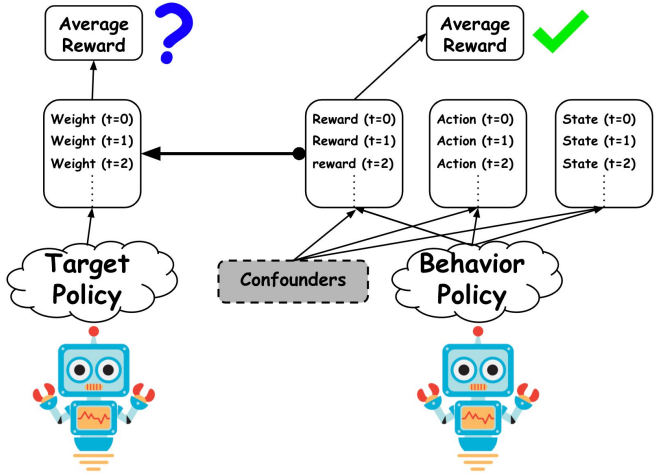


Balancing Estimator

- Off-policy reward as a weighted average:

$$\hat{\rho}_\pi = \sum_{i=1}^{n} w_i r_i$$

- Expected Reward: $\mu_a(s,u) = \mathbb{E}[\mathcal{R}(s,a,u)]$

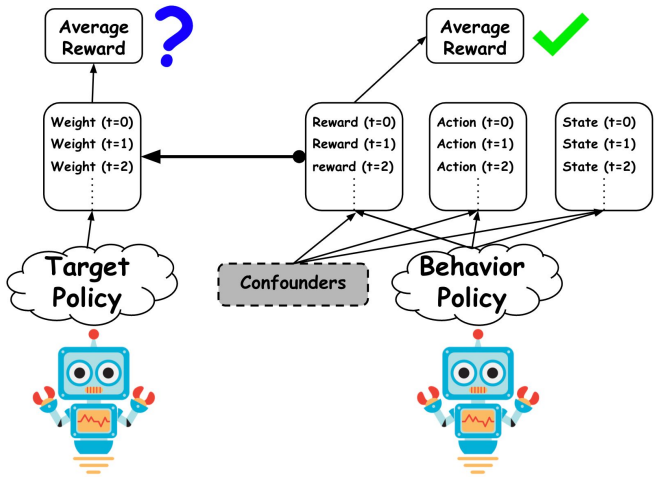


Balancing Estimator

- Off-policy reward as a weighted average:

$$\hat{\rho}_\pi = \sum_{i=1}^{n} w_i r_i$$

- Expected Reward: $\mu_a(s,u) = \mathbb{E}[\mathcal{R}(s,a,u)]$



Target Policy Value

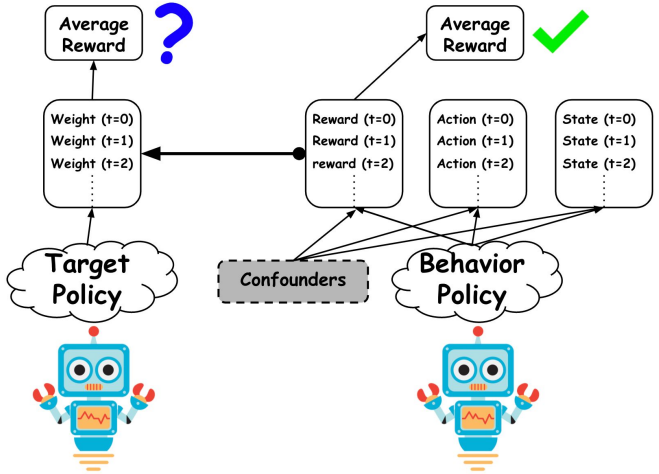
$$\begin{aligned} v(\pi_e) &= \mathbb{E}_e[\mu_A(S,U)] \\ &= \sum_{a=1}^{m} \mathbb{E}_e[\pi_e(a \mid S,U)\mu_a(S,U)] \\ &= \sum_{a=1}^{m} \mathbb{E}_b[d(S)\pi_e(a \mid S,U)\mu_a(S,U)] \end{aligned}$$

Balancing Estimator

- Off-policy reward as a weighted average:

$$\hat{\rho}_\pi = \sum_{i=1}^{n} w_i r_i$$

- Expected Reward: $\mu_a(s,u) = \mathbb{E}[\mathcal{R}(s,a,u)]$



Target Policy Value

$$\begin{aligned} v(\pi_e) &= \mathbb{E}_e[\mu_A(S,U)] \\ &= \sum_{a=1}^{m} \mathbb{E}_e[\pi_e(a \mid S,U)\mu_a(S,U)] \\ &= \sum_{a=1}^{m} \mathbb{E}_b[d(S)\pi_e(a \mid S,U)\mu_a(S,U)] \end{aligned}$$

Estimated Target Policy Value

$$\begin{aligned} \mathbb{E}_b[WR] &= \mathbb{E}_b[W\mu_A(S,U)] \\ &= \sum_{a=1}^{m} \mathbb{E}_b[W\delta_{Aa}\mu_a(S,U)] \end{aligned}$$

Balancing Estimator

Target Policy Value

$$
\begin{aligned}
v(\pi_e) &= \mathbb{E}_e[\mu_A(S,U)] \\
&= \sum_{a=1}^{m} \mathbb{E}_e[\pi_e(a \mid S,U)\mu_a(S,U)] \\
&= \sum_{a=1}^{m} \mathbb{E}_b[d(S)\pi_e(a \mid S,U)\mu_a(S,U)]
\end{aligned}
$$

Estimated Target Policy Value

$$
\begin{aligned}
\mathbb{E}_b[WR] &= \mathbb{E}_b[W\mu_A(S,U)] \\
&= \sum_{a=1}^{m} \mathbb{E}_b[W\delta_{Aa}\mu_a(S,U)]
\end{aligned}
$$

Approximate Bias

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{a=1}^{m} f_{ia}\mu_a(S_i,U_i)$$

$$f_{ia} = W_i\delta_{A_i a} - d(S_i)\pi_e(a \mid S_i,U_i)$$

Upper-bounding Variance

$$\frac{\|W\|_2^2}{n^2}$$

Balancing Estimator

- Bias squared + Upper-bound Variance squared:

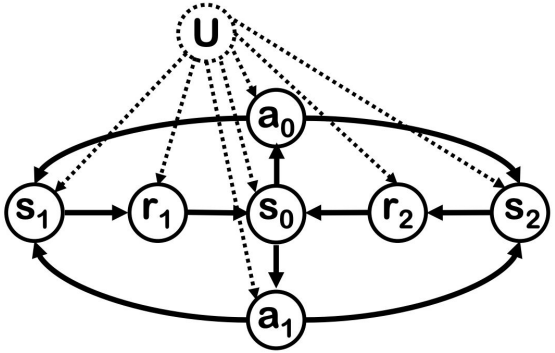
$$B(W,g) = \frac{1}{n}\sum_{i=1}^{n}\sum_{a=1}^{m}\mathbb{E}[f_{ia}g_a(S_i,U_i) \mid Z_i]$$

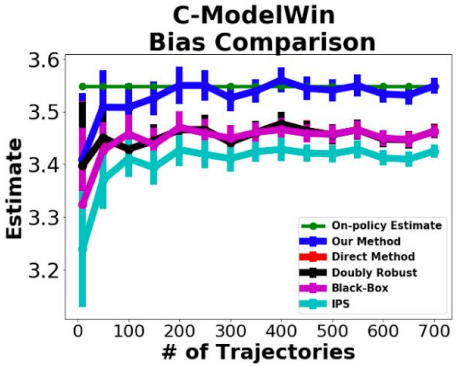
$$J_\lambda(W,g) = B(W,g)^2 + \frac{\lambda}{n^2}\|W\|^2 .$$

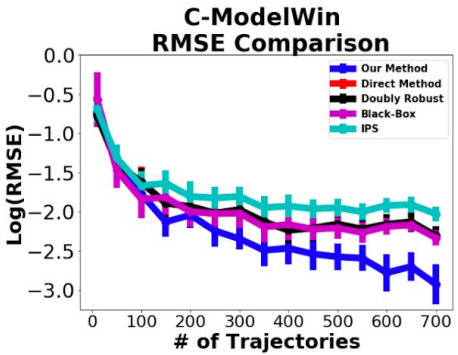
Theorem: $\inf_{W\in\mathbb{R}^n}\sup_{g\in\mathcal{G}} J_\lambda(W,g) = O_p(1/n)$

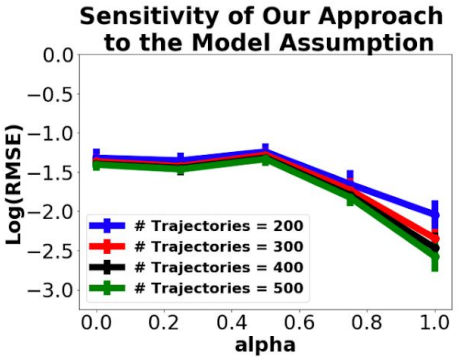
Theorem: If $J_\lambda(W,\mu) = O_p(1/n)$ then $\hat{\tau}_W = v(\pi_e) + O_p(1/\sqrt{n})$

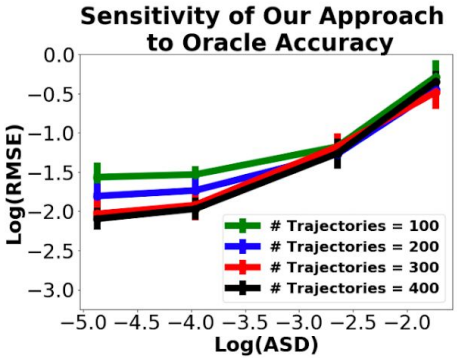
Experimental Results - Confounded ModelWin



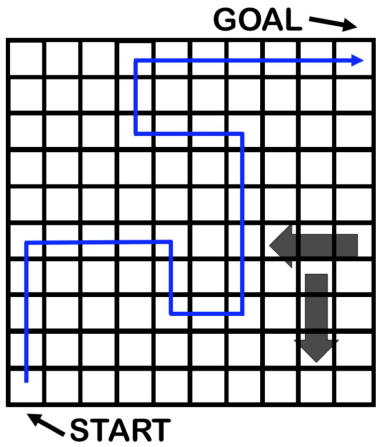


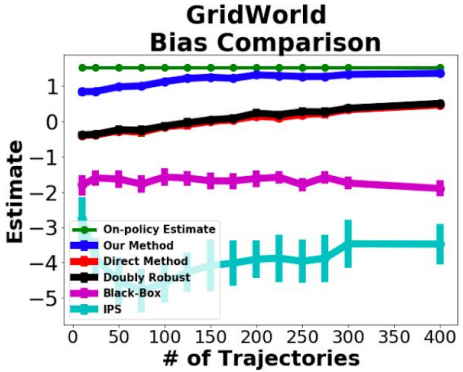


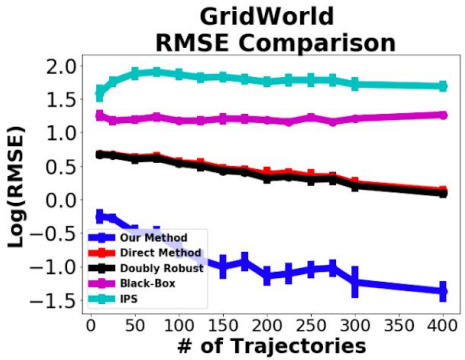


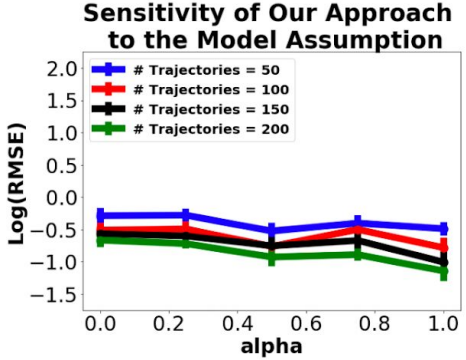


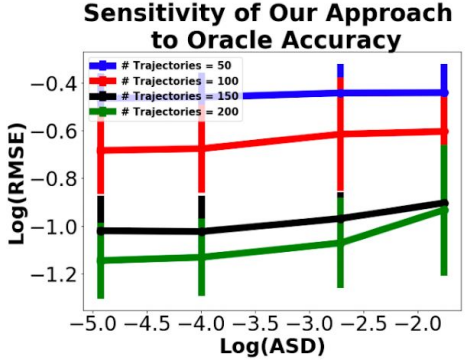
Experimental Results - Windy GridWorld











Summary

- **Dealing with a large output space:**
  - Embedding based classifiers with proper regularization methods.

- **Online experiments may not be feasible:**
  - Black-box off-policy estimation.

- **Existence of confounders:**
  - Balancing estimator.

Future Directions

- **How can we learn and leverage structural domain knowledge to enhance generalization and sample complexity in Reinforcement Learning?**

  - How can we define and quantify generalization?
  - How can the agent learn causal graph structures and infer invariances from them?
  - How to directly do policy optimization when confounders exist in an offline setting?
  - How to efficiently learn latent variable models for confounders?

References

- Breaking the Glass Ceiling for Embedding-Based Classifiers for Large Output Spaces
- Black-box Off-policy Estimation for Infinite-Horizon Reinforcement Learning
- Off-policy Evaluation in Infinite-Horizon Reinforcement Learning with Latent Confounders