Approximate Policy Iteration

Recap: Policy Iteration

Recall Policy Iteration (PI) for the setting where P and r are known:

We compute $Q^\pi(s,a)$ exactly for all s, a, PI updates policy as:

$$\pi'(s) = \arg\max_a Q^\pi(s,a)$$

i.e., be greedy with respect to π at every state s,

Monotonic improvement of PI: $Q^{\pi'}(s,a) \geq Q^\pi(s,a), \forall s, a$

Recap: Policy Iteration

Recall Policy Iteration (PI) for the setting where P and r are known:

We compute $Q^\pi(s,a)$ exactly for all s, a, PI updates policy as:

$$\pi'(s) = \arg\max_a Q^\pi(s,a)$$

i.e., be greedy with respect to π at every state s,

Monotonic improvement of PI: $Q^{\pi'}(s,a) \geq Q^\pi(s,a), \forall s, a$

What if P & r are unknown, and MDP is large (e.g., infinitely many states)?

Recap: Model-based RL

Simulation Lemma:

$$\widehat{V}^{\pi}(s_0) - V^{\pi}(s_0) = \frac{\gamma}{1-\gamma}\mathbb{E}_{s,a\sim d^{\pi}_{s_0}}\left[\mathbb{E}_{s'\sim \widehat{P}(s,a)}\widehat{V}^{\pi}(s') - \mathbb{E}_{s'\sim P(s,a)}\widehat{V}^{\pi}(s')\right]$$

$$\leq \frac{\gamma}{(1-\gamma)^2}\mathbb{E}_{s,a\sim d^{\pi}_{s_0}}\left\| \widehat{P}(\cdot \,|\, s,a) - P(\cdot \,|\, s,a) \right\|_1$$

Recap: Model-based RL

An Algorithm under Generative Model Setting for (small) discrete MDP:

Recap: Model-based RL

An Algorithm under Generative Model Setting for (small) discrete MDP:

1. Model fitting:

$\forall s, a$: collect N next states, $s_i' \sim P(\cdot \mid s, a), i \in [N]$;

set $\widehat{P}(s' \mid s, a) = \dfrac{\sum_{i=1}^{N} \mathbf{1}\{s_i' = s'\}}{N}$;

Recap: Model-based RL

An Algorithm under Generative Model Setting for (small) discrete MDP:

1. Model fitting:

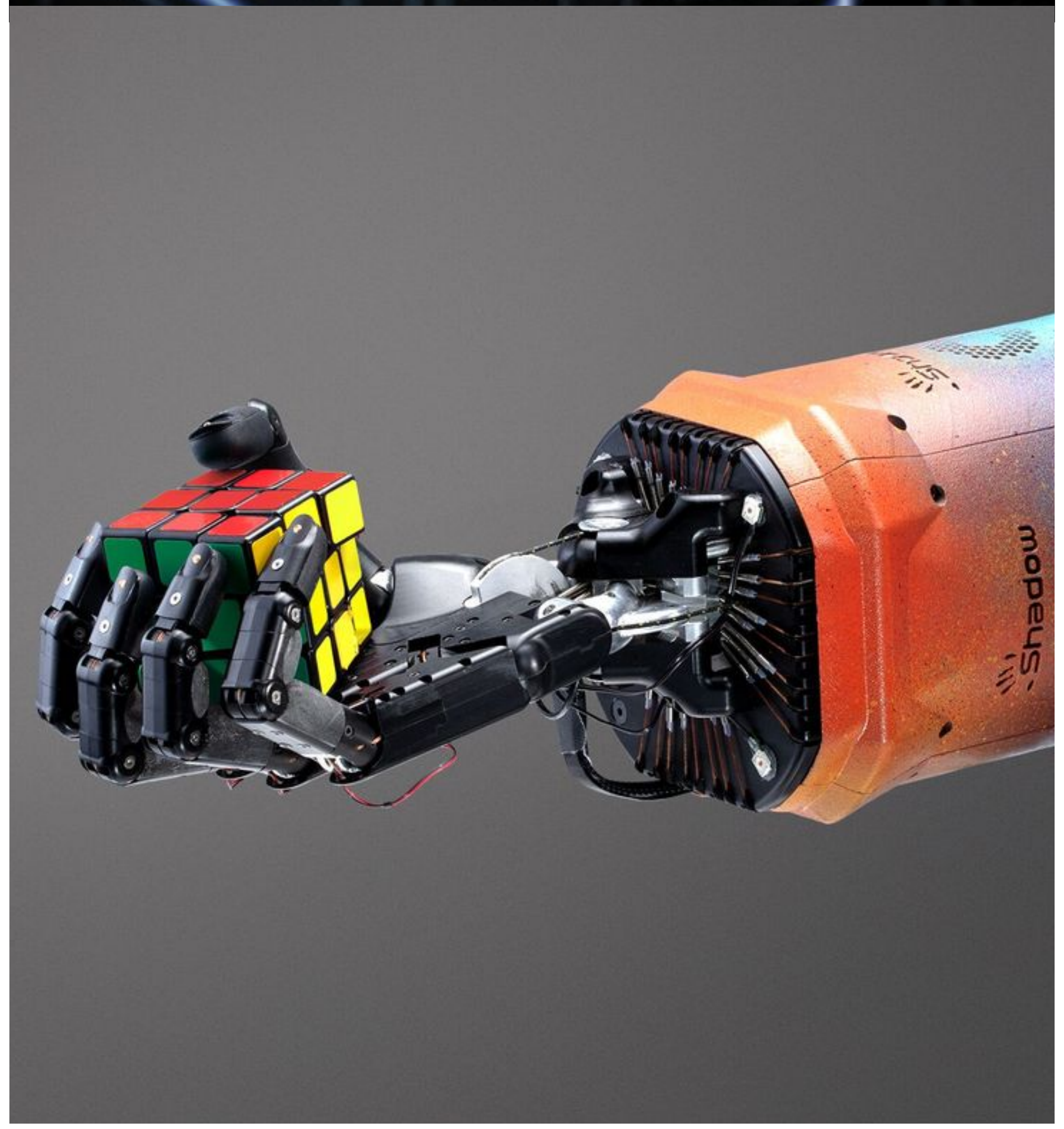
$\forall s, a$: collect N next states, $s_i' \sim P(\cdot \mid s, a), i \in [N]$;

set $\widehat{P}(s' \mid s, a) = \dfrac{\sum_{i=1}^{N} \mathbf{1}\{s_i' = s'\}}{N}$;

2. Planning w/ the learned model:

$\widehat{\pi}^\star = \textbf{PI}\left(\widehat{P}, r\right)$

We are moving on to large scale MDPs

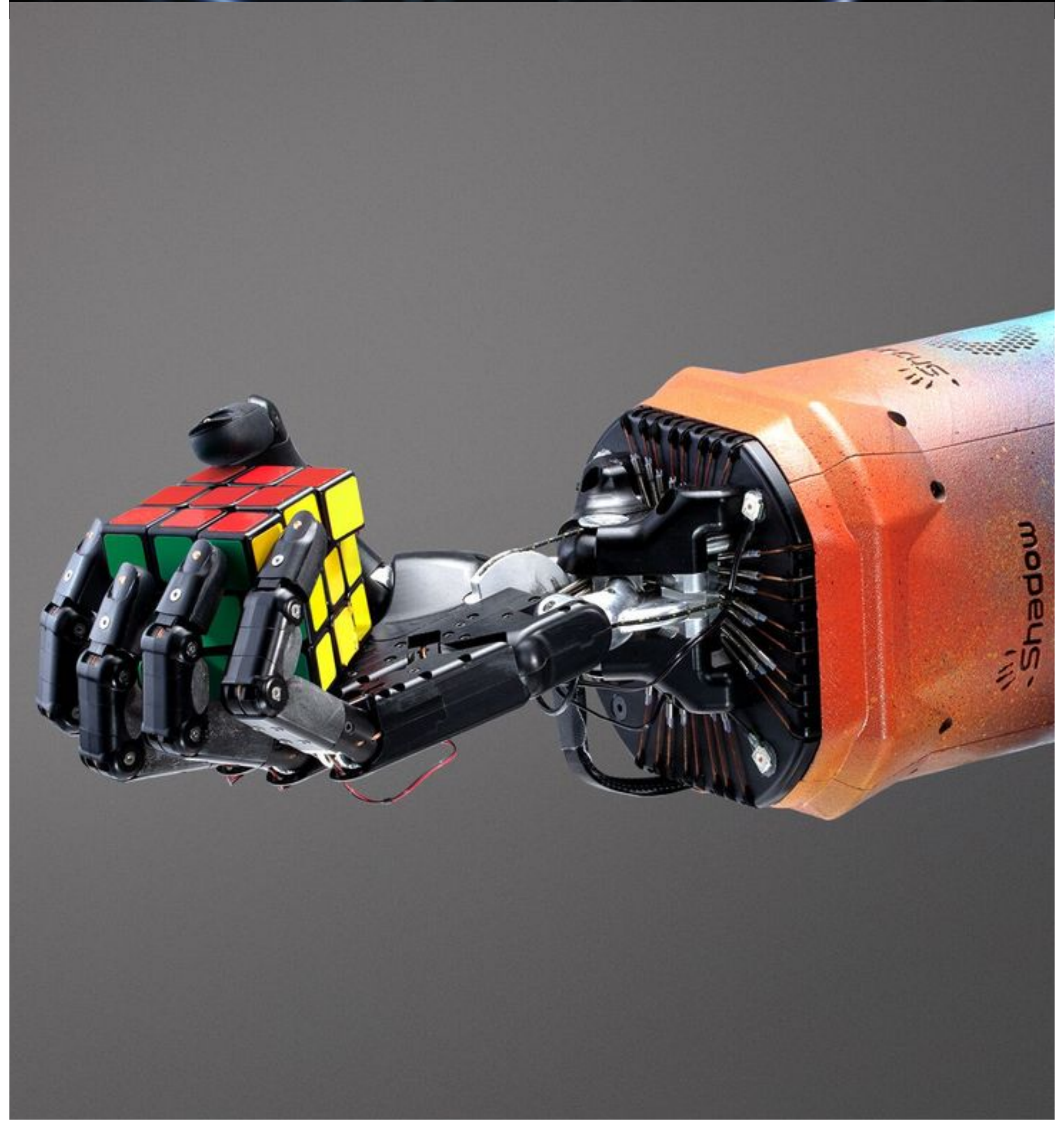


When we face extremely large state space or continuous state space:

Enumerate over all state-action pairs is not possible in both computation, space, and statistics;

What should we do?

We are moving on to large scale MDPs



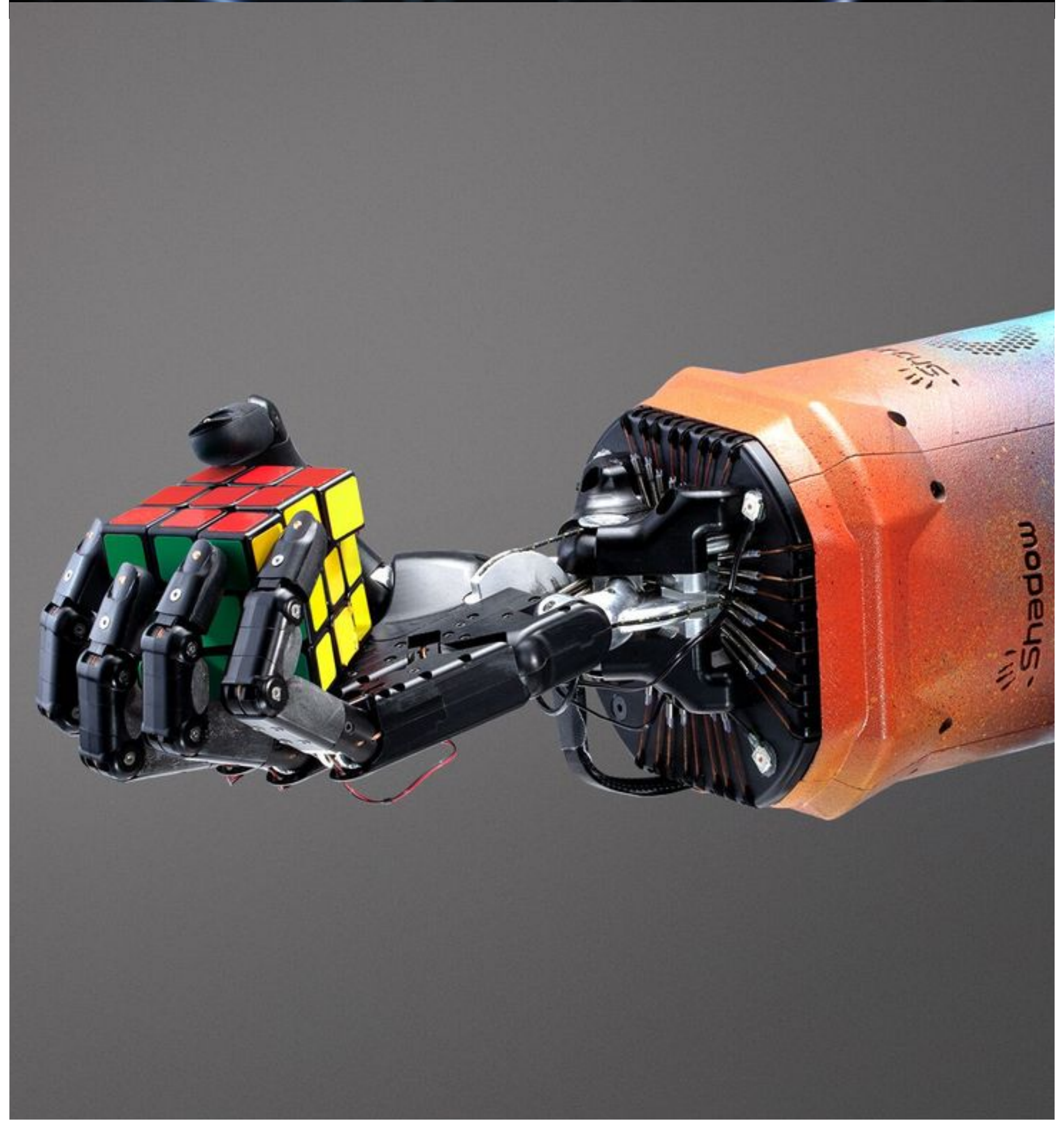
When we face extremely large state space or continuous state space:

Enumerate over all state-action pairs is not possible in both computation, space, and statistics;

What should we do?

Answer: generalization via function approximation (e.g., linear, decision tree, SVM, GP, neural nets)

We are moving on to large scale MDPs



When we face extremely large state space or continuous state space:

Enumerate over all state-action pairs is not possible in both computation, space, and statistics;

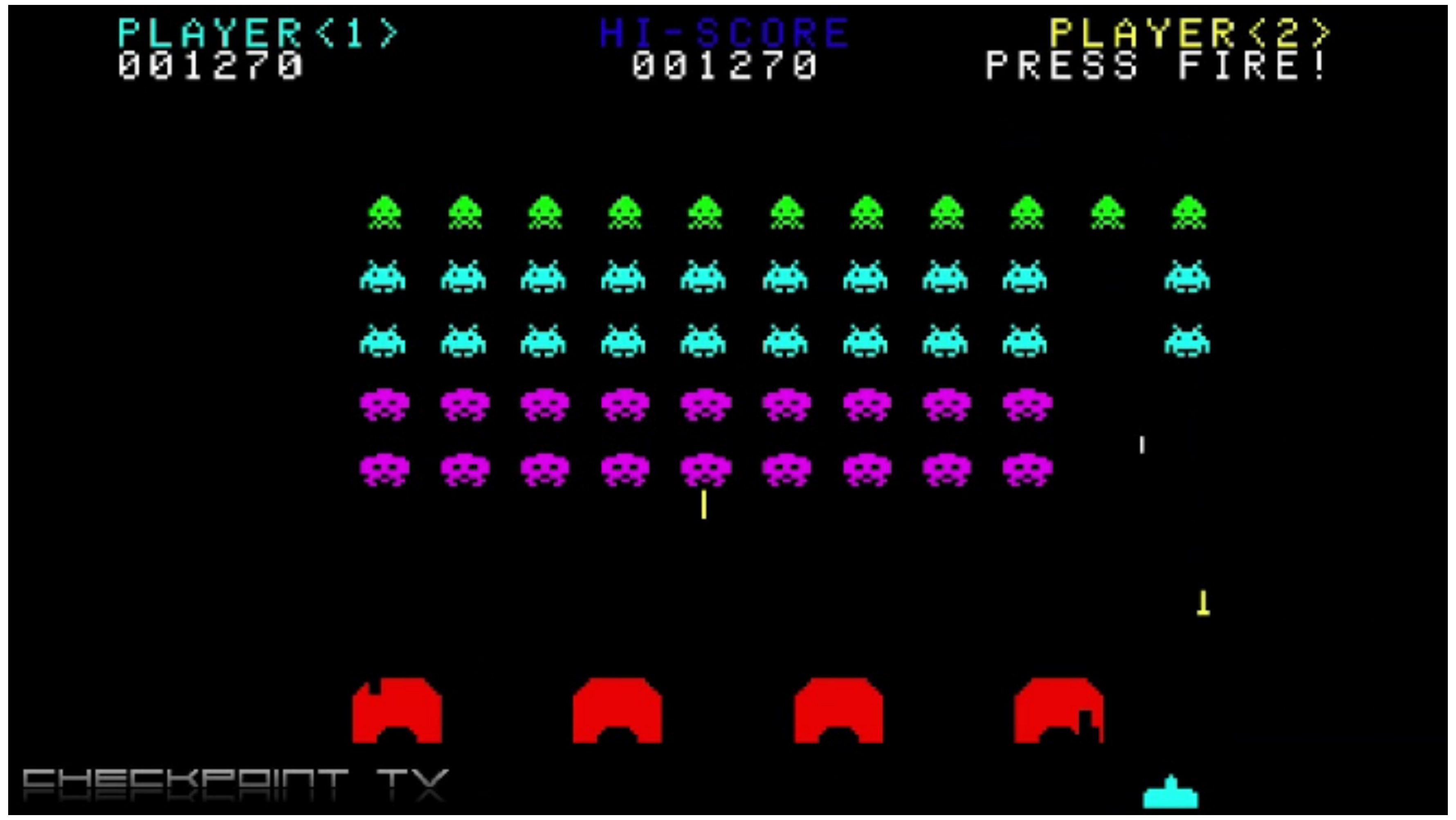
What should we do?

Answer: generalization via function approximation (e.g., linear, decision tree, SVM, GP, neural nets)

Indeed, in LQR, we are using quadratic function to represent Q & V

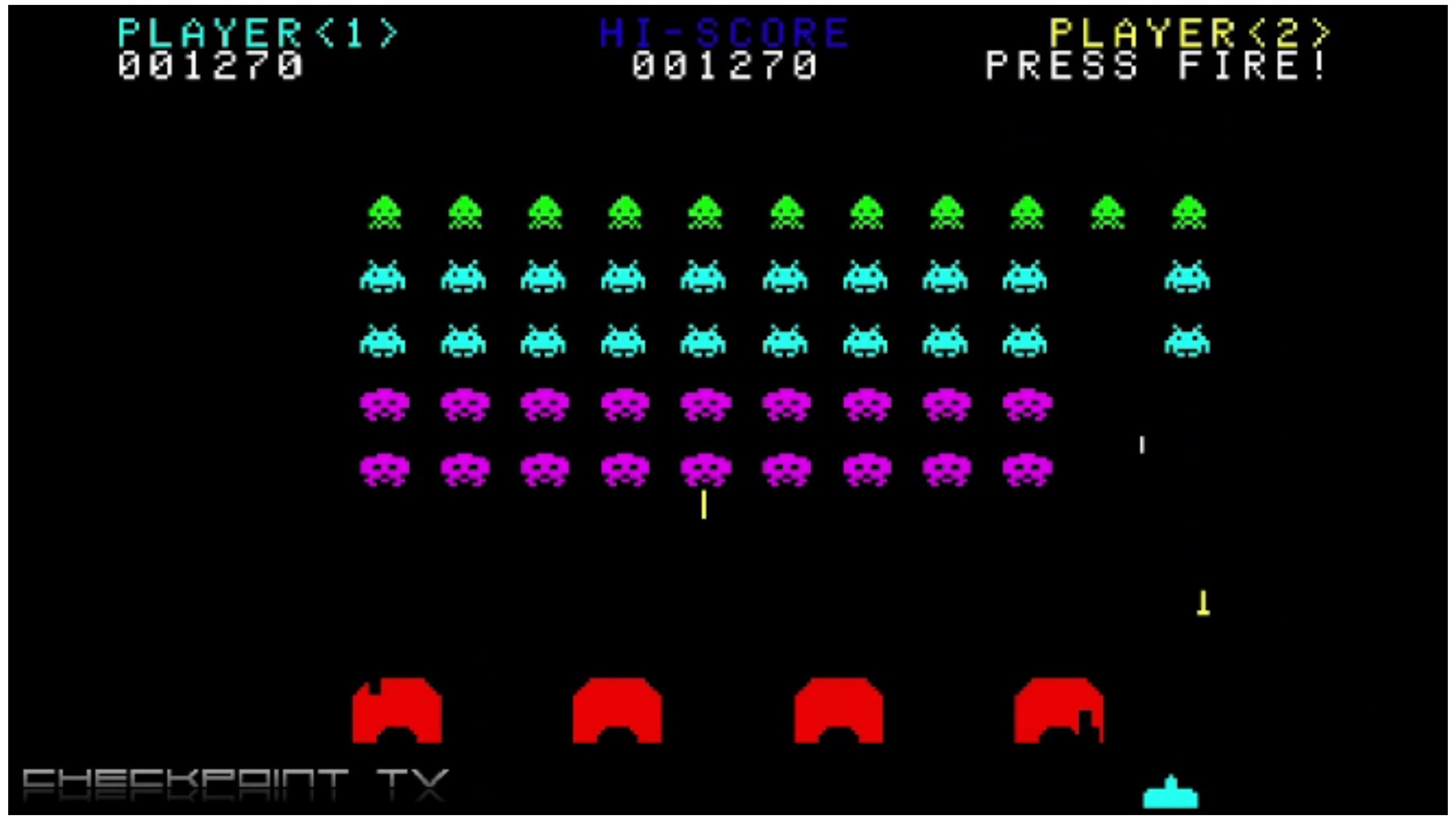
Another example: Video games

State s: RGB image



Another example: Video games

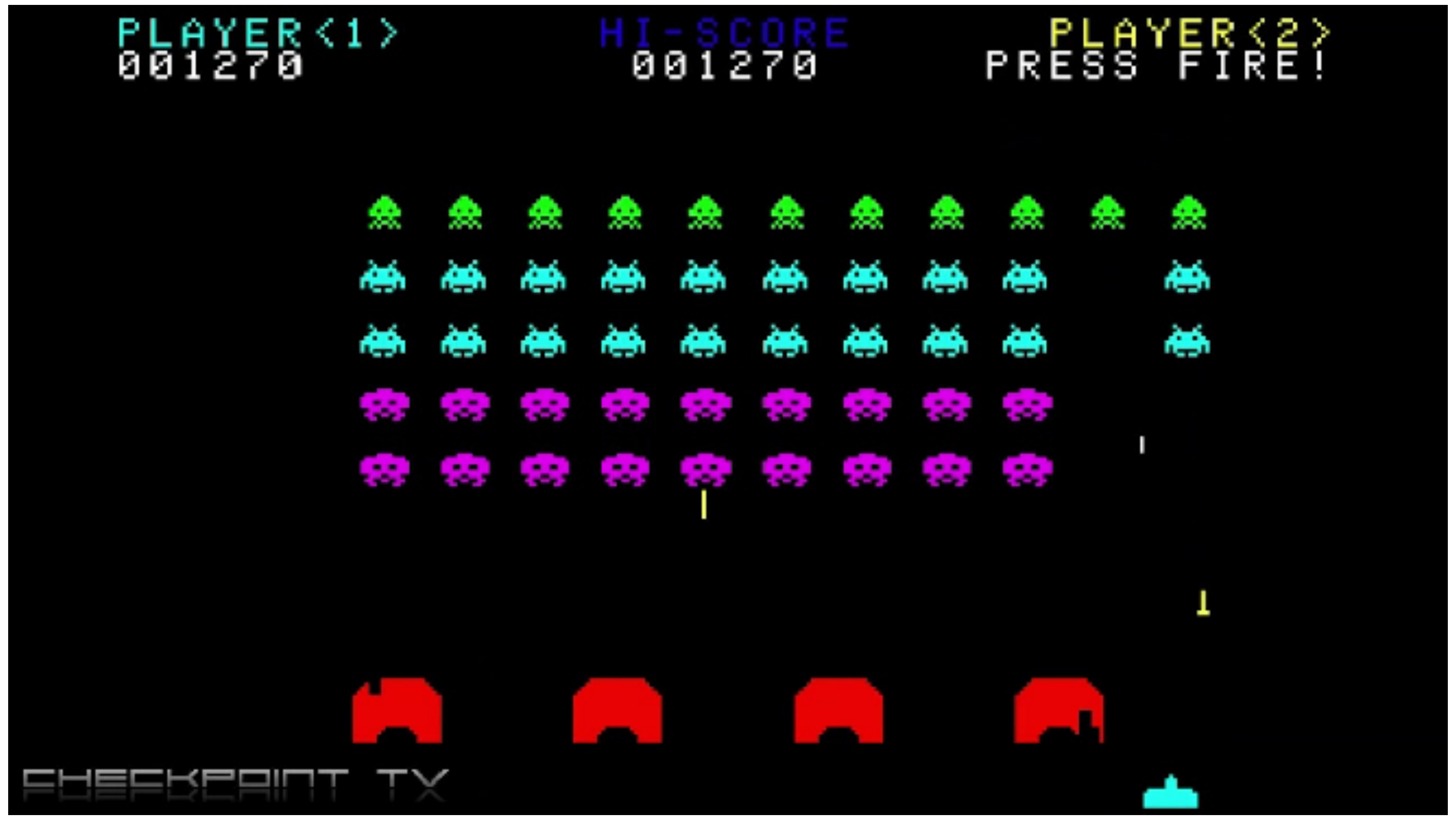
State s: RGB image



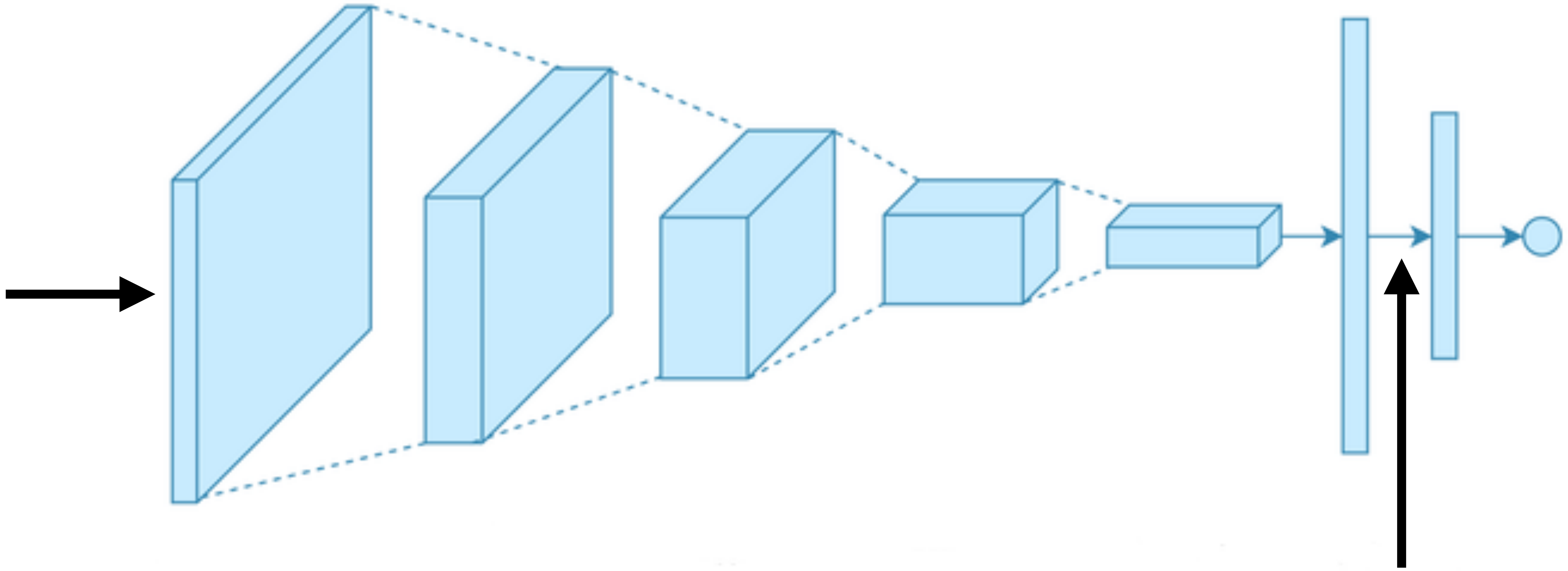
We can try to capture $Q^\star(s, a)$ via deep nets:

Another example: Video games

State s: RGB image



We can try to capture $Q^\star(s, a)$ via deep nets:



Game action a

Question for Today (and the next a few lectures):

How to (approximately) learn $\pi^\star$ using function approximation for large scale MDPs?
(i.e., numeration over state-action is not feasible)

Outline:

1. Quick recap on supervised learning's performance guarantee (classification & regression)

2. Approximate Policy Iteration (relies regression oracle)

Recap on Supervised Learning: Classification

Recap on Supervised Learning: Classification

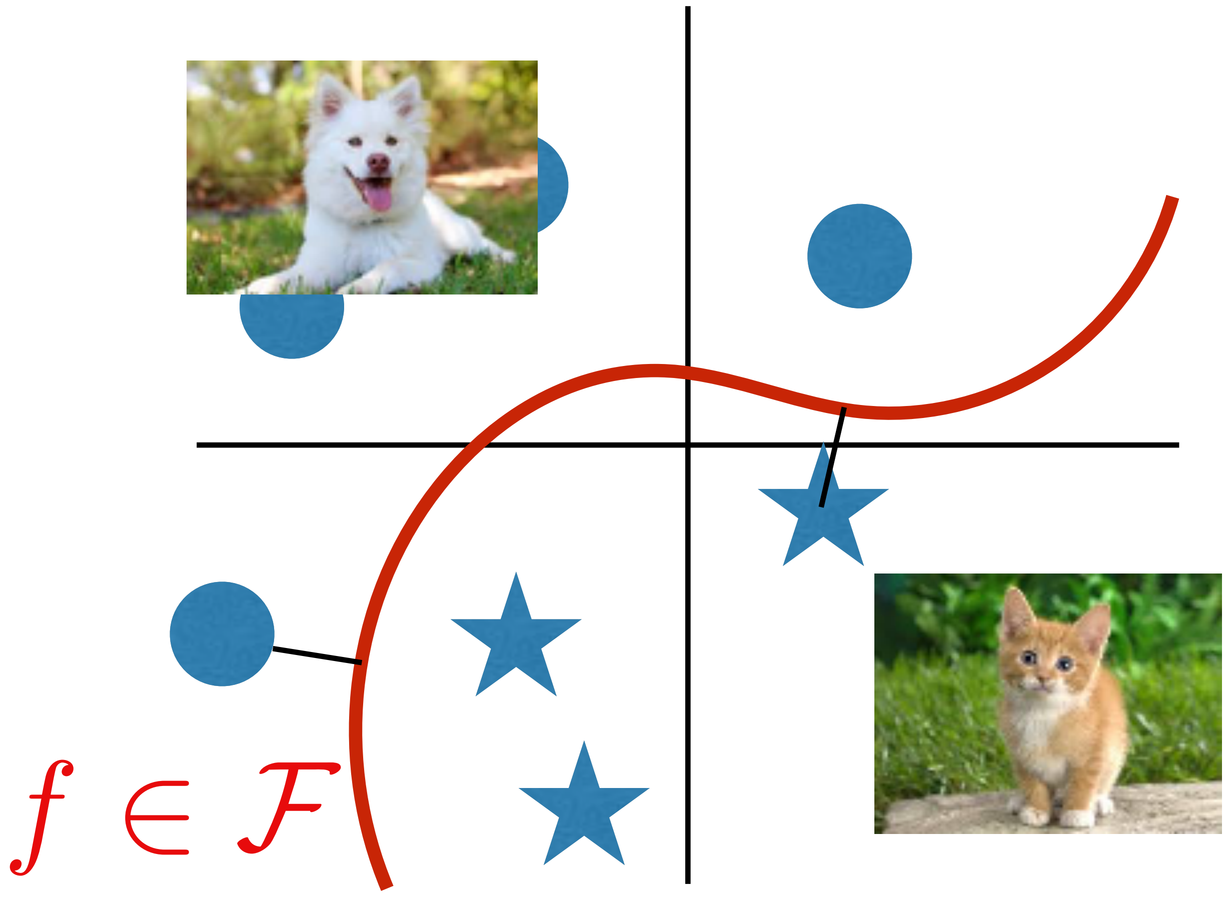
Given i.i.d examples at training:

( ,cat) ( ,cat) ( ,dog)

Recap on Supervised Learning: Classification

Given i.i.d examples at training:

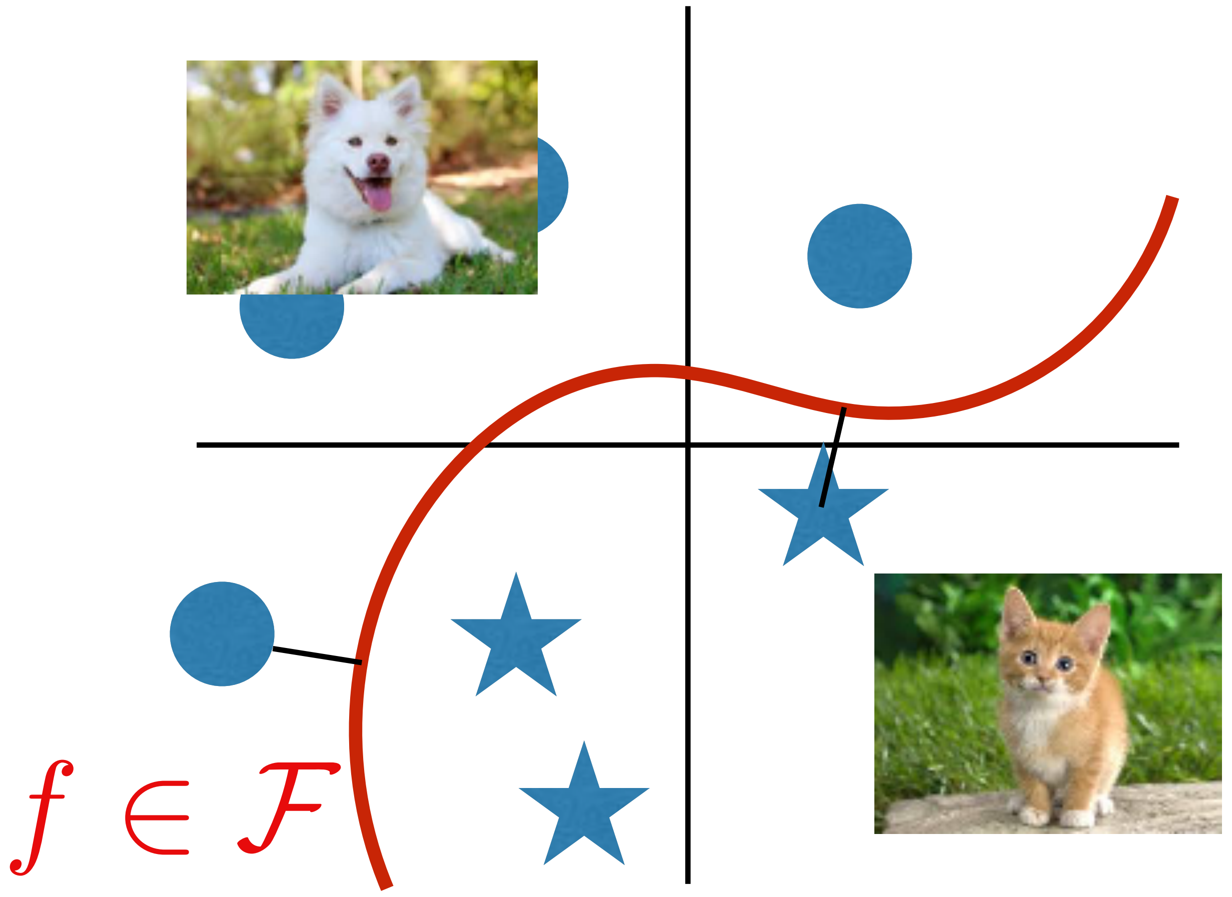
( ,cat) ( ,cat) ( ,dog)



Recap on Supervised Learning: Classification

Given i.i.d examples at training:

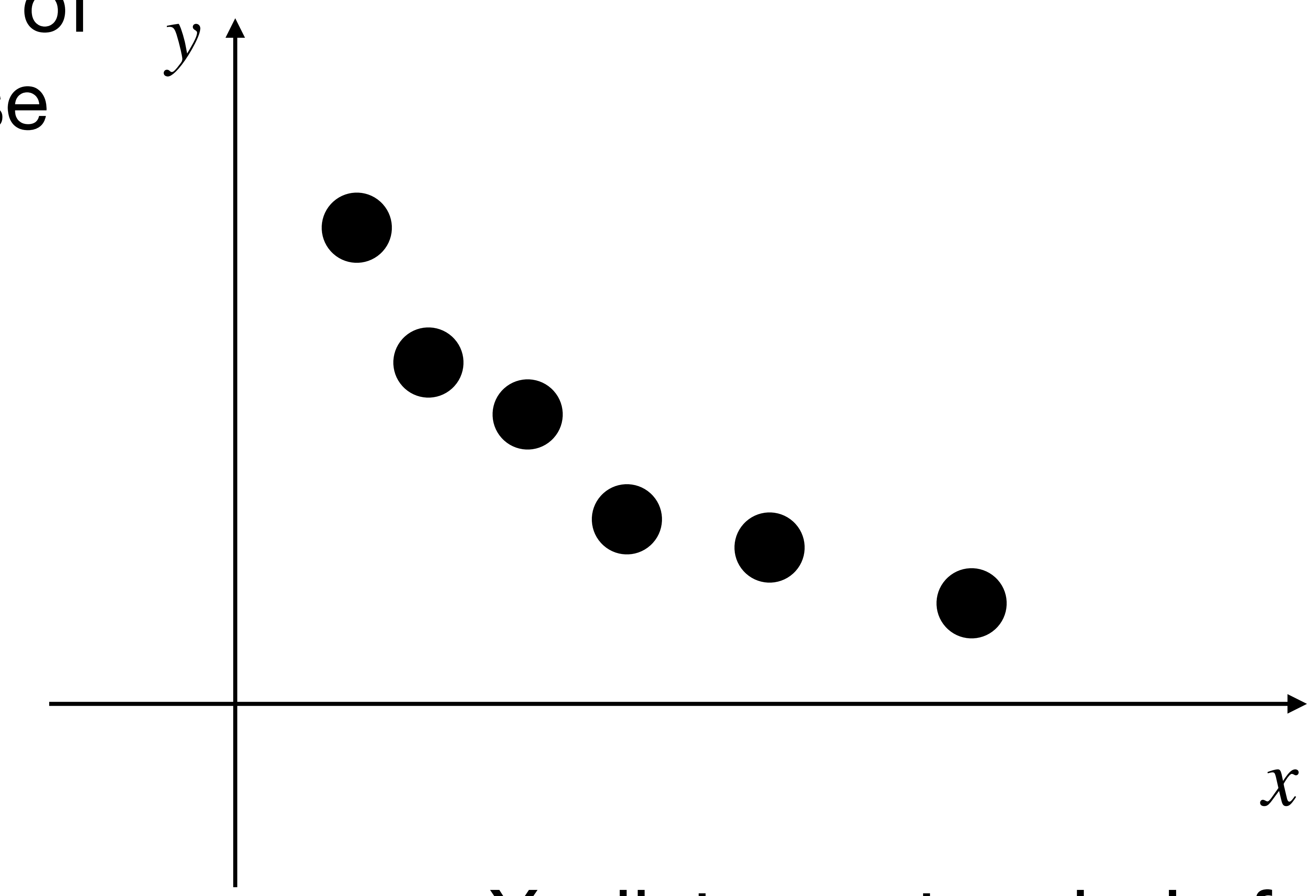
( ,cat) ( ,cat) ( ,dog)



Using function approximator, we are able to predict on cats/dogs that we **never see before** (i.e., we **generalize**)

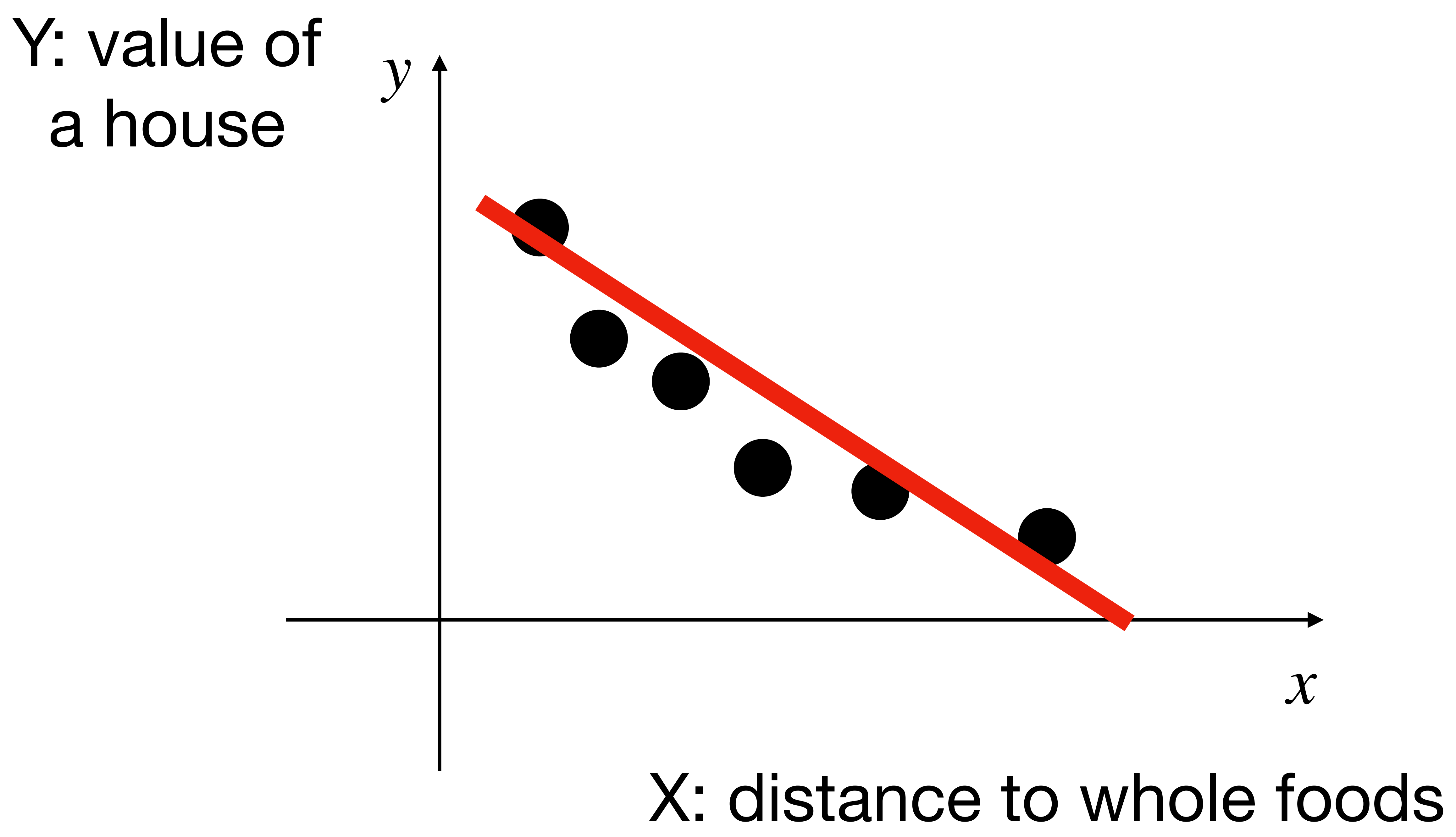
Recap on Supervised Learning: Regression

Y: value of
a house

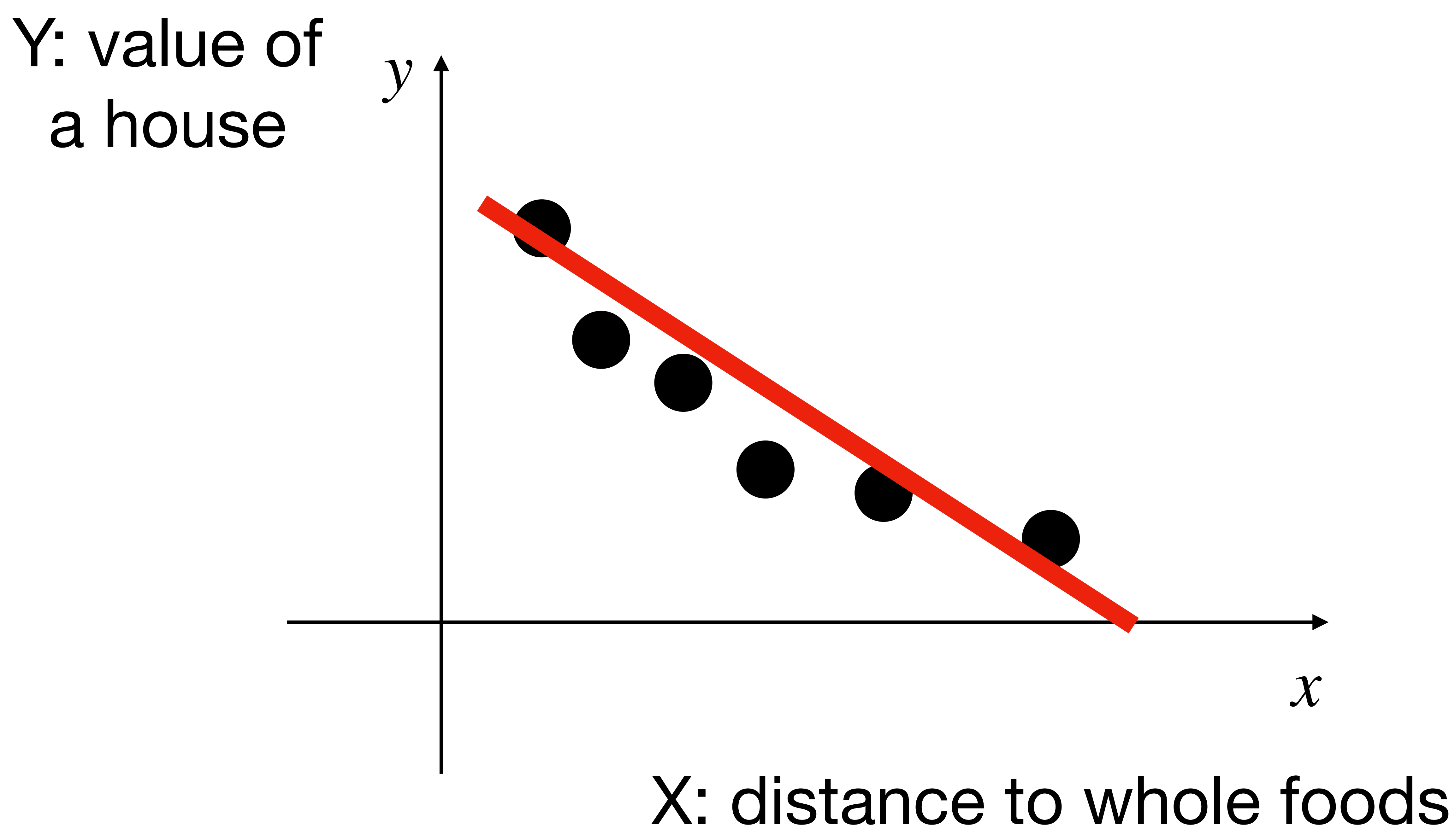


X: distance to whole foods

Recap on Supervised Learning: Regression



Recap on Supervised Learning: Regression



Using function approximator, we are able to predict on the value of some house not from the training data

Recap on Supervised Learning: regression

We have a data distribution $\mathscr{D}$, $x_i \sim \mathscr{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$

Recap on Supervised Learning: regression

We have a data distribution $\mathcal{D}$, $x_i \sim \mathcal{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$

We want to approximate $f^\star$ using finite training samples;

Recap on Supervised Learning: regression

We have a data distribution $\mathscr{D}$, $x_i \sim \mathscr{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$

We want to approximate $f^\star$ using finite training samples;

Let us introduce an abstract function class $\mathscr{F} = \{f : \mathscr{X} \mapsto \mathbb{R}\}$, and do least square:

Recap on Supervised Learning: regression

We have a data distribution $\mathscr{D}$, $x_i \sim \mathscr{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$

We want to approximate $f^\star$ using finite training samples;

Let us introduce an abstract function class $\mathscr{F} = \{f : \mathscr{X} \mapsto \mathbb{R}\}$, and do least square:

$$\hat{f} = \arg\min_{f \in \mathscr{F}} \sum_{i=1}^{N} \left(f(x_i) - y_i\right)^2$$

Recap on Supervised Learning: regression

We have a data distribution $\mathscr{D}$, $x_i \sim \mathscr{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$

We want to approximate $f^\star$ using finite training samples;

Let us introduce an abstract function class $\mathscr{F} = \{f : \mathscr{X} \mapsto \mathbb{R}\}$, and do least square:

Empirical Risk Minimizer (ERM) $\hat{f} = \arg\min_{f \in \mathscr{F}} \sum_{i=1}^{N} \left(f(x_i) - y_i\right)^2$

Recap on Supervised Learning: regression

We have a data distribution $\mathscr{D}$, $x_i \sim \mathscr{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$

We want to approximate $f^\star$ using finite training samples;

Let us introduce an abstract function class $\mathscr{F} = \{f : \mathscr{X} \mapsto \mathbb{R}\}$, and do least square:

Empirical Risk Minimizer (ERM) $$\hat{f} = \arg\min_{f \in \mathscr{F}} \sum_{i=1}^{N} \left(f(x_i) - y_i\right)^2$$

Q: quality of ERM $\hat{f}$?

Recap on Supervised Learning: regression

We have a data distribution $\mathscr{D}$, $x_i \sim \mathscr{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$

$$\hat{f} = \arg\min_{f \in \mathscr{F}} \sum_{i=1}^{N} \left(f(x_i) - y_i\right)^2$$

Recap on Supervised Learning: regression

We have a data distribution $\mathcal{D}$, $x_i \sim \mathcal{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} \left(f(x_i) - y_i\right)^2$$

Supervised learning theory (e.g., VC theory) says that we can indeed **generalize**, i.e., we can predict well **under the same distribution:**

Recap on Supervised Learning: regression

We have a data distribution $\mathcal{D}$, $x_i \sim \mathcal{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \sum_{i=1}^{N} \left(f(x_i) - y_i \right)^2$$

Supervised learning theory (e.g., VC theory) says that we can indeed **generalize**,
i.e., we can predict well **under the same distribution:**

Assume $f^\star \in \mathcal{F}$ (this is called realizability), we can expect:

Recap on Supervised Learning: regression

We have a data distribution $\mathscr{D}$, $x_i \sim \mathscr{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$

$$\hat{f} = \arg\min_{f \in \mathscr{F}} \sum_{i=1}^{N} \left(f(x_i) - y_i\right)^2$$

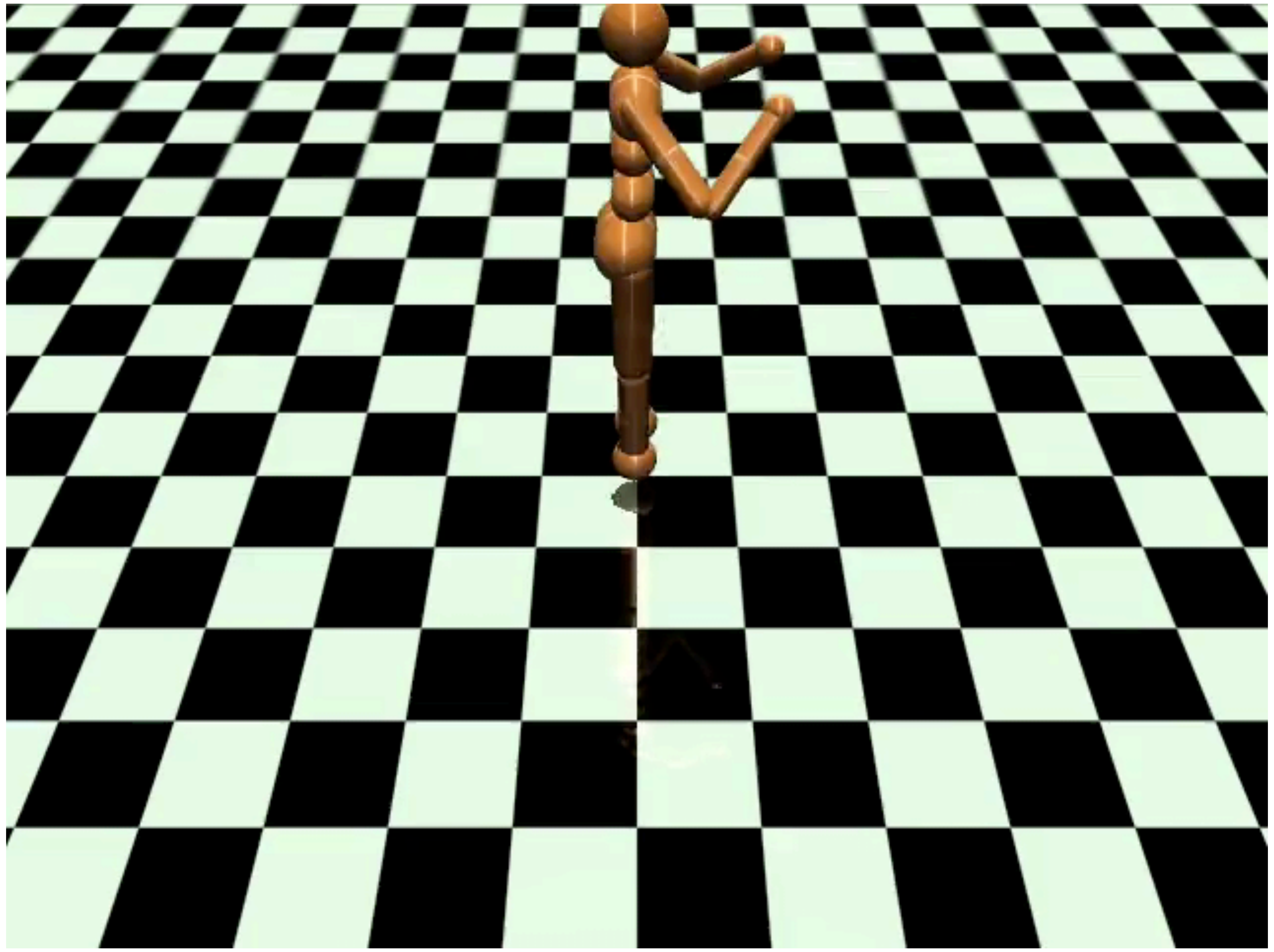
Supervised learning theory (e.g., VC theory) says that we can indeed **generalize**,
i.e., we can predict well **under the same distribution:**

Assume $f^\star \in \mathscr{F}$ (this is called realizability), we can expect:

$$\mathbb{E}_{x \sim \mathscr{D}} \left(\hat{f}(x) - f^\star(x)\right)^2 \leq \delta$$

Supervise Learning can fail if there is train-test distribution mismatch

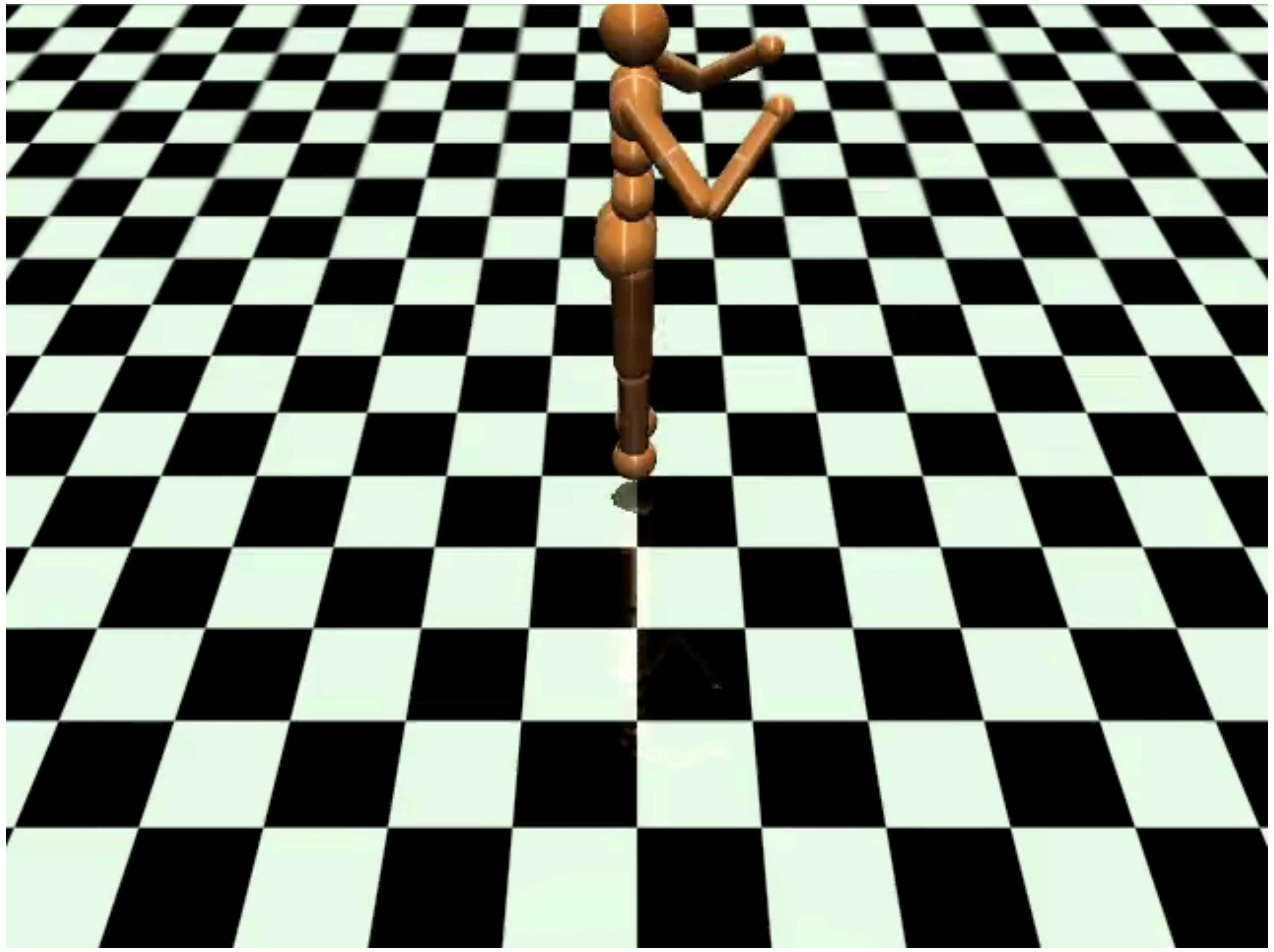
However, for some $\mathscr{D}' \neq \mathscr{D}$, $\mathbb{E}_{x\sim\mathscr{D}'}(f(x) - f^\star(x))^2$ might be arbitrarily large



[openAI Gym]

Supervise Learning can fail if there is train-test distribution mismatch

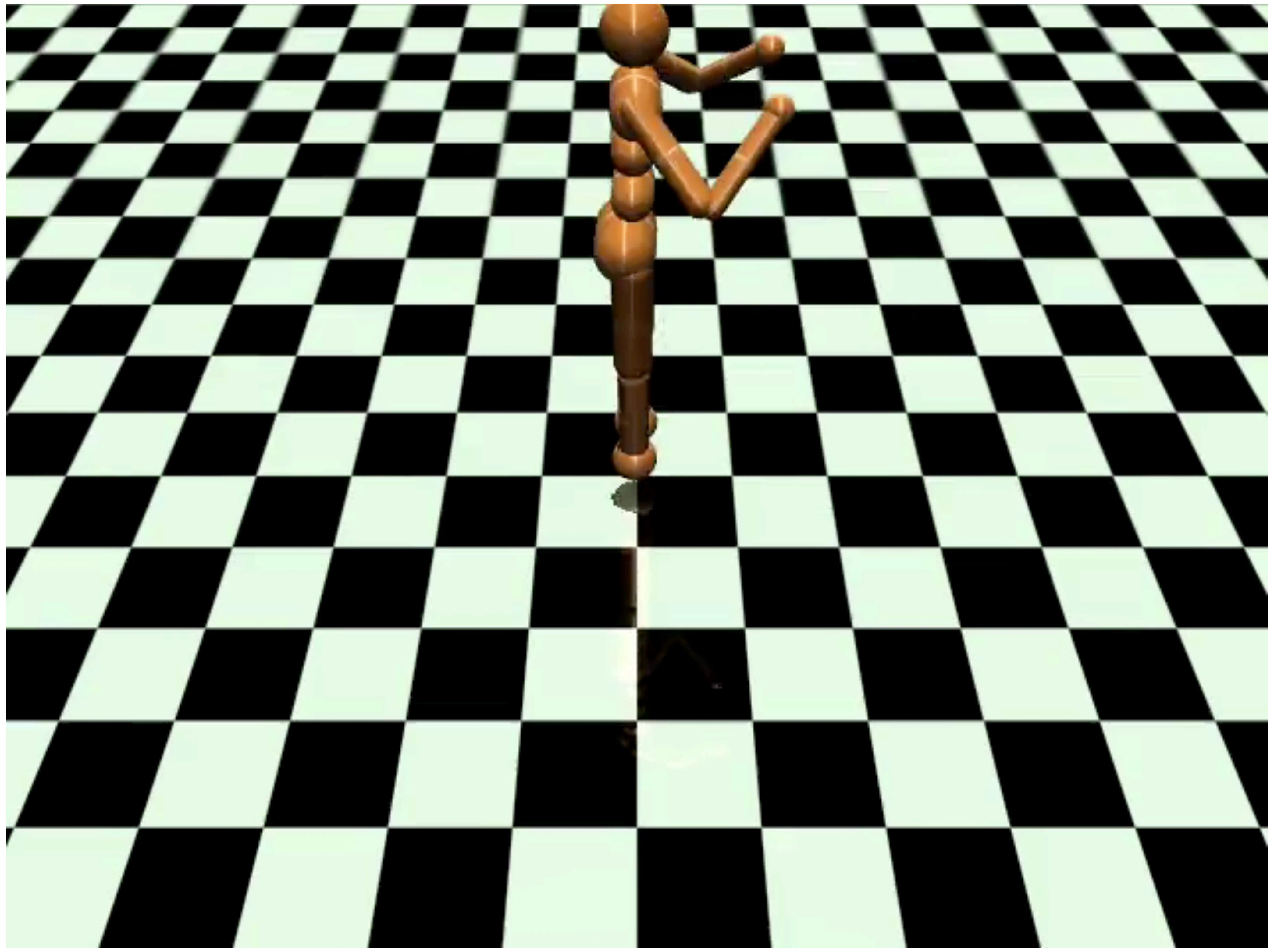
However, for some $\mathcal{D}' \neq \mathcal{D}$, $\mathbb{E}_{x \sim \mathcal{D}'} (f(x) - f^\star(x))^2$ might be arbitrarily large



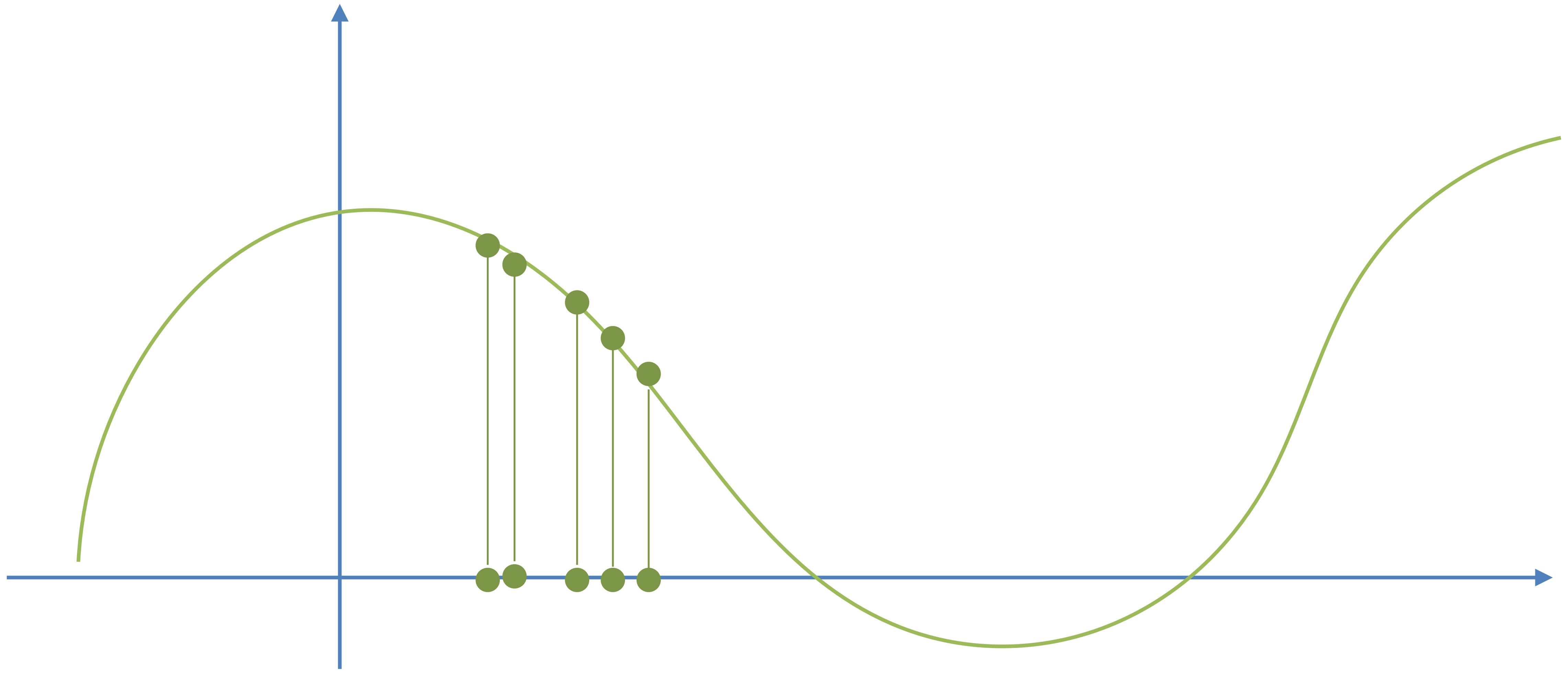
[openAI Gym]

Supervise Learning can fail if there is train-test distribution mismatch

However, for some $\mathscr{D}' \neq \mathscr{D}$, $\mathbb{E}_{x \sim \mathscr{D}'} (f(x) - f^\star(x))^2$ might be arbitrarily large

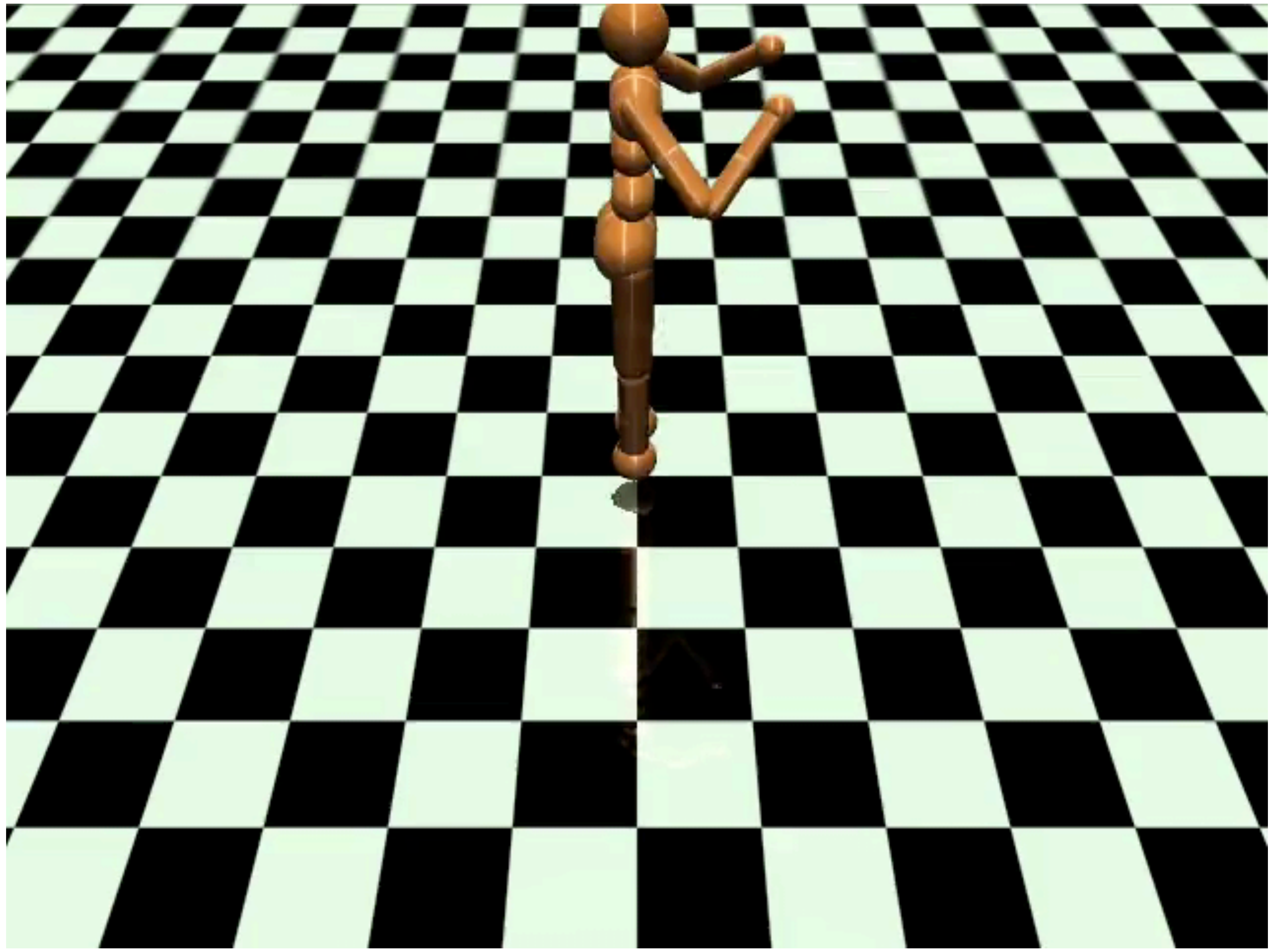


[openAI Gym]

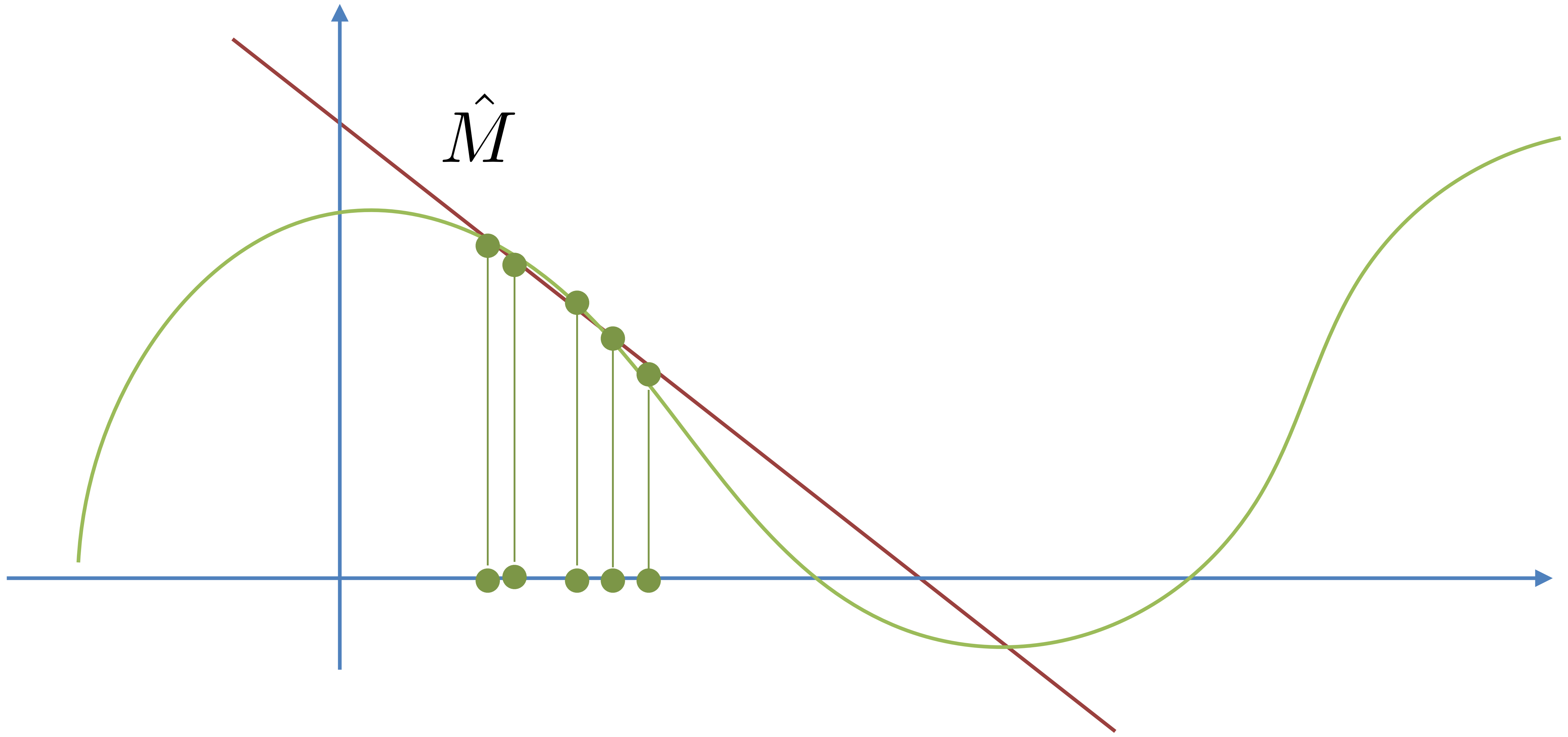


Supervise Learning can fail if there is train-test distribution mismatch

However, for some $\mathscr{D}' \neq \mathscr{D}$, $\mathbb{E}_{x \sim \mathscr{D}'} (f(x) - f^\star(x))^2$ might be arbitrarily large

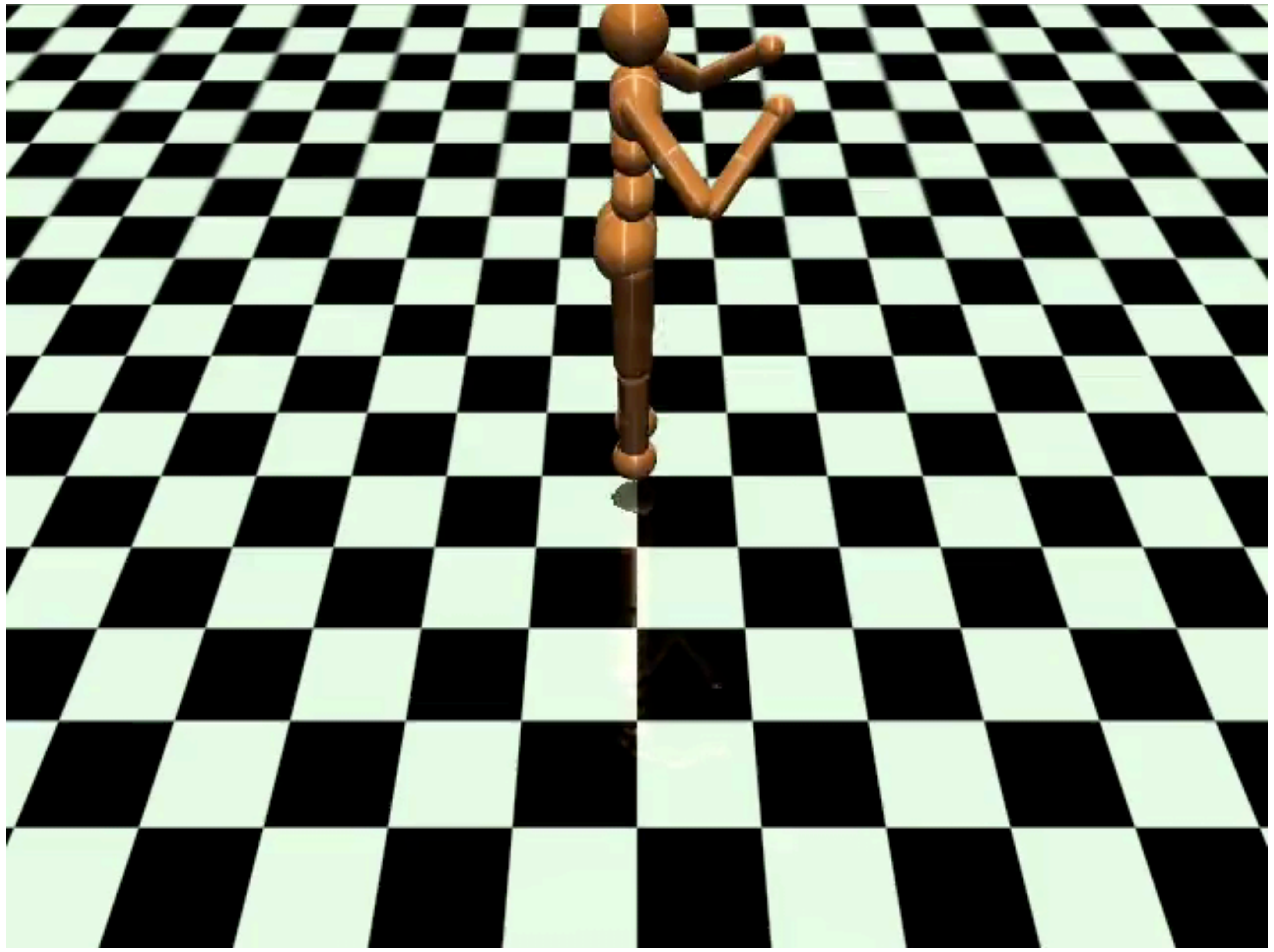


[openAI Gym]

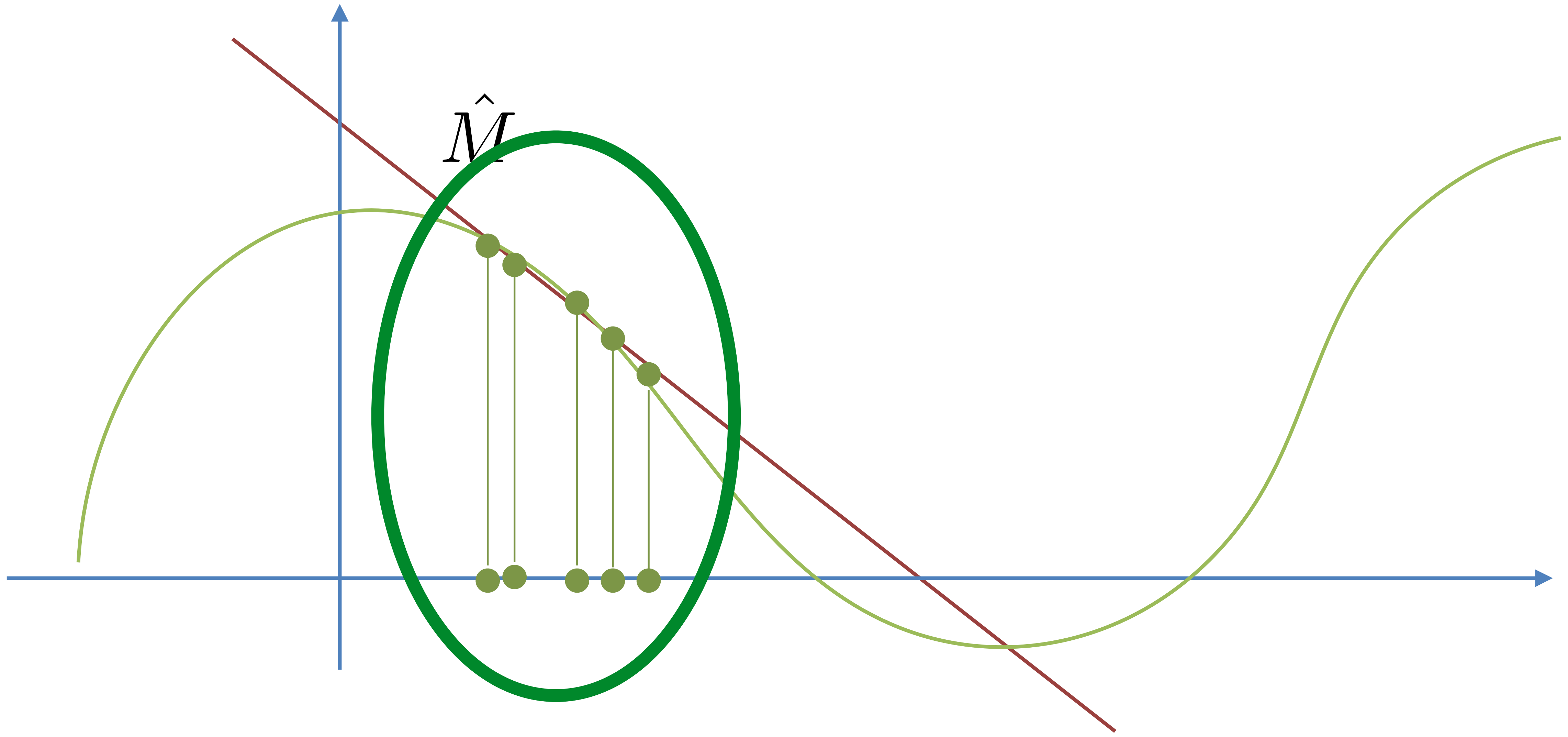


Supervise Learning can fail if there is train-test distribution mismatch

However, for some $\mathscr{D}' \neq \mathscr{D}$, $\mathbb{E}_{x \sim \mathscr{D}'} (f(x) - f^\star(x))^2$ might be arbitrarily large

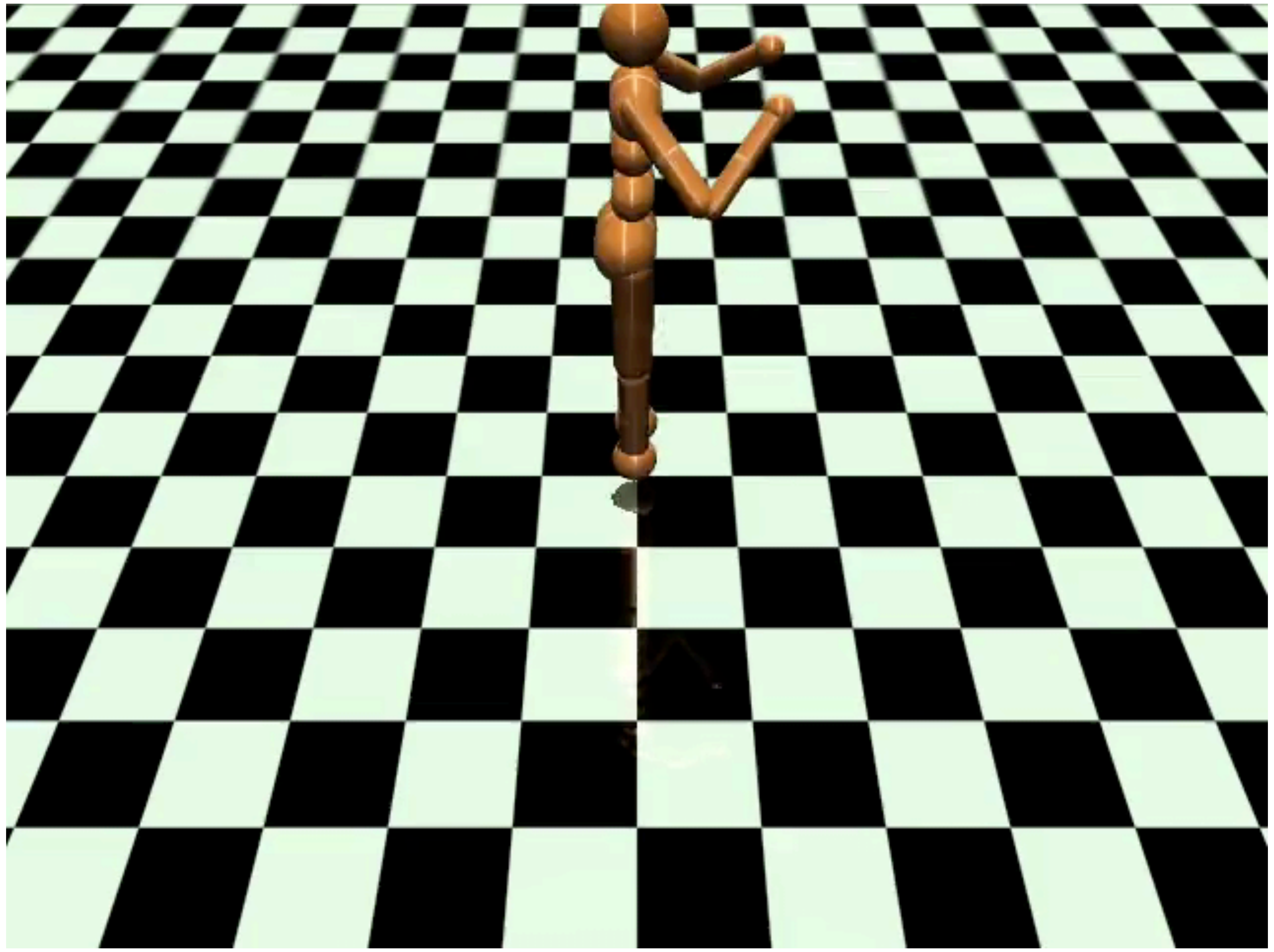


[openAI Gym]

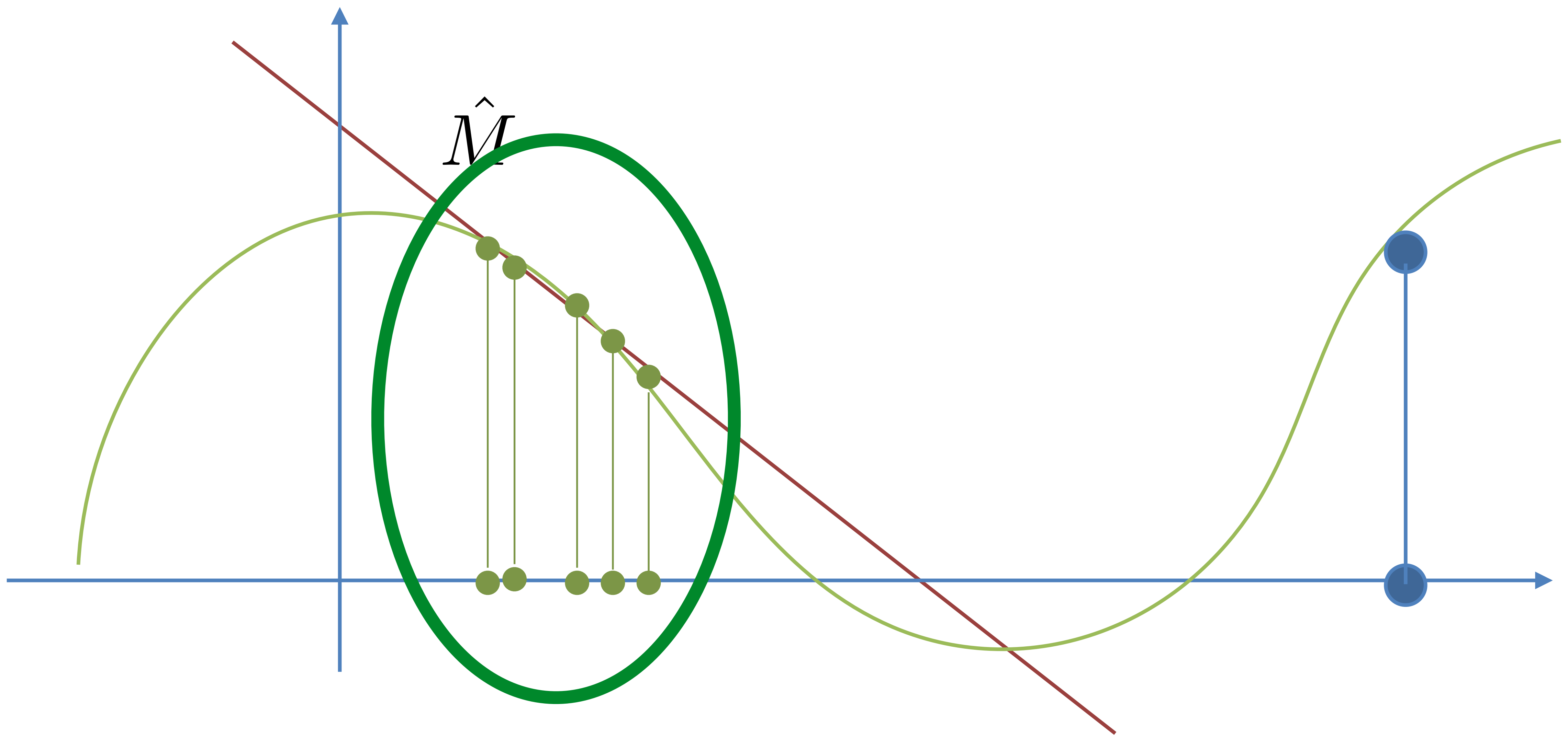


Supervise Learning can fail if there is train-test distribution mismatch

However, for some $\mathscr{D}' \neq \mathscr{D}$, $\mathbb{E}_{x \sim \mathscr{D}'} (f(x) - f^\star(x))^2$ might be arbitrarily large

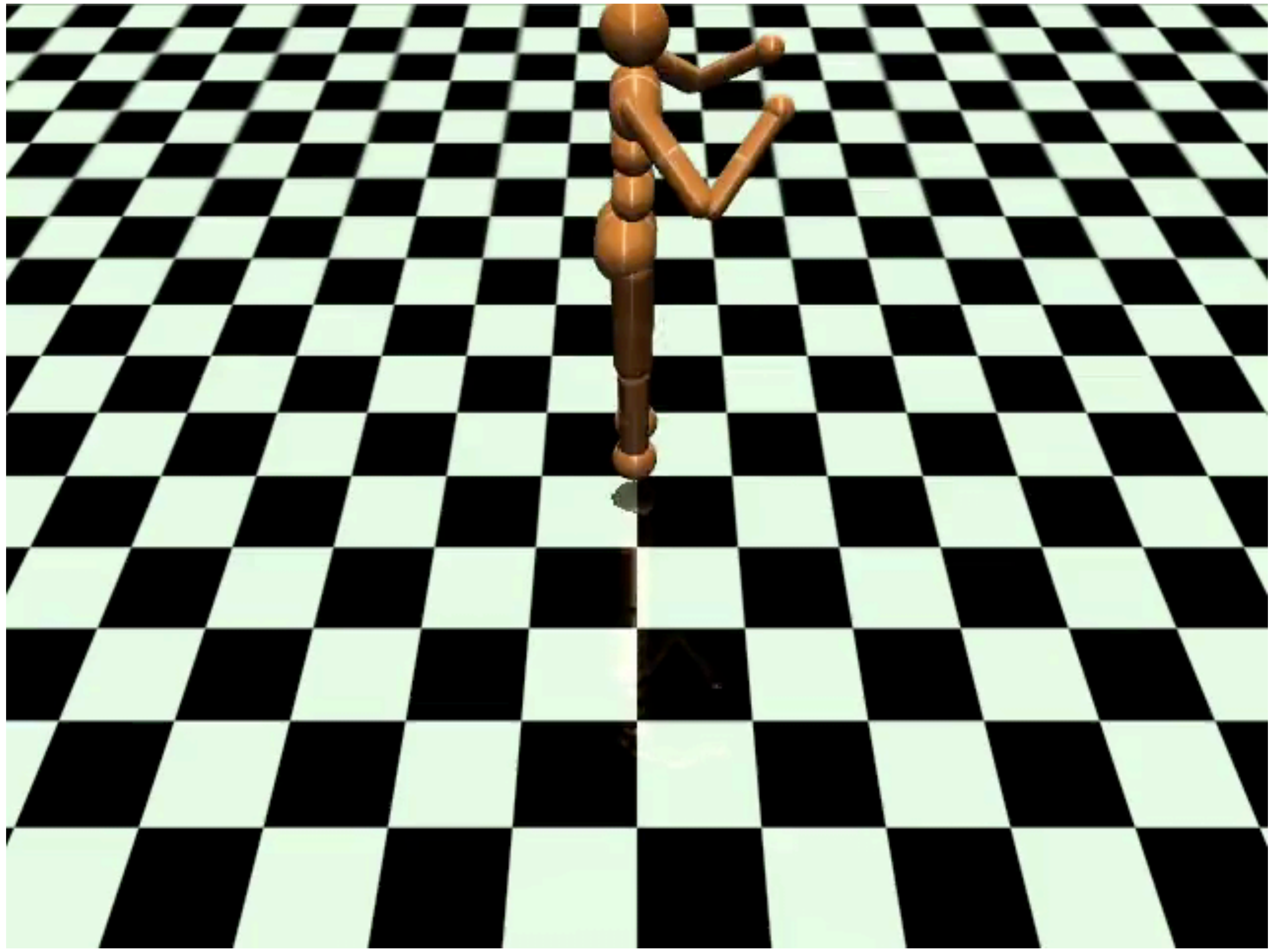


[openAI Gym]

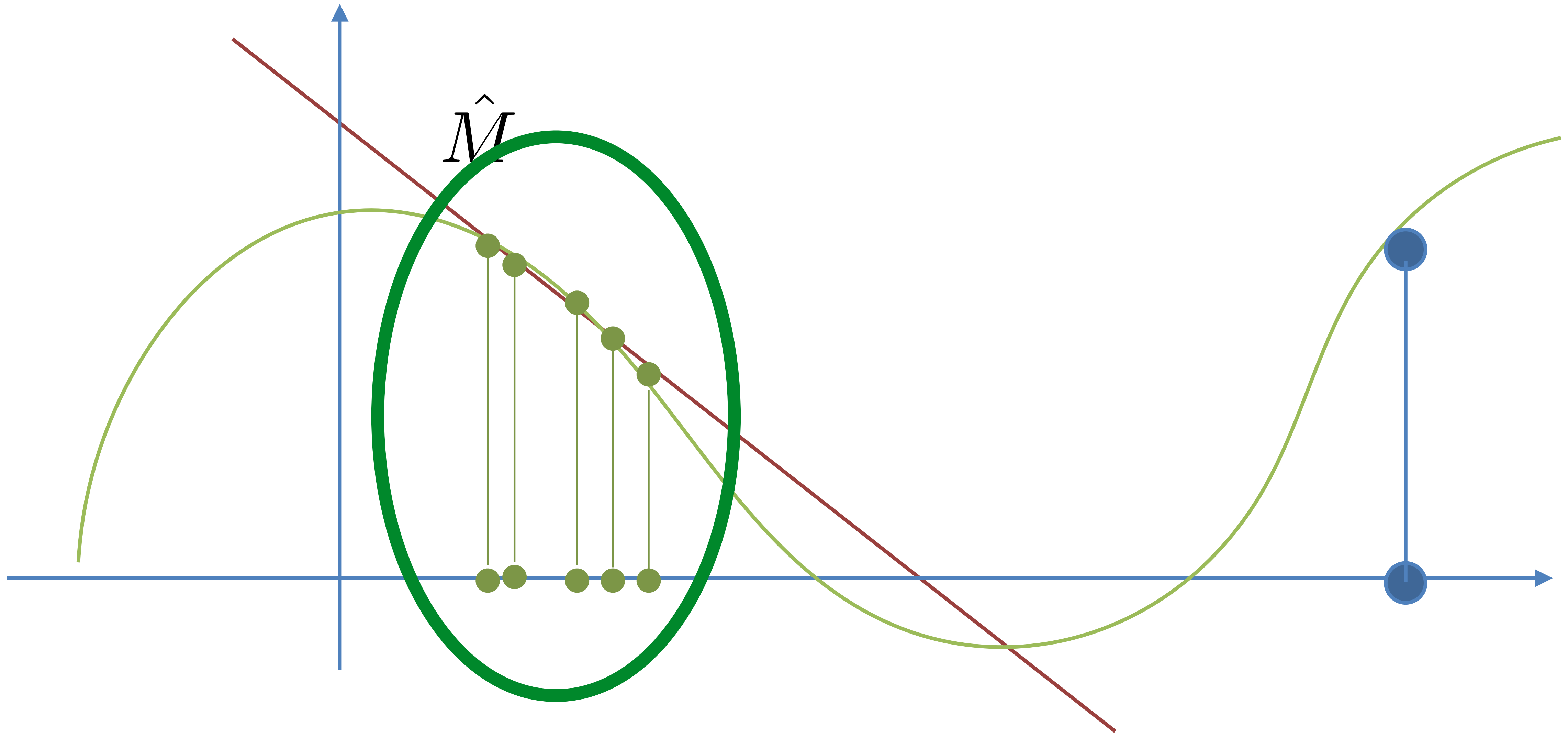


Supervise Learning can fail if there is train-test distribution mismatch

However, for some $\mathscr{D}' \neq \mathscr{D}$, $\mathbb{E}_{x \sim \mathscr{D}'}\left(f(x) - f^\star(x)\right)^2$ might be arbitrarily large



[openAI Gym]



Deeper neural nets or larger dataset do not help if there is distribution shift;
(ML is not black magic)

Recap on Supervised Learning: regression

Throughout the semester, we will often just assume supervised learning succeed:

Recap on Supervised Learning: regression

Throughout the semester, we will often just assume supervised learning succeed:

For any data distribution $\mathscr{D}$, $x_i \sim \mathscr{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$, define **ERM**:

$$\hat{f} = \arg\min_{f \in \mathscr{F}} \sum_{i=1}^{N} \left(f(x_i) - y_i\right)^2$$

Recap on Supervised Learning: regression

Throughout the semester, we will often just assume supervised learning succeed:

For any data distribution $\mathscr{D}$, $x_i \sim \mathscr{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$, define **ERM**:

$$\hat{f} = \arg\min_{f \in \mathscr{F}} \sum_{i=1}^{N} \left(f(x_i) - y_i\right)^2$$

Assume $f^\star \in \mathscr{F}$ (this is called realizability), we can expect **small test error under** $\mathscr{D}$**:**

Recap on Supervised Learning: regression

Throughout the semester, we will often just assume supervised learning succeed:

For any data distribution $\mathscr{D}$, $x_i \sim \mathscr{D}, y_i = f^\star(x_i) + \epsilon_i$, where noise $\mathbb{E}[\epsilon_i] = 0, |\epsilon_i| \leq c$, define **ERM**:

$$\hat{f} = \arg\min_{f \in \mathscr{F}} \sum_{i=1}^{N} \left(f(x_i) - y_i\right)^2$$

Assume $f^\star \in \mathscr{F}$ (this is called realizability), we can expect **small test error under $\mathscr{D}$:**

$$\mathbb{E}_{x \sim \mathscr{D}} \left(\hat{f}(x) - f^\star(x)\right)^2 \leq \delta$$

(where $\delta \approx \sqrt{1/N}$ (sometime it could be $1/N$))

Outline:

1. Quick recap on supervised learning's performance guarantee (classification & regression)

2. Approximate Policy Iteration (relies regression oracle)

Setting and Notation

Discounted infinite horizon MDP:

$$\mathscr{M} = \{S, A, \gamma, r, P, \mu_0\}$$

Setting and Notation

Discounted infinite horizon MDP:

$$\mathscr{M} = \{S, A, \gamma, r, P, \mu_0\}$$

State visitation: $d^{\pi}_{\mu_0}(s) = (1-\gamma)\sum_{h=0}^{\infty} \gamma^h \mathbb{P}^{\pi}_h(s; \mu_0)$

Setting and Notation

Discounted infinite horizon MDP:

$$\mathcal{M} = \{S, A, \gamma, r, P, \mu_0\}$$

State visitation: $d^{\pi}_{\mu_0}(s) = (1-\gamma)\sum_{h=0}^{\infty}\gamma^h \mathbb{P}^{\pi}_h(s;\mu_0)$

As we will consider large scale unknown MDP here, we start with a (restricted) function class $\mathcal{Q}$:

$$\mathcal{Q} = \{Q : S \times A \mapsto [0,1/(1-\gamma)]\}$$

Setting and Notation

Discounted infinite horizon MDP:

$$\mathscr{M} = \{S, A, \gamma, r, P, \mu_0\}$$

State visitation: $d^{\pi}_{\mu_0}(s) = (1-\gamma) \sum_{h=0}^{\infty} \gamma^h \mathbb{P}^{\pi}_h(s; \mu_0)$

As we will consider large scale unknown MDP here, we start with a (restricted) function class $\mathscr{Q}$:

$$\mathscr{Q} = \{Q : S \times A \mapsto [0,1/(1-\gamma)]\}$$

(e.g., all 2 layer neural networks, all 10 layer regression tree, all possible linear functions)

Setting and Notation

Discounted infinite horizon MDP:

$$\mathcal{M} = \{S, A, \gamma, r, P, \mu_0\}$$

State visitation: $d^{\pi}_{\mu_0}(s) = (1-\gamma)\sum_{h=0}^{\infty} \gamma^h \mathbb{P}^{\pi}_h(s; \mu_0)$

As we will consider large scale unknown MDP here, we start with a (restricted) function class $\mathcal{Q}$:

$$\mathcal{Q} = \{Q : S \times A \mapsto [0,1/(1-\gamma)]\}$$

(e.g., all 2 layer neural networks, all 10 layer regression tree, all possible linear functions)

We can only reset according to $s_0 \sim \mu_0$

Approximate Policy Iteration

Like Policy Iteration, we iterate between two steps:

Approximate Policy Iteration

Like Policy Iteration, we iterate between two steps:

1. Policy Evaluation $\widehat{Q}^t \approx Q^{\pi^t}$

Approximate Policy Iteration

Like Policy Iteration, we iterate between two steps:

1. Policy Evaluation $\widehat{Q}^t \approx Q^{\pi^t}$

2. Policy Improvement $\pi^{t+1}(s) = \arg\max_a \widehat{Q}^t(s, a)$

Approximate Policy Iteration

Like Policy Iteration, we iterate between two steps:

1. Policy Evaluation $\widehat{Q}^{\,t} \approx Q^{\pi^t}$

We use supervised learning (regression) to estimate Q^{π^t}

2. Policy Improvement $\pi^{t+1}(s) = \arg\max_a \widehat{Q}^{\,t}(s, a)$

Approximate Policy Iteration

Like Policy Iteration, we iterate between two steps:

We use supervised learning (regression) to estimate Q^{π^t}

1. Policy Evaluation $\widehat{Q}^t \approx Q^{\pi^t}$

a. How to get training data?

b. Quality of the learned $\widehat{Q}^t$?

2. Policy Improvement $\pi^{t+1}(s) = \arg\max_a \widehat{Q}^t(s, a)$

Policy Evaluation: Dataset Generation

Q1: how do we sample a state-action pair $(s, a) \sim d^{\pi}_{\mu_0}$?

Policy Evaluation: Dataset Generation

Q1: how do we sample a state-action pair $(s, a) \sim d^{\pi}_{\mu_0}$?

1. Sample time step h with probability $\gamma^h(1-\gamma)$

Policy Evaluation: Dataset Generation

Q1: how do we sample a state-action pair $(s, a) \sim d^{\pi}_{\mu_0}$?

1. Sample time step h with probability $\gamma^h(1-\gamma)$

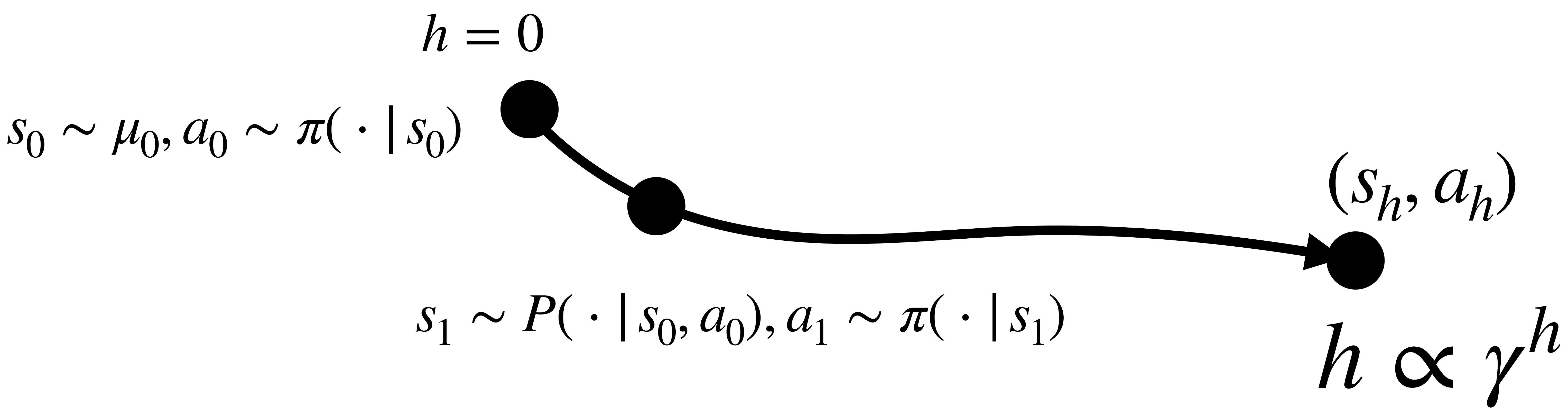
2. Roll-in π to time step h, and return (s_h, a_h)
(i.e., we sample $(s, a) \sim \mathbb{P}^{\pi}_{h}(\cdot\,,\cdot\,;\mu_0)$)

Policy Evaluation: Dataset Generation

Q1: how do we sample a state-action pair $(s, a) \sim d^{\pi}_{\mu_0}$?

1. Sample time step h with probability $\gamma^h(1-\gamma)$

2. Roll-in π to time step h, and return (s_h, a_h)
(i.e., we sample $(s, a) \sim \mathbb{P}^{\pi}_h(\cdot, \cdot; \mu_0)$)



Policy Evaluation: Dataset Generation

Q2: Given that we are at (s, a), how do we get an unbiased estimate of $Q^\pi(s, a)$?

Policy Evaluation: Dataset Generation

Q2: Given that we are at (s, a), how do we get an unbiased estimate of $Q^\pi(s, a)$?

Denote $(s_0, a_0) = (s, a)$

For $h = 0, \ldots,$

Policy Evaluation: Dataset Generation

Q2: Given that we are at (s, a), how do we get an unbiased estimate of $Q^\pi(s, a)$?

Denote $(s_0, a_0) = (s, a)$

For $h = 0, \ldots,$

Receive $r_h = r(s_h, a_h)$

Policy Evaluation: Dataset Generation

Q2: Given that we are at (s, a), how do we get an unbiased estimate of $Q^\pi(s, a)$?

Denote $(s_0, a_0) = (s, a)$

For $h = 0, \ldots,$

Receive $r_h = r(s_h, a_h)$

With probability $1 - \gamma$: **Break** and **Return** $\sum_{i=0}^{t} r_i$

Policy Evaluation: Dataset Generation

Q2: Given that we are at (s, a), how do we get an unbiased estimate of $Q^\pi(s, a)$?

Denote $(s_0, a_0) = (s, a)$

For $h = 0, \ldots,$

Receive $r_h = r(s_h, a_h)$

With probability $1 - \gamma$: **Break** and **Return** $\sum_{i=0}^{t} r_i$

Transition: $s_{h+1} \sim P(s_h, a_h), a_{h+1} \sim \pi(\cdot \mid s_{h+1})$

Policy Evaluation: Dataset Generation

Q2: Given that we are at (s, a), how do we get an unbiased estimate of $Q^\pi(s, a)$?

Denote $(s_0, a_0) = (s, a)$

For $h = 0, \ldots,$

Receive $r_h = r(s_h, a_h)$

With probability $1 - \gamma$: **Break** and **Return** $\sum_{i=0}^{t} r_i$

Transition: $s_{h+1} \sim P(s_h, a_h), a_{h+1} \sim \pi(\cdot \mid s_{h+1})$

$h = 0$ $(s, a), r_0 = r(s, a)$

$(s_h, a_h), r_h$

$s_1 \sim P(\cdot \mid s, a),$

$a_1 \sim \pi(\cdot \mid s_1)$

$r_1 = r(s_1, a_1)$

A Roll-out process

Policy Evaluation: Dataset Generation

Q2: Given that we are at (s, a), how do we get an unbiased estimate of $Q^\pi(s, a)$?

Denote $(s_0, a_0) = (s, a)$

For $h = 0, \ldots,$

> Receive $r_h = r(s_h, a_h)$
>
> With probability $1 - \gamma$: **Break** and **Return** $\sum_{i=0}^{t} r_i$
>
> Transition: $s_{h+1} \sim P(s_h, a_h), a_{h+1} \sim \pi(\cdot \mid s_{h+1})$

A Roll-out process

Policy Evaluation: Dataset Generation

$h = 0 \quad (s, a), r_0 = r(s, a)$

$(s_h, a_h), r_h$

$s_1 \sim P(\cdot \mid s, a),$

$a_1 \sim \pi(\cdot \mid s_1)$

$r_1 = r(s_1, a_1)$

If terminate (w/ p $1 - \gamma$),
we return

$$y := \sum_{i=0}^{t} r_i$$

Claim: A roll-out from (s, a) gives an unbiased estimate of $Q^\pi(s, a)$

$$\mathbb{E}\left[y\right] = Q^\pi(s, a)$$

Policy Evaluation: Dataset Generation

$h = 0 \quad (s, a), r_0 = r(s, a)$

$(s_h, a_h), r_h$

$s_1 \sim P(\cdot \mid s, a),$

$a_1 \sim \pi(\cdot \mid s_1)$

$r_1 = r(s_1, a_1)$

If terminate (w/ p $1 - \gamma$),
we return

$$y := \sum_{i=0}^{t} r_i$$

Claim: A roll-out from (s, a) gives an unbiased estimate of $Q^\pi(s, a)$

$$\mathbb{E}\left[y\right] = Q^\pi(s, a)$$

Proof sketch (full proof is left as an exercise):

Policy Evaluation: Dataset Generation

$h = 0$ $(s, a), r_0 = r(s, a)$

$(s_h, a_h), r_h$

$s_1 \sim P(\cdot \mid s, a),$

$a_1 \sim \pi(\cdot \mid s_1)$

$r_1 = r(s_1, a_1)$

If terminate (w/ p $1 - \gamma$),
we return

$$y := \sum_{i=0}^{t} r_i$$

Claim: A roll-out from (s, a) gives an unbiased estimate of $Q^\pi(s, a)$

$$\mathbb{E}\left[y\right] = Q^\pi(s, a)$$

Proof sketch (full proof is left as an exercise):

Q: What's the probability of returning $y = r_0$,

and what's the probability of returning $y = r_0 + r_1$?

Policy Evaluation: Dataset Generation

$h = 0 \quad (s, a), r_0 = r(s, a)$

$(s_h, a_h), r_h$

$s_1 \sim P(\cdot \mid s, a),$

$a_1 \sim \pi(\cdot \mid s_1)$

$r_1 = r(s_1, a_1)$

If terminate (w/ p $1 - \gamma$), we return

$$y := \sum_{i=0}^{t} r_i$$

Claim: A roll-out from (s, a) gives an unbiased estimate of $Q^\pi(s, a)$

$$\mathbb{E}\left[y\right] = Q^\pi(s, a)$$

Proof sketch (full proof is left as an exercise):

Q: What's the probability of returning $y = r_0$,

and what's the probability of returning $y = r_0 + r_1$?

$$(1-\gamma)r_0 + \gamma(1-\gamma)(r_0 + r_1) + \gamma^2(1-\gamma)(r_0 + r_1 + r_2) + \ldots = \sum_{h=0}^{\infty} \gamma^h r_h$$

Summary of the dataset generation process:

Given π^t:

1. we **roll-in** to generate $(s, a) \sim d^{\pi}_{\mu_0}$

2. At (s, a), we **roll-out** w/ π to generate an unbiased estimate of $Q^{\pi}(s, a)$: y

In other words, one roll-in & roll-out gives us a triple $\left(s, a, y\right)$

Summary of the dataset generation process:

Given π^t:

1. we **roll-in** to generate $(s, a) \sim d^{\pi}_{\mu_0}$

2. At (s, a), we **roll-out** w/ π to generate an unbiased estimate of $Q^{\pi}(s, a)$: y

In other words, one roll-in & roll-out gives us a triple $\left(s, a, y\right)$

Given π, repeat N times of the roll-in & roll-out process,
we get a training dataset of N samples:

Summary of the dataset generation process:

Given π^t:

1. we **roll-in** to generate $(s, a) \sim d^{\pi}_{\mu_0}$

2. At (s, a), we **roll-out** w/ π to generate an unbiased estimate of $Q^{\pi}(s, a)$: y

In other words, one roll-in & roll-out gives us a triple $\left(s, a, y\right)$

Given π, repeat N times of the roll-in & roll-out process, we get a training dataset of N samples:

$$\mathscr{D}^{\pi} = \left\{s^i, a^i, y^i\right\}_{i=1}^{N}$$