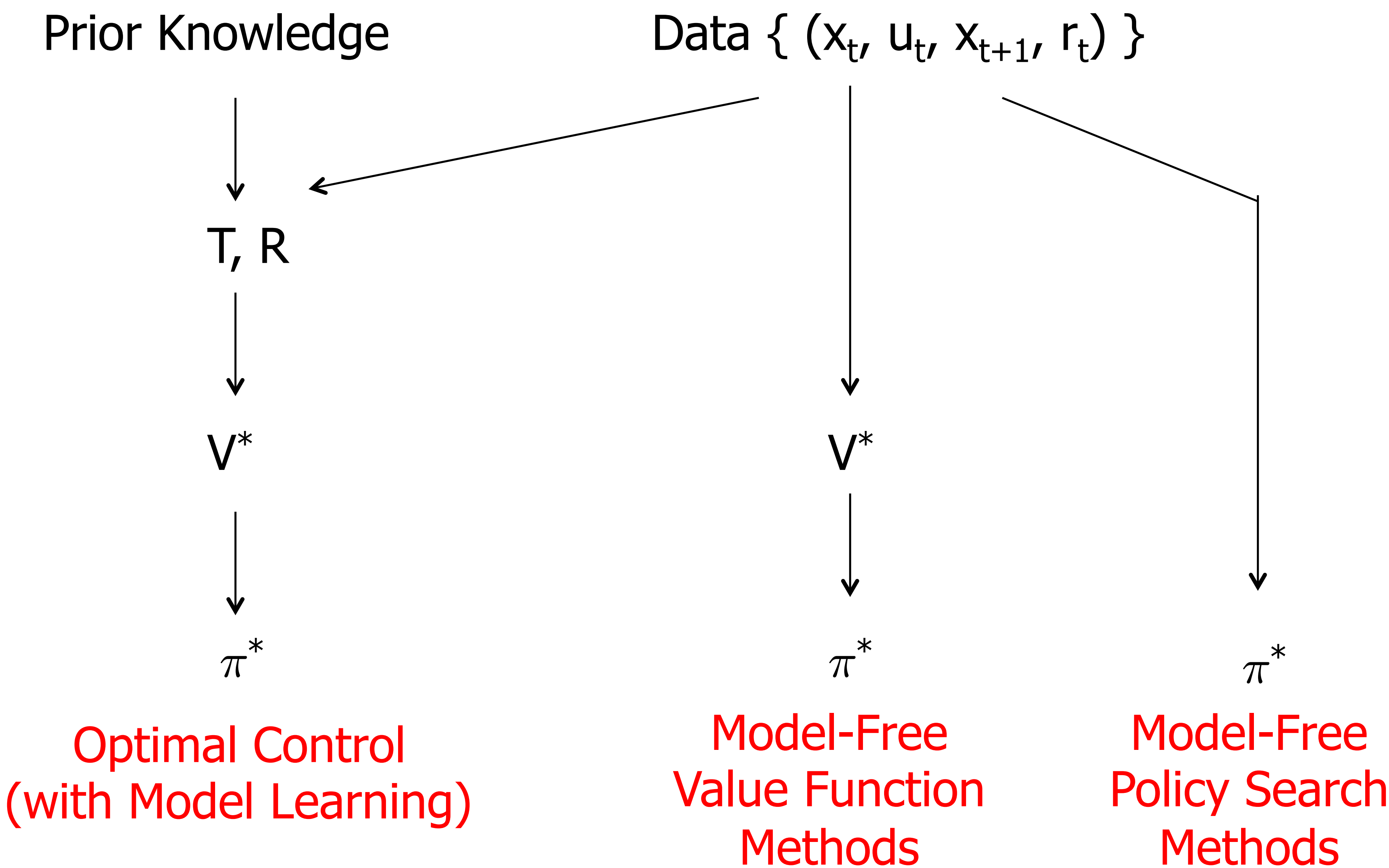
ICRA 2012 Tutorial on Reinforcement Learning
4. Value Function Methods

Pieter Abbeel
UC Berkeley

Jan Peters
TU Darmstadt

A Reinforcement Learning Ontology



Outline

- Challenge: Most real-world problems have large, often infinite and continuous, state spaces

Value Function Methods:

- Model-free learning
  - Monte Carlo, TD-learning and Q-learning (tabular)
- Function approximation
  - Q-learning with feature-based representations
  - Fitted Q-learning
    - Often good approach, even when model is available

Model-Based Learning

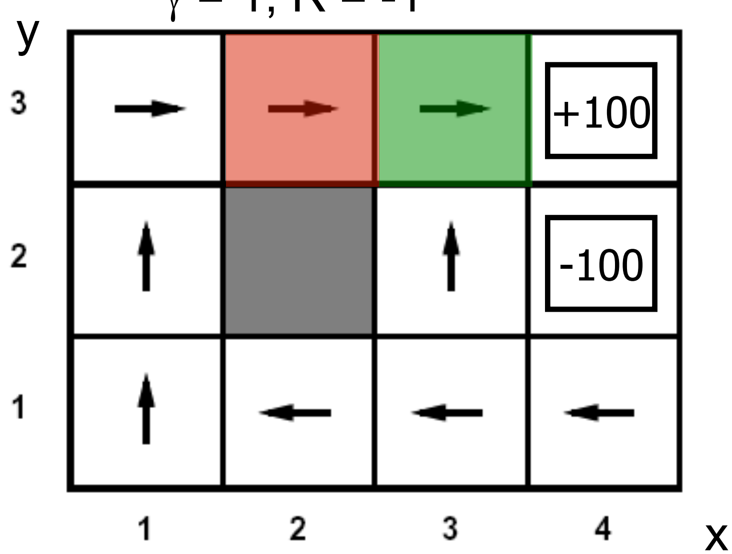
- Step 1: Learn the model:
  - Supervised learning to find **T(x,u,x')** and **R(x,u)** from experiences (x,u,x')
- Step 2: Solve for optimal policy:
  - Can be done with optimal control methods, such as value iteration

Model-free: 1. Monte Carlo / Direct Evaluation

- Repeatedly execute the policy π
- Estimate the value of the state s as the average over all times the state s was visited of the sum of discounted rewards accumulated from state s onwards

Exercise: Direct Evaluation

$\gamma = 1$, $R = -1$

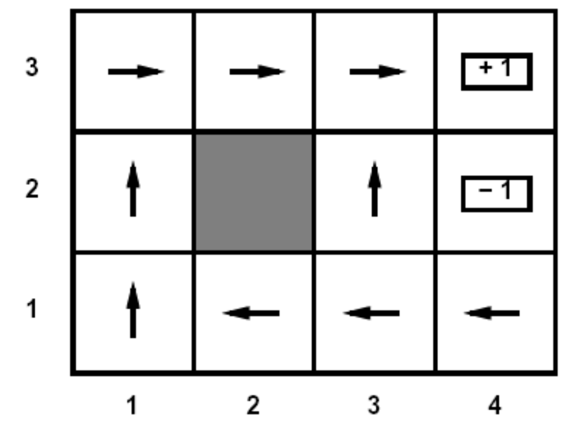


(1,1) up -1	(1,1) up -1
(1,2) up -1	(1,2) up -1
(1,3) right -1	(1,2) up -1
(2,3) right -1	(1,3) right -1
(3,3) right -1	(2,3) right -1
(3,2) up -1	(3,3) right -1
(4,2) exit -100	(3,2) up -1
(done)	(3,3) right -1
	(4,3) exit +100
	(done)

- (a) According to Direct Evaluation: What is V(3,3)?
- (b) According to Direct Evaluation: What is V(2,3)?
- (c) Just based on these samples, what could be a better estimate for V(2,3) ?

Limitations of Direct Evaluation

- Assume random initial state
- Assume the value of state (1,2) is known perfectly based on past runs
- Now for the first time encounter (1,1) --- can we do better than estimating V(1,1) as the rewards outcome of that run?



Model-free: 2. TD Learning

$$V_{i+1}^{\pi}(x) \leftarrow \sum_{x'} T(x, \pi(x), x')[R(x, \pi(x)) + \gamma V_i^{\pi}(x')]$$

- Who needs T and R? Approximate the expectation with samples of s' (drawn from T!)

$$sample_1 = R(x, \pi(x)) + \gamma V_i^{\pi}(x'_1)$$
$$sample_2 = R(x, \pi(x)) + \gamma V_i^{\pi}(x'_2)$$
$$\ldots$$
$$sample_k = R(x, \pi(x)) + \gamma V_i^{\pi}(x'_k)$$

$$V_{i+1}^{\pi}(x) \leftarrow \frac{1}{k} \sum_i sample_i$$

Almost! But we can't rewind time to get sample after sample from state s.

Exponential Moving Average

- Exponential moving average
  - Makes recent samples more important

$$\bar{x}_n = \frac{x_n + (1-\alpha)\cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

  - Forgets about the past (distant past values were wrong anyway)
  - Easy to compute from the running average

$$\bar{x}_n = (1-\alpha)\cdot \bar{x}_{n-1} + \alpha \cdot x_n$$

- Decreasing learning rate can give converging averages

Problems with TD Value Learning

- TD value leaning is a model-free way to do policy evaluation
- However, if we want to turn values into a (new) policy, we' re sunk:

$$\pi(x) = \arg\max_a Q^*(x,u)$$

$$Q^*(x,u) = \sum_{x'} T(x,u,x') \left[R(x,u) + \gamma V^*(x') \right]$$

- Idea: learn Q-values directly
- Makes action selection model-free too!

Detour: Q-Value Iteration

- Value iteration: find successive approx optimal values
  - Start with $V_0^*(x) = 0$, which we know is right (why?)
  - Given V_i^*, calculate the values for all states for depth i+1:

$$V_{i+1}(x) \leftarrow \max_u \sum_{x'} T(x,u,x') \left[R(x,u) + \gamma V_i(x')\right]$$

- But Q-values are more useful!
  - Start with $Q_0^*(x,u) = 0$, which we know is right (why?)
  - Given Q_i^*, calculate the q-values for all q-states for depth i+1:

$$Q_{i+1}(x,u) \leftarrow \sum_{x'} T(x,u,x') \left[R(x,u) + \gamma \max_{u'} Q_i(x',u')\right]$$

Q-Learning

- Q-Learning: sample-based Q-value iteration
- Learn Q* values
  - Receive a sample (x,u,x',r)
  - Consider your old estimate: $Q(x,u)$
  - Consider your new sample estimate:

$$sample = R(x,u) + \gamma \max_{u'} Q(x',u')$$

  - Incorporate the new estimate into a running average:

$$Q(x,u) \leftarrow (1-\alpha)Q(x,u) + (\alpha)\,[sample]$$

Q-Learning Properties

- Amazing result: Q-learning converges to optimal policy
  - If you explore enough
  - If you make the learning rate small enough
  - … but not decrease it too quickly!
  - Basically doesn’t matter how you select actions (!)
- Neat property: off-policy learning
  - learn optimal policy without following it

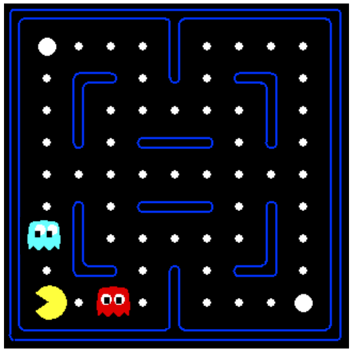
Q-Learning

- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory

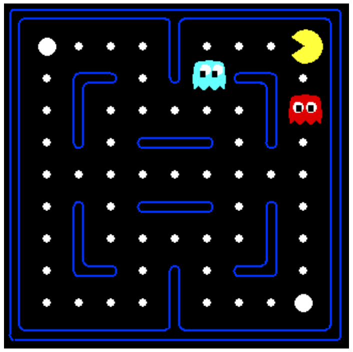
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar states
  - This is a fundamental idea in machine learning, and we'll see it over and over again

Example: Pacman

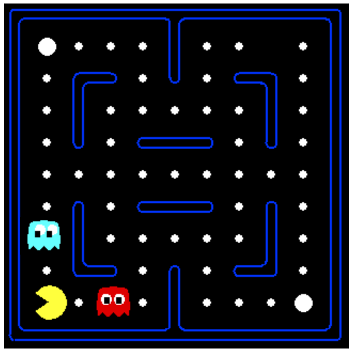
- Let’s say we discover through experience that this state is bad:



- In naïve q learning, we know nothing about this state or its q states:

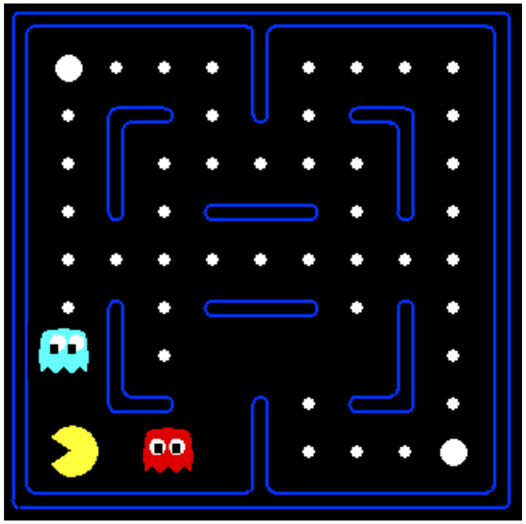


- Or even this one!



Feature-Based Representations

- Solution: describe a state using a vector of features
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - $1 / (\text{dist to dot})^2$
    - Is Pacman in a tunnel? (0/1)
    - …… etc.
  - Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Feature Functions

- Using a feature representation, we can write a q function (or value function) for any state using a few weights:

$$V(x) = w_1 f_1(x) + w_2 f_2(x) + \ldots + w_n f_n(x)$$

$$Q(x,u) = w_1 f_1(x,u) + w_2 f_2(x,u) + \ldots + w_n f_n(x,u)$$

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states may share features but be very different in value!

Tabular Q-function Linear Q-function

Q table

$$Q(x,u) = \sum_{i=1}^{n} w_i f_i(x,u)$$

Sample: $r + \gamma \max_{u'} Q(x', u')$

Difference: $\left[r + \gamma \max_{u'} Q(x', u')\right] - Q(x,u)$

Update:

$Q(x,u) \leftarrow$

$Q(x,u) + \alpha\,[\text{difference}]$

$\forall i, \quad w_i \leftarrow$

$w_i + \alpha\,[\text{difference}]\, f_i(x,u)$

Linear Q-function $Q(x,u) = \sum_{i=1}^{n} w_i f_i(x,u)$

Sample: $r + \gamma \max_{u'} Q(x', u')$

Difference: $\left[r + \gamma \max_{u'} Q(x', u')\right] - Q(x,u)$

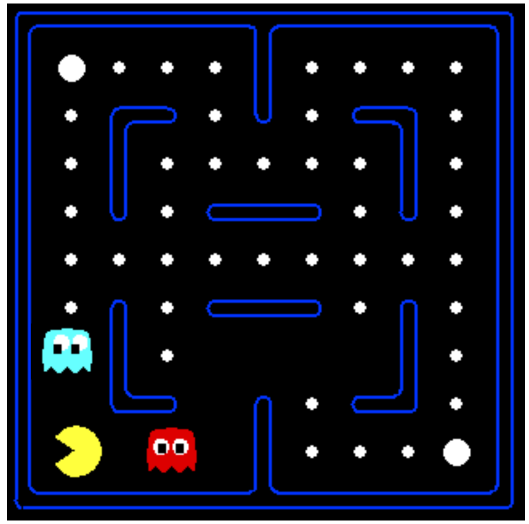
Update: $\forall i, \quad w_i \leftarrow w_i + \alpha\,[\text{difference}]\, f_i(x,u)$

- Intuitive interpretation:
  - Adjust weights of active features
  - E.g. if something unexpectedly bad happens, disprefer all states with that state's features
- Formal justification: online least squares on

$$\sum_{i=1}^{n} w_i f_i(x,u) = \text{sample}$$

Example: Q-Pacman

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$



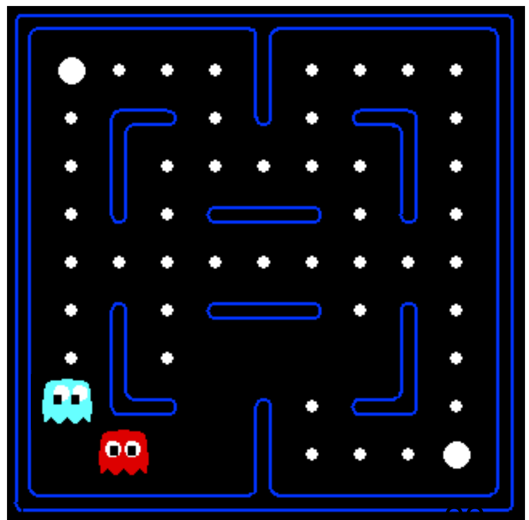
$$f_{DOT}(s, \mathrm{NORTH}) = 0.5$$

$$f_{GST}(s, \mathrm{NORTH}) = 1.0$$

$$Q(s,a) = +1$$

$$R(s,a,s') = -500$$

$$error = -501$$

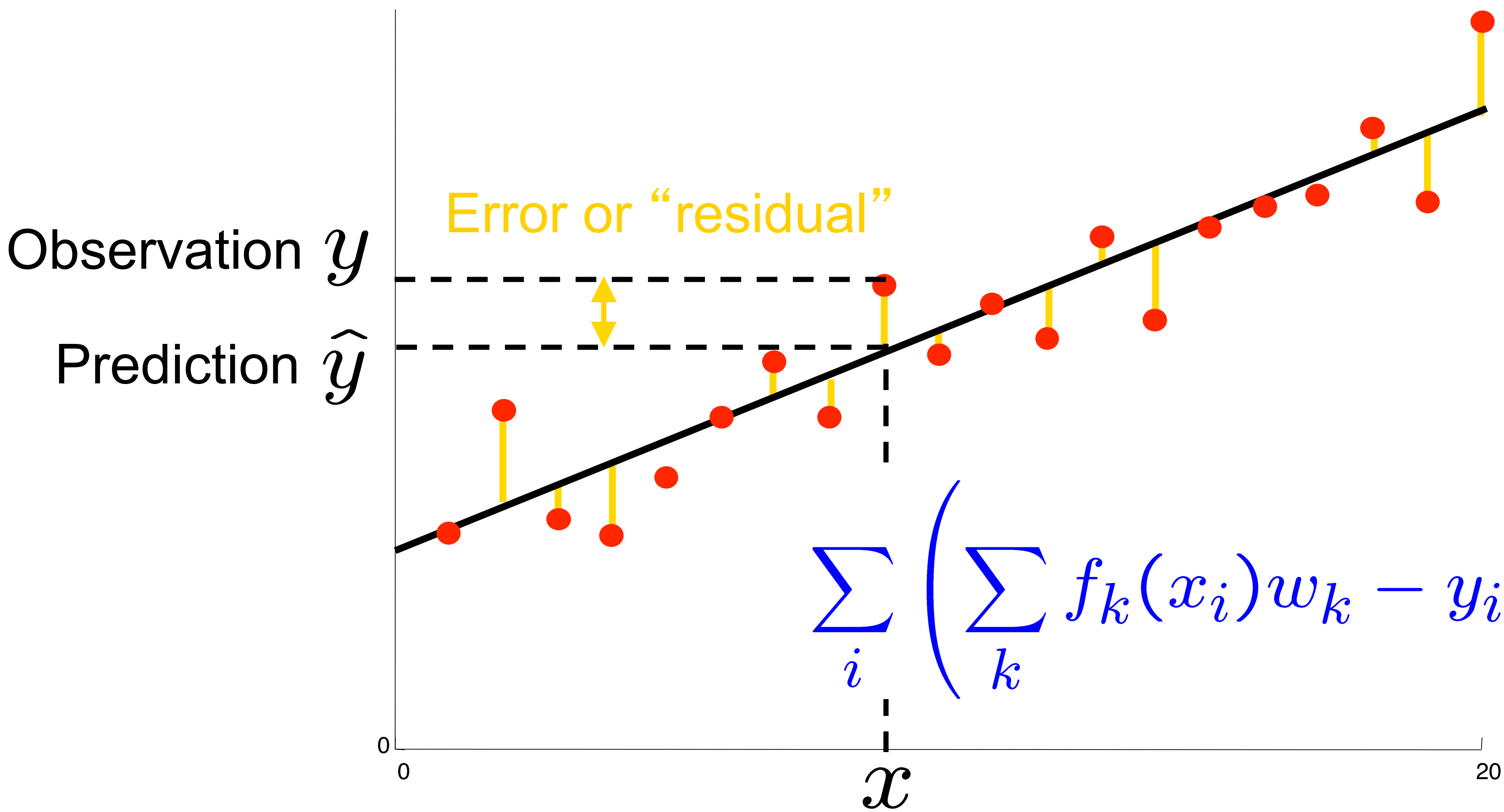


$$w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$$

$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

Ordinary Least Squares (OLS)



$$\sum_i \left(\sum_k f_k(x_i) w_k - y_i \right)^2$$

Minimizing Error

$$E(w) = \frac{1}{2} \sum_i \left(\sum_k f_k(x_i) w_k - y_i \right)^2$$

$$\frac{\partial E}{\partial w_m} = \sum_i \left(\sum_k f_k(x_i) w_k - y_i \right) f_m(x_i)$$

$$E \leftarrow E + \alpha \sum_i \left(\sum_k f_k(x_i) w_k - y_i \right) f_m(x_i)$$

Value update explained:

$$w_i \leftarrow w_i + \alpha \, [error] \, f_i(s,a)$$

Function approximation

- Update we covered
  - = gradient descent on one sample
- → How about batch version?
  - = called fitted Q-iteration

Fitted Q-Iteration

- Assume Q-function of the form $Q(x, u; w)$
  - E.g.: $Q(x, u; w) = \sum_i w_i f_i(x,u)$

- Iterate for $k = 1, 2, \ldots$ (improve w in each iter)

  Obtain samples $(x^{(j)}, u^{(j)}, x'^{(j)}, r^{(j)}), j=1,2,\ldots,J$ (from model or from experience, and can keep set fixed or grow over time)

  Supervised learning on:

  $$w^{(k+1)} = \operatorname{argmin}_w \sum_j \text{loss}(\, Q(x^{(j)}, u^{(j)}; w), \ \text{sample}^{(j)}\,)$$

  $$\text{where sample}^{(j)} = r^{(j)} + \gamma \max_{u'} Q(x^{(j)\prime}, u'; w^{(k)})$$

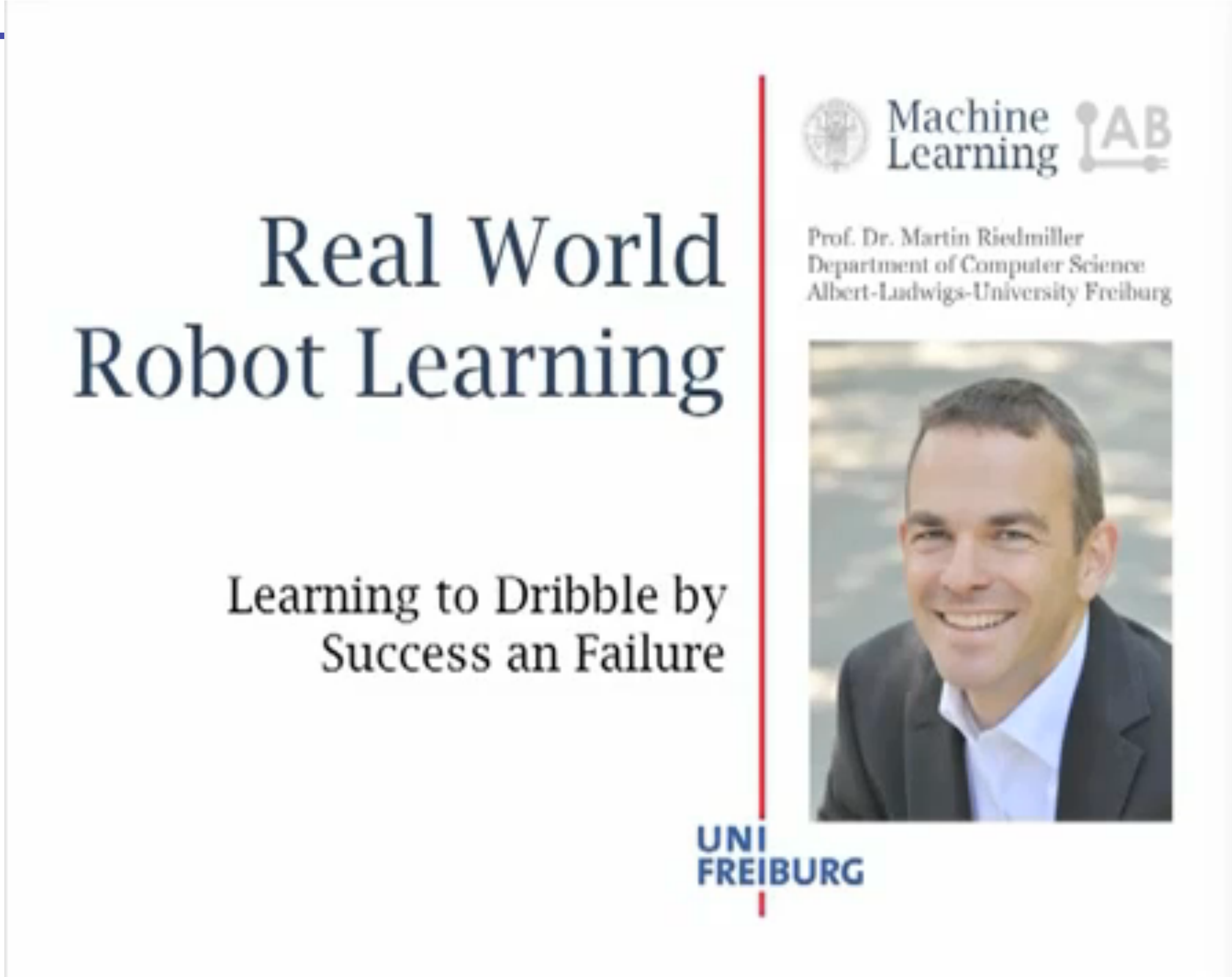
Outline

- Challenge: Most real-world problems have large, often infinite and continuous, state spaces

Value Function Methods:

- Model-free learning
  - Monte Carlo, TD-learning and Q-learning (tabular)
- Function approximation
  - Q-learning with feature-based representations
  - Fitted Q-learning
    - Often good approach, even when model is available

Fitted Q-iteration – demo



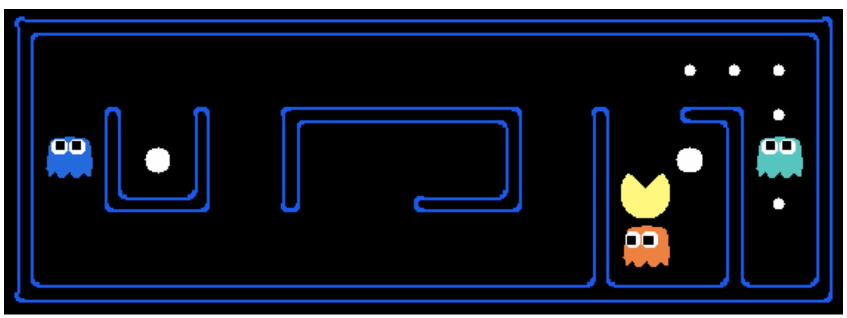
- Martin Riedmiller and collaborators

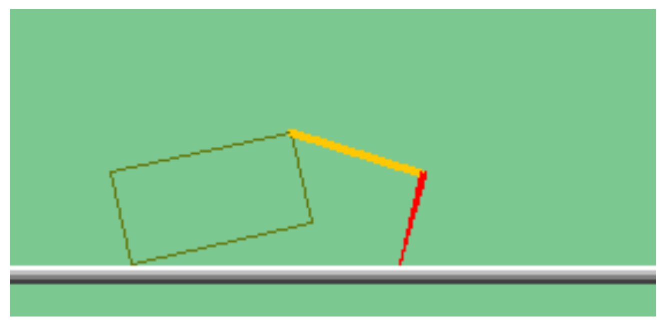
Fitted Q-iteration – demo

- Martin Riedmiller and collaborators
- Neural fitted Q-iteration, learning from scratch, without a model; growing batch: typically, improving the Q function and collecting the transitions is done in alternating fashion.
- Dribbling with soccer robots: difficult to solve analytically, due to physical interactions of robot and ball. First some random playing with the ball and then learn to dribble by rewarding the robot if it turns to the desired target direction without loosing the ball and punish it otherwise.
- Also: slot-car racing, cart and double pole, active suspension of a convertible car, steering of an autonomous car, magnetic levitation, ...

Mini Project! (Optional)

- Consolidate your understanding!
- Implement and experiment with
  - Value iteration
  - Q-learning
  - Q-learning with function approximation





- Time-frame: now and lunch break