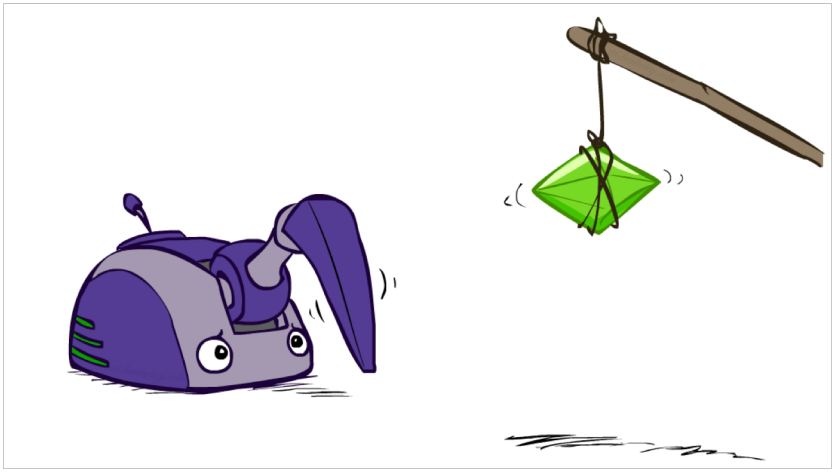
Outline for Today

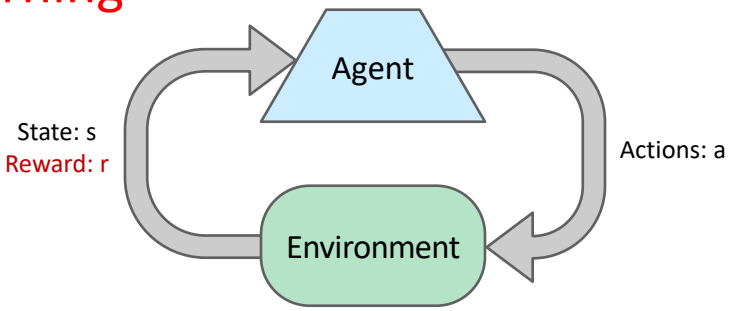
- Recap Q-Learning
- Approximate Q-Learning
- Hidden Markov Models
- Filtering
- Particle Filtering

3

Reinforcement Learning

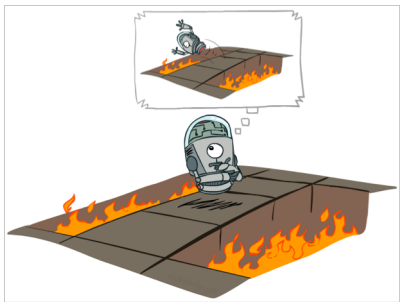


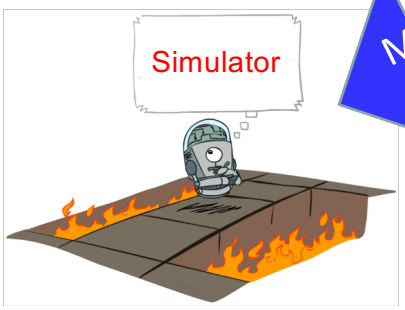
Reinforcement Learning



- MDP Framework, except: don't know transition funct. (T) or reward funct. (R)
- Must choose what action to execute so as to maximize expected rewards
- Receive feedback:
  - What state we end up in after executing a &
  - What reward we get
- Learn based on observed samples of outcomes!

Offline (MDPs) vs. Online (RL)





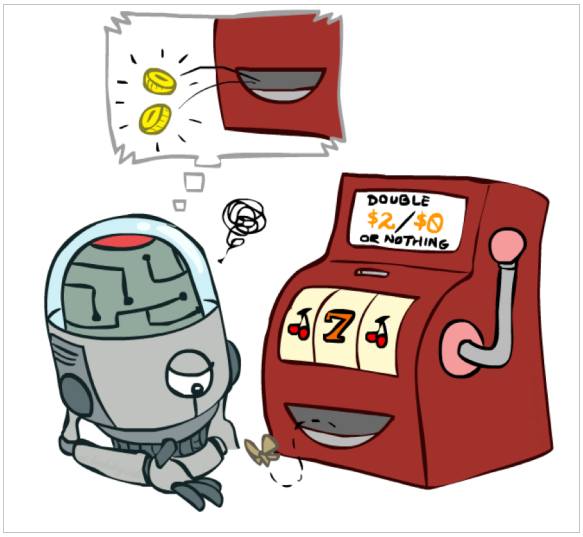
Offline Solution (Planning)

Monte Carlo Planning

Online Learning (RL)

Diff: 1) dying ok; 2) (re)set button

Model-Free Learning



Q Learning

- **Forall s, a**
  - **Initialize Q(s, a) = 0**
- **Repeat Forever**

  Where are you? s.

  Choose some action a

  Execute it in real world: *(s, a, r, s')*

  Do update:

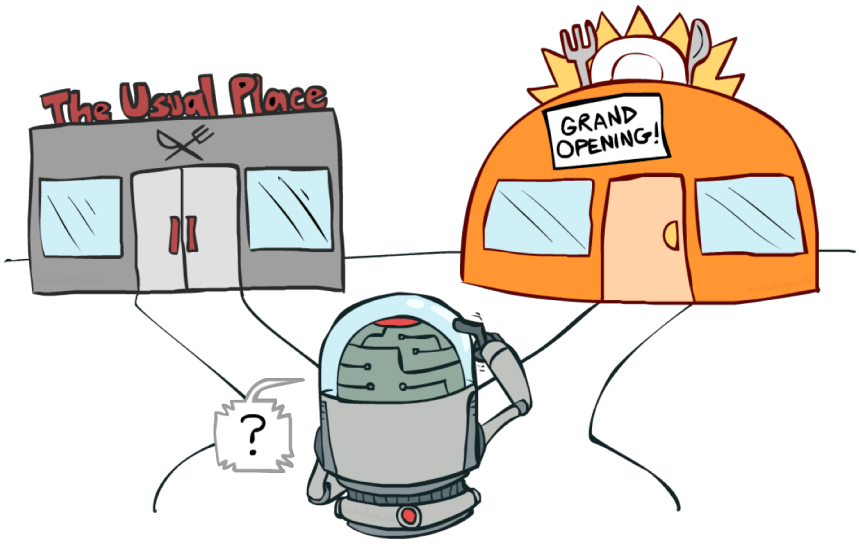
  difference ← $[r + \gamma \text{ Max}_{a'} Q(s', a')] - Q(s,a)$

  $Q(s,a) \leftarrow Q(s,a) + \alpha(\text{difference})$

Trial

Note parallel to RTDP
- Both have trials
- But no T, R

Exploration vs. Exploitation

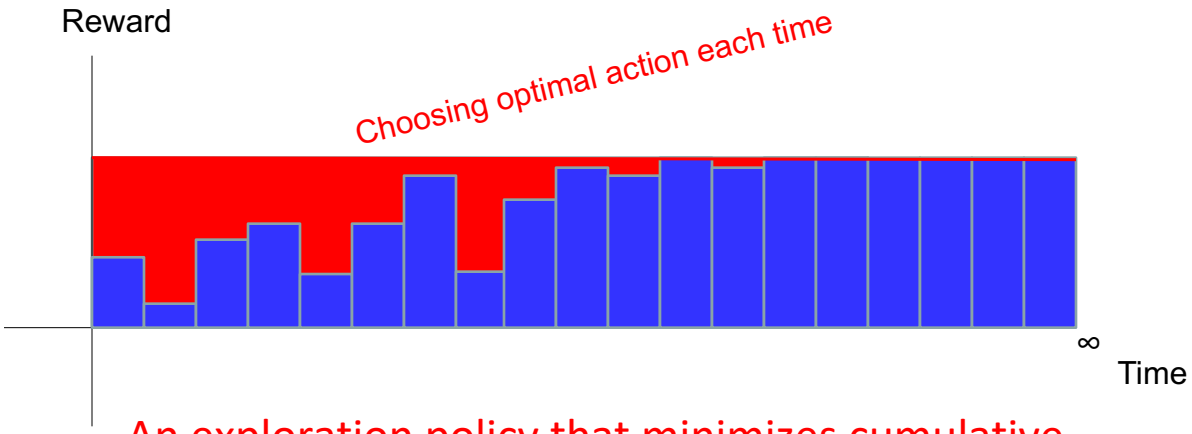


Want to Minimize ***Regret***

- **Cumulative Regret:**
  - Goal: achieve near optimal cumulative lifetime reward (in expectation)
  - Minimize this if acting in real world
- **Simple Regret:**
  - Goal: quickly identify policy with high reward (in expectation)
  - Minimize this if learning in simulator

51

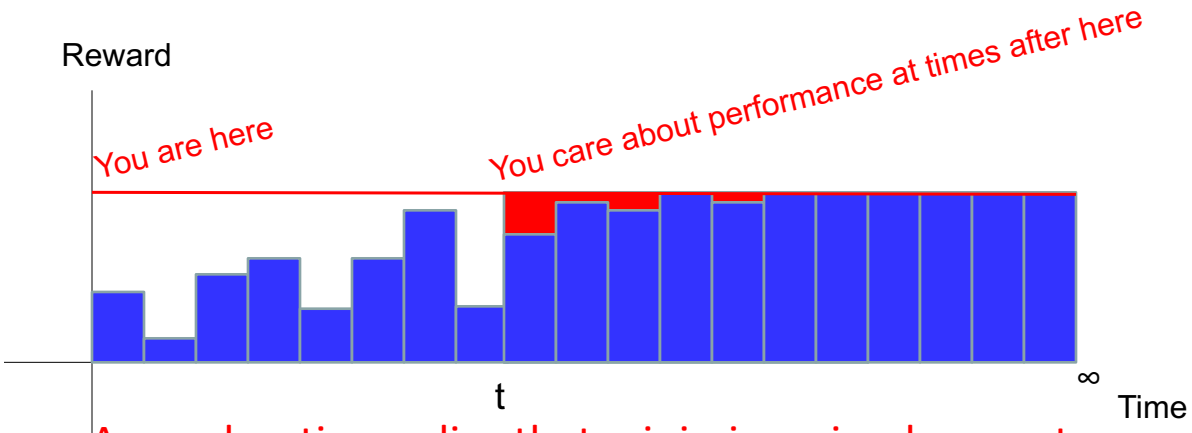
Cumulative Regret



An exploration policy that minimizes cumulative regret, minimizes the red area

52

Simple Regret

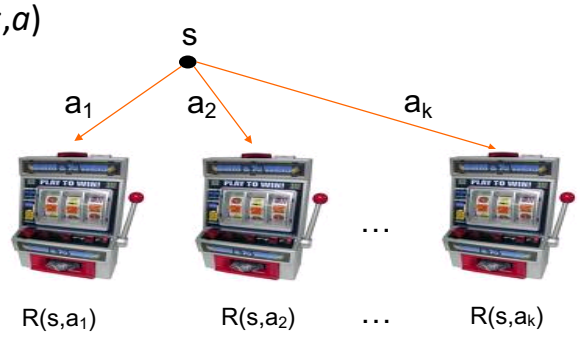


An exploration policy that minimizes simple regret…
Is given a time, t, in the ***future***, and explores in order to ***minimize red area after t***

53

RL on Single State MDP

- Suppose MDP has a single state and k actions
  - Can sample rewards of actions using call to simulator
  - Sampling action *a* is like pulling slot machine arm with random payoff function *R*(*s*,*a*)



Multi-Armed Bandit Problem

Slide adapted from **Alan Fern** (OSU)

Upper Confidence Bound (UCB)

1. Execute each action (pull each arm) once.
2. For N = 1 to infinity, repeat:

   Execute the action, a_i, that maximizes:

$$ExpectedReward(a_i) + \sqrt{\frac{2\log(N)}{NumberOfTimesExecuted(a_i)}}$$

Theorem: No algorithm can achieve lower expected **cumulative regret**, up to constant factors!
(but Thompson Sampling also optimal and often better in practice)

Putting UCB into Q-Learning !!!!

- Dealing with multiple states
- (Multi armed bandit assumes ONE state)

UCB Balances Exploration & Exploitation

Let N_{sa} be number of times one has executed a in s; let $N = \sum_{sa} N_{sa}$

Let $Q^e(s,a) = Q(s,a) + \sqrt{2\log(N)/(1+n_{sa})}$

- **Forall s, a**
  - Initialize Q(s, a) = 0, $n_{sa} = 0$
- **Repeat Forever**
  Where are you? s.
  Choose action with highest Q^e
  Execute it in real world: *(s, a, r, s')*
  Do update:

N_{sa} += 1;
difference ← [r + γ $Max_{a'}$ $Q^e(s', a')$] - $Q^e(s,a)$
$Q(s,a) \leftarrow Q^e(s,a) + \alpha$(difference)

Term rewards exploration, but converges to zero

$$ExpReward(a_i) + \sqrt{\frac{2\log(N)}{NumberOfTimesExecuted(a_i)}}$$

Summary of Bandits in Theory

PAC Objective:

- **UniformBandit** is a simple PAC algorithm
- **MedianElimination** improves by a factor of log(k) and is optimal up to constant factors

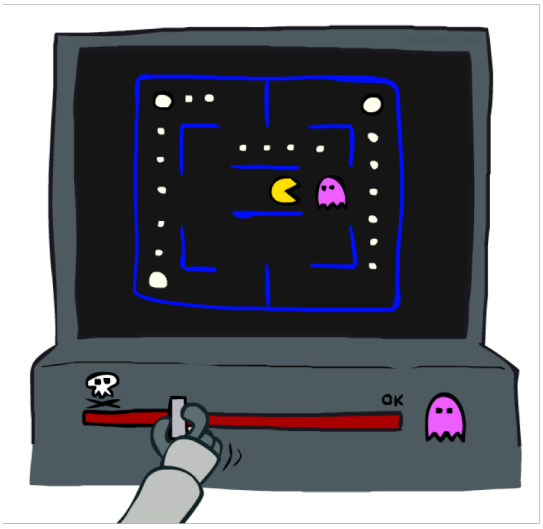
Cumulative Regret:

- **Uniform** is very bad!
- **UCB** is optimal (up to constant factors)
- **Thomson Sampling** also optimal; often performs better in practice

Simple Regret:

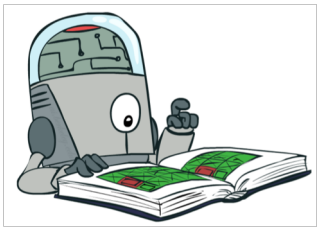
- **UCB** shown to reduce regret at polynomial rate
- **Uniform** reduces at an exponential rate
- **0.5-Greedy** may have even better exponential rate

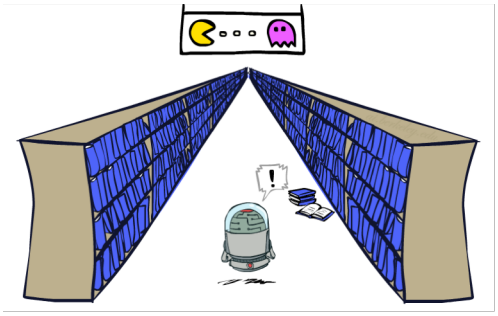
Approximate Q-Learning



Generalizing Across States

- Basic Q-Learning keeps a table of all q-values
- 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 situations
  - This is a fundamental idea in machine learning, and we'll see it over and over again

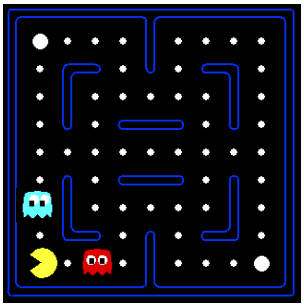




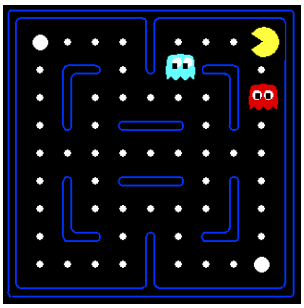
[demo – RL pacman]

Ex: Pacman – Failure to Generalize

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

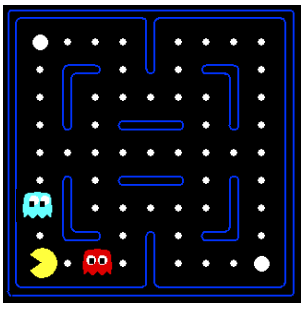


In naïve q-learning, we know nothing about this state:

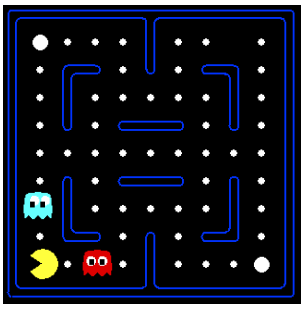


Ex: Pacman – Failure to Generalize

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



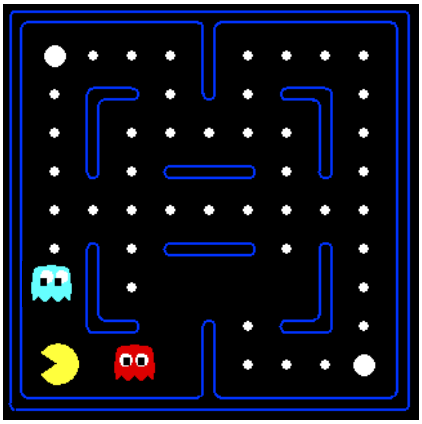
Or even this one!



Feature-Based Representations

Solution: describe a state using a **vector of features** (aka "properties")

- Features = functions from states to R (often 0/1) capturing important properties of the state
- Example features:
  - Distance to closest ghost or dot
  - Number of ghosts
  - 1 / (dist to dot)^2
  - Is Pacman in a tunnel? (0/1)
  - …… etc.
  - Is it the exact state on this slide?
    - Features -> inductive bias; ability to generalize
- Can also describe a q-state (s, a) with features (e.g. action moves closer to food)



Linear Combination of Features

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

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

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers
- Disadvantage: states sharing features may actually have very different values!

Q Learning

- **Forall s, a**
  - **Initialize Q(s, a) = 0**
- **Repeat Forever**

  Where are you? s.

  Choose some action a ; *e.g. using $\in$-greedy or by maximizing $Q_e(s, a)$*

  Execute it in real world: *(s, a, r, s')*

  Do update:

  difference ← [r + γ $\text{Max}_{a'}$ Q(s', a')] - Q(s,a)

  Q(s,a) ← Q(s,a) + α(difference)

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- **Forall *i***
  - Initialize $w_i = 0$
- **Repeat Forever**

  Where are you? s.

  Choose some action a

  Execute it in real world: *(s, a, r, s')*

  Do update:

  difference ← [r + γ $\text{Max}_{a'}$ Q(s', a')] - Q(s,a)

  Q(s,a) ← Q(s,a) + α(difference)

Approximate Q Learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- **Forall *i***
  - Initialize $w_i = 0$
- **Repeat Forever**

  Where are you? s.

  Choose some action a

  Execute it in real world: *(s, a, r, s')*

  Do update:

  difference ← [r + γ $\text{Max}_{a'}$ Q(s', a')] - Q(s,a)

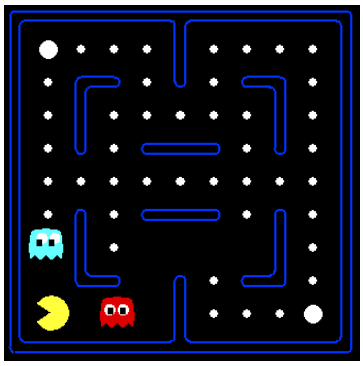
  Forall *i* do:

  $$w_i \leftarrow w_i + \alpha \, [\text{difference}] \, f_i(s,a)$$

Example: Pacman Features

$$Q(s,a) = w_1 f_{DOT}(s,a) + w_2 f_{GST}(s,a)$$

s



$$f_{DOT}(s,a) = \frac{1}{distance\ to\ closest\ food\ after\ taking\ a}$$

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

$$f_{GST}(s,a) = distance\ to\ closest\ ghost\ after\ taking\ a$$

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

$$Q(s,a) = w_1 f_{DOT}(s,a) + w_2 f_{GST}(s,a)$$

Suppose

W1 = 4.0

W2 = -1.0

α = 0.004

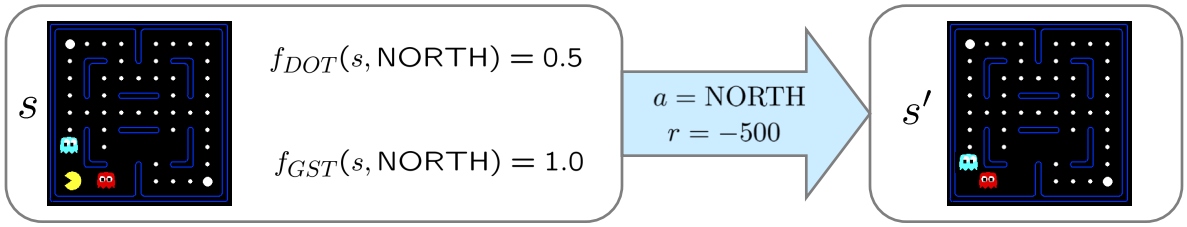
102

α = 0.004

Example: Q-Pacman

Forall *i* do:
$w_i \leftarrow w_i + \alpha$ [difference] $f_i(s,a)$

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



$Q(s, \text{NORTH}) = +1$

$Q(s', \cdot) = 0$

$r + \gamma \max_{a'} Q(s', a') = -500 + 0$

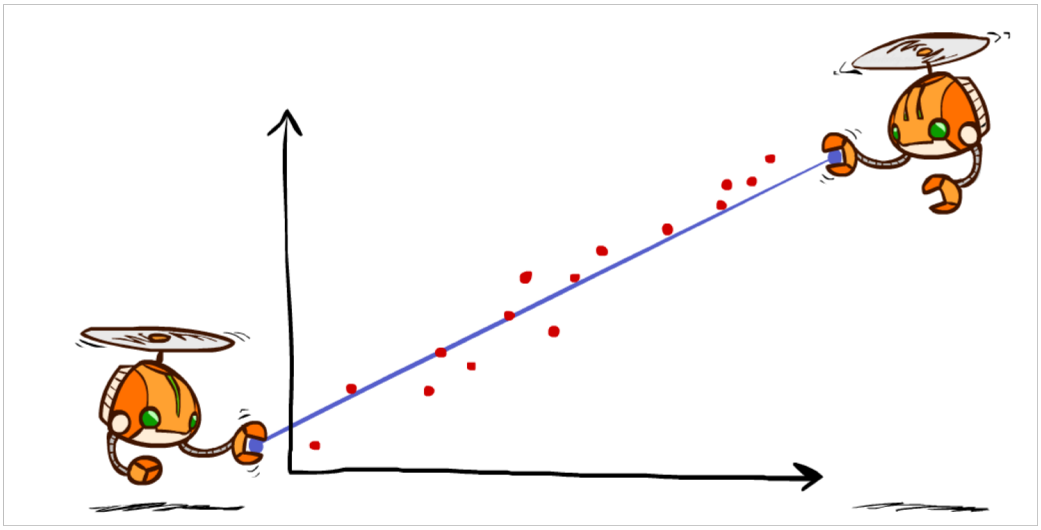
difference $= -501$ $\Rightarrow$ $w_{DOT} \leftarrow 4.0 + \alpha\,[-501]\,0.5$, $w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$

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

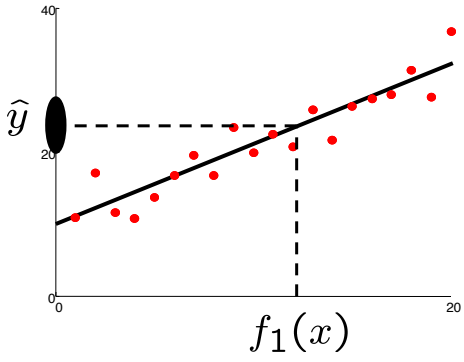
[Demo: approximate Q-learning pacman (L11D10)]

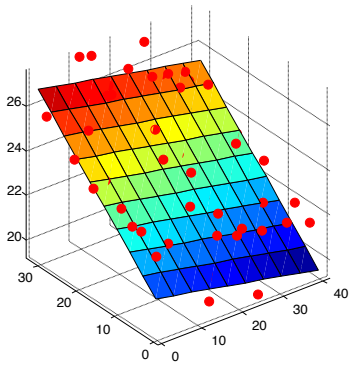
Video of Demo Approximate Q-Learning -- Pacman

Sidebar: Q-Learning and Least Squares



Linear Approximation: Regression





Prediction:

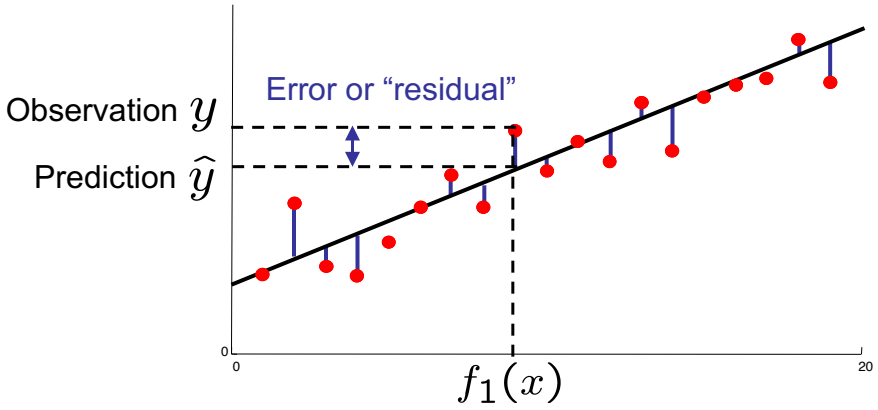
$\widehat{y} = w_0 + w_1 f_1(x)$

Prediction:

$\widehat{y}_i = w_0 + w_1 f_1(x) + w_2 f_2(x)$

Optimization: Least Squares

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i) \right)^2$$

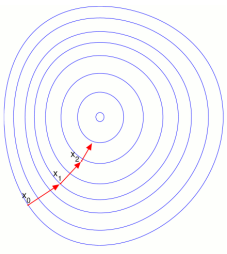


Minimizing Error

Suppose we have one point x, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2} \left(y - \sum_k w_k f_k(x) \right)^2$$

$$\frac{\partial \text{ error}(w)}{\partial w_m} = - \left(y - \sum_k w_k f_k(x) \right) f_m(x)$$



So to shrink error, repeat this update:

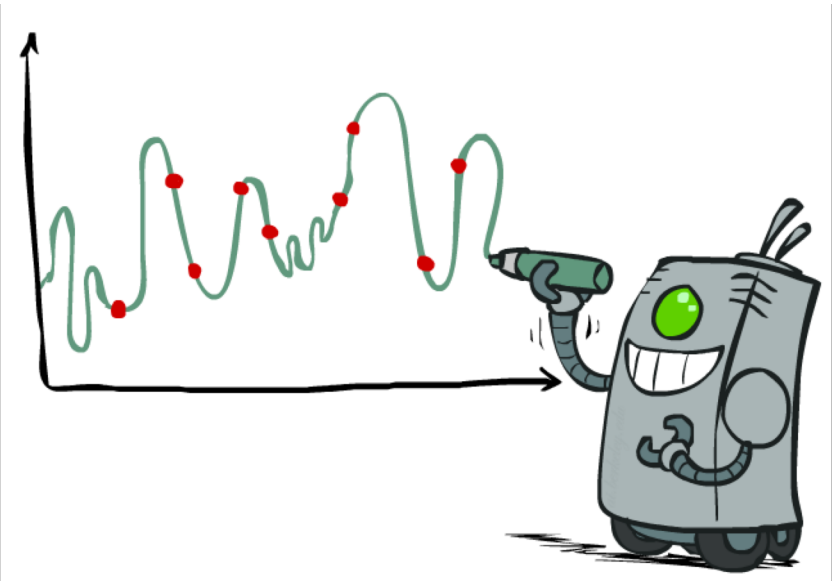
$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x) \right) f_m(x)$$

Approximate q update explained:

$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a) \right] f_m(s, a)$$

"target" "prediction"

Overfitting: Why Limiting Capacity Can Help



Simple Problem

Given: Features of current state
Predict: Will Pacman die on the next step?

110

Just one feature. See a pattern?

- Ghost one step away, pacman dies
- Ghost one step away, pacman dies
- Ghost one step away, pacman dies
- Ghost one step away, pacman dies
- Ghost one step away, pacman lives
- Ghost more than one step away, pacman lives
- Ghost more than one step away, pacman lives
- Ghost more than one step away, pacman lives
- Ghost more than one step away, pacman lives
- Ghost more than one step away, pacman lives
- Ghost more than one step away, pacman lives

Learn: Ghost one step away → pacman dies!

111

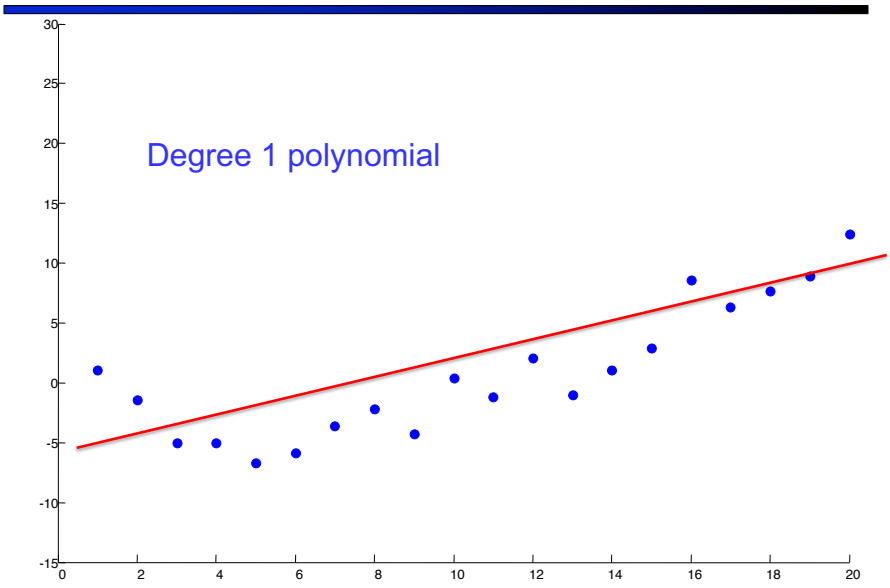
What if we add more features?

- Ghost one step away, score 211, pacman dies
- Ghost one step away, score 341, pacman dies
- Ghost one step away, score 231, pacman dies
- Ghost one step away, score 121, pacman dies
- Ghost one step away, score 301, pacman lives
- Ghost more than one step away, score 205, pacman lives
- Ghost more than one step away, score 441, pacman lives
- Ghost more than one step away, score 219, pacman lives
- Ghost more than one step away, score 199, pacman lives
- Ghost more than one step away, score 331, pacman lives
- Ghost more than one step away, score 251, pacman lives

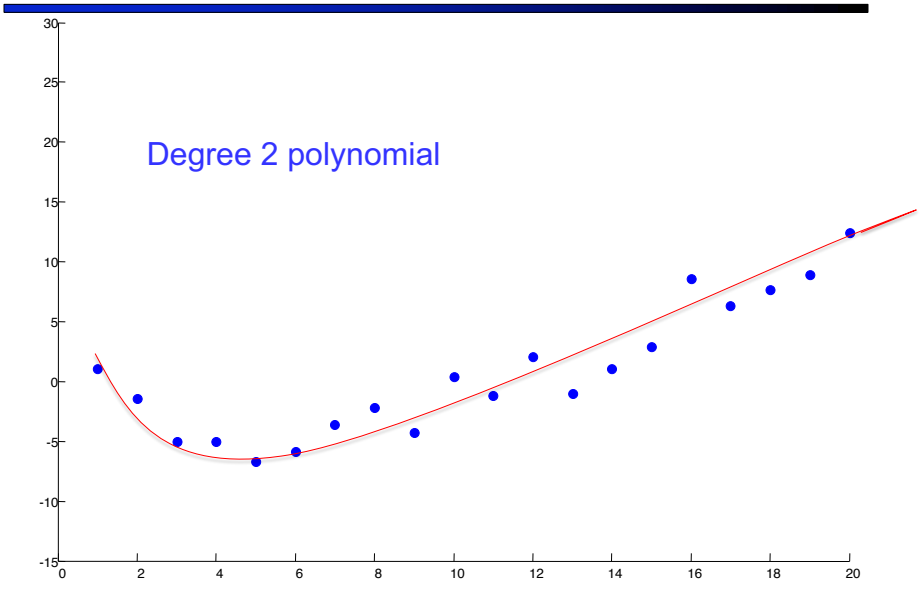
Learn: Ghost one step away AND score is NOT 301 → pacman dies!

112

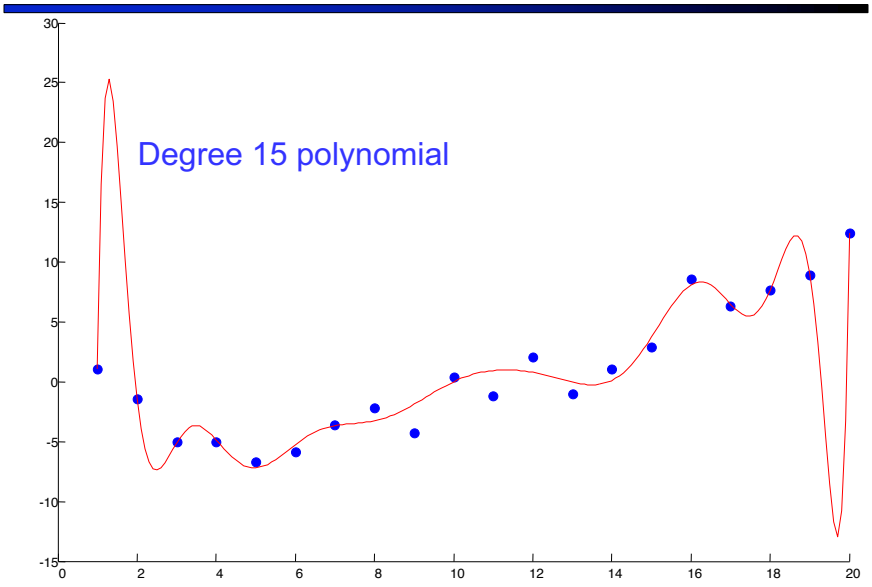
There's fitting, and there's



There's fitting, and there's

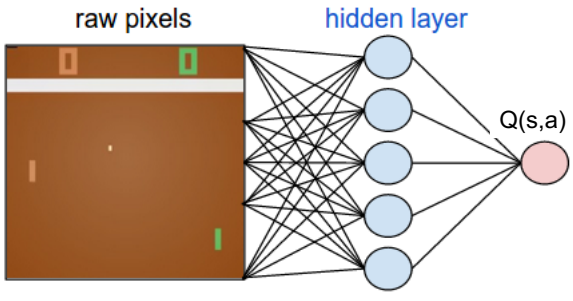


Overfitting



Approximating Q Function

- Linear Approximation
- Could also use Deep Neural Network
  - https://www.nervanasys.com/demystifying-deep-reinforcement-learning/



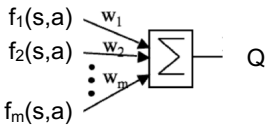
Deepmind Atari

https://www.youtube.com/watch?v=V1eYniJ0Rnk

Approximating the Q Function

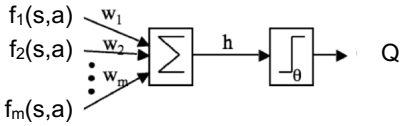
Linear Approximation

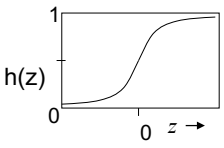
$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$



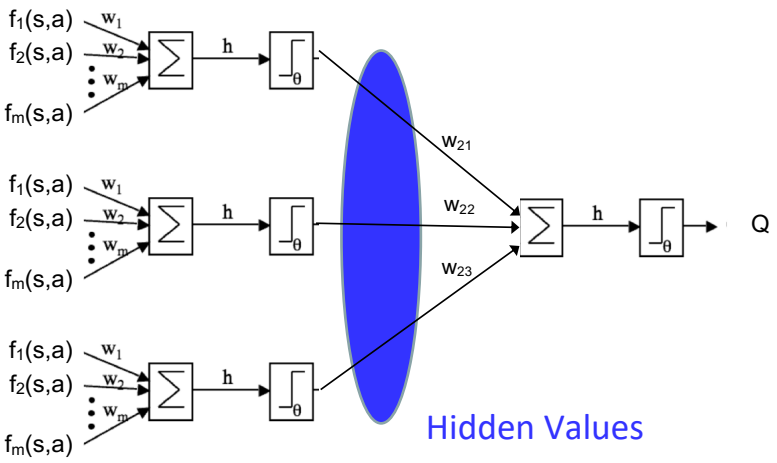
Neural Approximation (nonlinear)

$$h(z) = \frac{1}{1+e^{-z}}$$

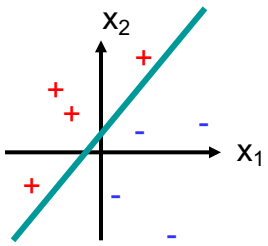




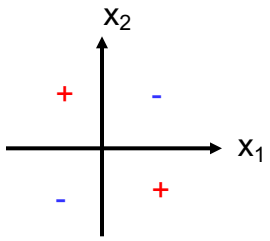
Deep Neural Networks



Decision Surface of a Perceptron



Linearly separable



Non-Linearly separable

- Perceptron is able to represent ***many*** useful functions
  - E.g., for AND(x_1, x_2) choose weights w_0=-1.5, w_1=1, w_2=1
- But functions that are not linearly separable are not representable
  - E.g. XOR(x_1, x_2)

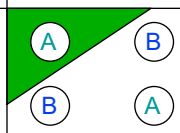
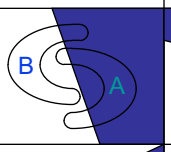
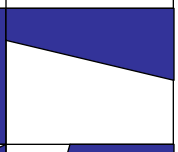
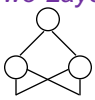
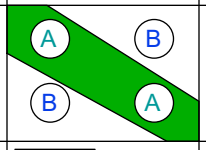
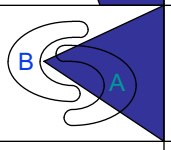
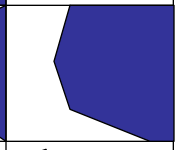
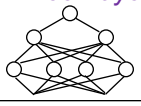
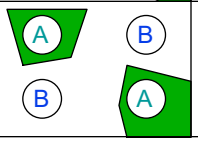
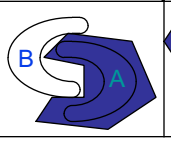
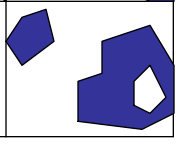
125

Slide by Moshe Sipper
www.cs.bgu.ac.il/~sipper/courses/ecal061/**ann-ics320Part4.ppt**

Hyperplane partitions

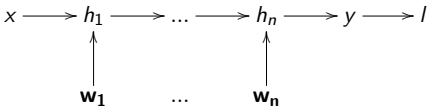
- An extra layer models a convex hull
  - "An area with no dents in it"
  - Perceptron models, but can't learn
  - Sigmoid function learning of convex hulls
  - Two layers add convex hulls together
  - Sufficient to classify anything "sane".
- In theory, further layers add nothing
- In practice, extra layers may be better

Different Non-Linearly Separable Problems

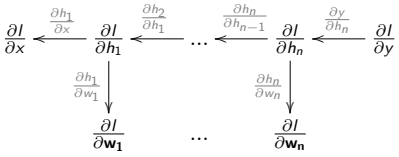
Structure	*Types of Decision Regions*	*Exclusive-OR Problem*	*Classes with Meshed regions*	*Most General Region Shapes*
Single-Layer 	*Half Plane Bounded By Hyperplane*			
Two-Layer 	*Convex Open Or Closed Regions*			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			

Deep Representations

- A deep representation is a composition of many functions



- Its gradient can be backpropagated by the chain rule



Slide adapted from David Silver

Training via Gradient Descent

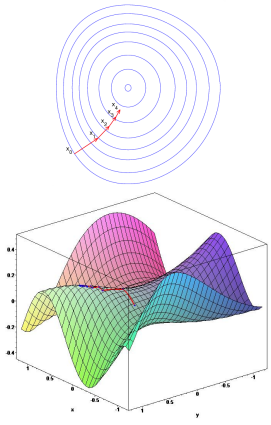
Loss = ML word for "error"

- Calculate gradient of expected loss $(\mathbf{w}) = \mathbb{E}[l]$

$$\frac{\partial l}{\partial \mathbf{w}} \sim \mathbb{E}\left[\frac{\partial l}{\partial \mathbf{w}}\right] = \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$$

- Adjust **w** down the gradient

$$\Delta w \propto \frac{\partial l}{\partial \mathbf{w}}$$



Slide adapted from David Silver

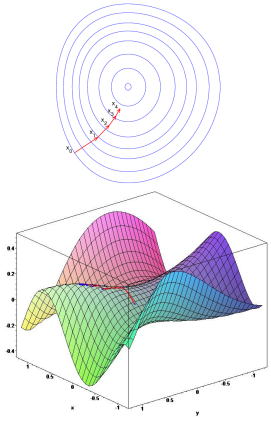
Training *via* ***Stochastic*** Gradient Descent

- Sample gradient of expected loss $L(\mathbf{w}) = \mathbb{E}[l]$

$$\frac{\partial l}{\partial \mathbf{w}} \sim \mathbb{E}\left[\frac{\partial l}{\partial \mathbf{w}}\right] = \frac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$$

- Adjust $\mathbf{w}$ down the Sampled gradient

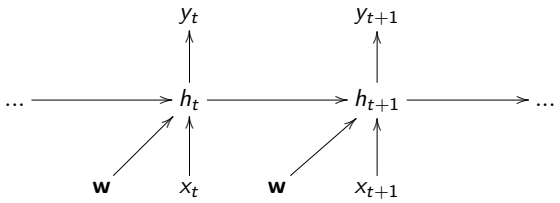
$$\Delta w \propto \frac{\partial l}{\partial \mathbf{w}}$$



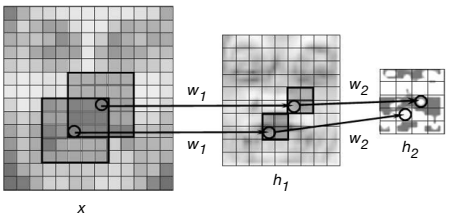
Slide adapted from David Silver

Weight Sharing

Recurrent neural network shares weights between time-steps



Convolutional neural network shares weights between local regions

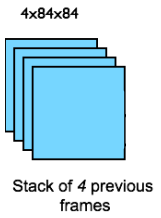


Slide adapted from David Silver

How Do Authors Represent an Atari State?

Raw pixels:
210 x 160 x 128 colors

Down-sampled to
84 x 84 x grey



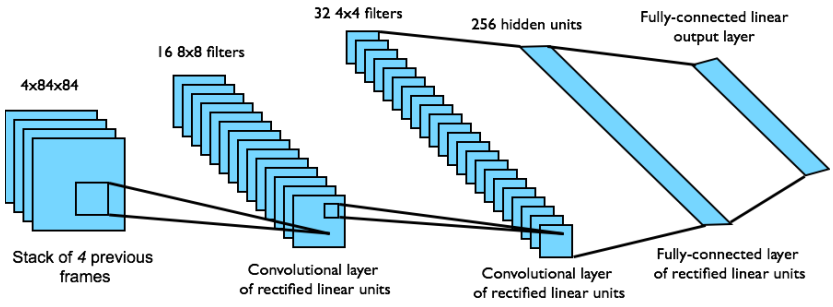
134

Represent Q Function Using DQN

- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step

Raw pixels:
210 x 160 x 128 colors

Down-sampled to
84 x 84 x grey



Network architecture and hyperparameters fixed across all games

Slide adapted from David Silver

Recap: Approx Q-Learning

- Optimal Q-values should obey Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s',a')^* \mid s,a\right]$$

- Treat right-hand side $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as a target
- Minimise MSE loss by stochastic gradient descent

$$I = \left(r + \gamma \max_{a} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

- Converges to Q^* using table lookup representation
- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets

Slide adapted from David Silver

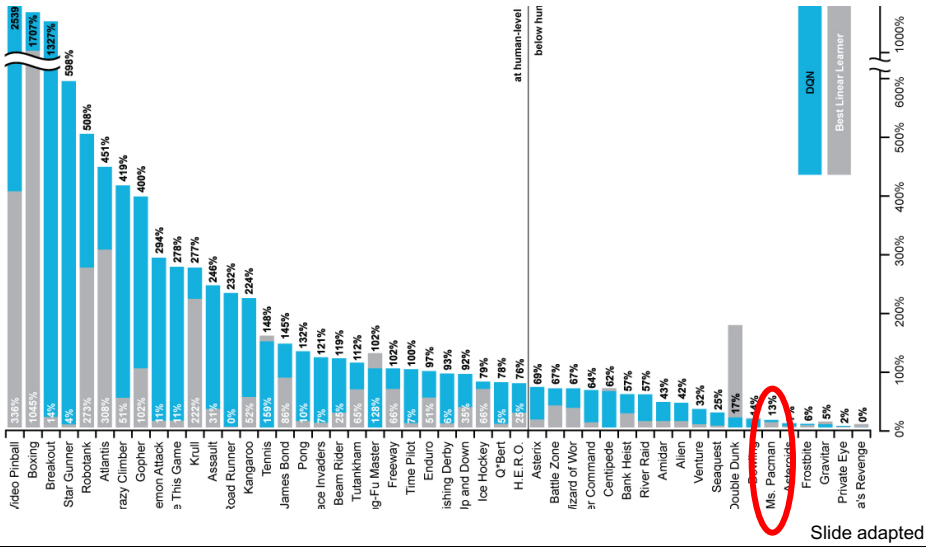
Why use Experience Replay?

- Maximize utility of data
- Avoid updates on correlated game states
  - I.I.D. assumption
- Avoid local maxima / divergence

137

DQN Results on Atari

Which game characteristics → good / bad performance?



Slide adapted from David Silver

Evaluation?

Generally quite convincing…

Google bought Deepmind one month after paper appeared on ArXiv: $500M

- Fair comparison with human 'experts'?
- Effect of parameter values?
  - Action every k frames, clipping rewards, …
- Effect of experience replay?
- Effect of NN architecture (# layers etc)
- Does it work on go / chess?

140

Takeaways

- Deep RL obviates hand features
- Approximate Q with nonlinear approx (NN)
  - Convergence?
- Use sequences (histories?) not individual actions
  - Markov?
- Challenge of sparse rewards, long time spans

141

Deep Mind Resources

DQN paper
www.nature.com/articles/nature14236

DQN source code:
sites.google.com/a/deepmind.com/dqn/

See also: http://icml.cc/2016/tutorials/deep_rl_tutorial.pdf

Playing Atari with Deep RL

Breakout:
https://www.youtube.com/watch?v=V1eYniJ0Rnk

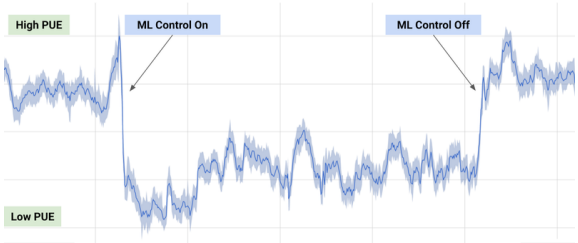
More games (less interesting)
https://www.youtube.com/watch?v=iqXKQf2BOSE

Paper
https://arxiv.org/pdf/1312.5602.pdf)

143

That's all for Reinforcement Learning!

- Very tough problem: How to perform any task well in an unknown, noisy environment!
- Traditionally used mostly for robotics, but…



Google DeepMind – RL applied to data center power usage

148