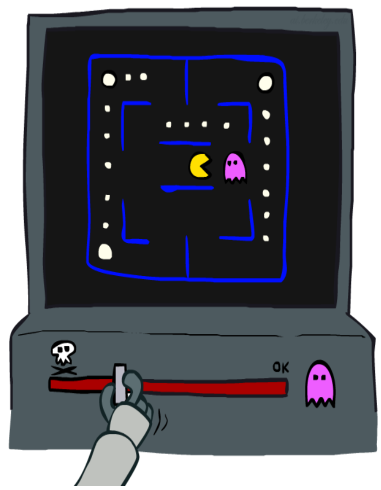
Approximate Q-Learning



Dan Weld / University of Washington

[Many slides taken from Dan Klein and Pieter Abbeel / CS188 Intro to AI at UC Berkeley – materials available at http://ai.berkeley.edu.]

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:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + (\alpha)\left[r + \gamma \max_{a'} Q(s',a')\right]$$

$$\left.\begin{aligned} \text{difference} &= \left[r + \gamma \max_{a'} Q(s',a')\right] - Q(s,a) \\ Q(s,a) &\leftarrow Q(s,a) + \alpha\,[\text{difference}] \end{aligned}\right\} \text{Equivalently}$$

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')*

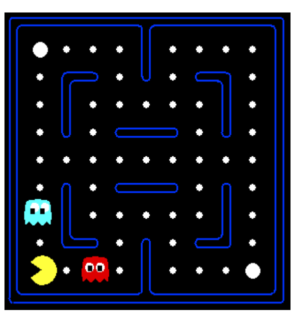
Do update:

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s', a')\right] - Q(s, a)$$

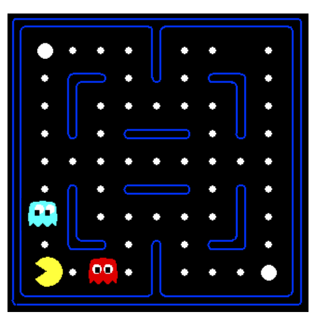
$$Q(s, a) \leftarrow Q(s, a) + \alpha\,[\text{difference}]$$

Example: Pacman

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



Or even this one!



Q-learning, no features, 50 learning trials

QuickTime™ and a
GIF decompressor
are needed to see this picture.

Q-learning, no features, 1000 learning trials:

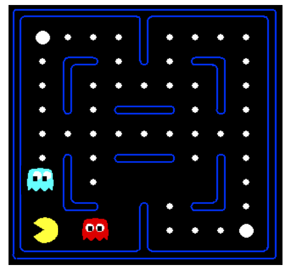
QuickTime™ and a
GIF decompressor
are needed to see this picture.

Feature-Based Representations

Soln: describe states w/ **vector of features** (aka "properties")

- Features = functions from states to R (often 0/1) capturing important properties of the state
- Examples:
  - Distance to closest ghost or dot
  - Number of ghosts
  - 1 / (dist to dot)^2
  - Is Pacman in a tunnel? (0/1)

    …… etc.
  - Is state the exact state on this slide?



- Can also describe a q-state (s, a) with features (e.g. action moves closer to food)

How to use features?

Using features we can represent V and/or Q as follows:

$$V(s) = g(f_1(s), f_2(s), \ldots, f_n(s))$$

$$Q(s,a) = g(f_1(s,a), f_2(s,a), \ldots, f_n(s,a))$$

What should we use for g?

(and f)?

Linear Combination

- 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!

Approximate Q-Learning

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

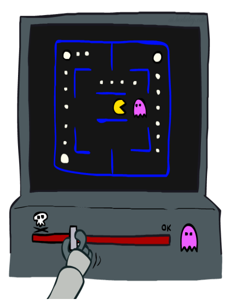
- Q-learning with linear Q-functions:

$$\text{transition } = (s,a,r,s')$$

$$\text{difference} = \left[r + \gamma \max_{a'} Q(s',a')\right] - Q(s,a)$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\,[\text{difference}]$$ Exact Q's

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



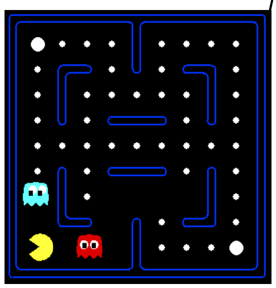
- Intuitive interpretation:
  - Adjust weights of active features
  - E.g., if something unexpectedly bad happens, blame the features that were on: ***disprefer all states with that state's features***

- Formal justification: in a few slides!

Example: Pacman Features

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

s



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

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

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

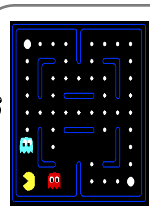
$$f_{GST}(s, NORTH) = 1.0$$

Example: Q-Pacman

α = 0.004

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

s



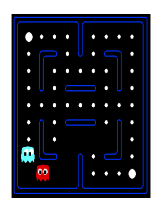
$f_{DOT}(s, \text{NORTH}) = 0.5$

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

$a = \text{NORTH}$

$r = -500$

s'



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

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

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

difference $= -501$

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

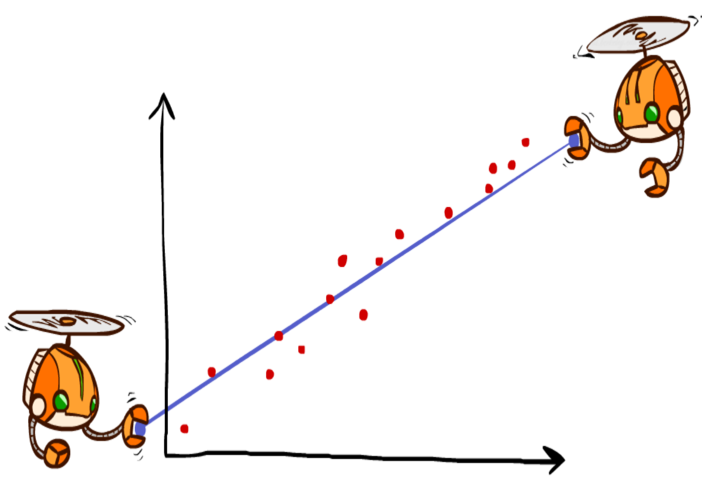
$$w_{GST} \leftarrow -1.0 + \alpha\,[-501]\,1.0$$

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

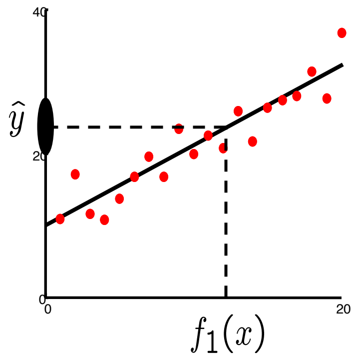
[Demo: approximate Q-

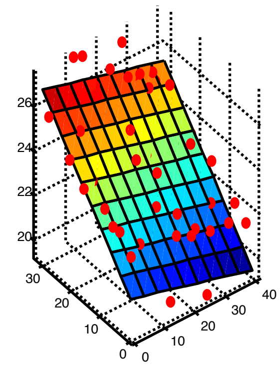
Video of Demo Approximate Q-Learning -- Pacman

Sidebar: Q-Learning and Least Squares



Linear Approximation: Regression



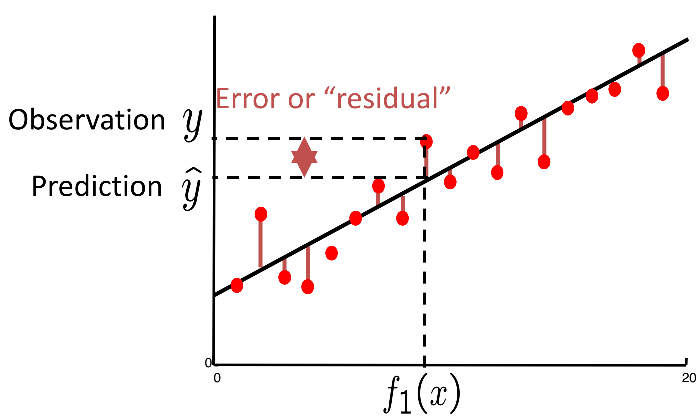


Prediction:
$$\hat{y} = w_0 + w_1 f_1(x)$$

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

Optimization: Least Squares

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



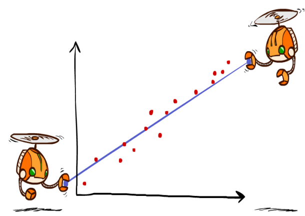
Minimizing Error

Imagine we had only one point x, with features f(x), target value y, and weights w:

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

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

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

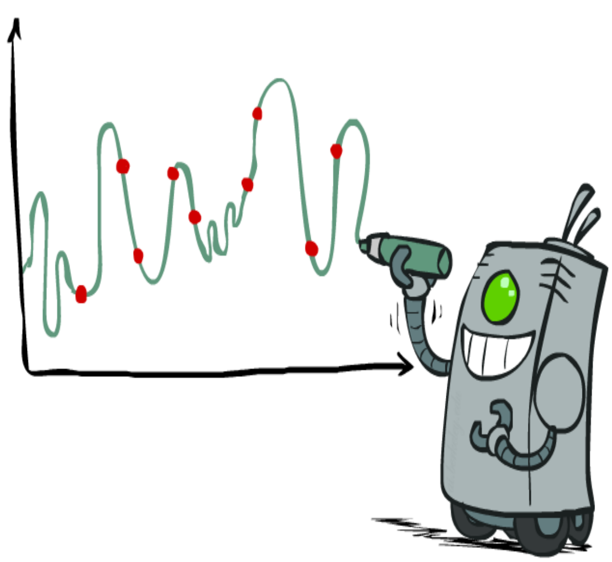


Approximate q update explained:

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

"target" "prediction"

Overfitting: Why Limiting Capacity Can Help



Simple Problem

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

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!

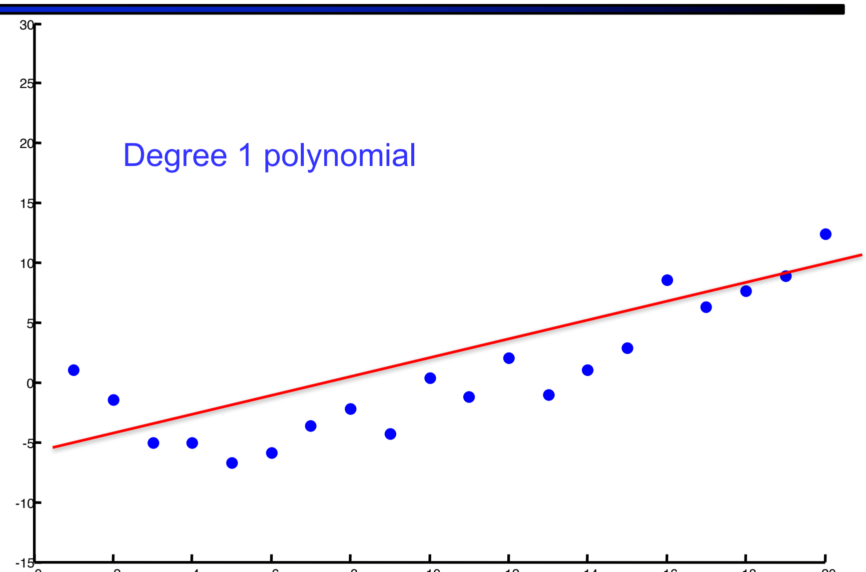
What if we add more features?

- 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

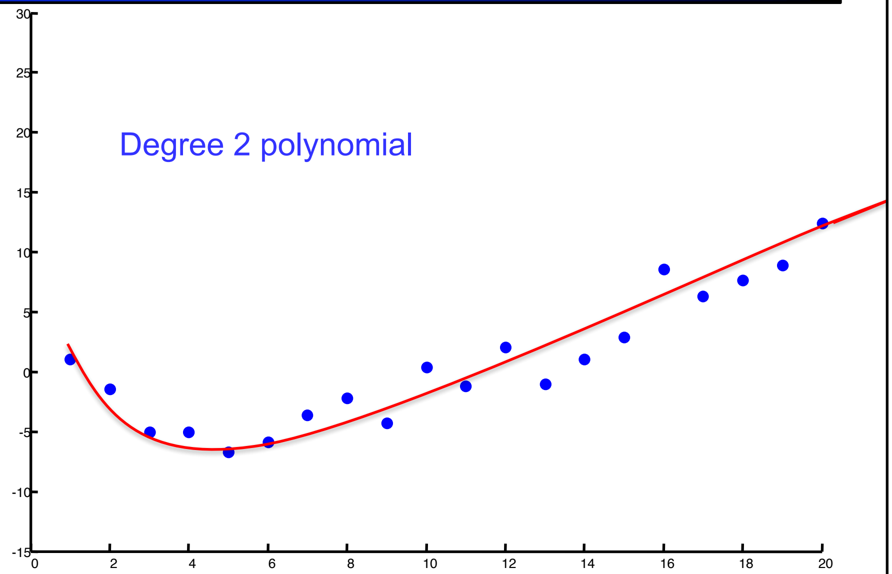
Learn: Ghost one step away AND score is NOT prime number → pacman dies!

24

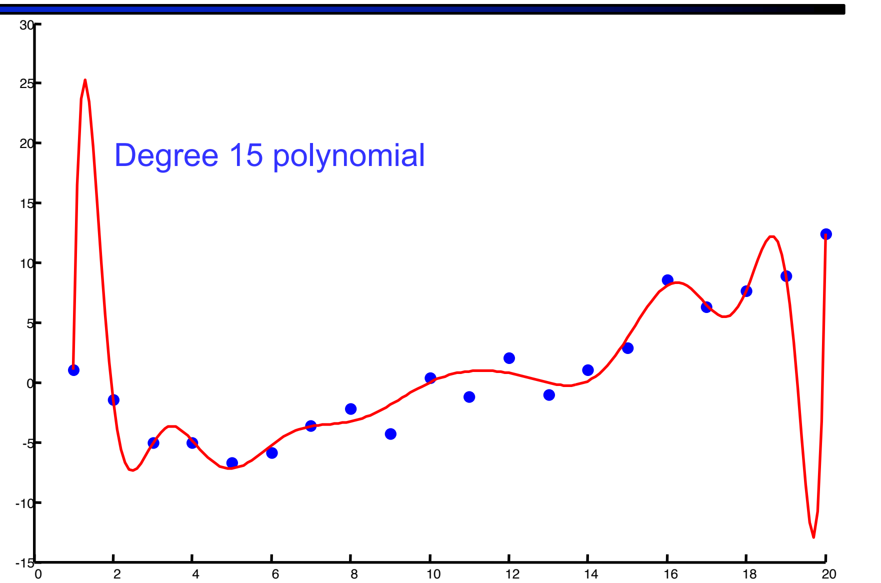
There's fitting, and there's



There's fitting, and there's

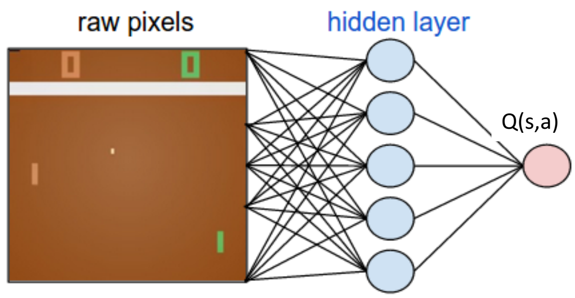


Overfitting



Approximating Q Function

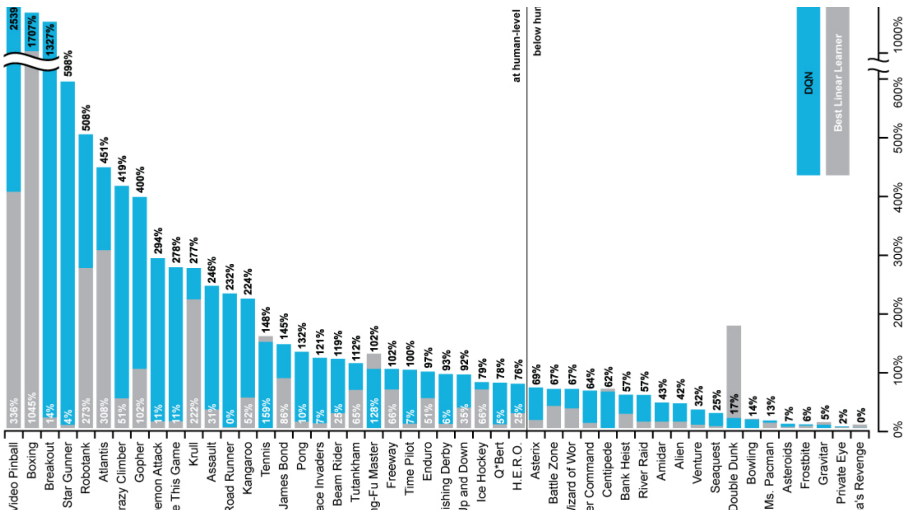
- Linear Approximation
- Could also use Deep Neural Network
  - https://www.nervanasys.com/demystifying-deep-reinforcement-learning/



Deepmind Atari

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

DQN Results on Atari

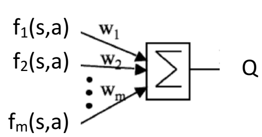


Slide adapted from David Silver

Approximating the Q Function

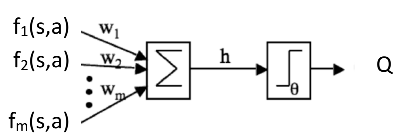
Linear Approximation

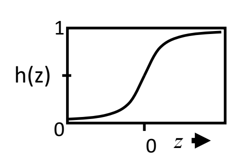
$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$



Neural Approximation (nonlinear)

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





Deep Representations

- A deep representation is a composition of many functions

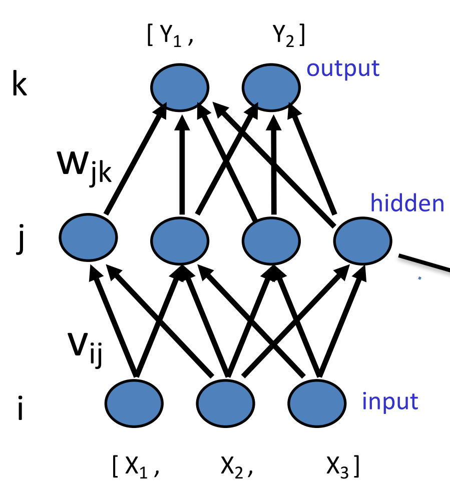
$$
\begin{tikzcd}
x \arrow[r] & h_1 \arrow[r] & \ldots \arrow[r] & h_n \arrow[r] & y \arrow[r] & l \\
& \mathbf{w_1} \arrow[u] & \ldots & \mathbf{w_n} \arrow[u] & &
\end{tikzcd}
$$

- Its gradient can be backpropagated by the chain rule

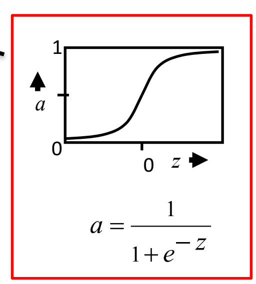
$$
\begin{tikzcd}
\frac{\partial l}{\partial x} & \frac{\partial l}{\partial h_1} \arrow[l, "\frac{\partial h_1}{\partial x}"'] \arrow[d, "\frac{\partial h_1}{\partial w_1}"'] & \ldots \arrow[l, "\frac{\partial h_2}{\partial h_1}"'] & \frac{\partial l}{\partial h_n} \arrow[l, "\frac{\partial h_n}{\partial h_{n-1}}"'] \arrow[d, "\frac{\partial h_n}{\partial w_n}"'] & \frac{\partial l}{\partial y} \arrow[l, "\frac{\partial y}{\partial h_n}"'] \\
& \frac{\partial l}{\partial \mathbf{w_1}} & \ldots & \frac{\partial l}{\partial \mathbf{w_n}} &
\end{tikzcd}
$$

Slide adapted from David Silver

Multi Layer Perceptron



- Multiple Layers
- Feed Forward
- Connected Weights $z_j = \sum_i x_i w_{ij}$
- 1-of-N Output



Training via Stochastic Gradient Descent

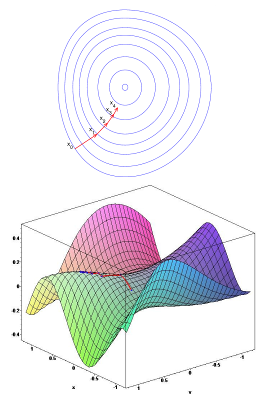
Loss = fancy ML word for "error"

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

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

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

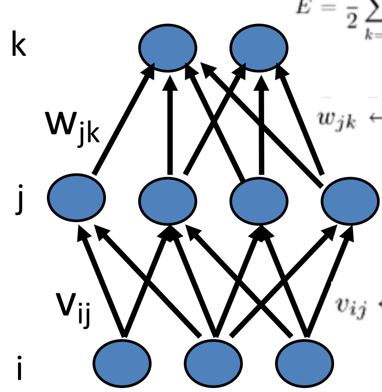
$$\Delta w \propto \frac{\partial l}{\partial \mathbf{w}}$$



Slide adapted from David Silver

Aka ... Backpropagation

$$E = \frac{1}{2}\sum_{k=1}^{N}(t_k - y_k)^2$$



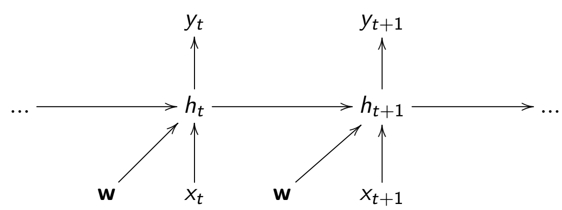
$$w_{jk} \leftarrow w_{jk} - \eta\frac{\partial E}{\partial w_{jk}}$$

$$v_{ij} \leftarrow v_{ij} - \eta\frac{\partial E}{\partial v_{ij}}$$

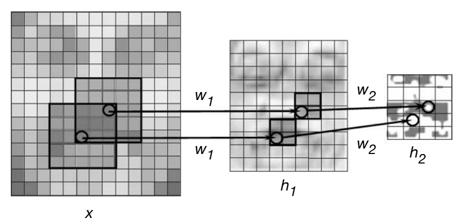
- Minimize error of calculated output
- Adjust weights
  - Gradient Descent
- Procedure
  - Forward Phase
  - Backpropagation of errors
- For each sample, multiple epochs

Weight Sharing

Recurrent neural network shares weights between time-steps



Convolutional neural network shares weights between local regions



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

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

- 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

Deep Q-Networks (DQN) Experience Replay

To remove correlations, build data-set from agent's own experience

		s_1, a_1, r_2, s_2	
		s_2, a_2, r_3, s_3	$\rightarrow$ s, a, r, s'
		s_3, a_3, r_4, s_4	
		...	
$s_t, a_t, r_{t+1}, s_{t+1}$	$\rightarrow$	$s_t, a_t, r_{t+1}, s_{t+1}$	

Sample experiences from data-set and apply update

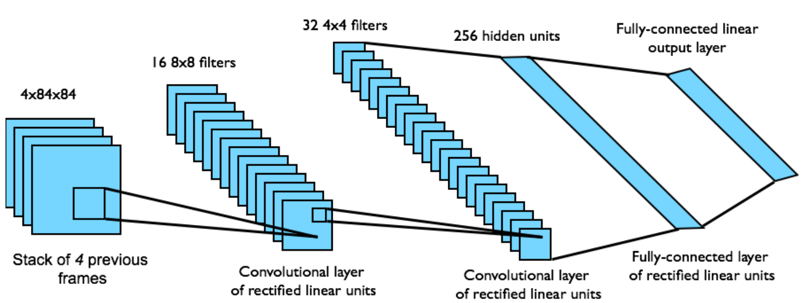
$$l = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

To deal with non-stationarity, target parameters $\mathbf{w}^-$ are held fixed

Slide adapted from David Silver

DQN in Atari

- 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



Network architecture and hyperparameters fixed across all games

Slide adapted from David Silver

Deep Mind Resources

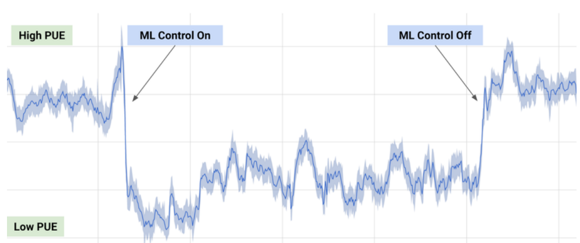
DQN paper
www.nature.com/articles/nature14236

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

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

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

49

That's all for Reinforcement Learning!

Data (experiences with environment) → Reinforcement Learning Agent → Policy (how to act in the future)

Lots of open research areas:

- How to best balance exploration and exploitation?
- How to deal with cases where we don't know a good state/feature representation?

50

Conclusion

- We're done with Part I: Search and Planning!
- We've seen how AI methods can solve problems in:
  - Search
  - Constraint Satisfaction Problems
  - Games
  - Markov Decision Problems
  - Reinforcement Learning
- Next up: Part II: Uncertainty and Learning!