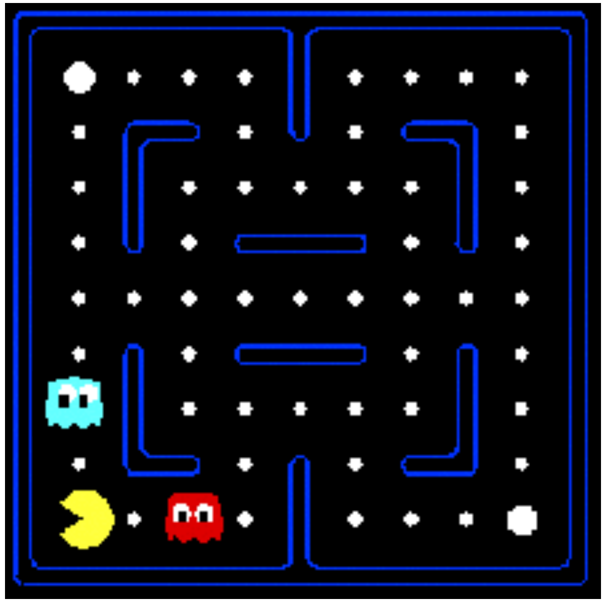
The next few slides based on those of Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley.

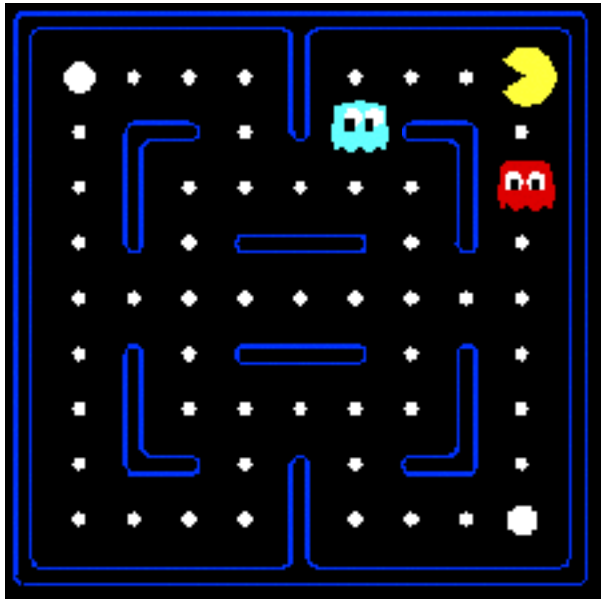
All CS188 materials are available at http://ai.berkeley.edu.

Example: Pacman

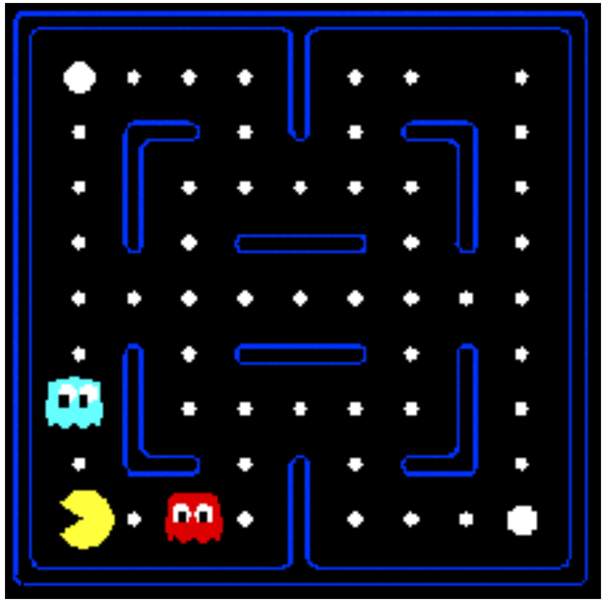
Let's say we discover through experience that this state is bad:



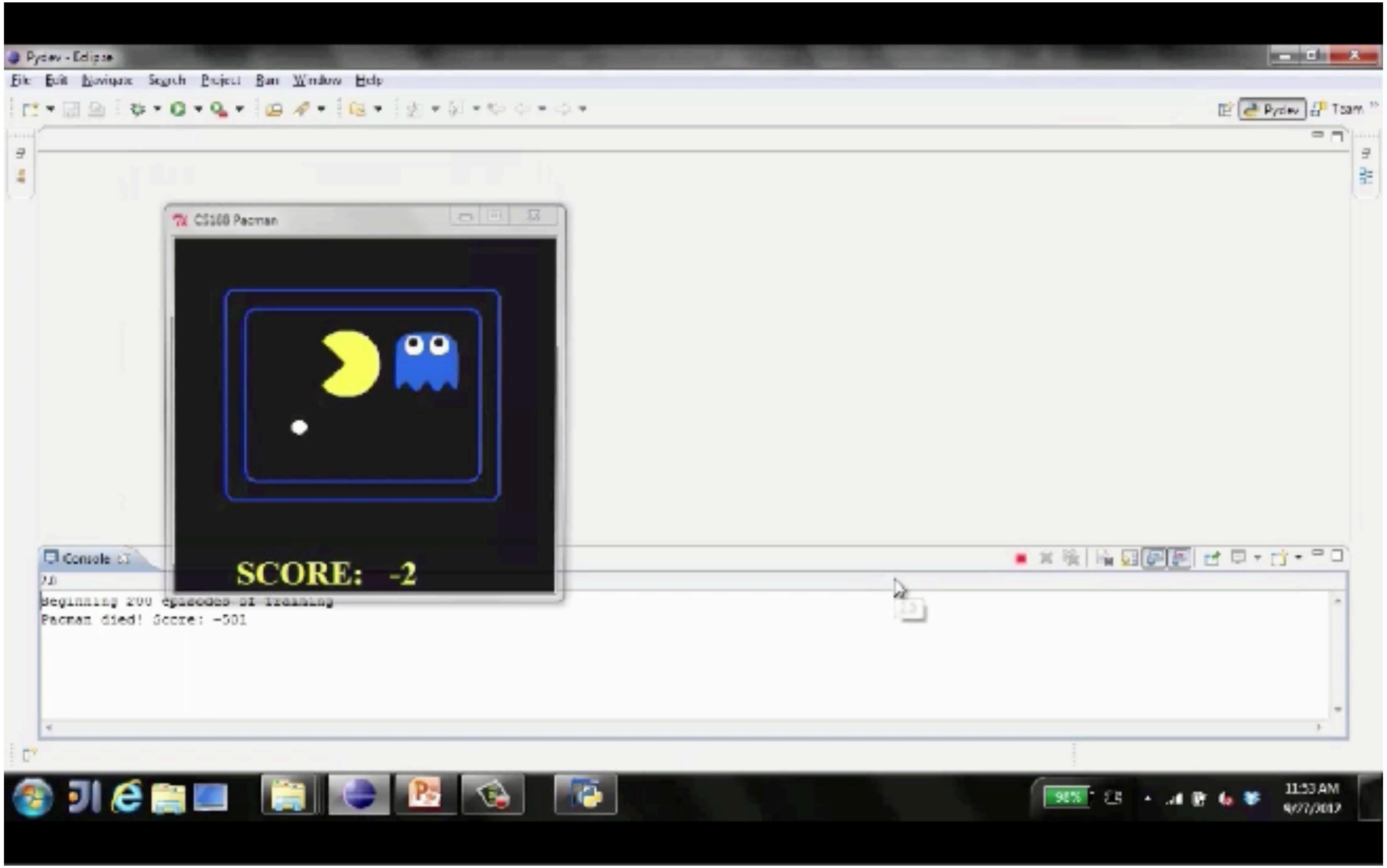
In naïve q-learning, we know nothing about this state:



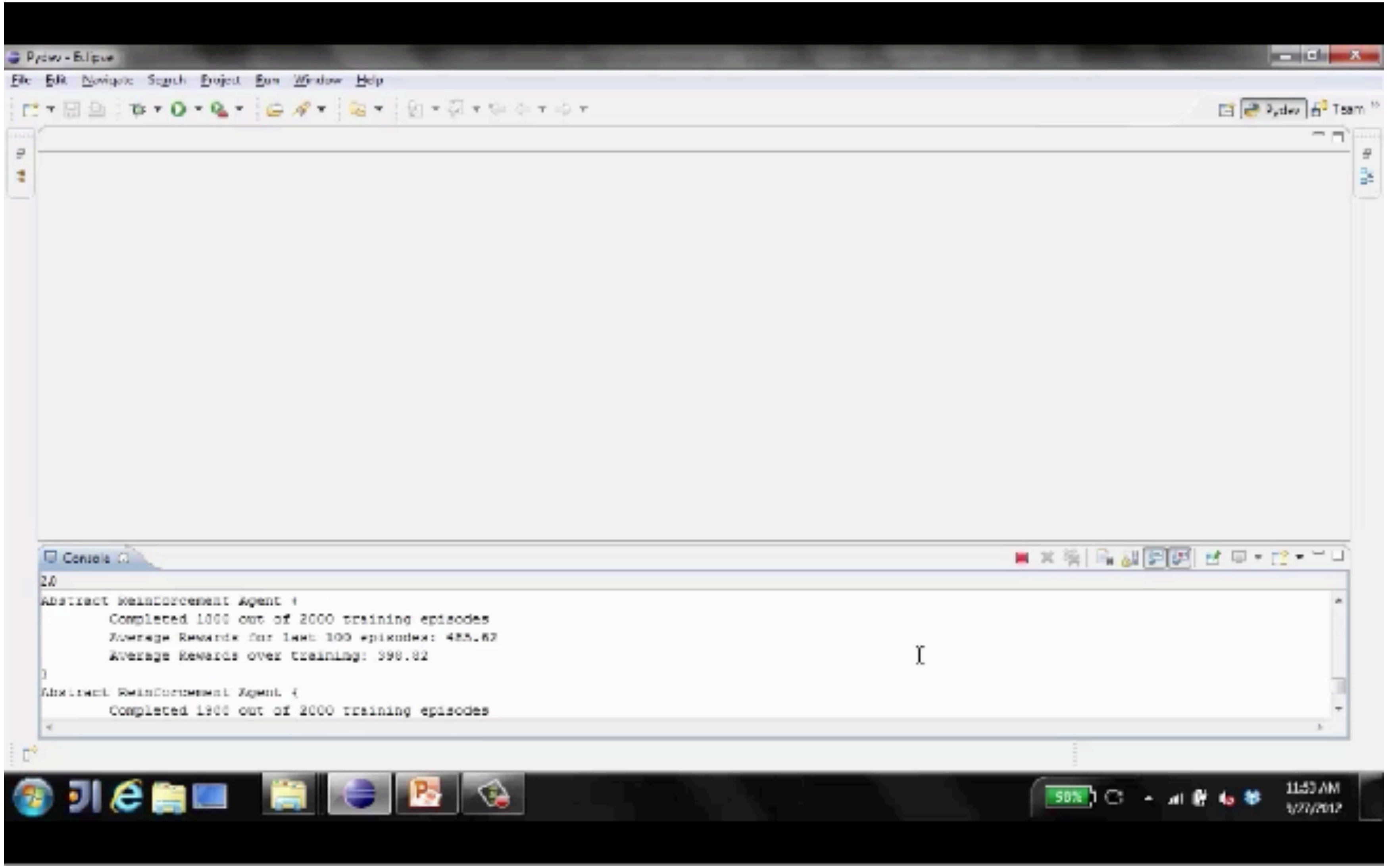
Or even this one!



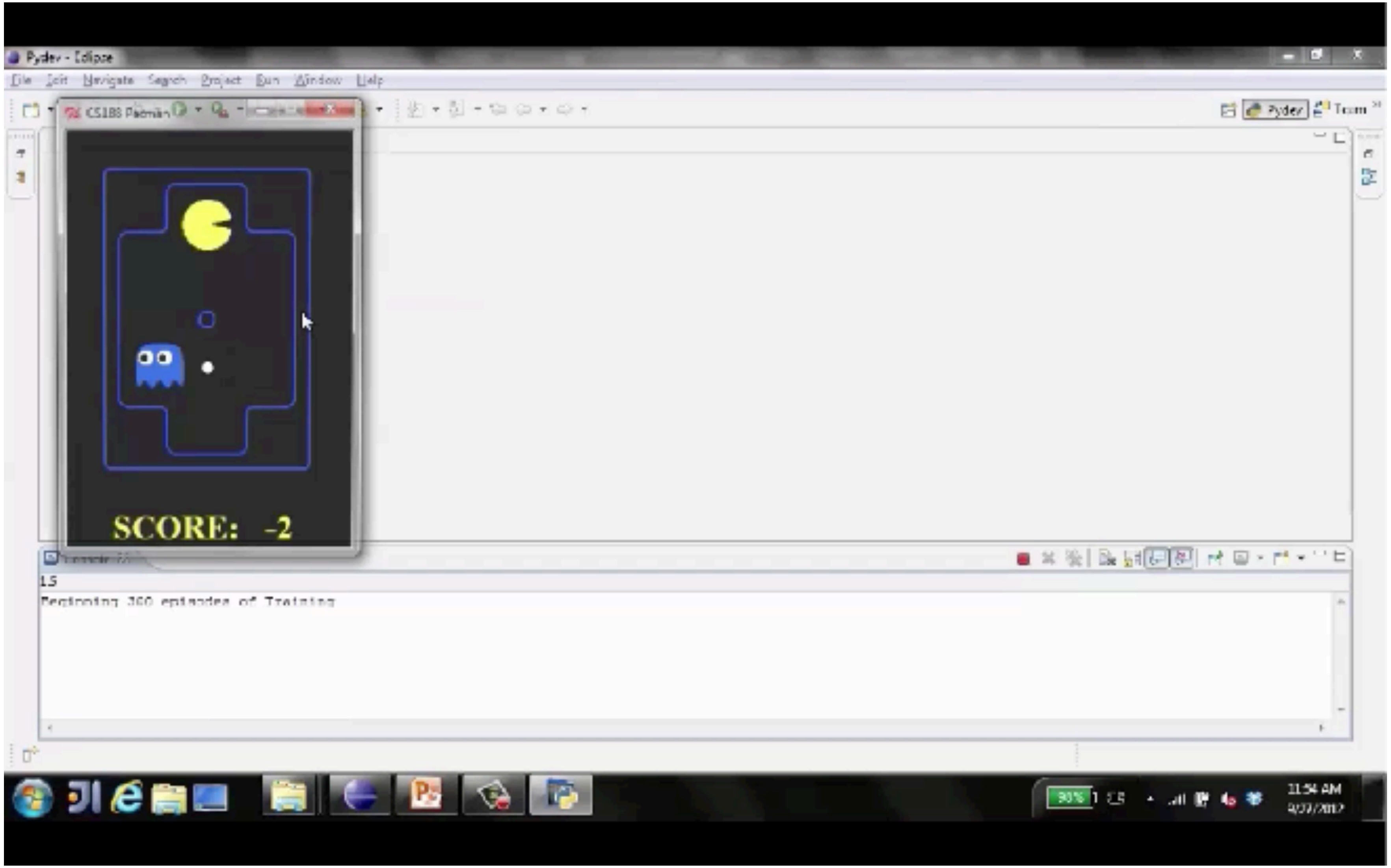
No generalization

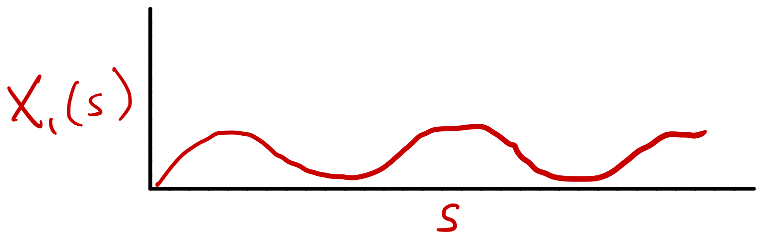


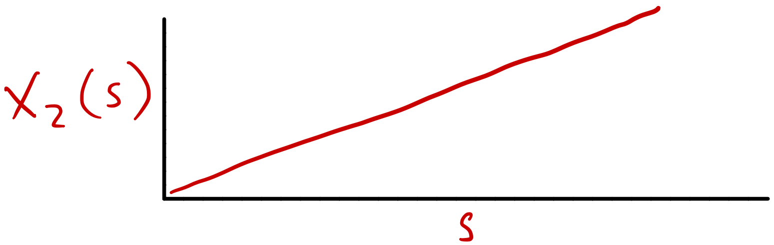
2000 episodes later…



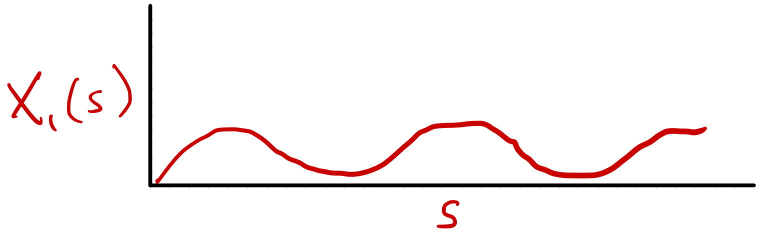
Harder maze, no generalization

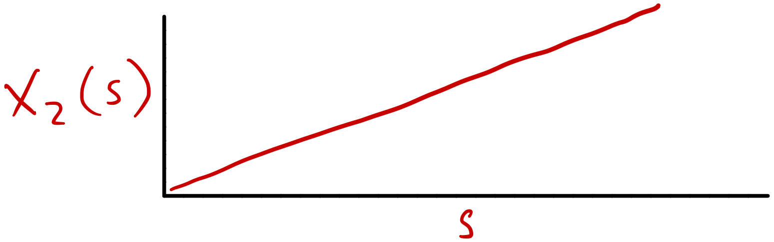






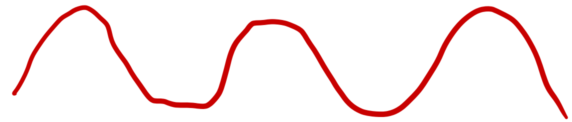
$$V(s) = w_1 X_1(s) + w_2 X_2(s)$$



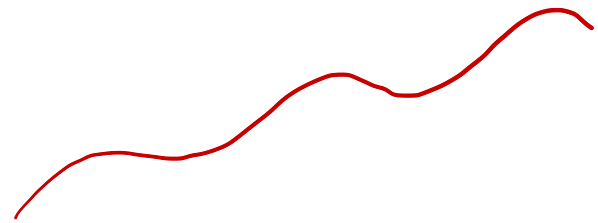


$$V(s) = w_1 X_1(s) + w_2 X_2(s)$$

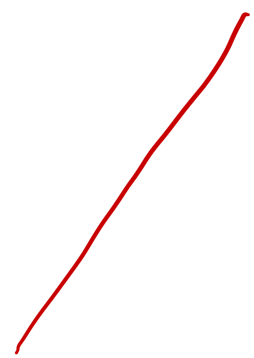
$w_1 = 2 \quad w_2 = 0$:

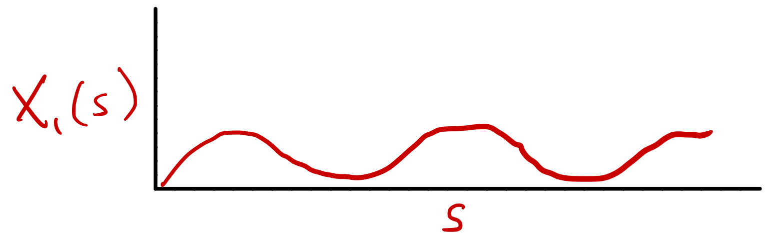


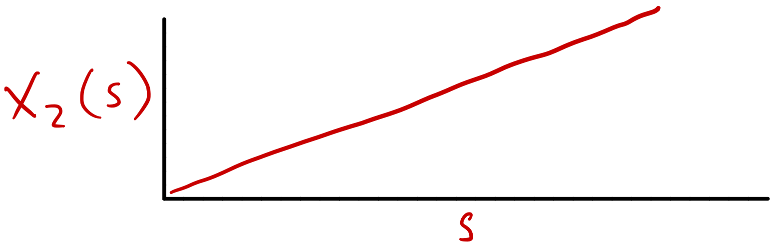
$w_1 = 1 \quad w_2 = 1$:



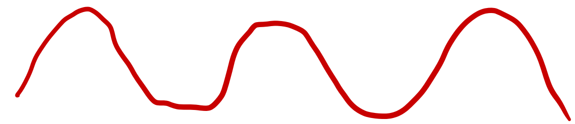
$w_1 = 0 \quad w_2 = 4$:



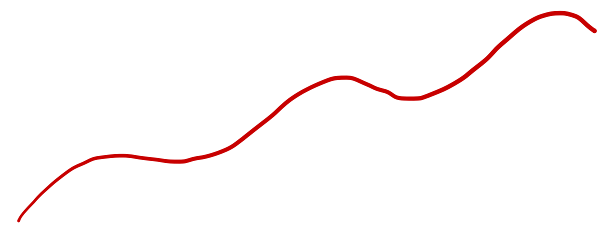




$w_1 = 2 \quad w_2 = 0$:

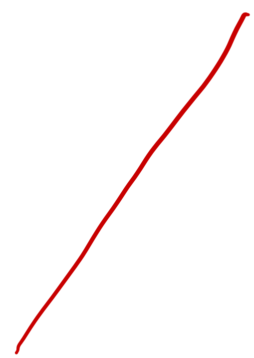


$w_1 = 1 \quad w_2 = 1$:



$$V(s) = w_1 x_1(s) + w_2 x_2(s)$$

$w_1 = 0 \quad w_2 = 4$:



Tabular equivalent :

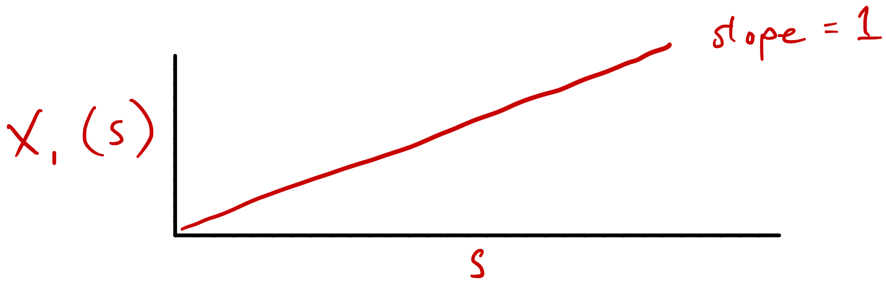
$$x_1(s) = \begin{cases} 1 & \text{at } s_1 \\ 0 & \text{elsewhere} \end{cases}$$

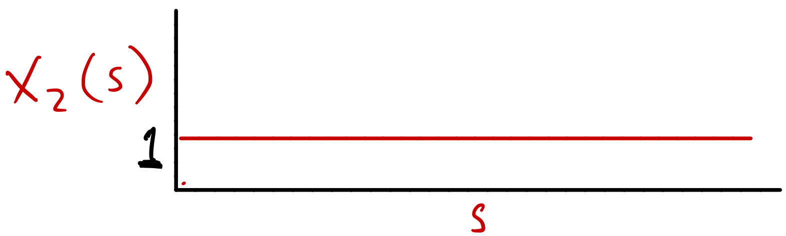
$$\vdots$$

$$x_N(s) = \begin{cases} 1 & \text{at } s_N \\ 0 & \text{elsewhere} \end{cases}$$

Thus, $w_i = V(s_i)$

Linear Regression:

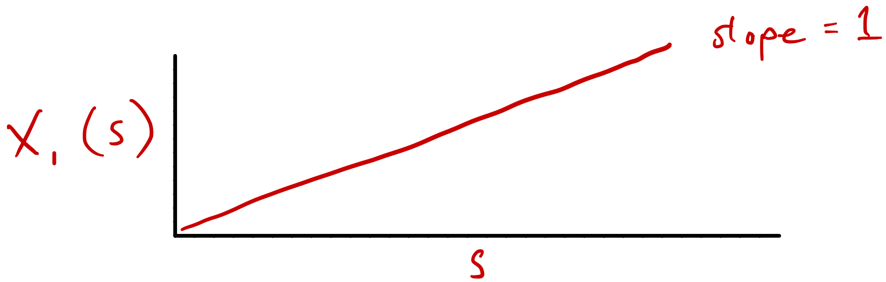


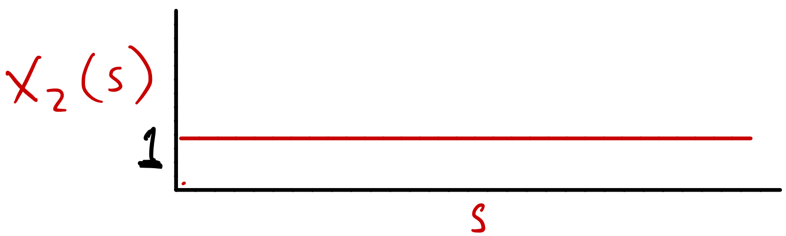


$$Y = w_1 x_1 + w_2 x_2$$
$$= w_1 x_1 + w_2 \cdot 1$$

slope intercept

Linear Regression:





$$Y = W_1 X_1 + W_2 X_2$$
$$= W_1 X_1 + W_2 \cdot 1$$

slope intercept

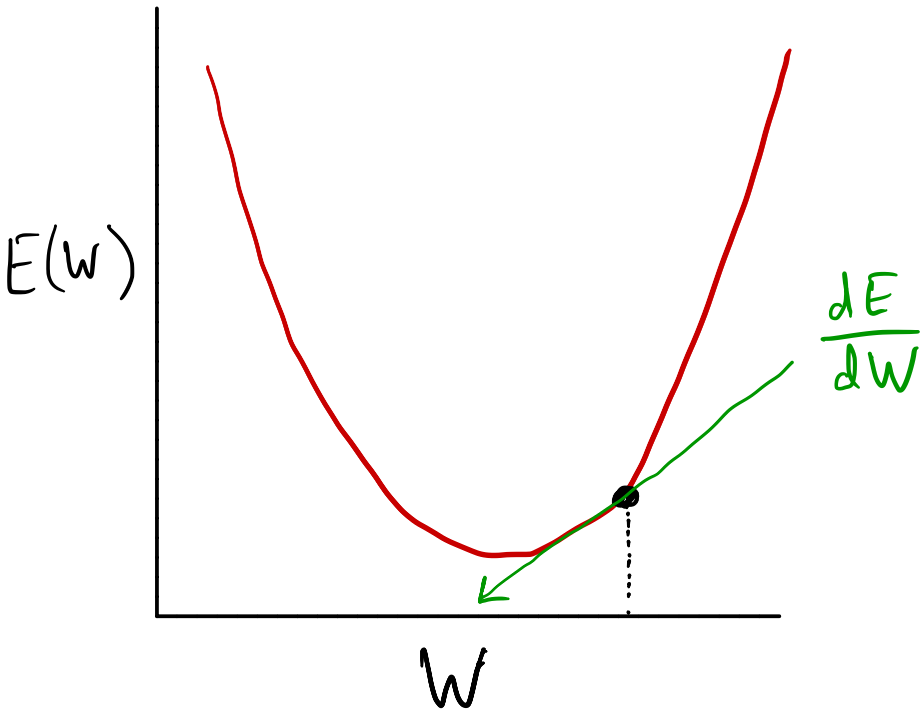
Supervised learning:

Given $\langle x, y \rangle$ pairs,
learn $f(x) \longrightarrow y$

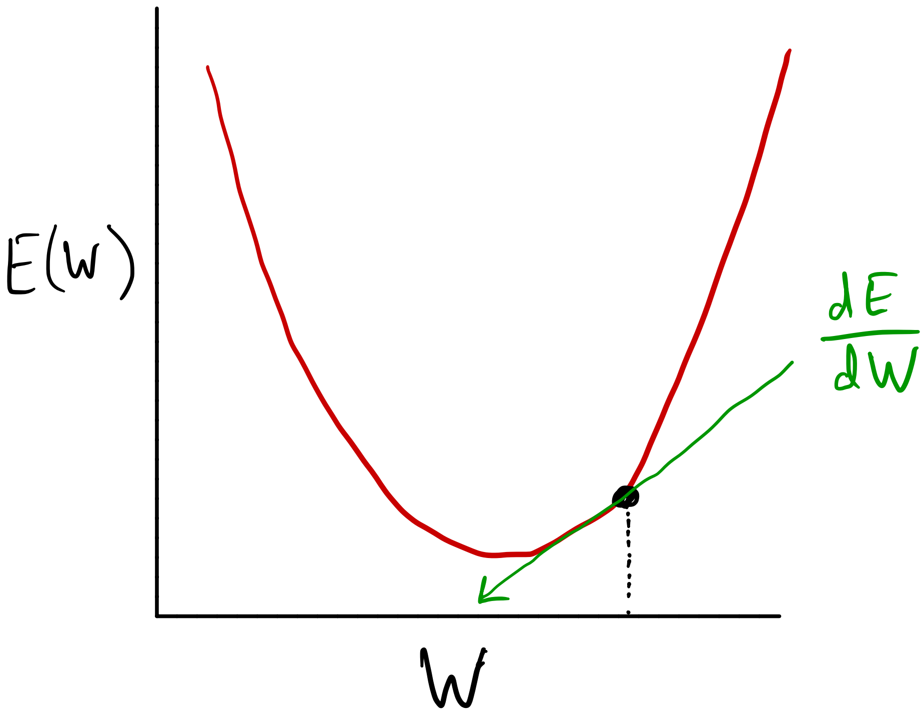
Discrete $Y \longrightarrow$ classification
e.g. "is this image a cat or a dog?"

Continuous $Y \longrightarrow$ regression
e.g. "predict height from weight"

$$E(W) = \sum_{i=0}^{N} \left(W^T x_i - y_i\right)^2$$



$$E(w) = \sum_{i=0}^{N} \left(w^T x_i - y_i\right)^2$$

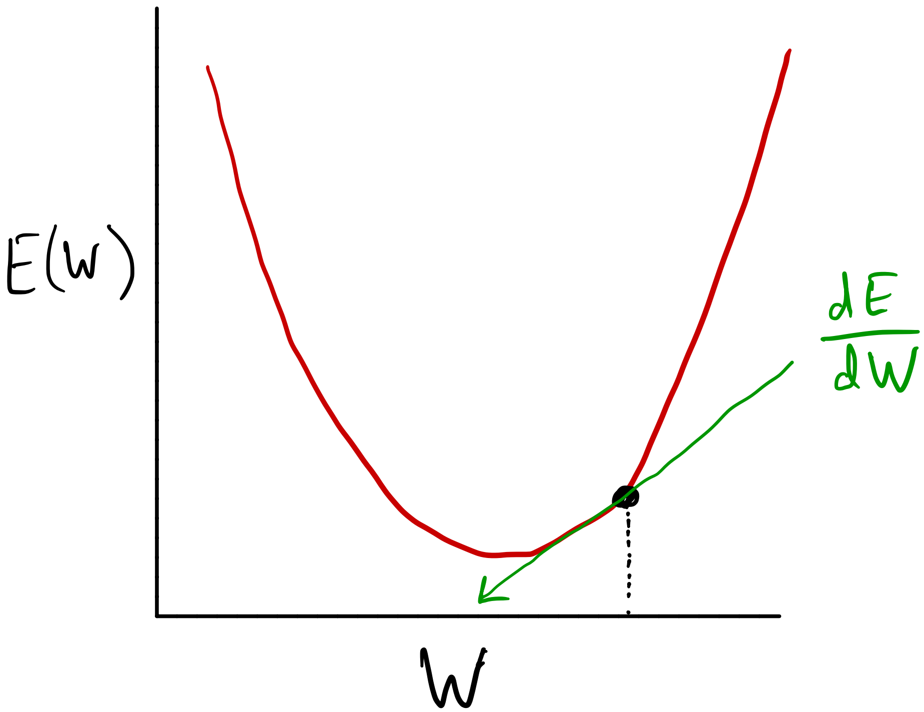


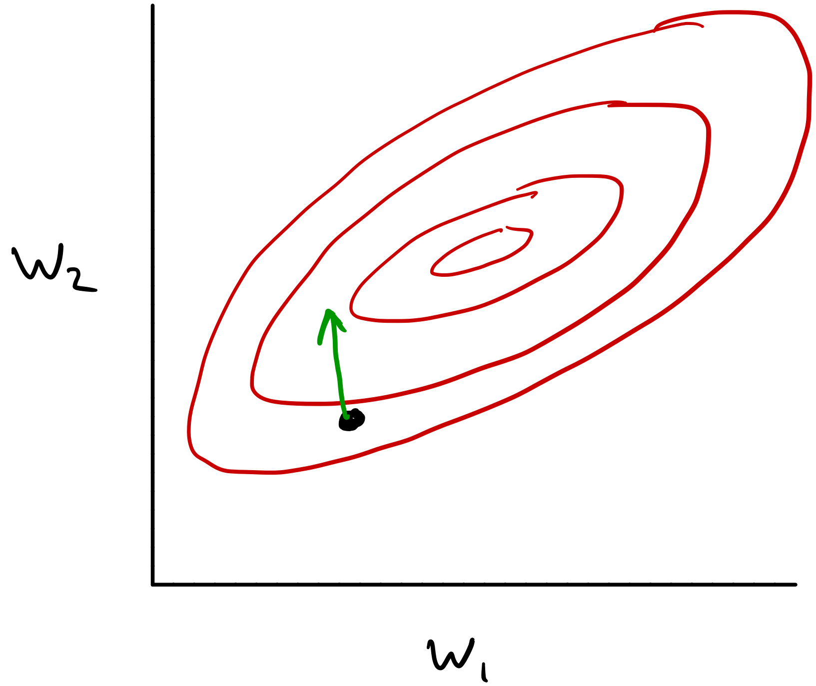
Batch least squares: Solve for $\arg\min_{w} E(w)$ directly

Exact gradient descent: Compute $\frac{dE}{dw}$ exactly from all x_i

SGD: Approximate $\frac{dE}{dw}$ with a small number of samples of X, often only one.

$$E(w) = \sum_{i=0}^{N} \left(w^T x_i - y_i\right)^2$$





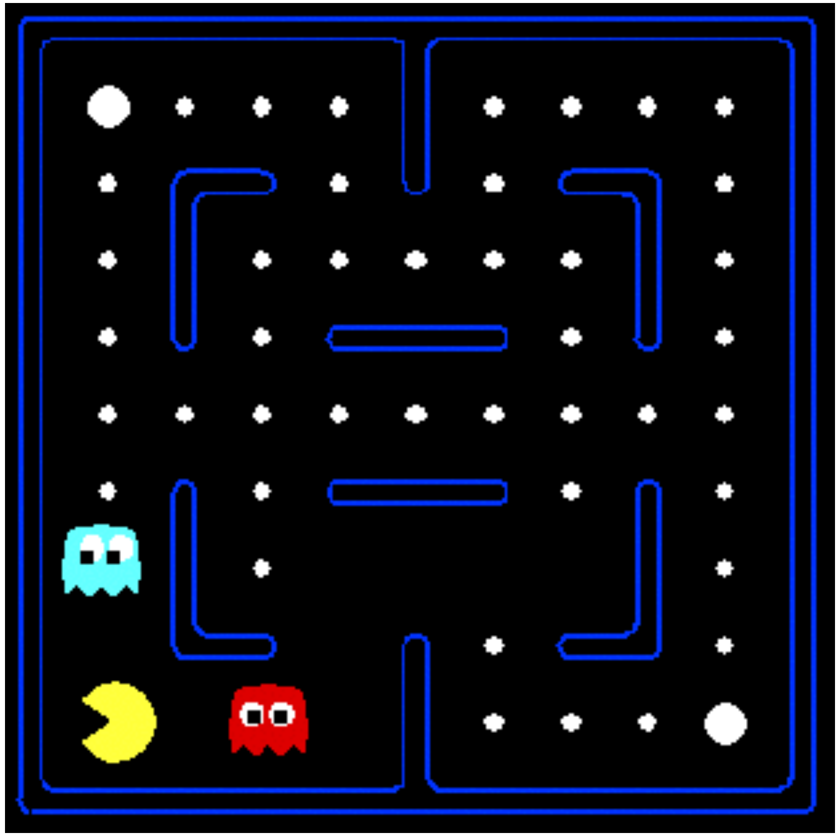
Batch least squares: Solve for $\underset{w}{\operatorname{argmin}}\, E(w)$ directly

Exact gradient descent: Compute $\frac{dE}{dW}$ exactly from all x_i

SGD: Approximate $\frac{dE}{dW}$ with a small number of samples of X, often only one.

Feature-Based Representations

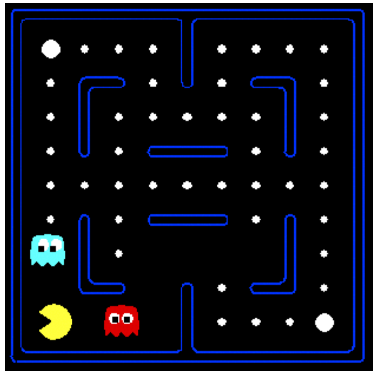
- Solution: describe a state using a vector of features (properties)
  - Features are functions from states to real numbers (often 0/1) that capture important properties of the state
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - 1 / (dist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



Example: Q-Pacman

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

s



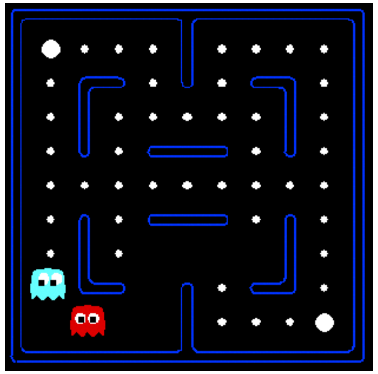
$f_{DOT}(s, \text{NORTH}) = 0.5$

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

$a = \text{NORTH}$

$r = -500$

s'



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

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

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

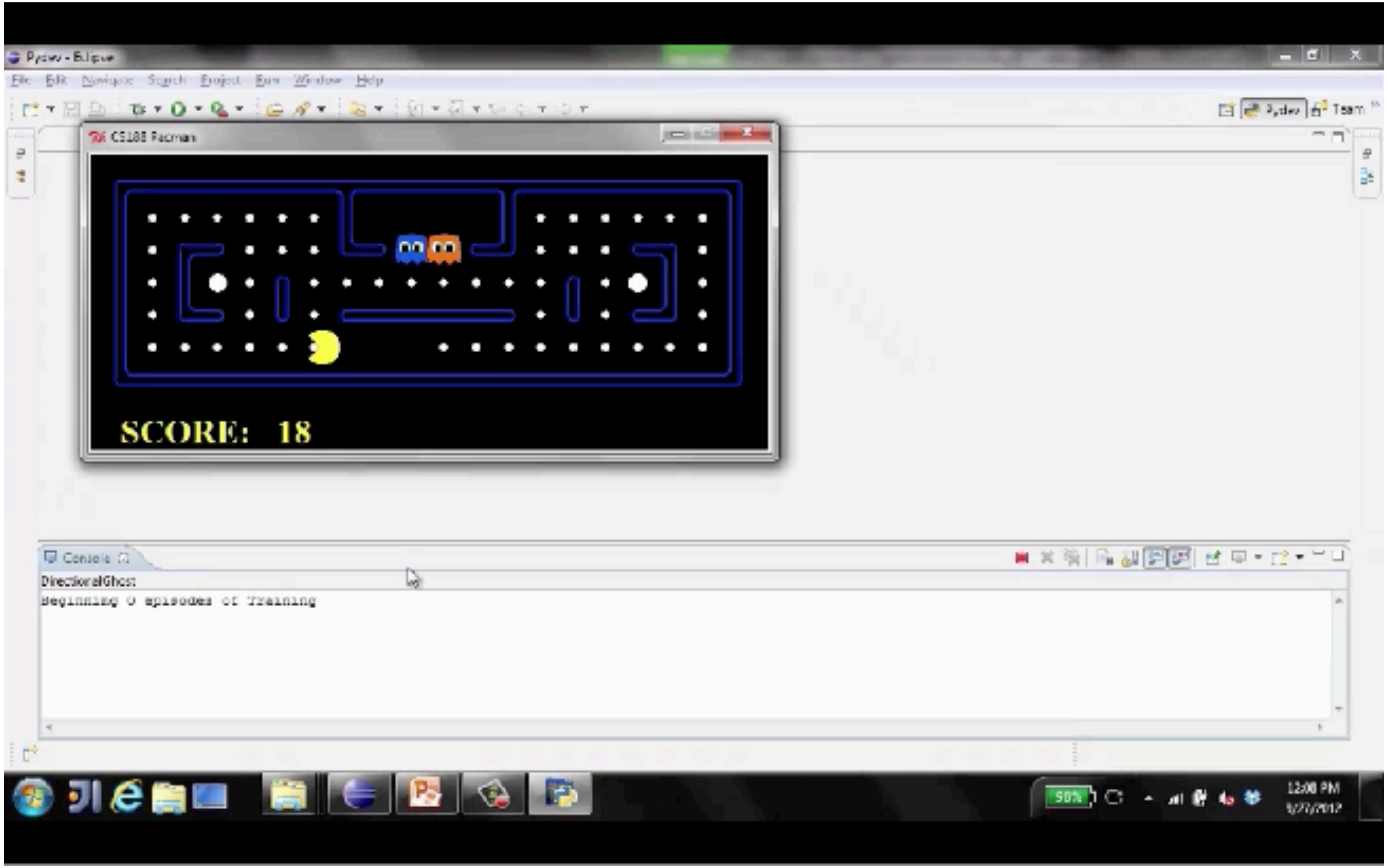
difference $= -501$

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

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

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

Approximate Q-Learning



RL + Function Approximation

Update Rule:

$$w \leftarrow w + \alpha \left[G_t - \hat{v}(S_t, w) \right] \nabla \hat{v}(S_t, w)$$

RL + Function Approximation
Update Rule:

$$w \leftarrow w + \alpha \left[G_t - \hat{v}(S_t, w) \right] \nabla \hat{v}(S_t, w)$$

When approximator is linear in features / bases
i.e. $\hat{v}(S_t, w) = w \cdot x(s)$, then:

$$w \leftarrow w + \alpha \left[G_t - \hat{v}(S_t, w) \right] x(S_t)$$

since $\dfrac{d\hat{v}(s)}{dw_i} = x_i(s)$

RL + Function Approximation Update Rule:

$$w \leftarrow w + \alpha \left[G_t - \hat{v}(S_t, w) \right] \nabla \hat{v}(S_t, w)$$

When approximator is linear in features / bases

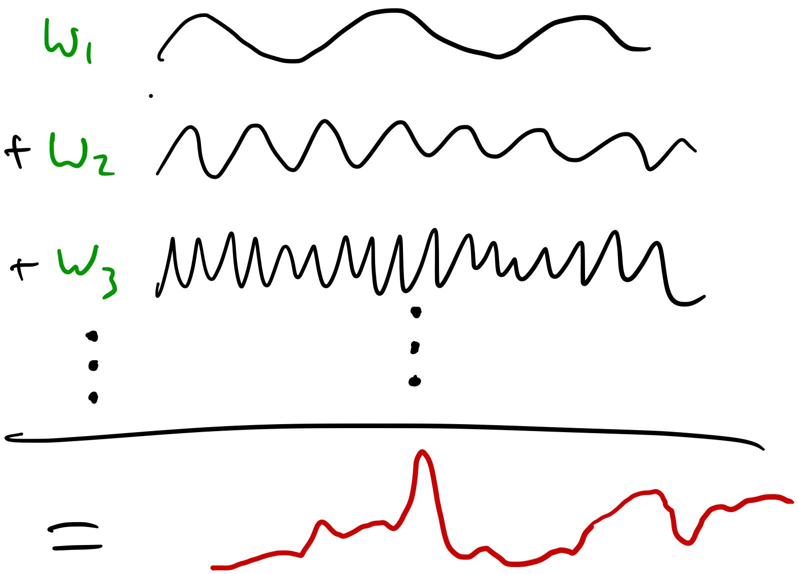
i.e. $\hat{v}(S_t, w) = w \cdot x(s)$, then:

$$w \leftarrow w + \alpha \left[G_t - \hat{v}(S_t, w) \right] x(S_t)$$

since $\frac{d\hat{v}(s)}{dw} = x_i(s)$

Assigns "credit" to most activated features for success + failure

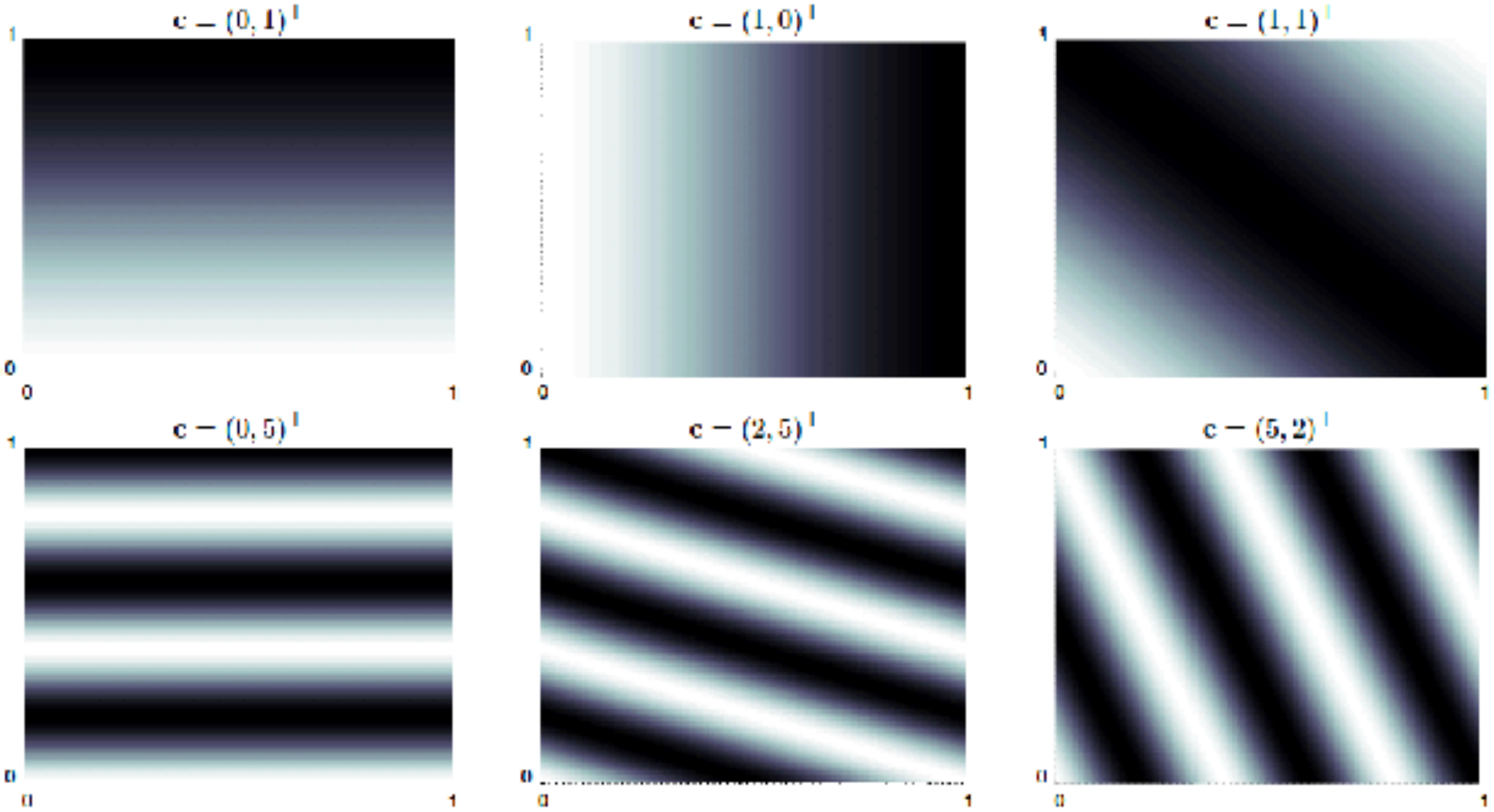
Fourier Basis



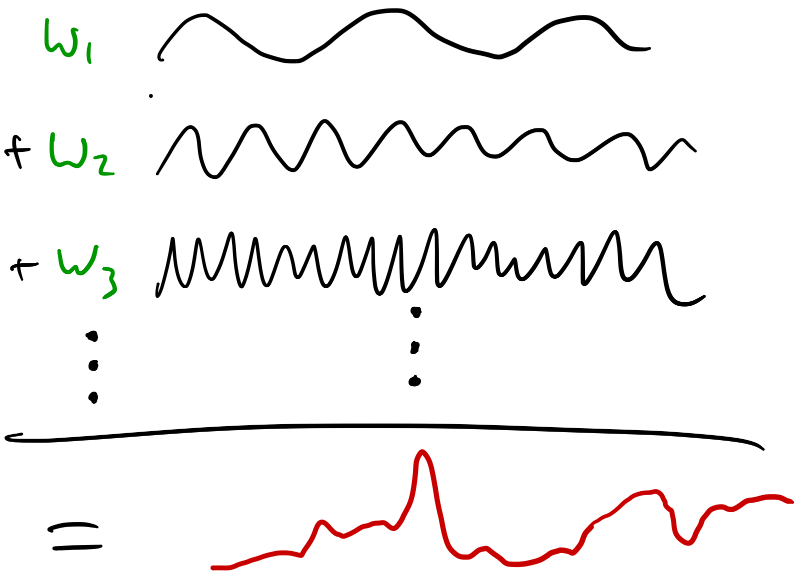
Params:

- order
- Interaction terms?

Fourier interaction terms



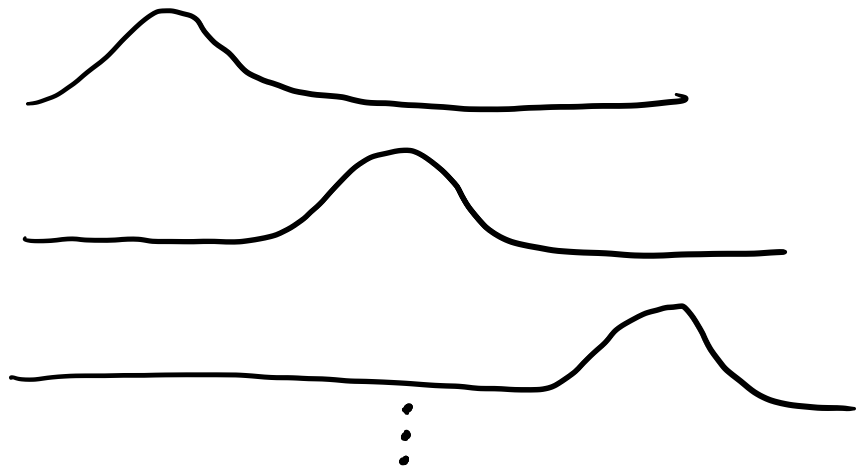
Fourier Basis



Params:

- order
- Interaction terms?

Radial Basis Functions

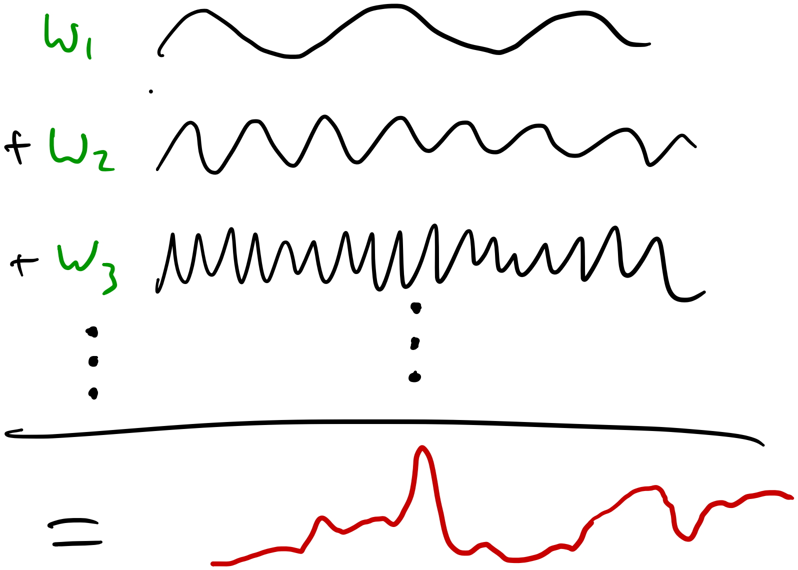


Params:

- Kernel width
- Tiling density

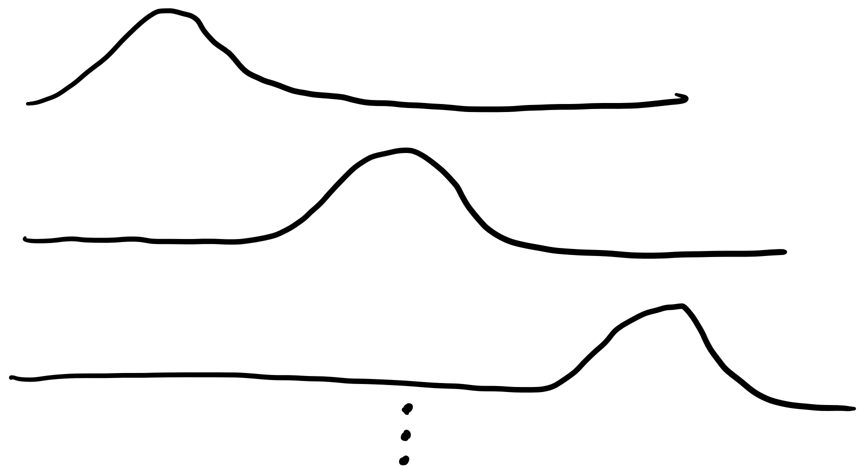
(both variable per dimension)

Fourier Basis



Params:
- order
- Interaction terms?

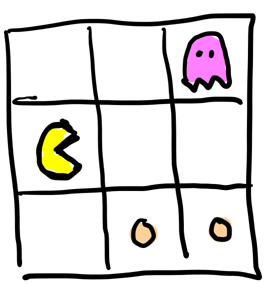
Radial Basis Functions



Params:
- Kernel width
- Tiling density

(both variable per dimension)

Hand-Designed Features

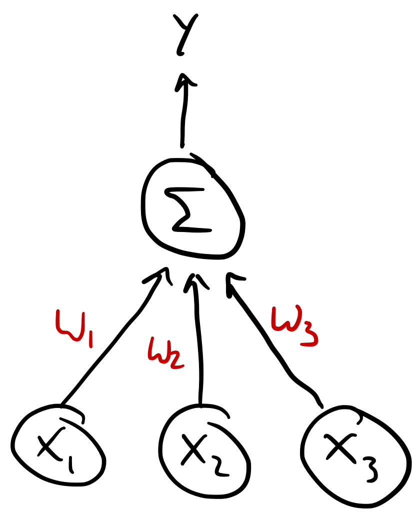


"Distance to nearest food pellet"
"Number of ghosts within radius of 3"
"1 if exactly this state, 0 otherwise"

→

How do choose?
Representative Power?
Locality?

Perceptron

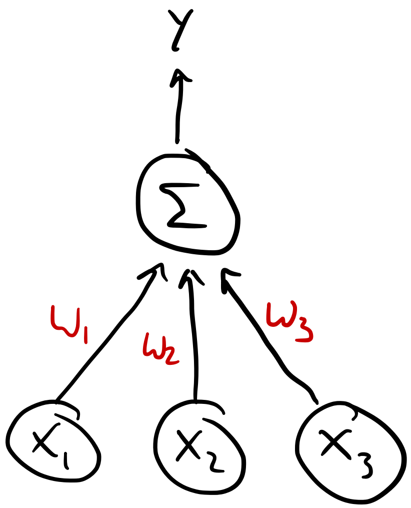


$$Y = X_1 w_1 + X_2 w_2 + X_3 w_3$$

$$\frac{dy}{dw_1} = X_1$$

Y is Linear in X

Perceptron

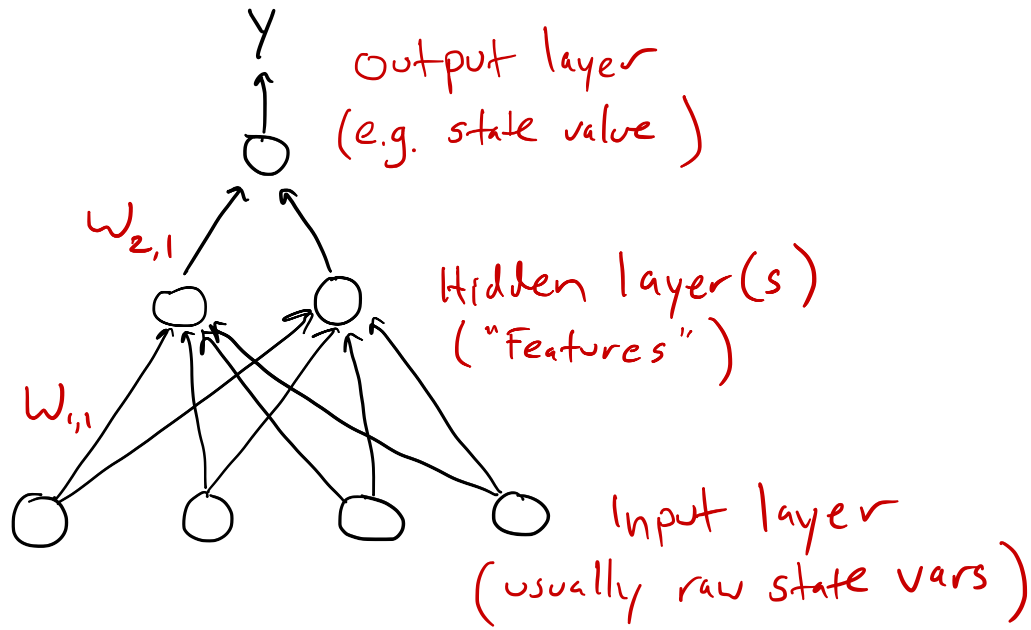


$$Y = X_1W_1 + X_2W_2 + X_3W_3$$

$$\frac{dy}{dW_1} = X_1$$

Y is Linear in X

Neural Network

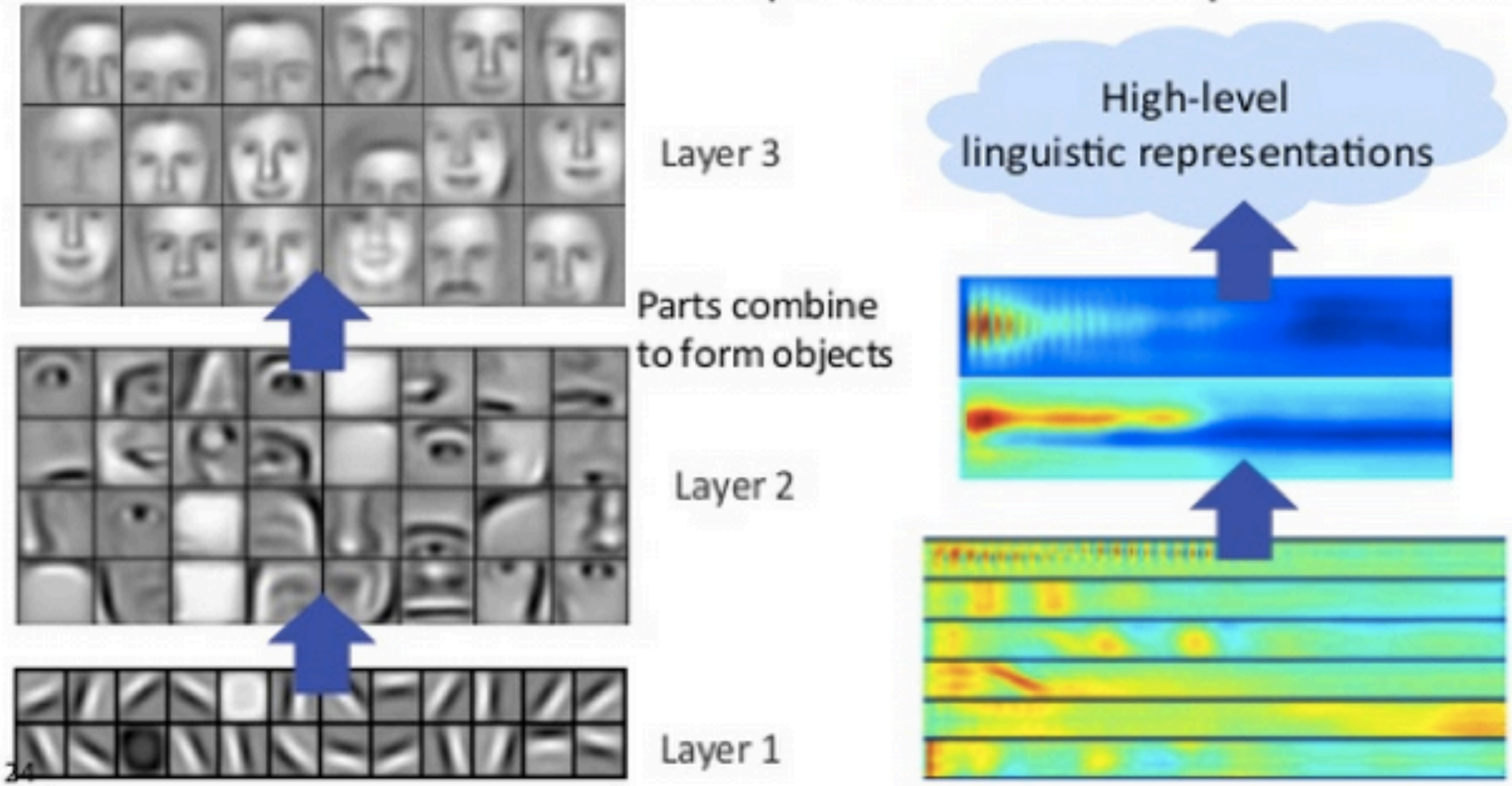


$$\frac{dy}{dW_{1,1}} = ?$$

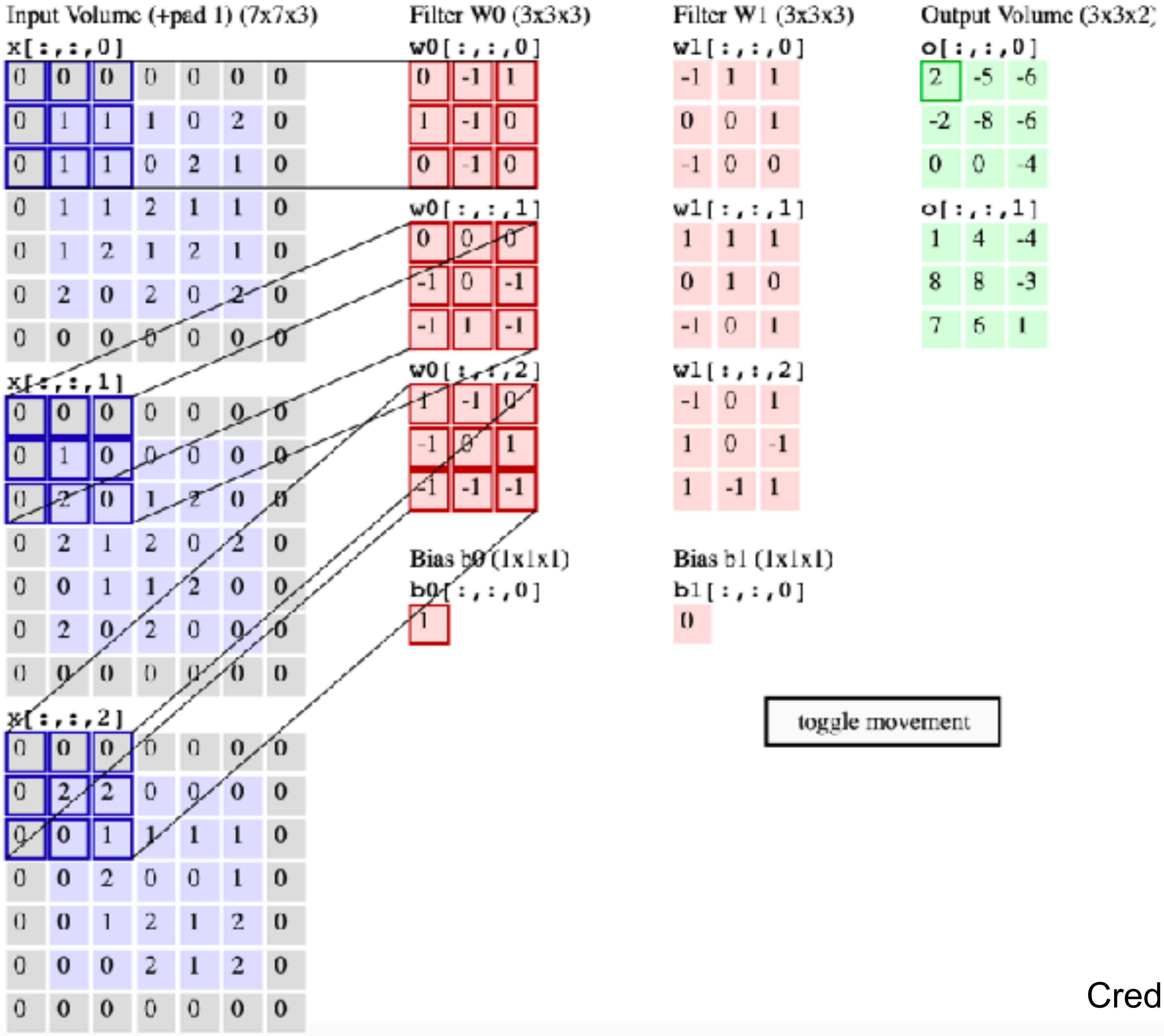
⇒ Differentiation Via Backpropagation

Y is Nonlinear in X!

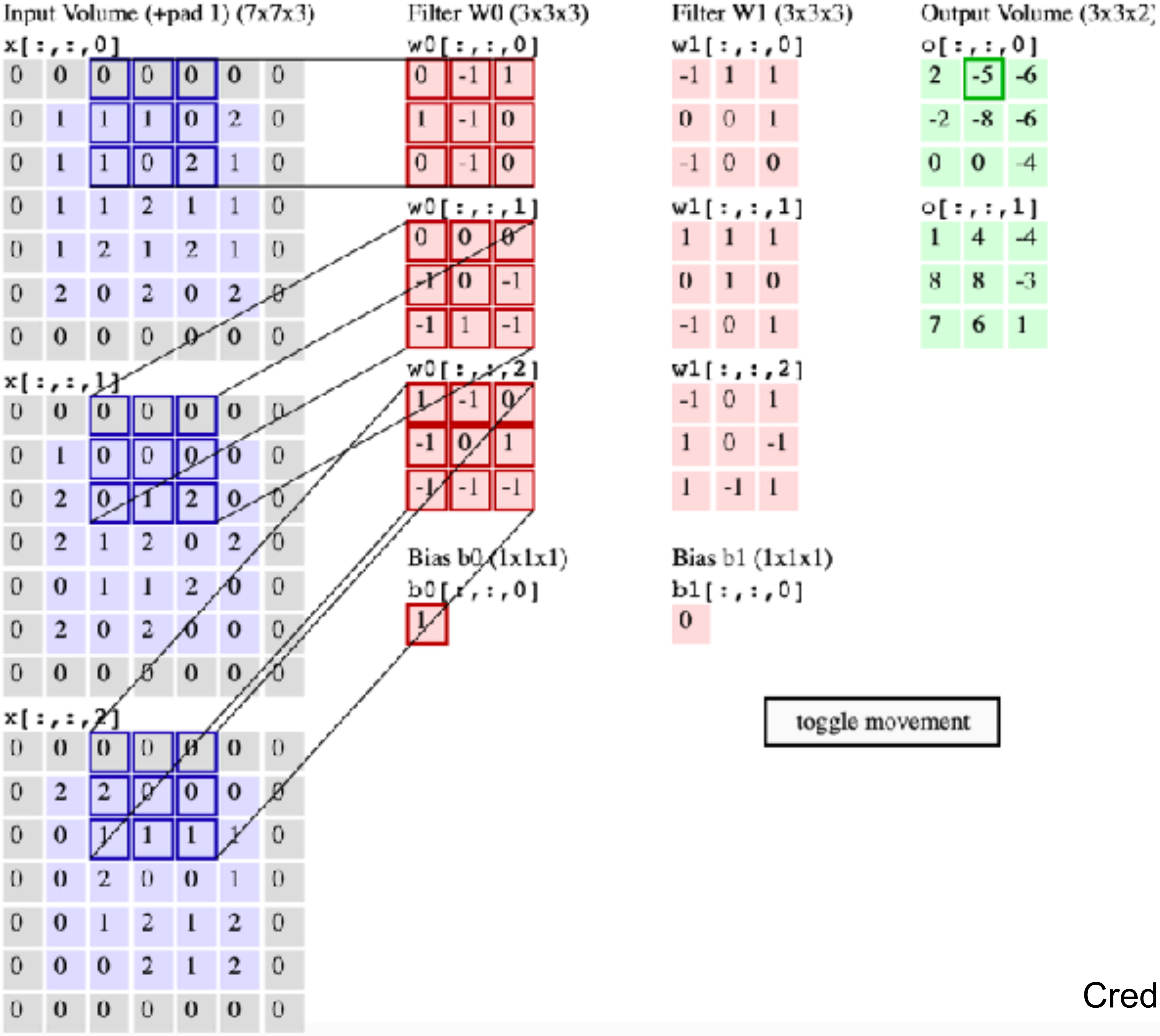
Successive model layers learn deeper intermediate representations



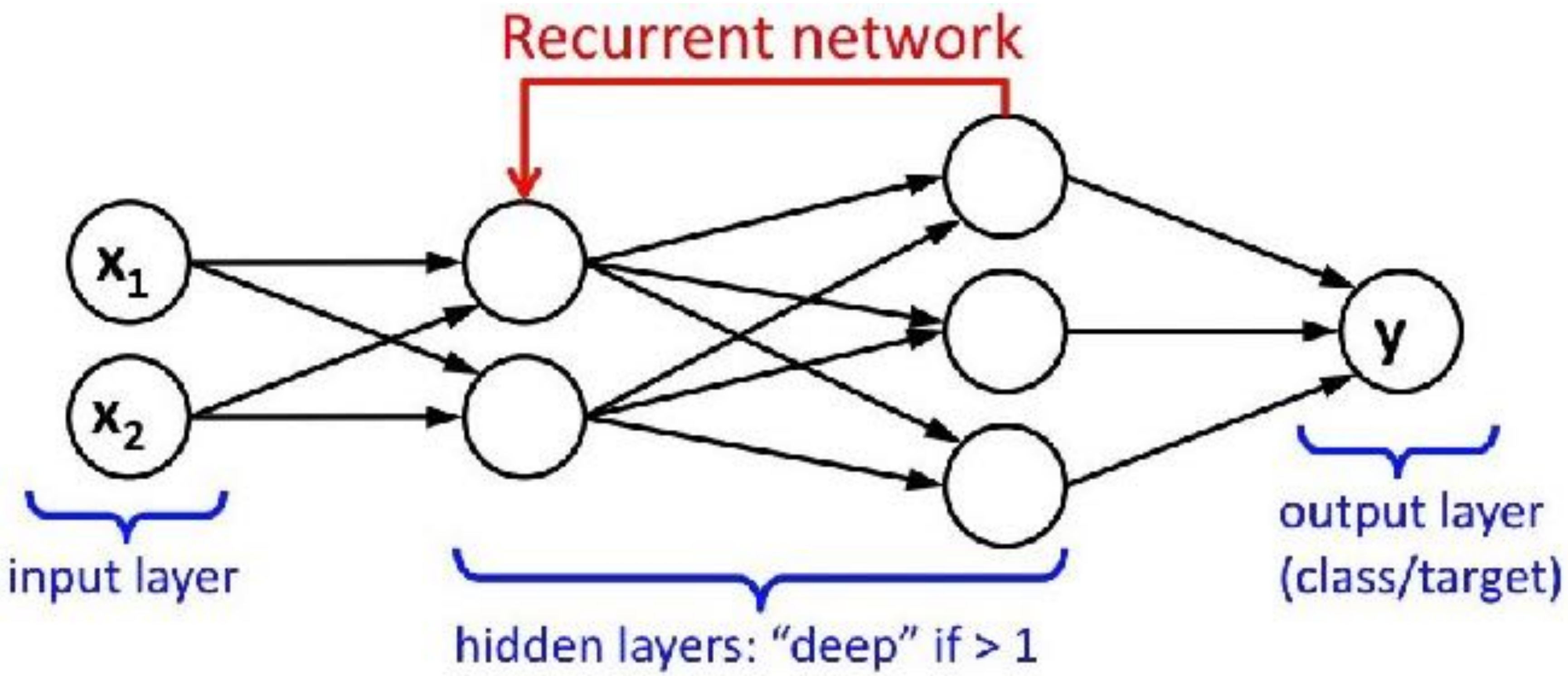
Prior: underlying factors & concepts compactly expressed w/ multiple levels of abstraction



Credit: Andrej Karpathy

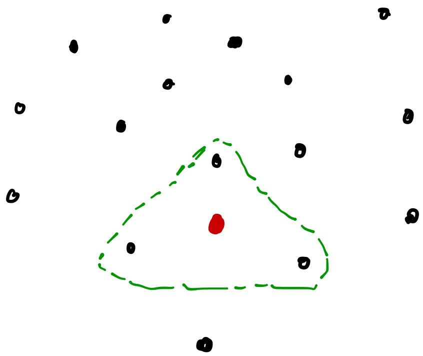


Credit: Andrej Karpathy


Recurrent network
x1
x2
y
input layer
hidden layers: "deep" if > 1
output layer
(class/target)

Nonparametric Methods

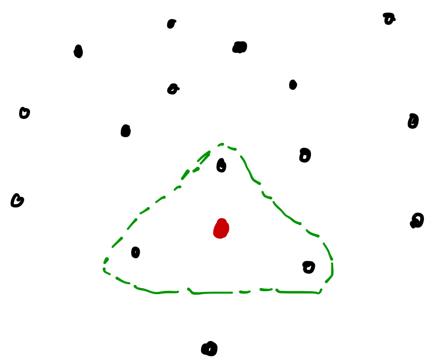
K-nearest neighbor



$$\hat{V}(s_q) = \frac{1}{K} \sum_{i=1}^{K} G(s_i)$$

Nonparametric Methods

K-nearest neighbor



$$\hat{V}(s_q) = \frac{1}{K}\sum_{i=1}^{K} G(s_i)$$

Kernel Methods

Similarity "Kernel" $K(s, s')$

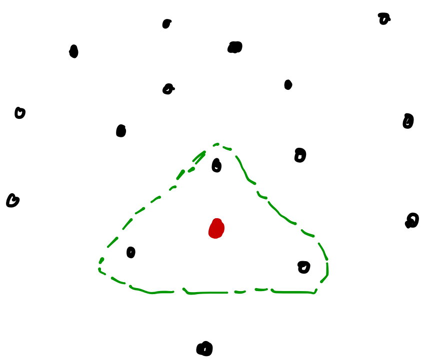
For data set of N states

And query state s_q

$$\hat{V}(s_q) = \sum_{i=1}^{N} K(s_q, s_i)\, G(s_i)$$

Nonparametric Methods

K-nearest neighbor



$$\hat{v}(s_q) = \frac{1}{K}\sum_{i=1}^{K} G(s_i)$$

KNN special case kernel where:

$K(s,s') = 1/K$ if s' is a KNN of s

0 otherwise

Kernel Methods

Similarity "Kernel" $K(s,s')$

For data set of N states

And query state s_q

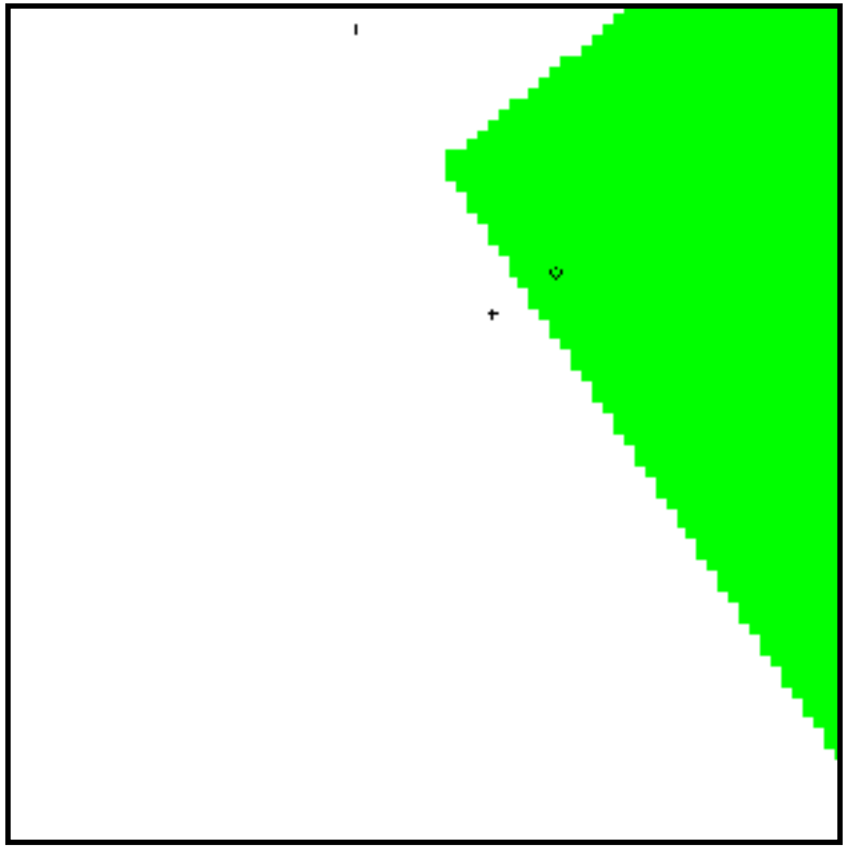
$$\hat{V}(s_q) = \sum_{i=1}^{N} K(s_q, s_i)\, G(s_i)$$

"Kernal trick" eqivalent to computing $\hat{v}$ in high-dim space of features

→ Complexity of kernel methods grows with number of data points, not features

Case-Based Reasoning

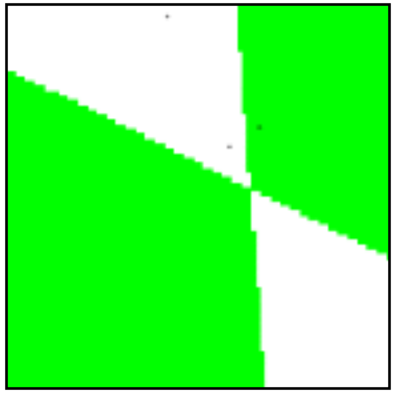
- Classification from similarity
  - Case-based reasoning
  - Predict an instance’s label using similar instances
- Nearest-neighbor classification
  - 1-NN: copy the label of the most similar data point
  - K-NN: vote the k nearest neighbors (need a weighting scheme)
  - Key issue: how to define similarity
  - Trade-offs: Small k gives relevant neighbors, Large k gives smoother functions



http://www.cs.cmu.edu/~zhuxj/courseproject/knndemo/KNN.html

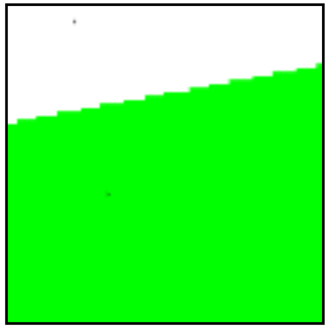
Parametric / Non-Parametric

- Parametric models:
  - Fixed set of parameters
  - More data means better settings
- Non-parametric models:
  - Complexity of the classifier increases with data
  - Better in the limit, often worse in the non-limit
- (K)NN is non-parametric

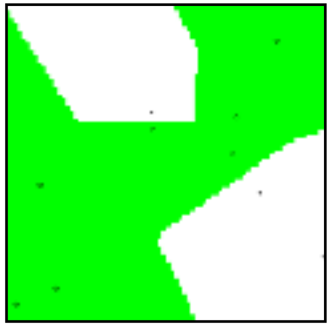


Truth

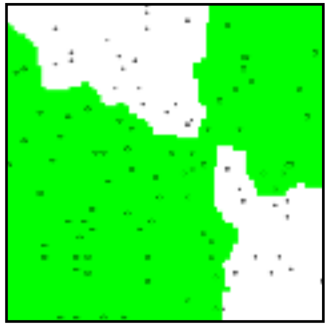
2 Examples



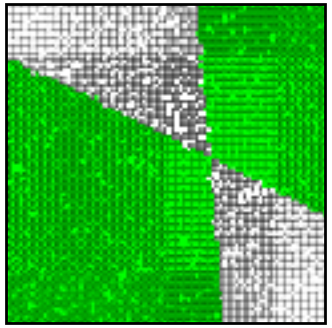
10 Examples



100 Examples



10000 Examples



Basic Similarity

- Many similarities based on feature dot products:

$$\mathsf{sim}(x, x') = f(x) \cdot f(x') = \sum_i f_i(x) f_i(x')$$

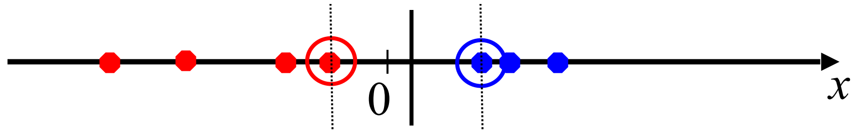
- If features are just the pixels:

$$\mathsf{sim}(x, x') = x \cdot x' = \sum_i x_i x'_i$$

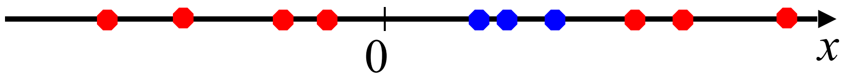
- Note: not all similarities are of this form

Kernel methods

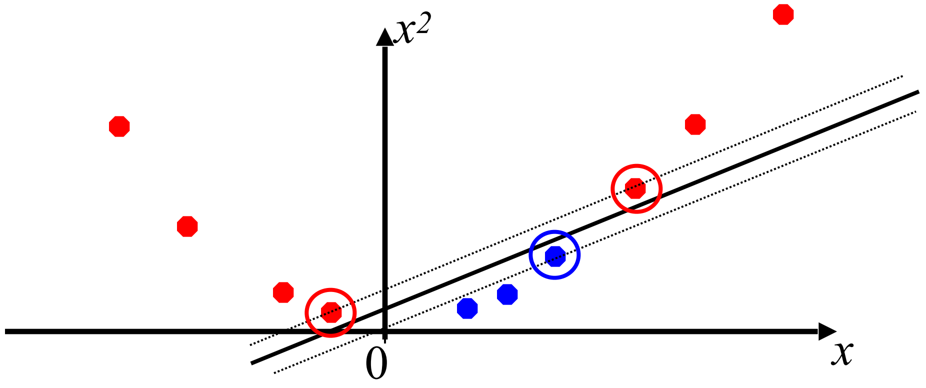
- Data that is linearly separable works out great for linear decision rules:



- But what are we going to do if the dataset is just too hard?



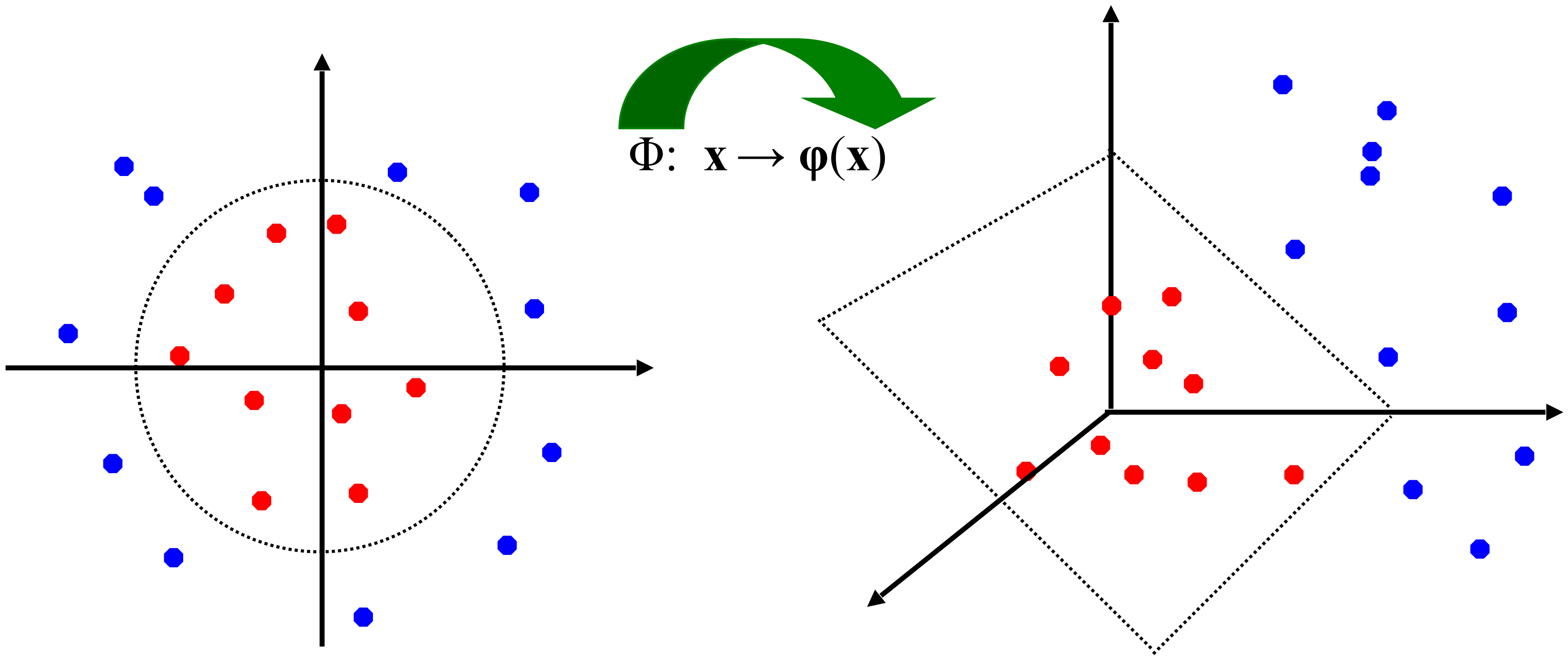
- How about… mapping data to a higher-dimensional space:



This and next few slides adapted from Ray Mooney, UT

Kernel methods

- General idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable:



Some Kernels

- Kernels implicitly map original vectors to higher dimensional spaces, take the dot product there, and hand the result back

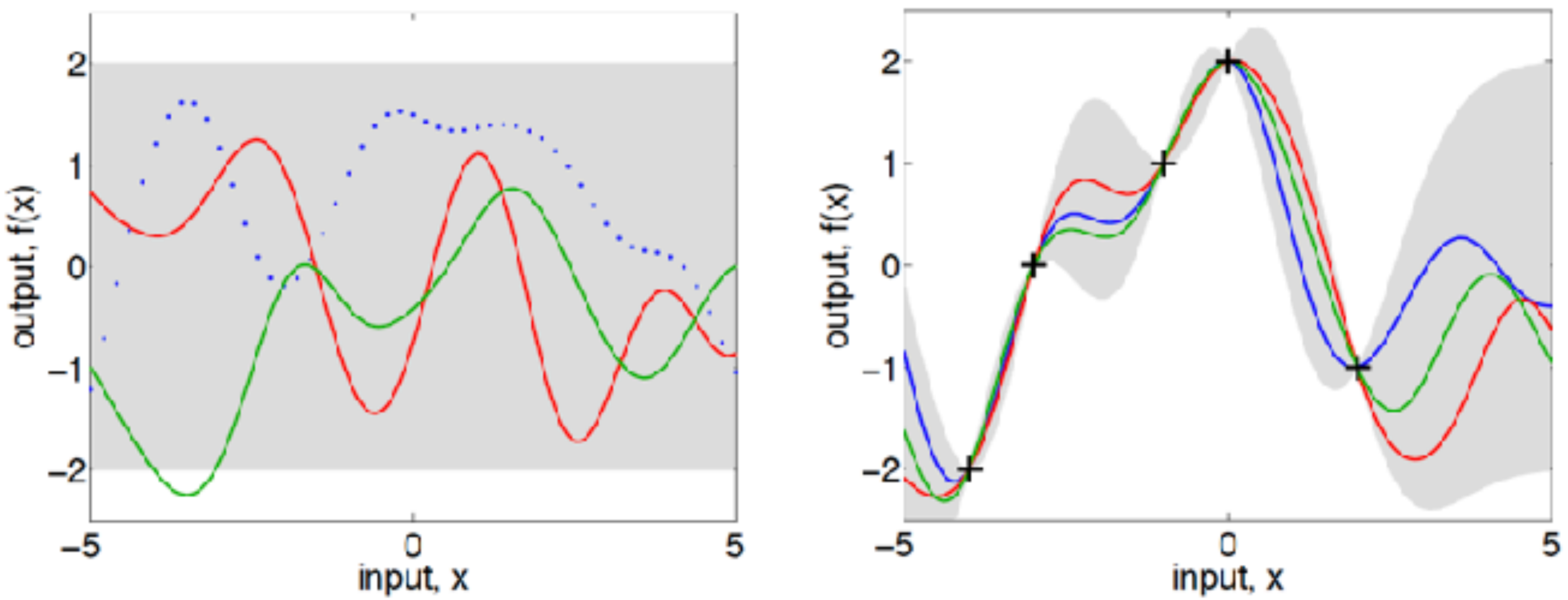
- Linear kernel: $$K(x, x') = x' \cdot x' = \sum_i x_i x'_i$$

- Quadratic kernel: $$K(x, x') = (x \cdot x' + 1)^2$$
$$= \sum_{i,j} x_i x_j \, x'_i x'_j + 2 \sum_i x_i \, x'_i + 1$$

- RBF: infinite dimensional representation
$$K(x, x') = \exp(-||x - x'||^2)$$

- Discrete kernels: e.g. string kernels

Prior and Posterior



Predictive distribution:

$$p(y^*|x^*, \mathbf{x}, \mathbf{y}) \sim \mathcal{N}\big(\mathbf{k}(x^*, \mathbf{x})^\top [K + \sigma^2_{\text{noise}} I]^{-1}\mathbf{y},$$
$$k(x^*, x^*) + \sigma^2_{\text{noise}} - \mathbf{k}(x^*, \mathbf{x})^\top [K + \sigma^2_{\text{noise}} I]^{-1}\mathbf{k}(x^*, \mathbf{x})\big)$$

$$w \leftarrow w - \frac{1}{2}\alpha \nabla \left[V_\pi(S_t) - \hat{v}(S_t, w) \right]^2$$

$$w \leftarrow w + \alpha \left[U_t - \hat{v}(S_t, w) \right] \cdot \nabla \hat{v}(S_t, w)$$

$$w \leftarrow w - \frac{1}{2}\alpha \nabla \left[V_\pi(S_t) - \hat{v}(S_t, w) \right]^2$$

$$w \leftarrow w + \alpha \left[U_t - \hat{v}(S_t, w) \right] \cdot \nabla \hat{v}(S_t, w)$$

MC Estimate:

$$U_t = \sum_{i=t}^{T} \gamma^{i-t-1} R_i$$

- Unbiased
- fixed when w changes

converges near global opt

$$w \leftarrow w - \frac{1}{2}\alpha \nabla\left[V_\pi(S_t) - \hat{v}(S_t, w)\right]^2$$

$$w \leftarrow w + \alpha\left[U_t - \hat{v}(S_t, w)\right] \cdot \nabla \hat{v}(S_t, w)$$

MC Estimate:

$$U_t = \sum_{i=t}^{T} \gamma^{i-t-1} R_i$$

- Unbiased
- fixed when w changes

converges near global opt

Bootstrap Estimate:

$$U_t = R_t + \gamma \hat{v}(S_{t+1}, w)$$

- Biased
- Function of w → changes with w!
- Gradient calc treated U_t as a constant!

Converges to TD fixed point (local opt)

For TD fixed point w_{TD}:

$$\overline{VE}(w_{TD}) \leq \frac{1}{1-\gamma} \min_w \left[\overline{VE}(w)\right]$$

Least Squares TD

TD fixed point is: $\mathbf{w}_{\mathrm{TD}} = \mathbf{A}^{-1}\mathbf{b},$

where

$$\mathbf{A} \doteq \mathbb{E}\left[\mathbf{x}_t(\mathbf{x}_t - \gamma\mathbf{x}_{t+1})^\top\right] \quad \text{and} \quad \mathbf{b} \doteq \mathbb{E}[R_{t+1}\mathbf{x}_t].$$

Instead, LSTD estimates: $$\widehat{\mathbf{A}}_t \doteq \sum_{k=0}^{t-1} \mathbf{x}_k(\mathbf{x}_k - \gamma\mathbf{x}_{k+1})^\top + \varepsilon\mathbf{I} \quad \text{and} \quad \widehat{\mathbf{b}}_t \doteq \sum_{k=0}^{t-1} R_{k+1}\mathbf{x}_k,$$