Lecture 3: Machine Learning 2

Roadmap

Stochastic Gradient Descent

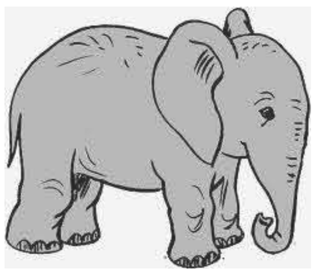
Non-linear features

Neural networks

Feature templates

Gradient descent is slow

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$



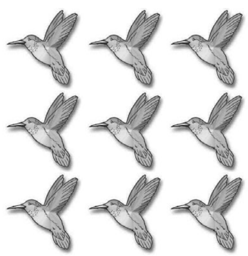
Algorithm: gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$
For $t = 1, \ldots, T$:
$\quad \mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

Problem: each iteration requires going over all training examples — expensive when have lots of data!

Stochastic gradient descent

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$



Algorithm: stochastic gradient descent

Initialize $\mathbf{w} = [0, \ldots, 0]$

For $t = 1, \ldots, T$:

  For $(x, y) \in \mathcal{D}_{\text{train}}$:

    $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$

Step size

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w})$$

Question: what should η be?

Strategies:

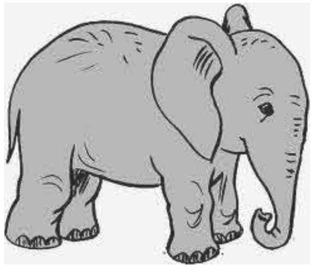
- Constant: $\eta = 0.1$
- Decreasing: $\eta = 1/\sqrt{\#\text{ updates made so far}}$

Stochastic gradient descent in Python

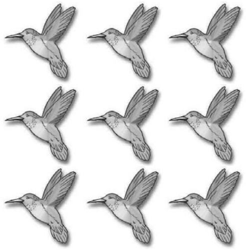
[code]

Summary

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x, y, \mathbf{w})$$



gradient descent



stochastic gradient descent

Key idea: stochastic updates

It's not about quality, it's about **quantity**.

Roadmap

Stochastic Gradient Descent

Non-linear features

Neural networks

Feature templates

Linear regression

training data

x	y
1	1
2	3
4	3

learning algorithm → f **predictor**

3 → f → 2.71

Which predictors are possible?

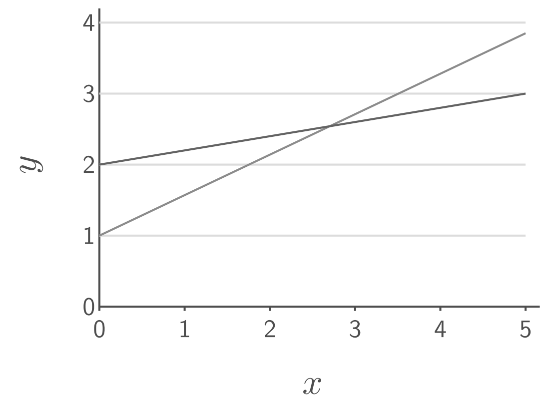
Hypothesis class

$$\mathcal{F} = \{f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x) : \mathbf{w} \in \mathbb{R}^d\}$$

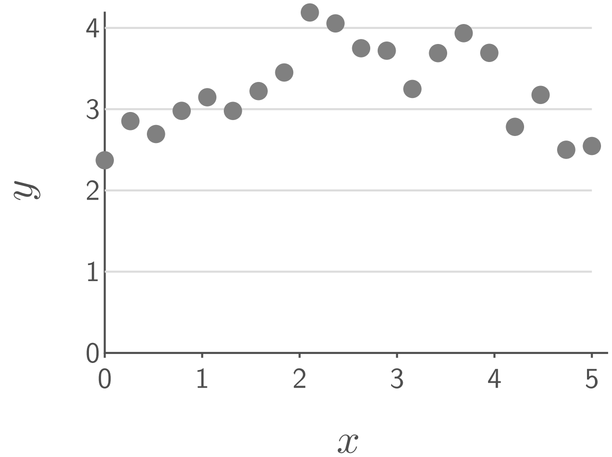
$$\phi(x) = [1, x]$$

$$f(x) = [1, 0.57] \cdot \phi(x)$$

$$f(x) = [2, 0.2] \cdot \phi(x)$$



More complex data



How do we fit a non-linear predictor?

Quadratic predictors

$$\phi(x) = [1, x, x^2]$$

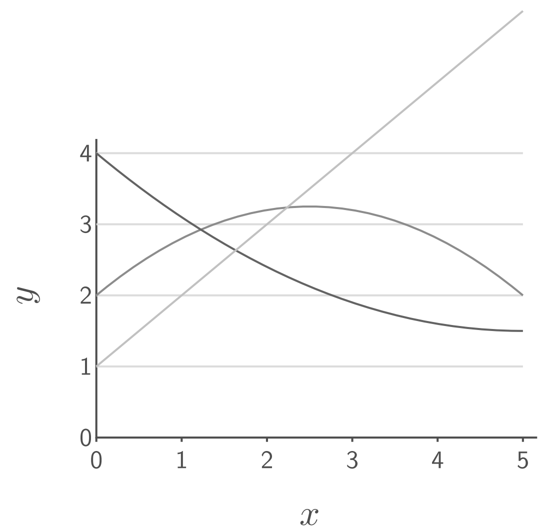
Example: $\phi(3) = [1, 3, 9]$

$f(x) = [2, 1, -0.2] \cdot \phi(x)$

$f(x) = [4, -1, 0.1] \cdot \phi(x)$

$f(x) = [1, 1, 0] \cdot \phi(x)$

$\mathcal{F} = \{f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x) : \mathbf{w} \in \mathbb{R}^3\}$



Non-linear predictors just by changing ϕ

Piecewise constant predictors

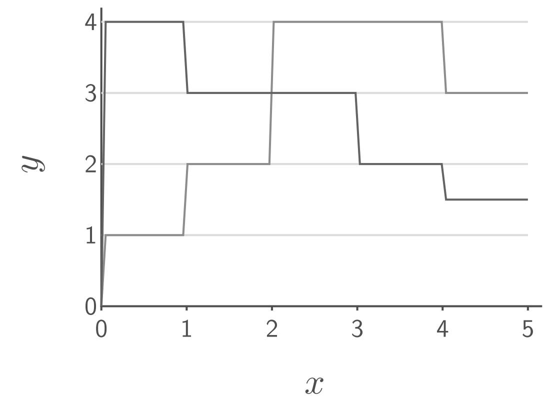
$$\phi(x) = [\mathbf{1}[0 < x \le 1], \mathbf{1}[1 < x \le 2], \mathbf{1}[2 < x \le 3], \mathbf{1}[3 < x \le 4], \mathbf{1}[4 < x \le 5]]$$

Example: $\phi(2.3) = [0, 0, 1, 0, 0]$

$f(x) = [1, 2, 4, 4, 3] \cdot \phi(x)$

$f(x) = [4, 3, 3, 2, 1.5] \cdot \phi(x)$

$\mathcal{F} = \{f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x) : \mathbf{w} \in \mathbb{R}^5\}$



Expressive non-linear predictors by partitioning the input space

Predictors with periodicity structure

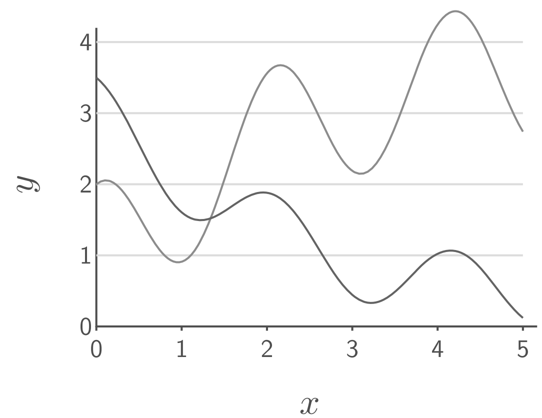
$$\phi(x) = [1, x, x^2, \cos(3x)]$$

Example: $\phi(2) = [1, 2, 4, 0.96]$

$f(x) = [1, 1, -0.1, 1] \cdot \phi(x)$

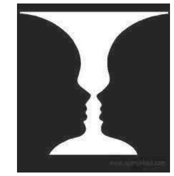
$f(x) = [3, -1, 0.1, 0.5] \cdot \phi(x)$

$\mathcal{F} = \{f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x) : \mathbf{w} \in \mathbb{R}^4\}$



Just throw in any features you want

Linear in what?



Prediction:

$$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$$

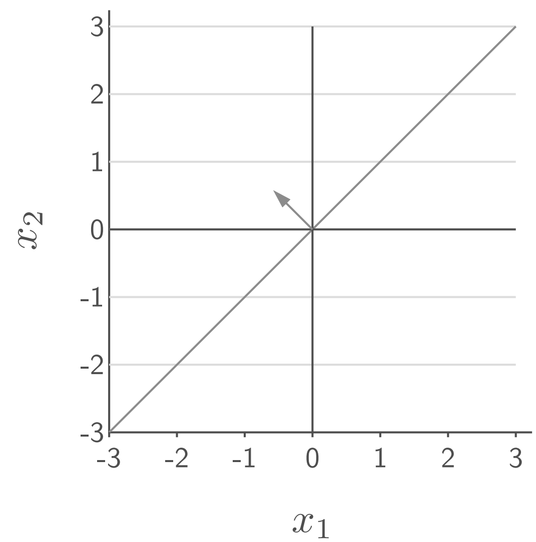
Linear in $\mathbf{w}$?	Yes
Linear in $\phi(x)$?	Yes
Linear in x?	No!

Key idea: non-linearity

- Expressiveness: score $\mathbf{w} \cdot \phi(x)$ can be a **non-linear** function of x
- Efficiency: score $\mathbf{w} \cdot \phi(x)$ always a **linear** function of $\mathbf{w}$

Linear classification

$$\phi(x) = [x_1, x_2]$$
$$f(x) = \mathrm{sign}([-0.6, 0.6] \cdot \phi(x))$$



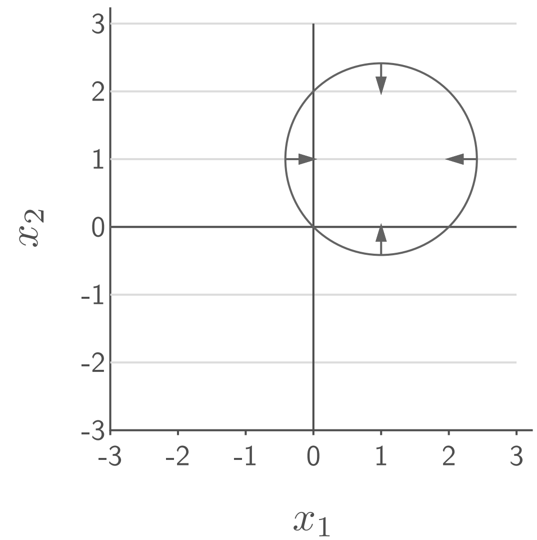
Decision boundary is a line

Quadratic classifiers

$$\phi(x) = [x_1, x_2, x_1^2 + x_2^2]$$
$$f(x) = \text{sign}([2, 2, -1] \cdot \phi(x))$$

Equivalently:

$$f(x) = \begin{cases} 1 & \text{if } \{(x\_1 - 1)^2 + (x\_2 - 1)^2 \leq 2\} \\ -1 & \text{otherwise} \end{cases}$$

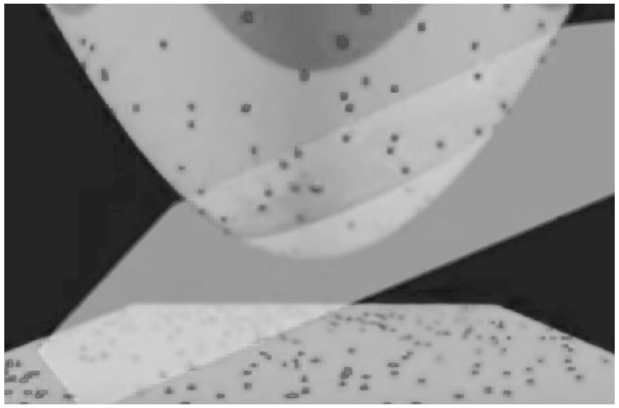


Decision boundary is a circle

Visualization in feature space

Input space: $x = [x_1, x_2]$, decision boundary is a circle

Feature space: $\phi(x) = [x_1, x_2, x_1^2 + x_2^2]$, decision boundary is a hyperplane

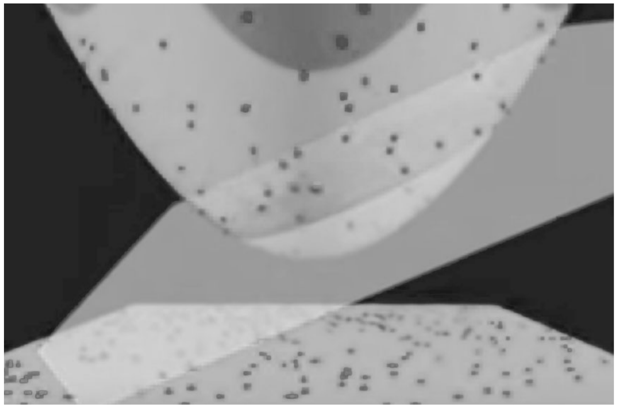


Summary

$$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x)$$

linear in $\mathbf{w}, \phi(x)$

non-linear in x



- Regression: non-linear predictor, classification: non-linear decision boundary
- Types of non-linear features: quadratic, piecewise constant, etc.

Non-linear predictors with linear machinery

Roadmap

Stochastic Gradient Descent

Non-linear features

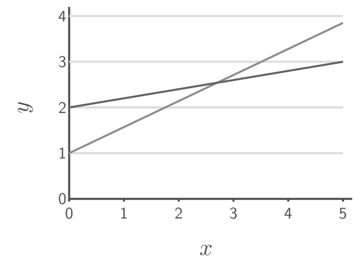
Neural networks

Feature templates

Non-linear predictors

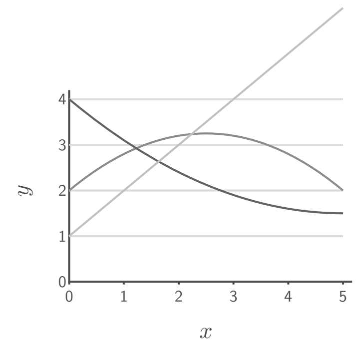
Linear predictors:

$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x),\ \phi(x) = [1, x]$



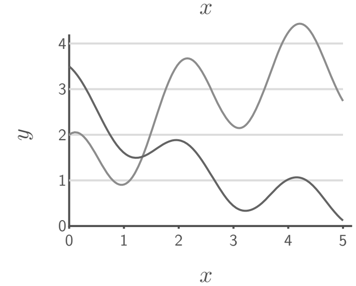
Non-linear (quadratic) predictors:

$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \phi(x),\ \phi(x) = [1, x, x^2]$



Non-linear neural networks:

$f_{\mathbf{w}}(x) = \mathbf{w} \cdot \sigma(\mathbf{V}\phi(x)),\ \phi(x) = [1, x]$



Motivating example

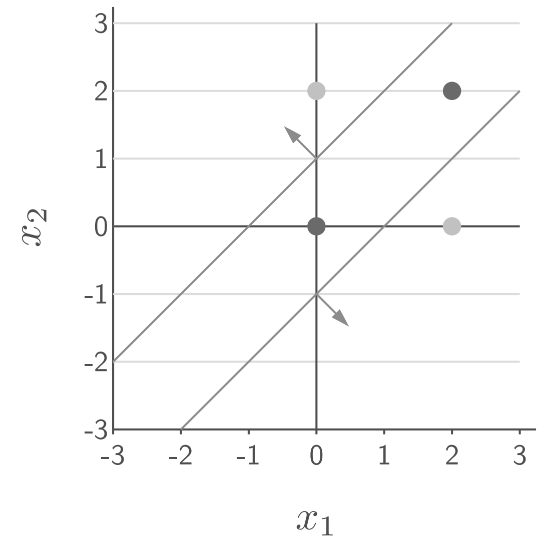
Example: predicting car collision

Input: positions of two oncoming cars $x = [x_1, x_2]$

Output: whether safe ($y = +1$) or collide ($y = -1$)

Unknown: safe if cars sufficiently far: $y = \text{sign}(|x_1 - x_2| - 1)$

x_1	x_2	y
0	2	1
2	0	1
0	0	-1
2	2	-1



Decomposing the problem

Test if car 1 is far right of car 2:

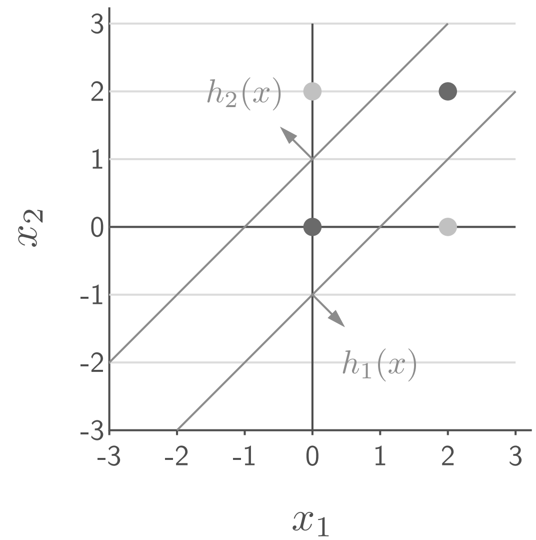
$$h_1(x) = \mathbf{1}[x_1 - x_2 \geq 1]$$

Test if car 2 is far right of car 1:

$$h_2(x) = \mathbf{1}[x_2 - x_1 \geq 1]$$

Safe if at least one is true:

$$f(x) = \text{sign}(h_1(x) + h_2(x))$$



x	$h_1(x)$	$h_2(x)$	$f(x)$
$[0, 2]$	0	1	$+1$
$[2, 0]$	1	0	$+1$
$[0, 0]$	0	0	-1
$[2, 2]$	0	0	-1

Rewriting using vector notation

Intermediate subproblems:

$$h_1(x) = \mathbf{1}[x_1 - x_2 \geq 1] = \mathbf{1}[[-1, +1, -1] \cdot [1, x_1, x_2] \geq 0]$$

$$h_2(x) = \mathbf{1}[x_2 - x_1 \geq 1] = \mathbf{1}[[-1, -1, +1] \cdot [1, x_1, x_2] \geq 0]$$

$$\mathbf{h}(x) = \mathbf{1}\left[\begin{bmatrix} -1 & +1 & -1 \\ -1 & -1 & +1 \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \geq 0\right]$$

Predictor:

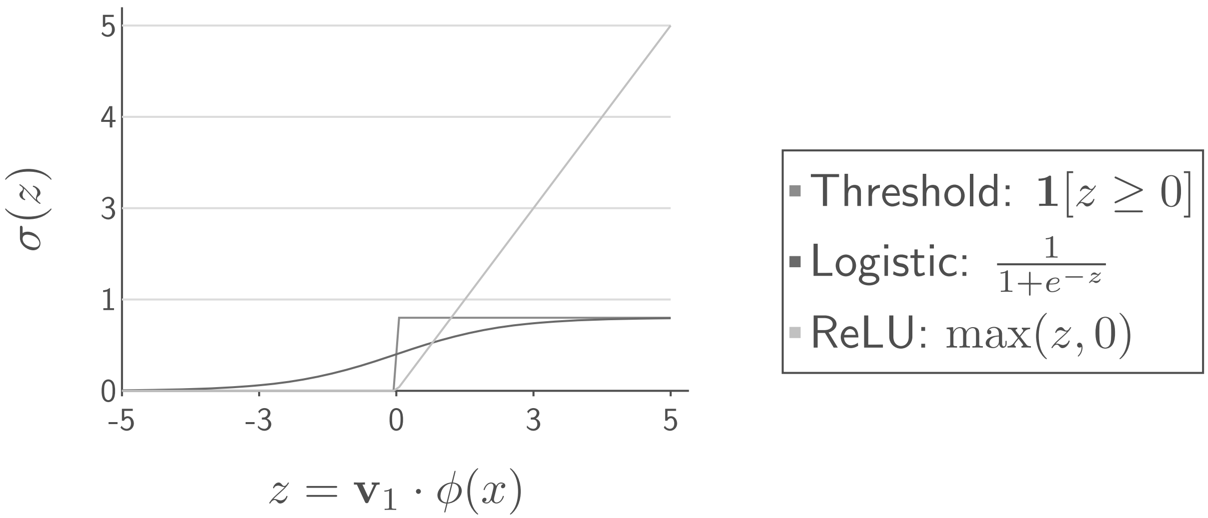
$$f(x) = \mathsf{sign}(h_1(x) + h_2(x)) = \mathsf{sign}([1, 1] \cdot \mathbf{h}(x))$$

Avoid zero gradients

Problem: gradient of $h_1(x)$ with respect to $\mathbf{v}_1$ is 0

$$h_1(x) = \mathbf{1}[\mathbf{v}_1 \cdot \phi(x) \geq 0]$$

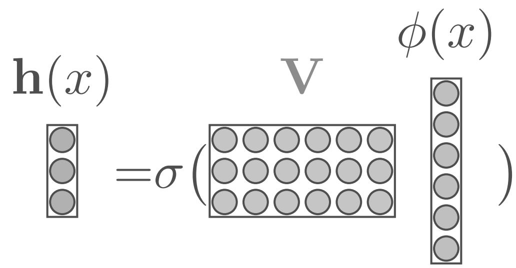
Solution: replace with an **activation function** σ with non-zero gradients



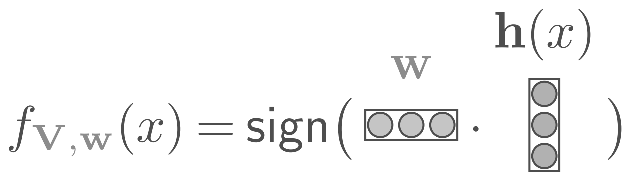
$$h_1(x) = \sigma(\mathbf{v}_1 \cdot \phi(x))$$

Two-layer neural networks

Intermediate subproblems:



Predictor (classification):



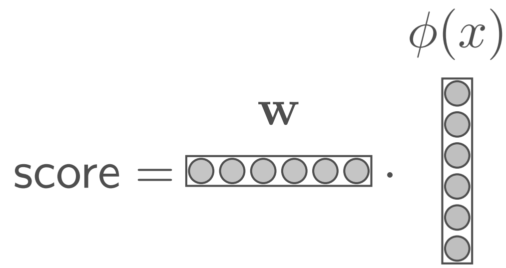
> Interpret $\mathbf{h}(x)$ as a learned feature representation!

Hypothesis class:

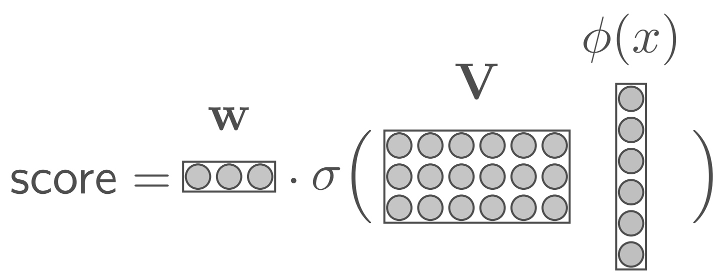
$$\mathcal{F} = \{f_{\mathbf{V},\mathbf{w}} : \mathbf{V} \in \mathbb{R}^{k\times d}, \mathbf{w} \in \mathbb{R}^k\}$$

Deep neural networks

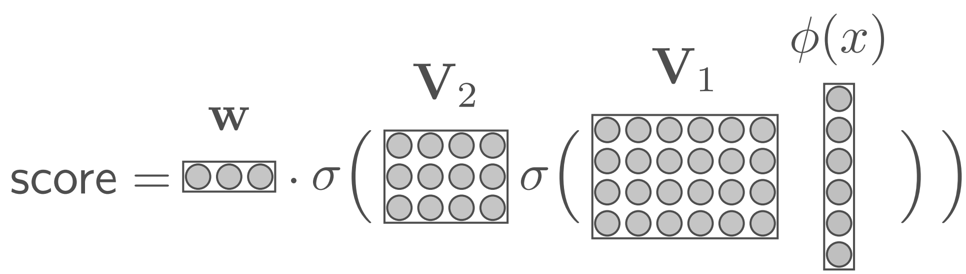
1-layer neural network:



2-layer neural network:

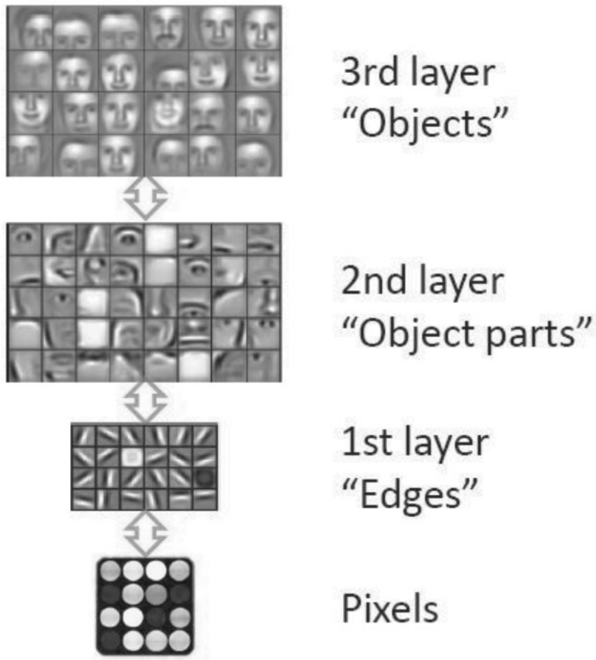


3-layer neural network:

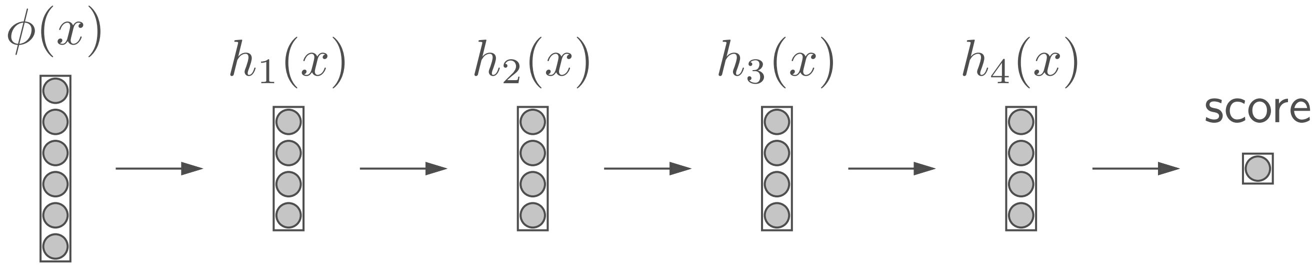


[figure from Honglak Lee]

Layers represent multiple levels of abstractions



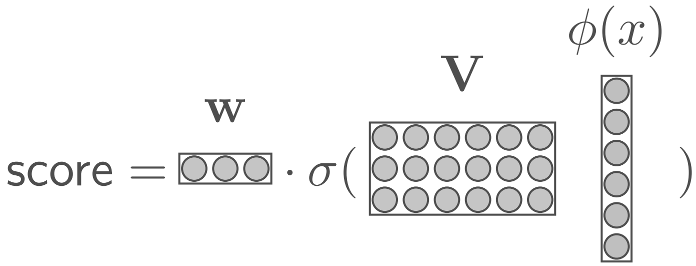
Why depth?



Intuitions:

- Multiple levels of abstraction
- Multiple steps of computation
- Empirically works well
- Theory is still incomplete

Summary



- Intuition: decompose problem into intermediate parallel subproblems
- Deep networks iterate this decomposition multiple times
- Hypothesis class contains predictors ranging over weights for all layers

Roadmap

Stochastic Gradient Descent

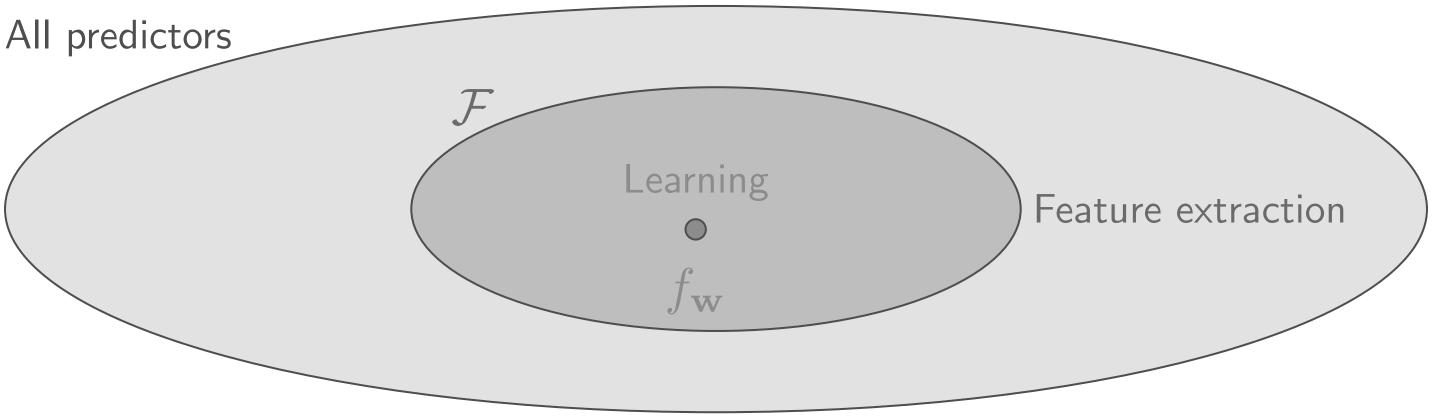
Non-linear features

Neural networks

Feature templates

Feature extraction + learning

$$\mathcal{F} = \{f_{\mathbf{w}}(x) = \mathsf{sign}(\mathbf{w} \cdot \phi(x)) : \mathbf{w} \in \mathbb{R}^d\}$$



- Feature extraction: choose $\mathcal{F}$ based on domain knowledge
- Learning: choose $f_{\mathbf{w}} \in \mathcal{F}$ based on data

Want $\mathcal{F}$ to contain good predictors but not be too big

Feature extraction with feature names

Example task:

classifier

string (x) $\longrightarrow$ $\boxed{f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x))}$ $\longrightarrow$ valid email address? (y)

Question: what properties of x **might be** relevant for predicting y?

Feature extractor: Given x, produce set of (feature name, feature value) pairs

abc@gmail.com

x

$\xrightarrow[\text{arbitrary!}]{\textbf{feature extractor } \phi}$

length>10	: 1
fracOfAlpha	: 0.85
contains_@	: 1
endsWith_com	: 1
endsWith_org	: 0

$\phi(x)$

Prediction with feature names

Weight vector $\mathbf{w} \in \mathbb{R}^d$

length>10	:-1.2
fracOfAlpha	:0.6
contains_@	:3
endsWith_com	:2.2
endsWith_org	:1.4

Feature vector $\phi(x) \in \mathbb{R}^d$

length>10	:1
fracOfAlpha	:0.85
contains_@	:1
endsWith_com	:1
endsWith_org	:0

Score: weighted combination of features

$$\mathbf{w} \cdot \phi(x) = \sum_{j=1}^{d} w_j \phi(x)_j$$

Example: $-1.2(1) + 0.6(0.85) + 3(1) + 2.2(1) + 1.4(0) = 4.51$

Organization of features?

abc@gmail.com

x

$$\xrightarrow[\text{arbitrary!}]{\textbf{feature extractor } \phi}$$

length>10	: 1
fracOfAlpha	: 0.85
contains_@	: 1
endsWith_com	: 1
endsWith_org	: 0

$\phi(x)$

Which features to include? Need an organizational principle...

Feature templates

Definition: feature template

A **feature template** is a group of features all computed in a similar way.

abc@gmail.com → last three characters equals ___

endsWith_aaa	: 0
endsWith_aab	: 0
endsWith_aac	: 0
...	
endsWith_com	: 1
...	
endsWith_zzz	: 0

Define types of pattern to look for, not particular patterns

Feature templates example 1

Input:

abc@gmail.com

Feature template	Example feature	
Last three characters equals ___	Last three characters equals *com*	: 1
Length greater than ___	Length greater than *10*	: 1
Fraction of alphanumeric characters	Fraction of alphanumeric characters	: 0.85

Feature templates example 2

Input:

Latitude: 37.4068176
Longitude: -122.1715122

Feature template	Example feature name	
Pixel intensity of image at row ___ and column ___ (___ channel)	Pixel intensity of image at row *10* and column *93* (*red* channel)	: 0.8
Latitude is in [___, ___] and longitude is in [___, ___]	Latitude is in [*37.4*, *37.5*] and longitude is in [*-122.2*, *-122.1*]	: 1

Sparsity in feature vectors

abc@gmail.com $\xrightarrow{}$ last character equals ___

endsWith_a	: 0
endsWith_b	: 0
endsWith_c	: 0
endsWith_d	: 0
endsWith_e	: 0
endsWith_f	: 0
endsWith_g	: 0
endsWith_h	: 0
endsWith_i	: 0
endsWith_j	: 0
endsWith_k	: 0
endsWith_l	: 0
endsWith_m	: **1**
endsWith_n	: 0
endsWith_o	: 0
endsWith_p	: 0
endsWith_q	: 0
endsWith_r	: 0
endsWith_s	: 0
endsWith_t	: 0
endsWith_u	: 0
endsWith_v	: 0
endsWith_w	: 0
endsWith_x	: 0
endsWith_y	: 0
endsWith_z	: 0

Compact representation:

{`"endsWith_m": 1`}

Two feature vector implementations

Arrays (good for dense features):

pixelIntensity(0,0)	: 0.8
pixelIntensity(0,1)	: 0.6
pixelIntensity(0,2)	: 0.5
pixelIntensity(1,0)	: 0.5
pixelIntensity(1,1)	: 0.8
pixelIntensity(1,2)	: 0.7
pixelIntensity(2,0)	: 0.2
pixelIntensity(2,1)	: 0
pixelIntensity(2,2)	: 0.1

`[0.8, 0.6, 0.5, 0.5, 0.8, 0.7, 0.2, 0, 0.1]`

Dictionaries (good for sparse features):

fracOfAlpha	: 0.85
contains_a	: 0
contains_b	: 0
contains_c	: 0
contains_d	: 0
contains_e	: 0
...	
contains_@	: 1
...	

`{"fracOfAlpha": 0.85, "contains_@": 1}`

Summary

$$\mathcal{F} = \{f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \phi(x)) : \mathbf{w} \in \mathbb{R}^d\}$$

Feature template:

abc@gmail.com $\xrightarrow{}$

last three characters equals ___

endsWith_aaa	: 0
endsWith_aab	: 0
endsWith_aac	: 0
...	
endsWith_com	: 1
...	
endsWith_zzz	: 0

Dictionary implementation:

{"endsWith_com": 1}

Overall Summary

- Stochastic Gradient Descent: faster gradient descent using sample gradients
- Non-Linear Features: Linear in weights $\mathbf{w}$, but nonlinear in inputs x
- Neural networks: Learning hierarchical feature representations
- Feature templates: useful for organizing the definition of many features,
- Next: Backpropagation, k-means, generalization, best practices