Machine Learning III

Dr. Angelica Lim
Assistant Professor
School of Computing Science
Simon Fraser University, Canada

Oct. 1, 2024

Course Overview

Week 1 : Getting to know you
Week 2 : Introduction to Artificial Intelligence
Week 3 : Machine Learning I: Basic Supervised Models (Classification)
Week 4 : Machine Learning II: Supervised Regression, Classification and Gradient Descent, K-Means
Week 5 : Machine Learning III: Neural Networks and Backpropagation
Week 6 : Search (A*)
Week 7 : Markov Decision Processes
Week 8 : Midterm
Week 9 : Reinforcement Learning
Week 10 : Games
Week 11 : Hidden Markov Models and Bayesian Networks
Week 12 : Constraint Satisfaction Problems
Week 13 : Ethics and Explainability

Reflex-based models	State-based models	Variable-based models	Logic-based models
	Search Markov decision processes Games	Constraint satisfaction problems Markov networks Bayesian networks	

Today's Plan

Supervised Learning

19.6.2 - Stochastic Gradient Descent
19.6.3 - Multivariable Linear Regression
19.4.3 - Dimensionality and Regularization

21.1 - Neural Networks
21.1.2 - Backpropagation

Course Announcements

Assignment 1 is released and due October 7

No class on Tues, October 15 due to holiday as per SFU guidelines

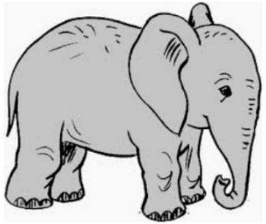
Midterm is set for Tues, October 29 in class.

Supervised Learning

Stochastic Gradient Descent

Gradient Descent is Slow

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



Algorithm: gradient descent

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

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

$\quad \mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})$

We only update **w** after finding the average over all samples

Stochastic gradient descent

Algorithm: stochastic gradient descent

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

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

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

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

We update **w** after finding the loss of each sample

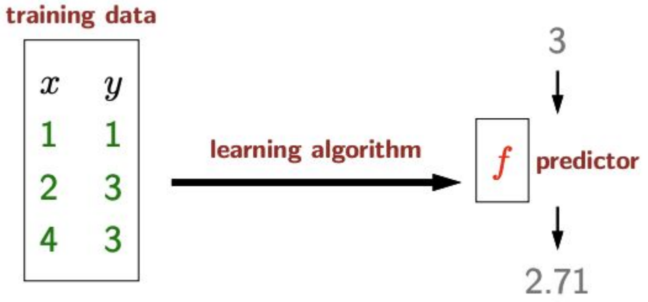
Variants of SGD include 1) minibatch, to update using an average over B samples, 2) randomizing the order over the samples. (*Why would this be important? What if you had all the positive samples first, followed by negative samples?)*

For assignment, you have to think if the feature will be useful considering the data you have, and whether it will scale well.

Supervised Learning

Nonlinear Functions

Recall: Linear Regression



Which predictors are possible?

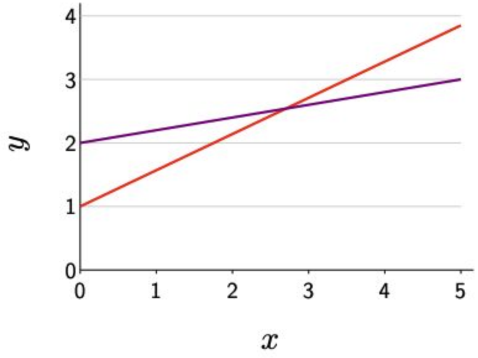
Hypothesis class

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

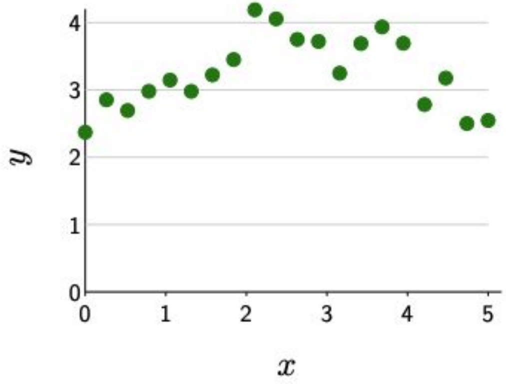
$\phi(x) = [1, x]$

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

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



More complex data



How do we fit a non-linear predictor?

From linear to quadratic predictors

We can get a non-linear predictor just by changing the feature extractor → $\varphi(x) = [1, x, x^2]$ e.g. $\varphi(3) = [1, 3, 9]$

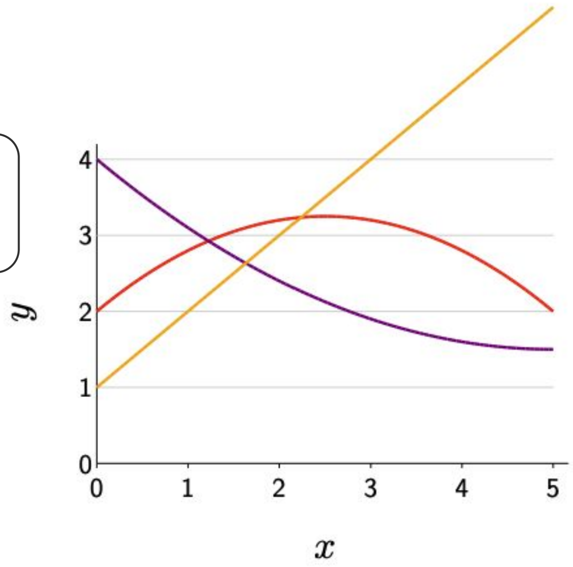
$$f(x) = [2, 1, -0.2] \cdot \phi(x)$$

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

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

If we set x^2 to zero, we recover a linear predictor

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



Piecewise constant predictors

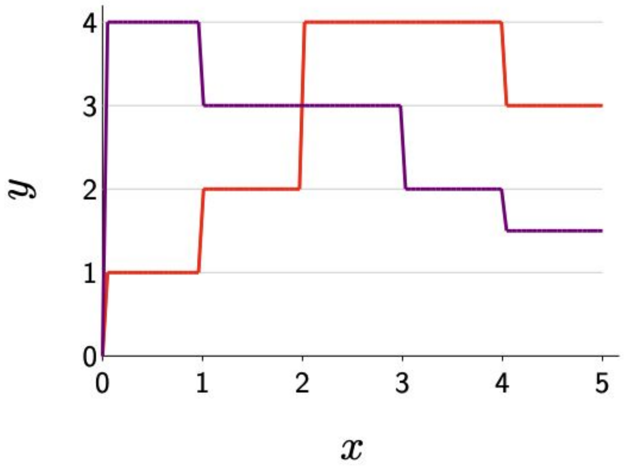
We can also obtain a non-linear predictor by partitioning the input space

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

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

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

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



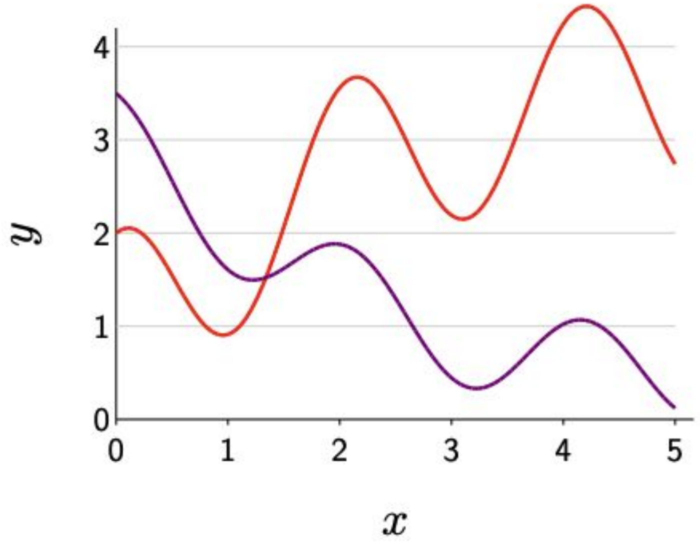
Predictors with periodicity

$\varphi(x) = [1, x, x^2, \cos(3x)]$ e.g. $\varphi(2) = [1, 2, 4, 0.96]$

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

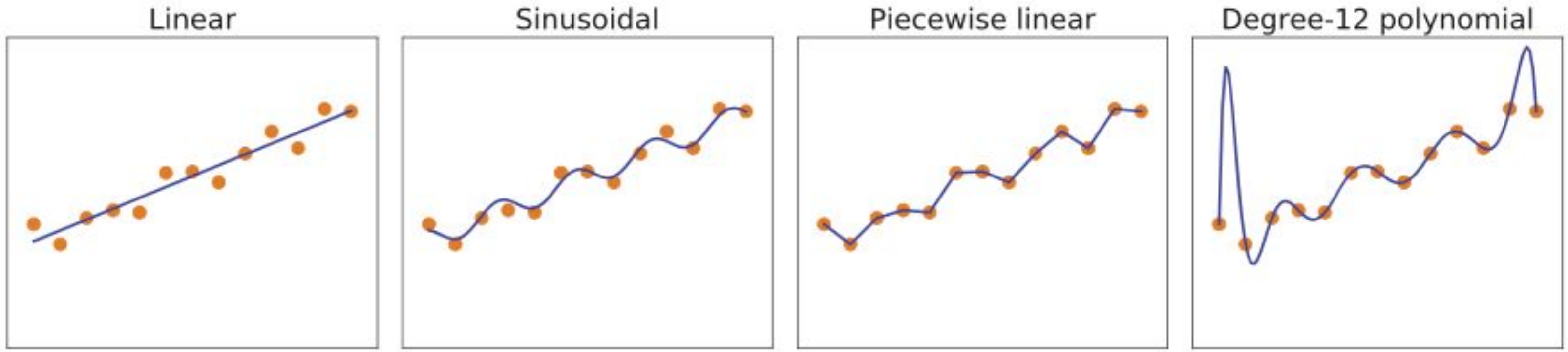
$$f(x) = [3, -1, 0.1, 0.5] \cdot \phi(x)$$

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



What prediction task might be periodic?

Many hypothesis classes are possible!



Linear in what?

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

Linear in $\mathbf{w}$? Yes

Linear in $\phi(x)$? Yes

Linear in x? No!

Indeed, the raw values of our data may not even be numbers

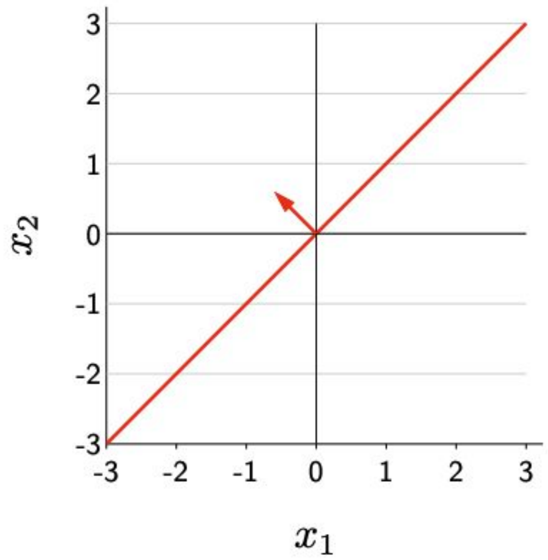
$\mathbf{w} \cdot \phi(x)$ can be a **non-linear** function of x

$\mathbf{w} \cdot \phi(x)$ always a **linear** function of $\mathbf{w}$

If the **function output w · φ(x) is linear in w** and the **loss function is convex** (squared, hinge, logistic losses but not zero-one loss), then minimizing the training loss with gradient descent and a proper step size is guaranteed to converge to the global minimum.

Recall: Linear classification

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



The decision boundary is a line

From linear to quadratic classifiers

Add a new feature $x_1^2+x_2^2$ into the feature vector

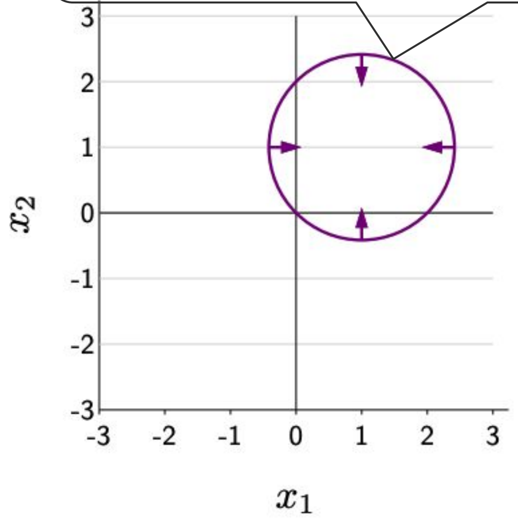
$$\phi(x) = [x_1, x_2, x_1^2 + x_2^2]$$
$$f(x) = \text{sign}([2, 2, -1] \cdot \phi(x))$$

Equivalently:

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

What is the result of x = [0,0]?

Inside is classified as +1, and outside is classified as -1



The decision boundary is a circle

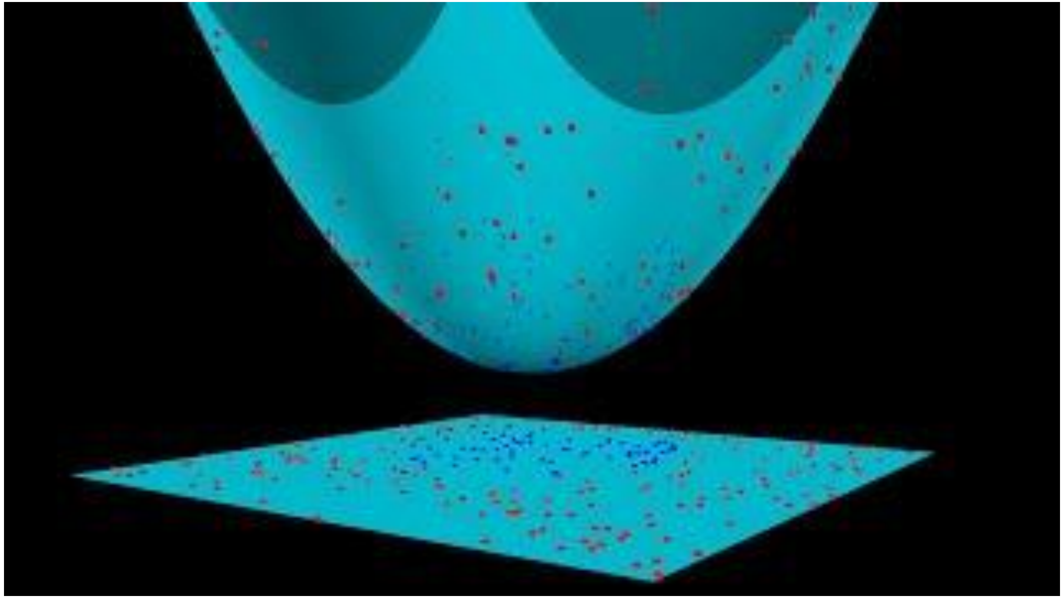
Visualization in feature space

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

Want to create a separator where the points inside a circle belong to one group and the points outside belong to another group.

If your initial feature space is not expressive enough, projecting into a higher dimension will allow you to linearly separate your data points.

Decision trees are very expressive.
For decision trees, you need to handcraft many of your features.

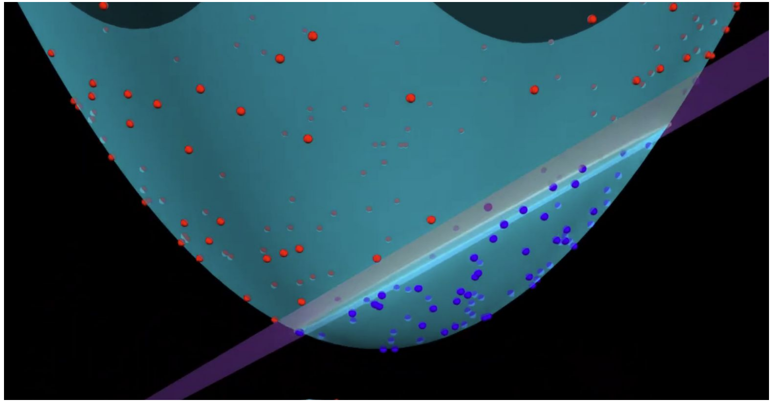


Summary

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

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.

Supervised Learning

Neural Networks

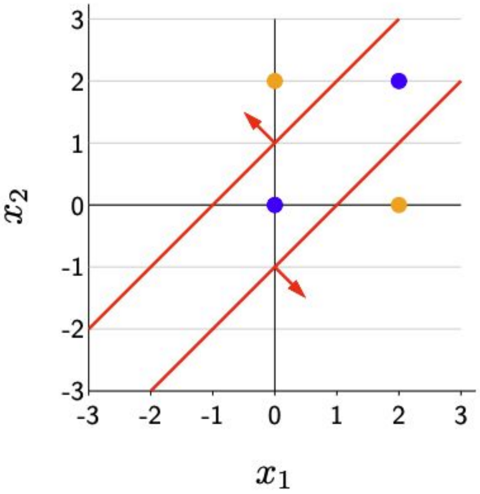
Motivating example

Example: predicting car collision

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

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

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



Safe if cars are sufficiently far:

$$y = \text{sign}(|x_1 - x_2| - 1)$$

The famous XOR problem that is impossible to fit using a linear classifier

Decomposing the problem

Test if car 1 is far right of car 2:

$$h_1(x) = 1[x1 - x2 \geq 1]$$

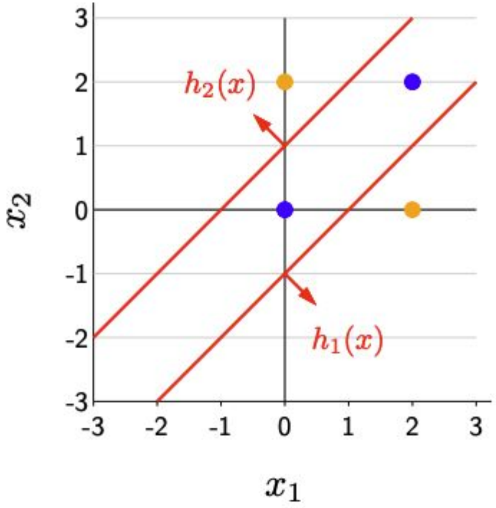
Test if car 2 is far right of car 1:

$$h_2(x) = 1[x2 - x1 \geq 1]$$

Safe if at least one is true:

$$f(x) = \text{sign}(h1(x) + h2(x))$$

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

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

$$h_1(x) = 1[x_1 - x_2 \geq 1]$$
$$\rightarrow \quad 1[-1 + x_1 - x_2 \geq 0]$$
$$\rightarrow \quad 1[[-1, 1, -1] \cdot [1, x_1, x_2] \geq 0]$$

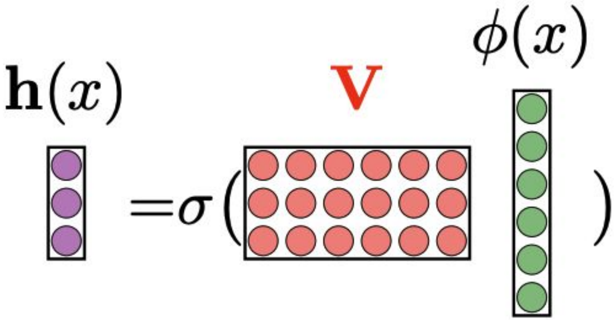
$$\mathbf{h}(x) = \mathbf{1}\left[\begin{array}{c} \\ \text{v1} \\ \text{v2} \end{array}\begin{bmatrix} -1 & +1 & -1 \\ -1 & -1 & +1 \end{bmatrix}\begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \geq 0\right]$$

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

$$h_2(x) = 1[x_2 - x_1 \geq 1]$$
$$\rightarrow \quad 1[-1 - x_1 + x_2 \geq 0]$$
$$\rightarrow \quad 1[[-1, -1, 1] \cdot [1, x_1, x_2] \geq 0]$$

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

Two-layer neural networks

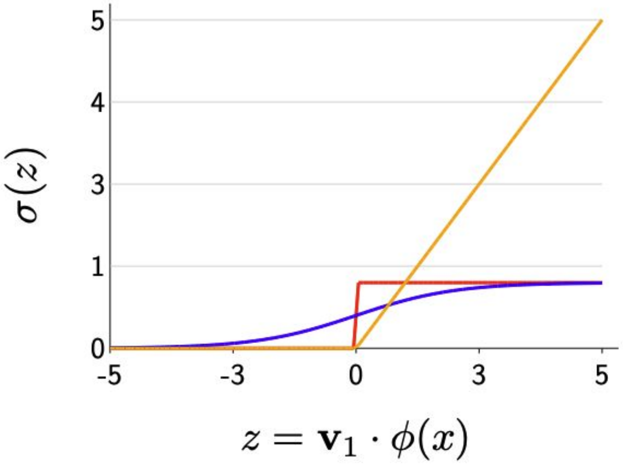


Avoiding zero gradients

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



Threshold: $\mathbf{1}[z \geq 0]$
Logistic: $\frac{1}{1+e^{-z}}$
ReLU: $\max(z, 0)$

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

Two-layer neural networks

Intermediate subproblems:

$$\mathbf{h}(x) = \sigma(\mathbf{V}\,\phi(x))$$

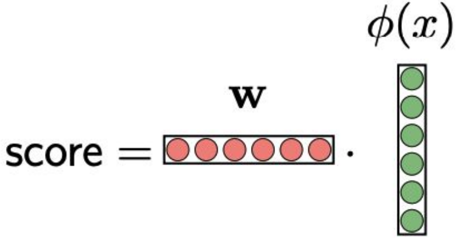
Predictor (classification):

$$f_{\mathbf{V},\mathbf{w}}(x) = \text{sign}(\mathbf{w} \cdot \mathbf{h}(x))$$

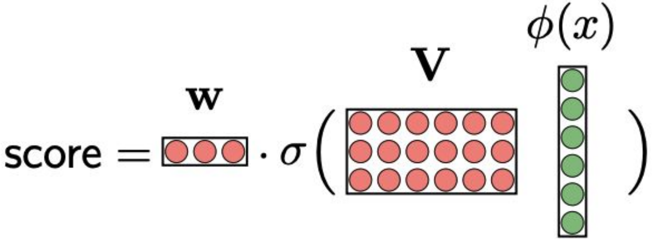
We can interpret $\mathrm{h}(\mathrm{x})$ as a learned feature representation!

Deep neural networks

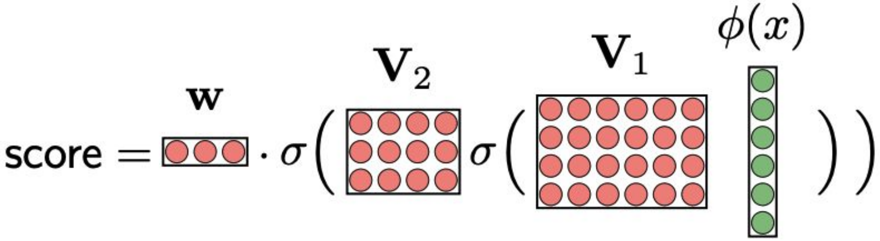
1-layer network:



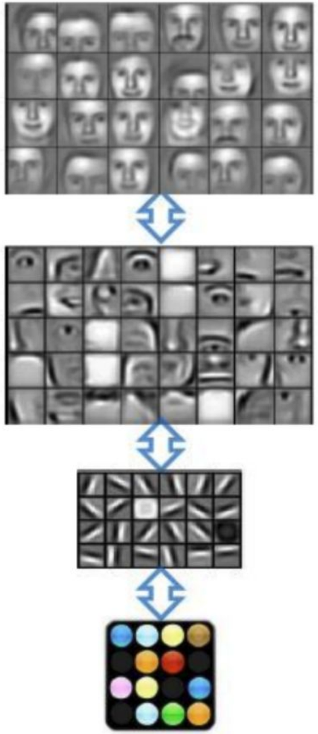
2-layer network:



3-layer network:



Layers represent multiple layers of abstraction



3rd layer
"Objects"

2nd layer
"Object parts"

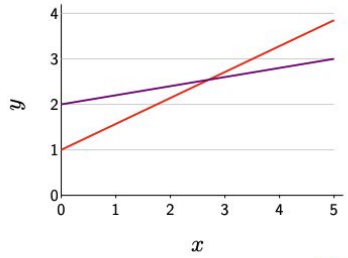
1st layer
"Edges"

Pixels

Summary

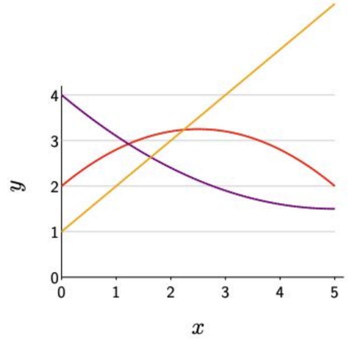
Linear predictors:

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



Non-linear (quadratic) predictors:

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



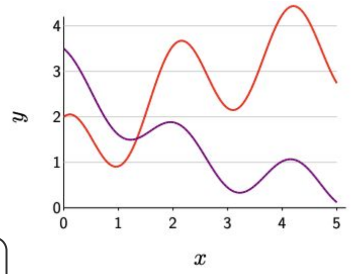
V is like w, but it allows for multiple subproblems.

Non-linear neural networks:

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

Logistic function

Weights on previous layer

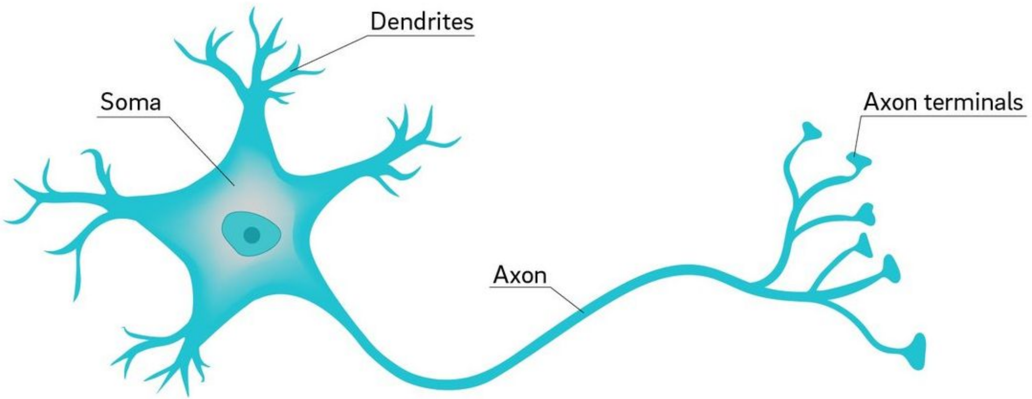


Tell Angelica that her slide ordering does not match the slides posted on Canvas

Artificial neural networks

They are inspired from actual neurons in human/animal brain.

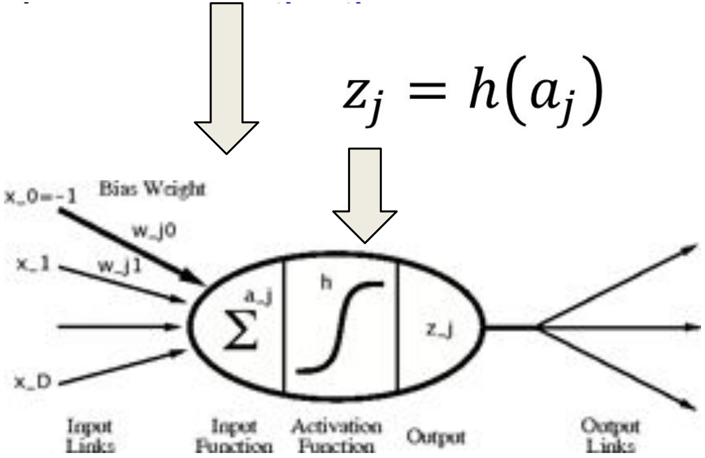
- **Dendrites**: Take its input from other neurons in the form of electrical impulses.
- **Soma**: Generates inferences from inputs and decides what action to take.
- **Axon terminals**: Transmit outputs in the form of impulses.



Artificial neuron cell (aka perceptron)

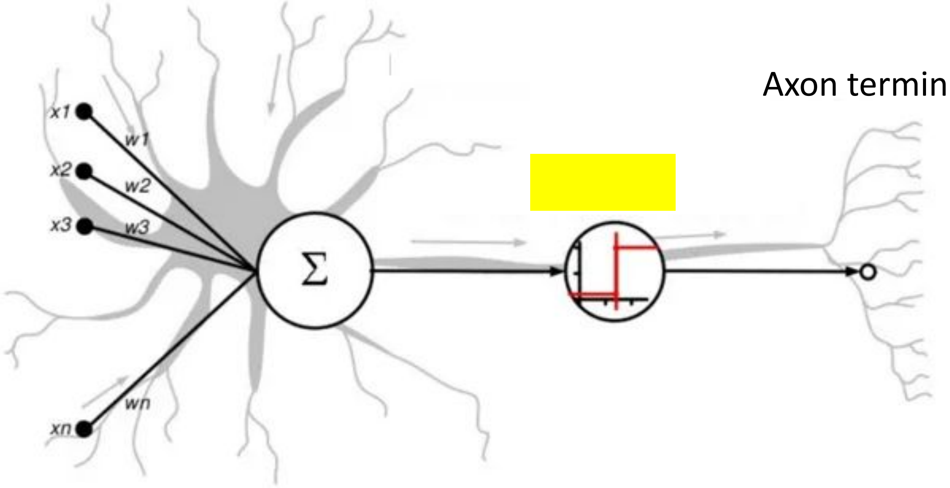
$$a_j = \sum_{i=1}^{D} \left(w_{ji}^{(1)} x_i + x_{j0}^{(1)} \right)$$

$$z_j = h(a_j)$$

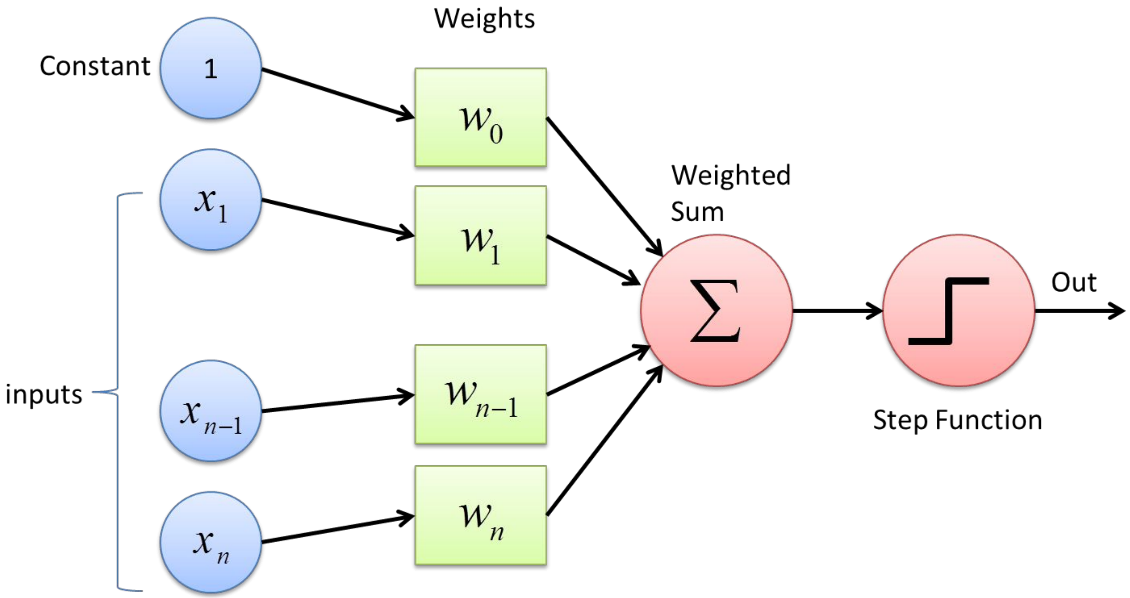


Dendrites

Axon terminals



Artificial neuron cell (aka perceptron)



Backpropagation

Motivation: regression with 4-layer neural network

Loss on one example:

$$\text{Loss}(x, y, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{w}) = (\mathbf{w} \cdot \sigma(\mathbf{V}_3 \sigma(\mathbf{V}_2 \sigma(\mathbf{V}_1 \phi(x)))) - y)^2$$

(Stochastic) gradient descent:

$$\mathbf{V}_1 \leftarrow \mathbf{V}_1 - \eta \nabla_{\mathbf{V}_1} \text{Loss}(x, y, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{w})$$

$$\mathbf{V}_2 \leftarrow \mathbf{V}_2 - \eta \nabla_{\mathbf{V}_2} \text{Loss}(x, y, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{w})$$

$$\mathbf{V}_3 \leftarrow \mathbf{V}_3 - \eta \nabla_{\mathbf{V}_3} \text{Loss}(x, y, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{w})$$

This can get tedious!

Computation graphs

$$\text{Loss}(x, y, \mathbf{V}_1, \mathbf{V}_2, \mathbf{V}_3, \mathbf{w}) = (\mathbf{w} \cdot \sigma(\mathbf{V}_3 \sigma(\mathbf{V}_2 \sigma(\mathbf{V}_1 \phi(x)))) - y)^2$$

Definition: computation graph

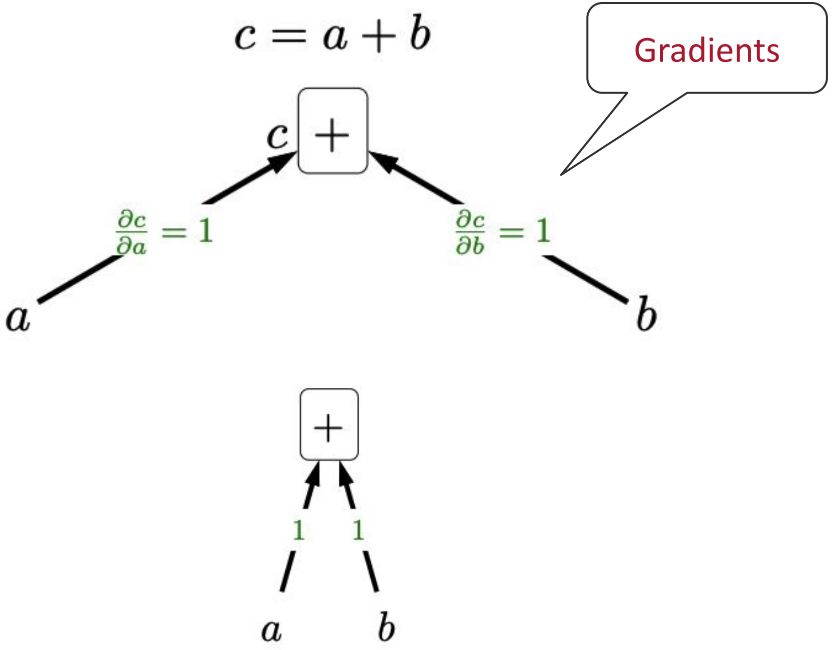
A directed acyclic graph whose root node represents the final mathematical expression and each node represents intermediate subexpressions.

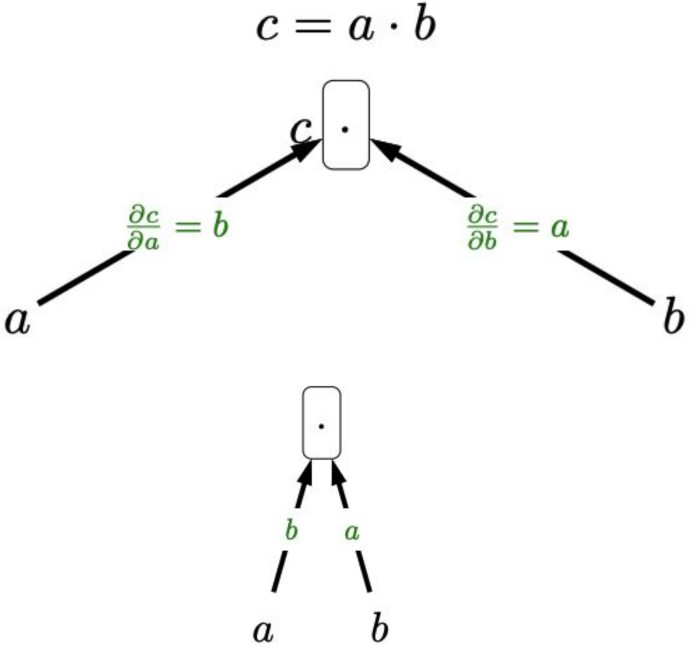
Upshot: compute gradients via general **backpropagation** algorithm

Purposes:

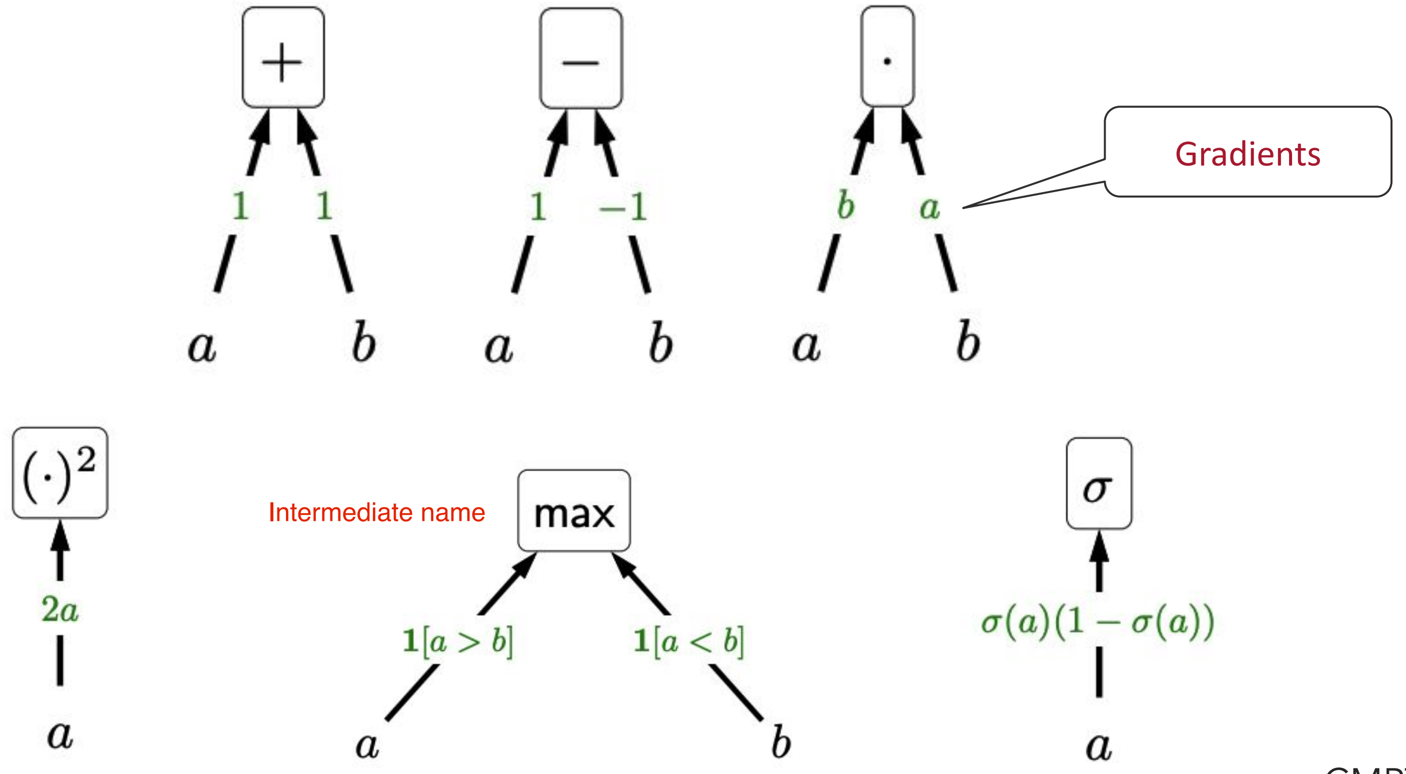
- Automatically compute gradients (how TensorFlow and PyTorch work)
- Gain insight into modular structure of gradient computations

Functions as boxes





Basic building blocks

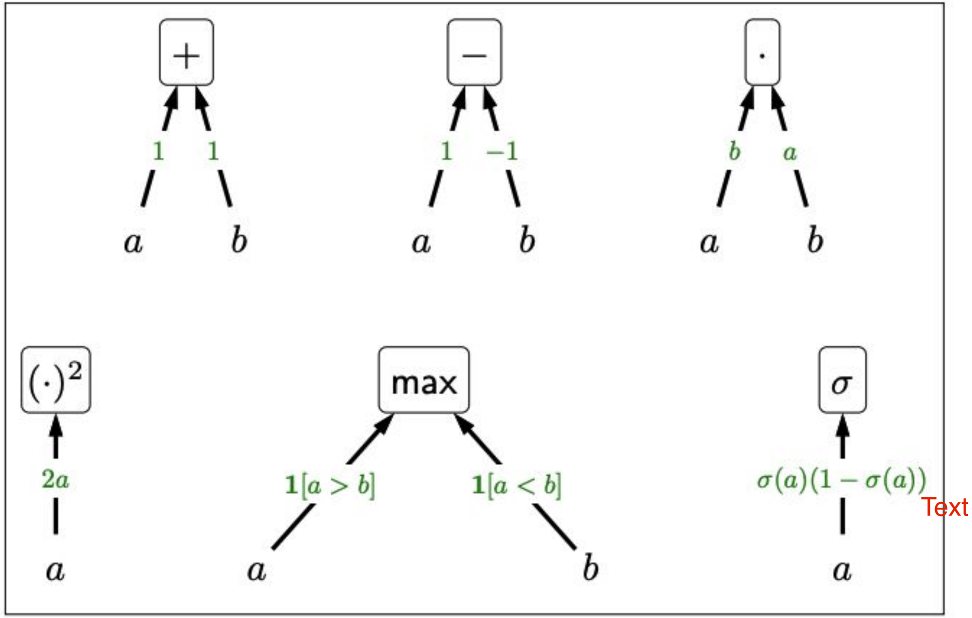


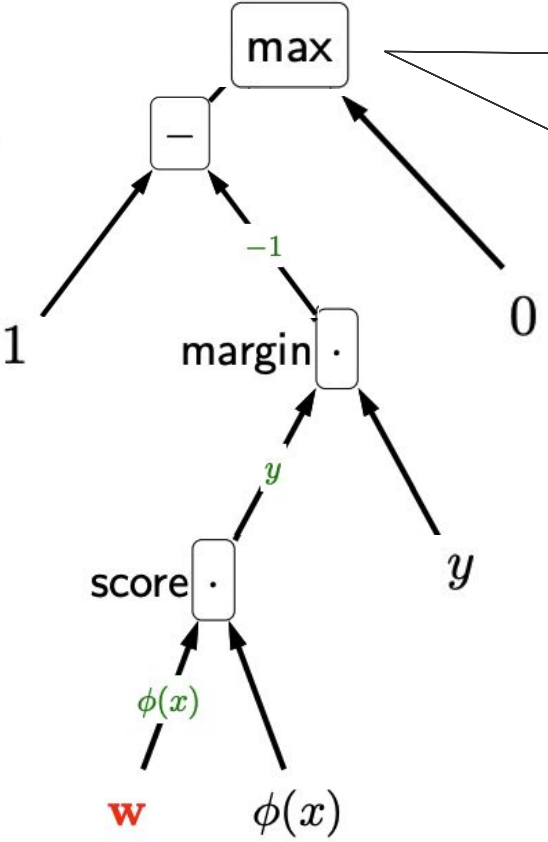
Linear classification with hinge loss

$$\text{Loss}(x, y, \mathbf{w}) = \max\{1 - \mathbf{w} \cdot \phi(x) y, 0\}$$

$$\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w}) = -\mathbf{1}[\text{margin} < 1]\phi(x)y$$

Can use the computation graph to obtain the gradient





Just the product of the gradients on the path from **w** to the root!

Two-layer neural networks

Recall squared loss:

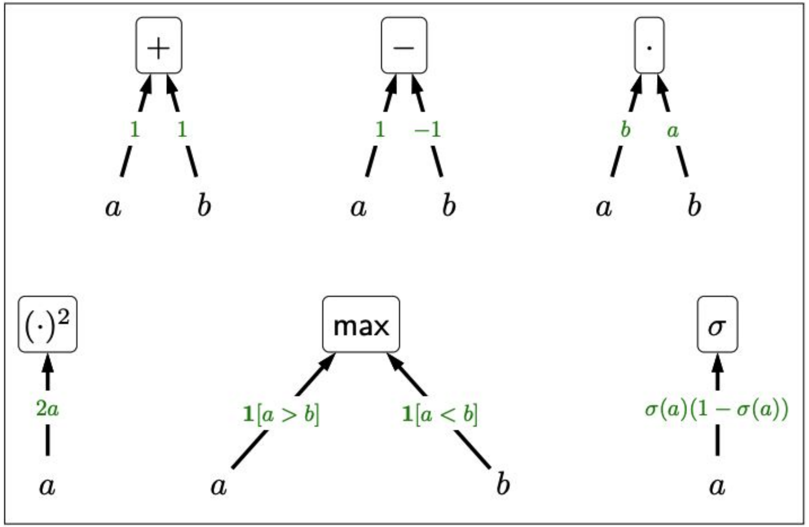
Review computation graph

$$\text{Loss}(x, y, \mathbf{w}) = (f_{\mathbf{w}}(x) - y)^2$$

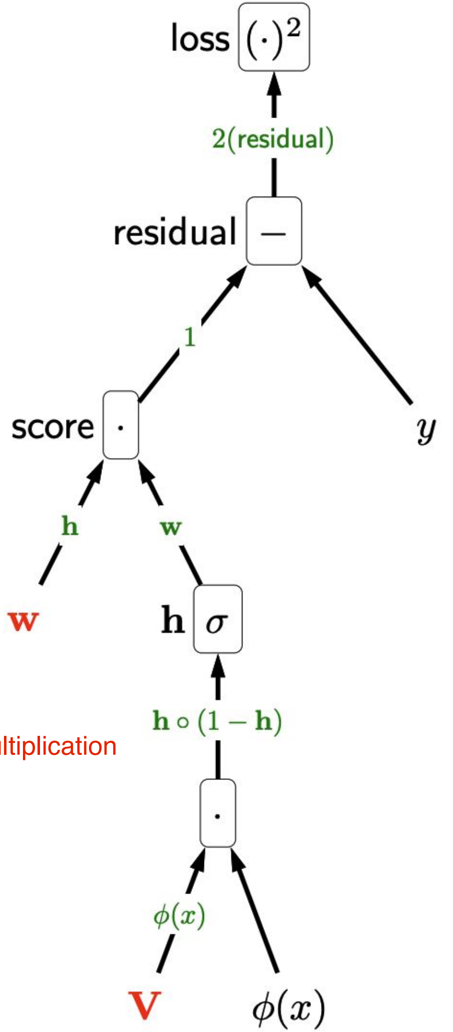
$$\text{Loss}(x, y, \mathbf{V}, \mathbf{w}) = (\mathbf{w} \cdot \sigma(\mathbf{V}\phi(x)) - y)^2$$

$$\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{V}, \mathbf{w}) = 2(\text{residual})\mathbf{h}$$

$$\nabla_{\mathbf{V}} \text{Loss}(x, y, \mathbf{V}, \mathbf{w}) = 2(\text{residual})\mathbf{w} \circ \mathbf{h} \circ (1 - \mathbf{h})\phi(x)^{\top}$$



o is element-wise multiplication



Backpropagation

$$\text{Loss}(x, y, \mathbf{w}) = (\mathbf{w} \cdot \phi(x) - y)^2$$

$$\mathbf{w} = [3, 1], \phi(x) = [1, 2], y = 2$$

Algorithm: backpropagation algorithm

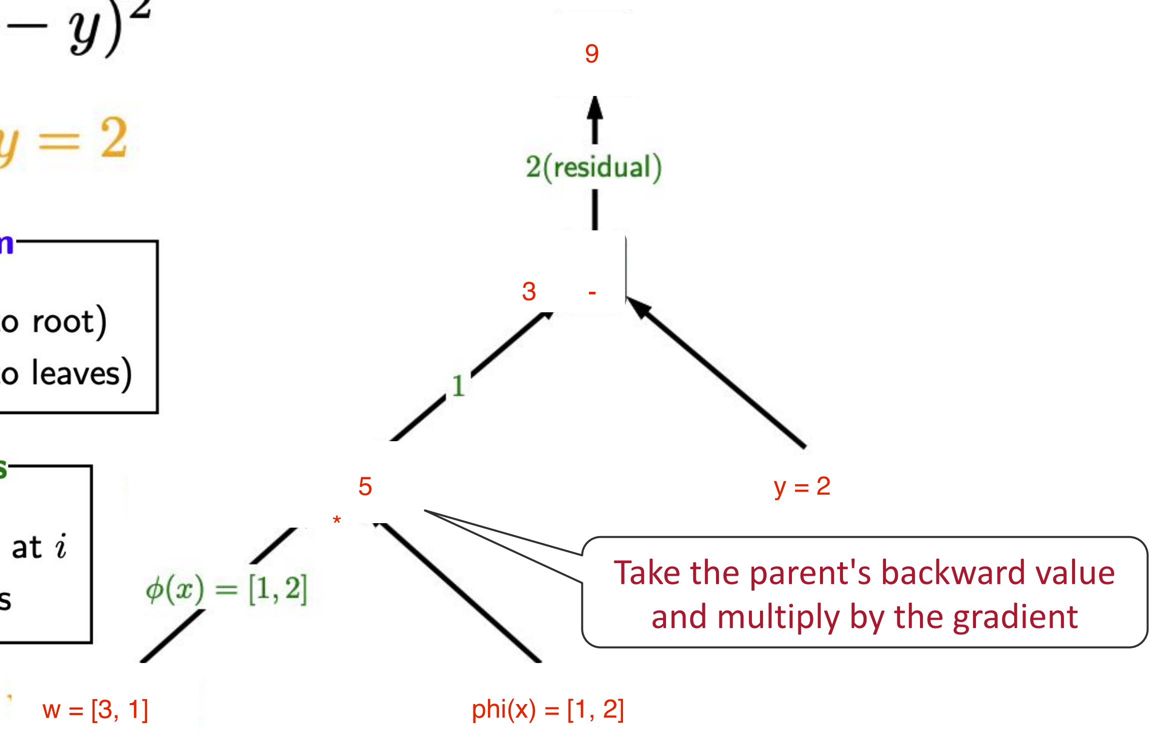
Forward pass: compute each f_i (from leaves to root)

Backward pass: compute each g_i (from root to leaves)

Definition: Forward/backward values

Forward: f_i is value for subexpression rooted at i

Backward: $g_i = \frac{\partial \text{loss}}{\partial f_i}$ is how f_i influences loss



$$\nabla_{\mathbf{w}} \text{Loss}(x, y, \mathbf{w}) = [6, 12]$$

Machine Learning Strategies

How do you prevent overfitting?

$\mathcal{D}_{\text{train}} \longrightarrow$ learning algorithm $\longrightarrow f$

How good is the predictor f?

Key idea: the real learning objective

Our goal is to minimize **error on unseen future examples**.

As the hypothesis class size increases, the estimation error increases

Strategy 1: Reduce Dimensionality

Manual feature selection:

- Add features if they help
- Remove features if they don’t help

Automatic feature selection (beyond the scope of this class):

- Boosting

Strategy 2: Regularization

L2 regularization penalizes the norm (length) of **w** by λ.

$$\min_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

Algorithm: gradient descent

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

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

$\quad \mathbf{w} \leftarrow \mathbf{w} - \eta(\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w}) + \lambda \mathbf{w})$

Note: Support vector machines are exactly hinge loss + L2 regularization