香港浸會大學
HONG KONG BAPTIST UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE
計算機科學系

COMP7015 Artificial Intelligence

Lecture 7: Multiclass Classification and Beyond Linearity

Instructor: Dr. Kejing Yin

October 27, 2022

Logistics

- Programming Assignment 1 is out. Due: 23:59 pm on Nov. 6
  - Start early
  - Finish on your own

- Office Hour this week (Oct. 29, Sat.) will be canceled
  - Post questions in Piazza

- If you choose Topic 3 for the group project, submit your project proposal by the end of Friday (Oct. 28).

Agenda for Today

- Review of Quiz Sample Solutions
- Multi-class Classification: Methods and Evaluation
- Feature Extraction/Engineering: Beyond Linearity
- Deep Learning Basics
  - Neurons and layers
  - Activation functions
  - Backpropagation

Quiz Question Review: Recap of first-order logic

- Why the universal quantifier ∀ always pairs with "⇒"?

- Recall its semantics:
  "$\forall x\ P$" is true in a given model if P is true in all possible extended interpretations.

  All kings are persons: $\forall x\ \text{King}(x) \Rightarrow \text{Person}(x)$ ✓

Three possible extended interpretations

$x \rightarrow$ William Shakespeare, W. Shakespeare is a King (false) ⇒ W. Shakespeare is a person ✓
$x \rightarrow$ King George V, King George V is a King (true) ⇒ King George V is a person (true). ✓
$x \rightarrow$ Tom Cat Tom Cat is a King (false) ⇒ Tom Cat is a person. ✓

- How about $\forall x\ \text{King}(x) \wedge \text{Person}(x)$? ✗ *Everything is both a King and a Person*

Three possible extended interpretations

$x \rightarrow$ William Shakespeare, W. Shakespeare is a King (false) ∧ W. Shakespeare is a person (true) ✗
$x \rightarrow$ King George V, King George V is a King (true) ∧ King George V is a person (true). ✓
$x \rightarrow$ Tom Cat Tom Cat is a King (false) ∧ Tom Cat is a person (false). ✗

Quiz Question Review: Recap of first-order logic

- Similarly, why the existence quantifier ∃ always pairs with "∧" ?

- Recall its semantics:
  "$\exists x\ P$" is true in a given model if P is true in at least one possible extended interpretations.

Some students know Python: $\exists x$ Student(x) ∧ Knows(x, Python)

Three possible extended interpretations

$x \rightarrow$ Alice,	Alice is a Student (true) ∧ Alice knows Python (true). ✓
$x \rightarrow$ Harry,	Harry is a Student (false) ∧ Harry knows Python (true).
$x \rightarrow$ Tom Cat	Tom Cat is a Student (false) ∧ Tom Cat knows Python (false).

$\exists x$ Student(x) ∧ Knows(x, Python) ✓

Quiz Question Review: Recap of first-order logic

- Similarly, why the existence quantifier ∃ always pairs with "∧" ?

Some students are from Mars: $\exists x$ Student(x) ∧ FromMars(x) ✗

possible extended interpretations

$x \rightarrow$ Alice,	Alice is a Student (true) ∧ Alice is from Mars(false). ✗
$x \rightarrow$ Harry,	Harry is a Student (false) ∧ Harry is from Mars(false). ✗
$x \rightarrow$ Tom Cat	Tom Cat is a Student (false) ∧ Tom Cat is from Mars(false). ✗
x → …	✗

How about: $\exists x$ Student(x) ⇒ FromMars(x) ? *This formula would become true*

possible extended interpretations

$x \rightarrow$ Alice,	Alice is a Student (true) ⇒ Alice is from Mars(false). ✗
$x \rightarrow$ Harry,	Harry is a Student (false) ⇒ Harry is from Mars(false). ✓
$x \rightarrow$ Tom Cat	Tom Cat is a Student (false) ⇒ Tom Cat is from Mars(false). ✓

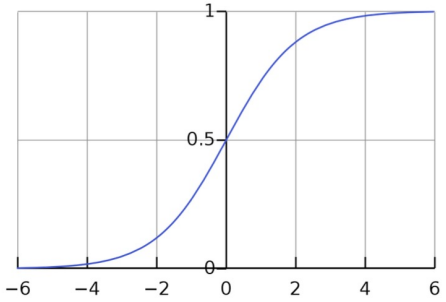
$\exists x$ Student(x) ⇒ FromMars(x) is synthetically valid but cannot express our desired semantics.

Multiclass Classification: Methods and Evaluation

Recap: Logistic Regression for Binary Classification

Consider this transfromation retruns a "probability":

- $p(y = +1|x) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x})}$
- $p(y = -1|x) = 1 - \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{\exp(-\mathbf{w}^\top \mathbf{x})}{1+\exp(-\mathbf{w}^\top \mathbf{x})} = \frac{1}{1+\exp(+\mathbf{w}^\top \mathbf{x})}$



- The *log likelihood* (numerically more stable):

$$\ell(\mathbf{w}) = \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} \mathbb{1}[y_i = +1] \log p(y_i = +1|\mathbf{x}) + \mathbb{1}[y_i = -1] \log p(y_i = -1|\mathbf{x})$$

- We can maximize the likelihood, or equivalently, minimize the negative likelihood:

$$\text{Loss}(\mathbf{x}, y, \mathbf{w}) = \log(1 + \exp(-(\mathbf{w}^\top \mathbf{x})y))$$

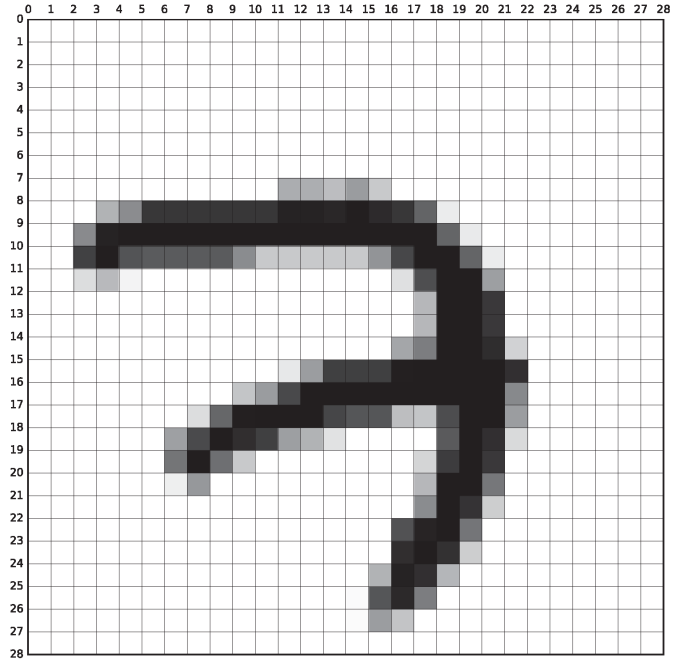
Features			Label
surface	texture	density	ripe?
hard	clear	0.77	yes
hard	clear	0.56	yes
hard	slightly blurry	0.64	no
hard	clear	0.61	yes
hard	clear	0.63	yes
hard	slightly blurry	0.66	no
hard	slightly blurry	0.67	no
hard	clear	0.44	yes
soft	clear	0.24	no
soft	clear	0.40	yes
hard	blurry	0.25	no
soft	blurry	0.34	no
soft	slightly blurry	0.48	yes

Binary class labels: Yes(+)/No(-)

Multiclass Classification Applications

- Hand-written digit recognition

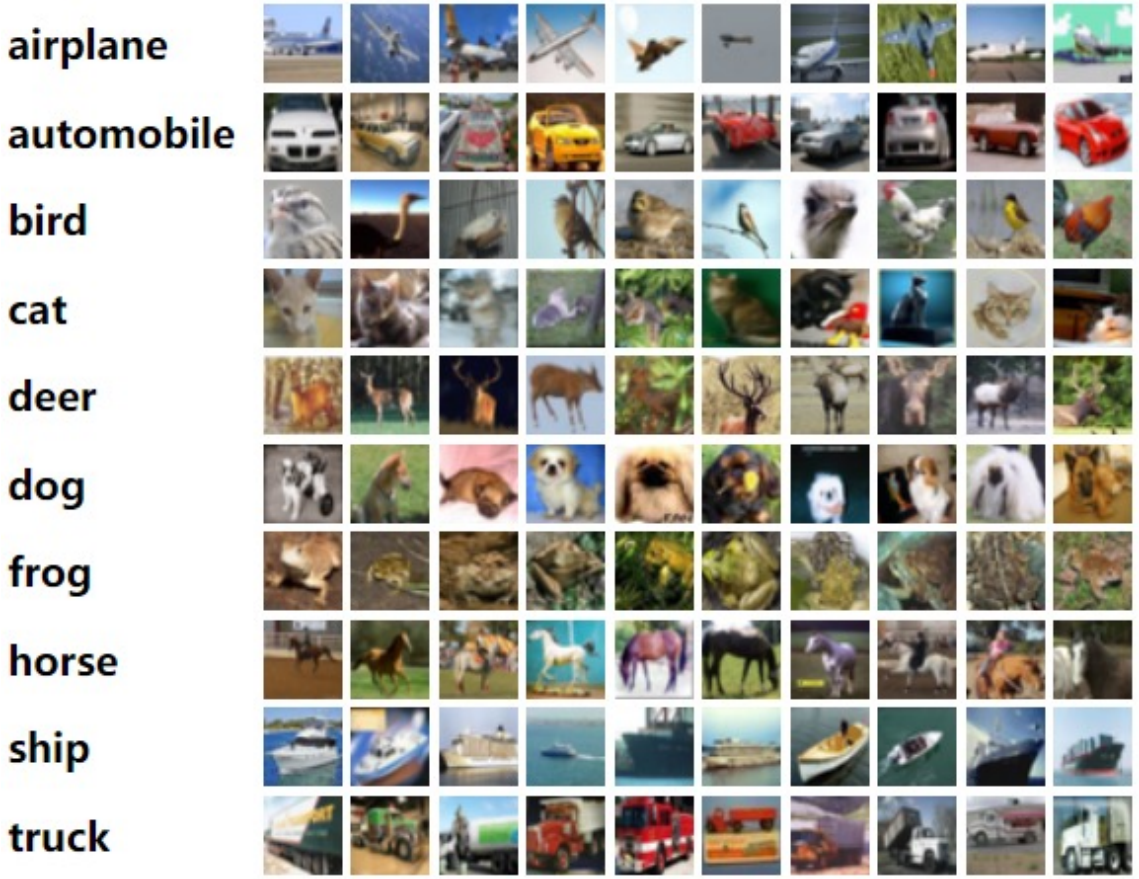
10 classes



A sample in test set

Multiclass Classification Applications

- Image classification



A sample in test set

Multiclass Classification

- Some algorithms can be directly applied to multiclass classification
  - Example: Decision Tree

- More often, we extend binary classification methods to multiclass classification.

- Decomposition: dividing the multiclass classification problem into several binary classification problems.

```
ID3(D,X) =
  Let T be a new tree
  If all instances in D have same class c
    Label(T) = c; Return T
  If X = ∅ or no attribute has positive information gain
    Label(T) = most common class in D; return T
  X ← attribute with highest information gain
  Label(T) = X
  For each value x of X
    D_x ← instances in D with X = x
    If D_x is empty
      Let T_x be a new tree
      Label(T_x) = most common class in D
    Else
      T_x = ID3(D_x, X − {X})
    Add a branch from T to T_x labeled by x
  Return T
```

Decomposition-based Multiclass Classification

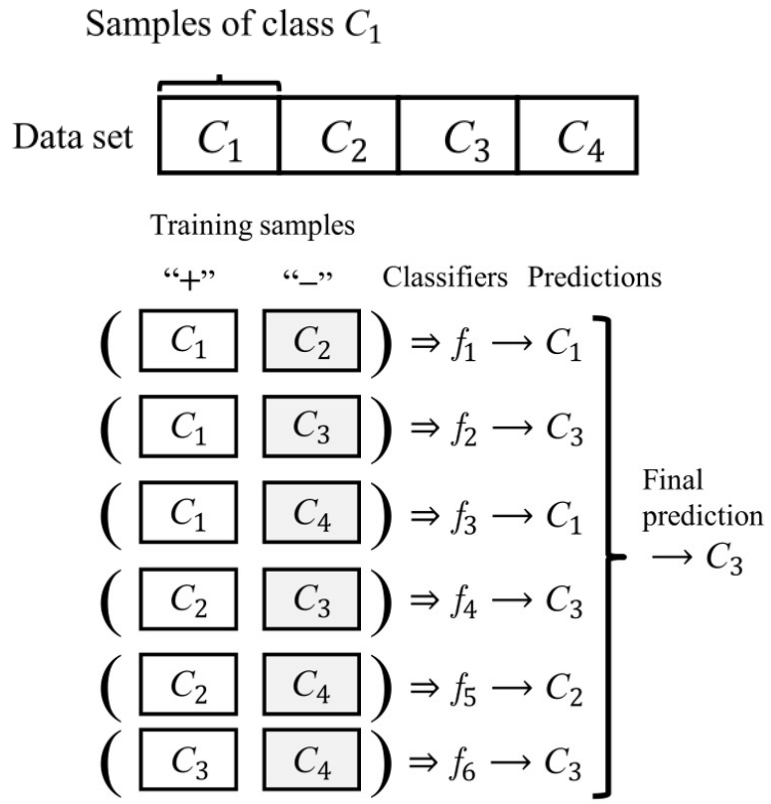
- Consider N classes: $C_1, C_2, \ldots, C_N$.
- Decomposition: dividing the multiclass classification problem into several binary classification problems.
- In testing phase, we ensemble the outputs collected from all binary classifiers into the final multiclass predictions.
- **Key question: How to divide and how to ensemble.**
- We focus on two classical dividing strategies:
  One vs One (OvO) & **One vs Rest (OvR)**

Decomposition-based Multiclass Classification

- Given a dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, where $y_i \in \{C_1, C_2, \ldots, C_N\}$
- One vs One (OvO) puts the N classes into pairs: in total $N(N-1)/2$ binary classification problems.

Example: OvO trains a classifier to distinguish class C_i and C_j , where it regards C_i as positive and C_j as negative.

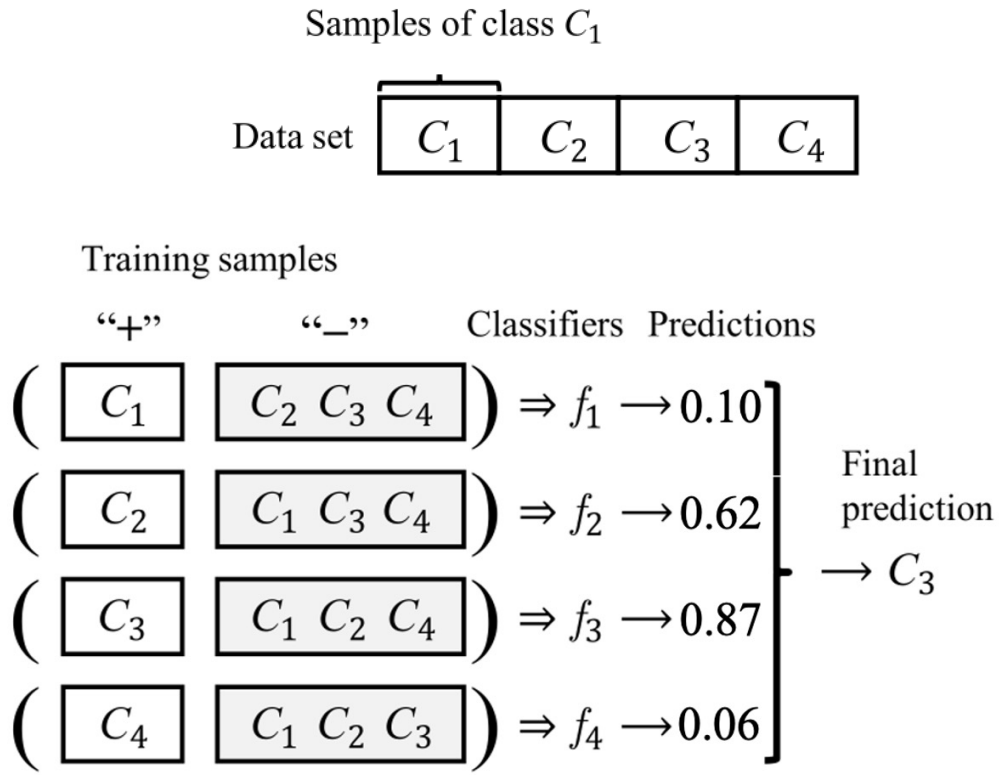
OvO strategy



Decomposition-based Multiclass Classification

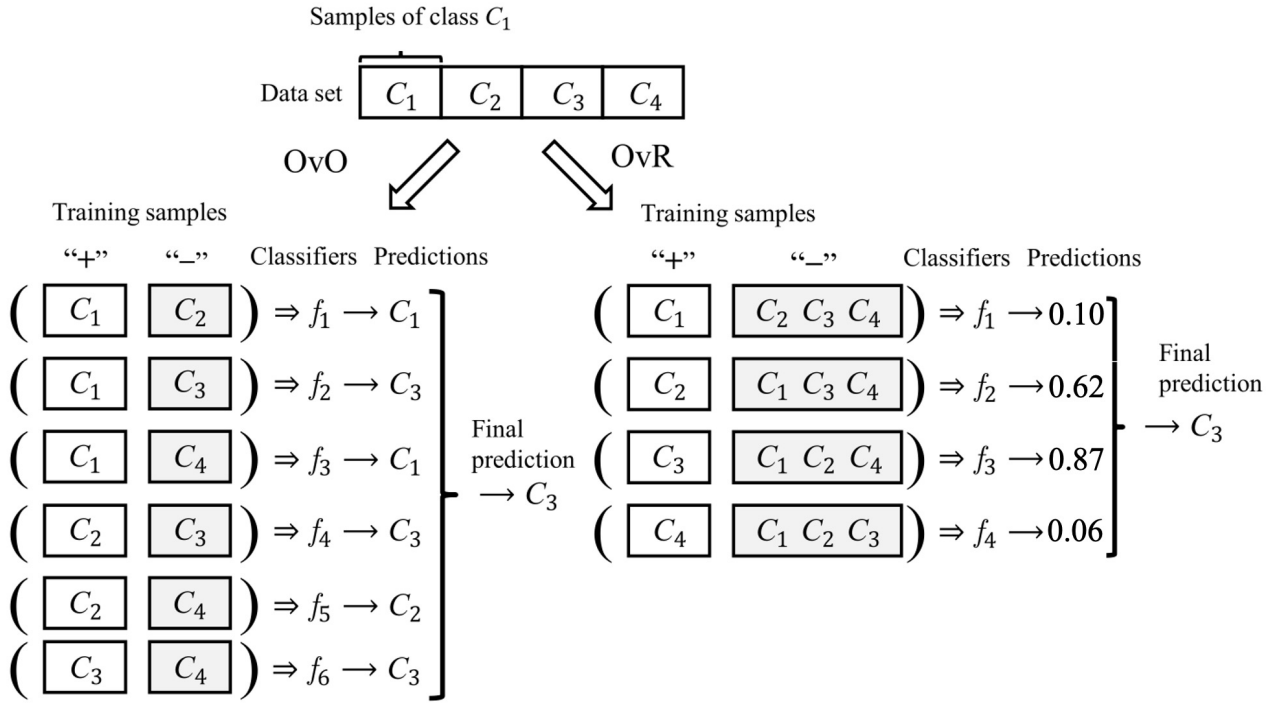
- Given a dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, where $y_i \in \{C_1, C_2, \ldots, C_N\}$
- OvR trains N classifiers by considering each class as positive in turn, and the rest classes are considered as negative.

OvR strategy



Decomposition-based Multiclass Classification

- Given a dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, where $y_i \in \{C_1, C_2, \ldots, C_N\}$



- OvR trains only N classifier but OvO trains $N(N-1)/2$ classifier.
- OvO is usually more expansive in storage and test time.
- In training, OvO uses data of two classes, but OvR uses all data. Usually, OvO can be trained faster.
- Performance are usually similar.

Evaluation Methods for Multiclass Classification

- Confusion matrix for multiclass classification

No	Actual	Predicted
1	Airplane	Airplane
2	Car	Boat
3	Car	Car
4	Car	Car
5	Car	Boat
6	Airplane	Boat
7	Boat	Boat
8	Car	Airplane
9	Airplane	Airplane
10	Car	Car

Three labels: Airplane, Car, Boat

		Predicted		
		Airplane	Boat	Car
Actual	Airplane	2	1	0
	Boat	0	1	0
	Car	1	2	3

Number of images of cars being predicted as boat

Confusion matrix: rows are actual labels (ground truth), columns are predictions

Example and figures from https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f

Evaluation Methods for Multiclass Classification

- We can compute the metrics for each class

		Predicted		
		Airplane	Boat	Car
Actual	Airplane	2	1	0
	Boat	0	1	0
	Car	1	2	3

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision	Recall	F1 Score
Airplane	2	1	1	0.67	0.67	2 * (0.67 * 0.67) / (0.67 + 0.67) = **0.67**
Boat	1	3	0	0.25	1.00	2*(0.25 * 1.00) / (0.25 + 1.00) = **0.40**
Car	3	0	3	1.00	0.50	2 * (1.00 * 0.50) / (1.00 + 0.50) = **0.67**

Number of images of other classes being predicted as boat

How to evaluate the overall performance for this multiclass classification model as a whole?

Taking averages!

- Macro average
- Weighted average
- Micro average

Example and figures from https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f

Evaluation Methods for Multiclass Classification

- Macro Average

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision	Recall	F1 Score
Airplane	2	1	1	0.67	0.67	2 * (0.67 * 0.67) / (0.67 + 0.67) = **0.67**
Boat	1	3	0	0.25	1.00	2*(0.25 * 1.00) / (0.25 + 1.00) = **0.40**
Car	3	0	3	1.00	0.50	2 * (1.00 * 0.50) / (1.00 + 0.50) = **0.67**

Label	Per-Class F1 Score	Macro-Averaged F1 Score
Airplane	0.67	$\frac{0.67 + 0.40 + 0.67}{3}$ = **0.58**
Boat	0.40	
Car	0.67	

Taking the average of per-class metrics (Precision, Recall, and F1).

$$\text{Macro-Precision} = \frac{\text{Precision}_1 + \text{Precision}_2 + \cdots + \text{Precision}_N}{N}$$

$$\text{Macro-Recall} = \frac{\text{Recall}_1 + \text{Recall}_2 + \cdots + \text{Recall}_N}{N}$$

$$\text{Macro-F1} = \frac{\text{F1}_1 + \text{F1}_2 + \cdots + \text{F1}_N}{N}$$

Example and figures from https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f

Evaluation Methods for Multiclass Classification

- Weighted Average

  Taking the weighted average of per-class metrics (Precision, Recall, and F1).

		Predicted		
		Airplane	Boat	Car
Actual	Airplane	2	1	0
	Boat	0	1	0
	Car	1	2	3

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision	Recall	F1 Score
Airplane	2	1	1	0.67	0.67	2 * (0.67 * 0.67) / (0.67 + 0.67) = **0.67**
Boat	1	3	0	0.25	1.00	2*(0.25 * 1.00) / (0.25 + 1.00) = **0.40**
Car	3	0	3	1.00	0.50	2 * (1.00 * 0.50) / (1.00 + 0.50) = **0.67**

Label	Per-Class F1 Score	Support	Support Proportion	Weighted Average F1 Score
Airplane	0.67	3	0.3	$(0.67 * 0.3) + (0.40 * 0.1) + (0.67 * 0.6)$ = **0.64**
Boat	0.40	1	0.1	
Car	0.67	6	0.6	
Total	-	10	1.0	

frequency of actual occurrence of each class

$$\text{Macro-Precision} = \frac{p_1\text{Precision}_1 + \cdots + p_N\text{Precision}_N}{N}$$

$$\text{Macro-Recall} = \frac{p_1\text{Recall}_1 + \cdots + p_N\text{Recall}_N}{N}$$

$$\text{Macro-F1} = \frac{p_1\text{F1}_1 + \cdots + p_N\text{F1}_N}{N}$$

Example and figures from https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f

Evaluation Methods for Multiclass Classification

- Micro Average

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)	Precision	Recall	F1 Score
Airplane	2	1	1	0.67	0.67	2 * (0.67 * 0.67) / (0.67 + 0.67) = **0.67**
Boat	1	3	0	0.25	1.00	2*(0.25 * 1.00) / (0.25 + 1.00) = **0.40**
Car	3	0	3	1.00	0.50	2 * (1.00 * 0.50) / (1.00 + 0.50) = **0.67**

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)	Micro-Averaged Values
Airplane	2	1	1	Precision = $\frac{6}{6+4}$ = **0.60**
Boat	1	3	0	Recall = $\frac{6}{6+4}$ = **0.60**
Car	3	0	3	F1 Score = $\frac{6}{6+\frac{1}{2}(4+4)}$ = **0.60**
TOTAL	**6**	**4**	**4**	

Add up the confusion matrix for each binary classification and compute the metrics

$$\text{Micro-Precision} = \frac{TP_1+\cdots+TP_N}{(TP_1+FP_1)+\cdots+(TP_N+FP_N)}$$

$$\text{Micro-Recall} = \frac{TP_1+\cdots+TP_N}{(TP_1+FN_1)+\cdots+(TP_N+FN_N)}$$

$$\text{Macro-F1} = \frac{2 \times \text{Micro}-\text{Precision} \times \text{Micro}-\text{Recall}}{\text{Micro}-\text{Precision}+ \text{Micro}-\text{Recall}}$$

Example and figures from https://towardsdatascience.com/micro-macro-weighted-averages-of-f1-score-clearly-explained-b603420b292f

Evaluation Methods for Multiclass Classification

- Which one to use?
  - Macro averaged: all classes are equally important.
  - Weighted averaged: assigns greater weighting to classes with more samples in the dataset.
  - Micro averaged: overall performance regardless of the class.

Feature Engineering

Feature Engineering

- Recall from the summary of the linear models:

$$\underbrace{\mathbf{w} \cdot \boxed{\phi(x)}}_{\text{score}}$$

Two components: $\phi(x)$ and $\mathbf{w}$

	Regression	Classification
Prediction $f_{\mathbf{w}}(x)$	score	sign(score)
Relate to target y	residual (score $- y$)	margin (score y)
Loss functions	squared absolute deviation	zero-one hinge logistic
Algorithm	gradient descent	gradient descent

So far, we just used $\phi(x) = x$ and focused on learning of $\mathbf{w}$.

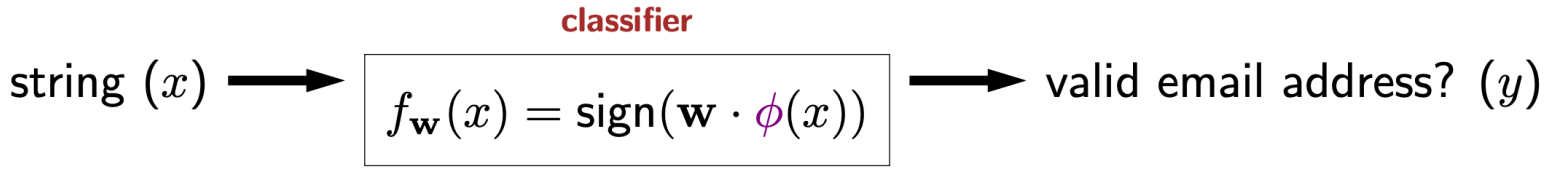
Another important part of ML pipeline:

Feature extraction (feature engineering) specifies $\phi(x)$ based on some domain knowledge.

Often the bottleneck for real applications

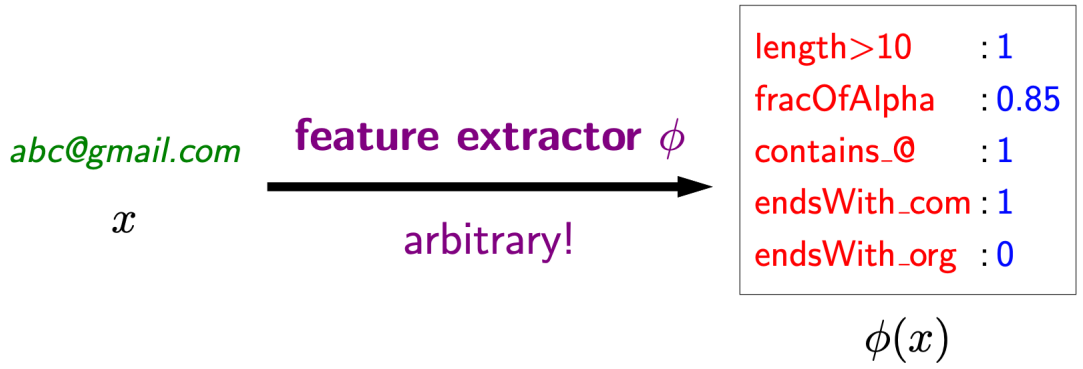
Feature Engineering

- An example task: predict whether a string is an email address



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

Feature extractor ϕ produces (feature name, feature value) pairs



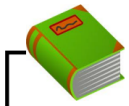
What features to include?
Need some organizational principles.

Feature engineering is sort of an "art".

Example and figures from https://stanford-cs221.github.io/spring2022-extra/modules/machine-learning/nonlinear-and-neural.pdf

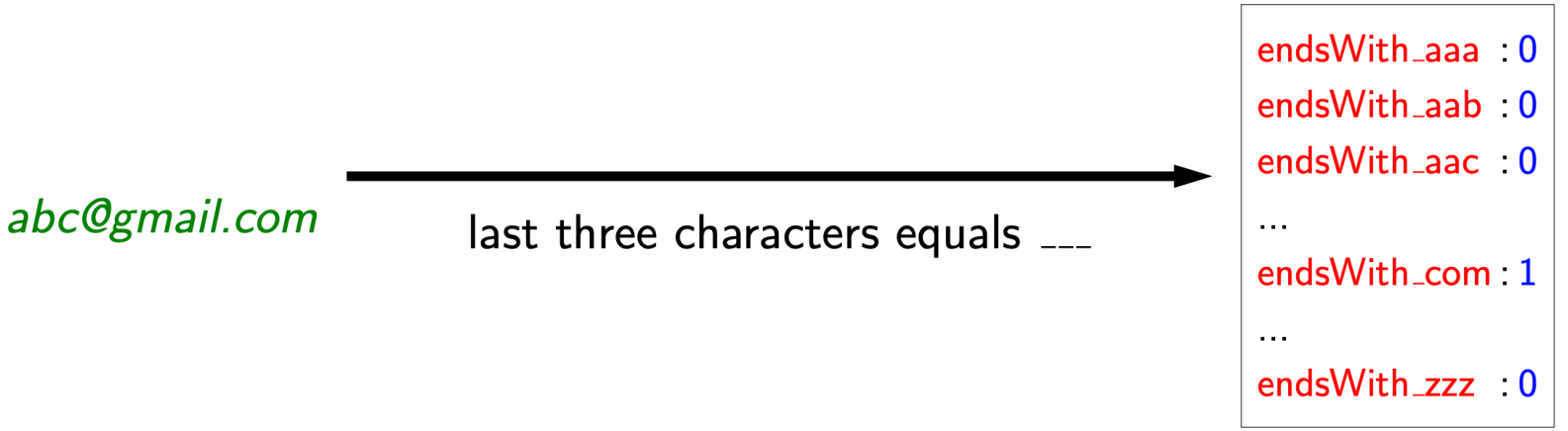
Feature Engineering

- Feature templates



Definition: feature template

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



Define types of pattern to look for, not particular patterns

Example and figures from https://stanford-cs221.github.io/spring2022-extra/modules/machine-learning/nonlinear-and-neural.pdf

Feature Engineering

- Feature templates Example

Latitude: 37.4068176

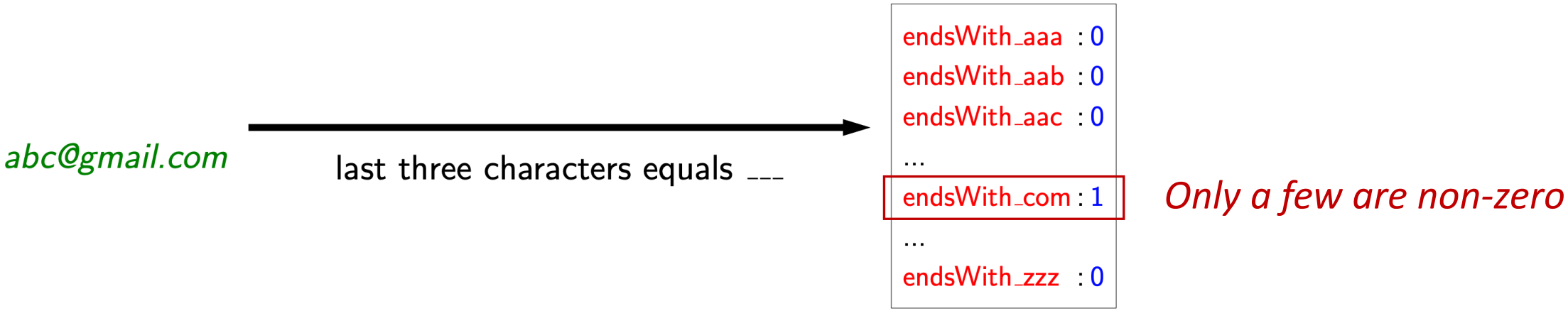
Longitude: -122.1715122

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

Example and figures from https://stanford-cs221.github.io/spring2022-extra/modules/machine-learning/nonlinear-and-neural.pdf

Sparse Features

- Dictionary is more efficient to represent sparse features (few non-zeros)



A more compact way to represent such sparse features:

(dictionary) {"endsWith_com": 1}

Instead of an array: [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0]

Example and figures from https://stanford-cs221.github.io/spring2022-extra/modules/machine-learning/nonlinear-and-neural.pdf

Hypothesis Class

- Which feature extractor ϕ to use?
  - Recall the hypothesis class in Lecture 5:
  - A hypothesis class is the set of possible predictors with a fixed $\phi(x)$ and varying w:

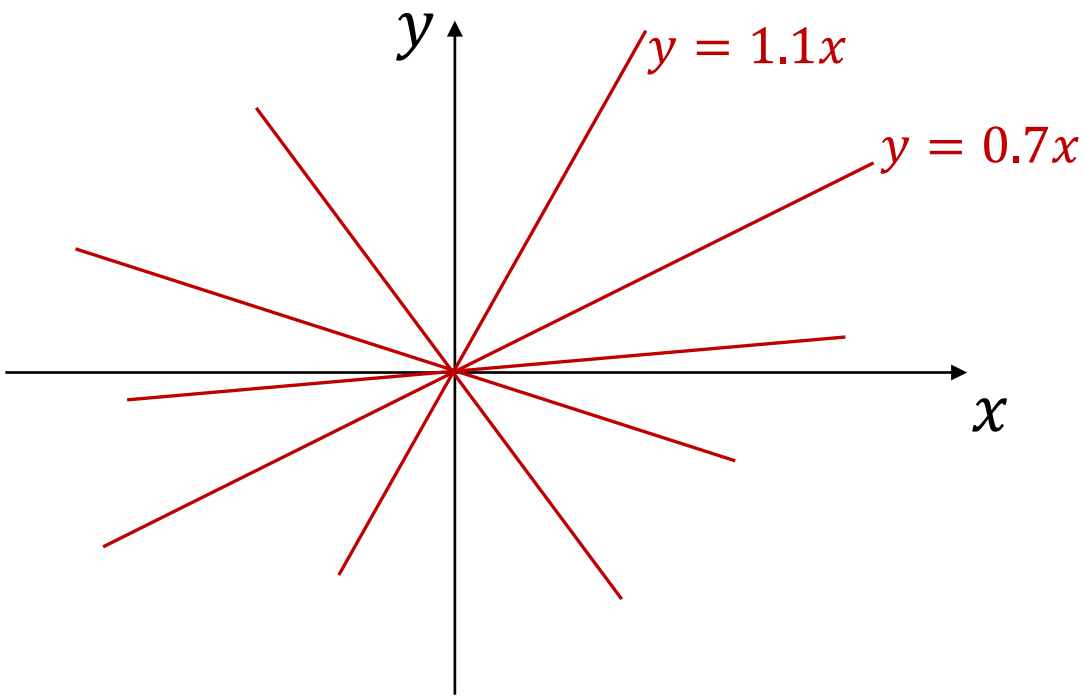
$$\mathcal{H} = \{f_{\mathbf{w}} : \mathbf{w} \in \mathbb{R}^d\}$$

where $f_{\mathbf{w}}(x) = \mathbf{w}^{\mathrm{T}}\phi(x)$ or $\mathrm{sign}(\mathbf{w}^{\mathrm{T}}\phi(x))$ is the predictor.

Hypothesis Class

- Example of hypothesis class:
  - Linear regression: $x \in \mathbb{R}$ and $y \in \mathbb{R}$
  - Linear function: $\phi(x) = x$, the hypothesis class is $\mathcal{H}_1 = \{w_1 x : w_1 \in \mathbb{R}\}$

All lines go through the origin



Constructing a feature vector:

Considering the hypothesis class defined by this feature map.

Hypothesis Class

- Example of hypothesis class:
  - Linear regression: $x \in \mathbb{R}$ and $y \in \mathbb{R}$
  - Linear function: $\phi(x) = x$, the hypothesis class is $\mathcal{H}_1 = \{w_1 x : w_1 \in \mathbb{R}\}$

All lines go through the origin

  - Quadratic function: $\phi(x) = [x, x^2]$, the hypothesis class is

$$\mathcal{H}_2 = \{w_1 x + w_2 x^2 : w_1 \in \mathbb{R}, w_2 \in \mathbb{R}\}$$

All quadratic functions that go through the origin
Also includes all linear functions

- $\mathcal{H}_2$ is a larger set than $\mathcal{H}_1$: more expressive (can represent more things).
- The feature map ϕ defnes the hypothesis class.

Hypothesis Class

- Prediction is driven by score: $\mathbf{w}^T \phi(x)$
  - Is score linear in $\boldsymbol{w}$? *Yes!*
  - Is score linear in $\phi(x)$? *Yes!*
  - Is score linear in x? *Not necessarily!* E.g., $\phi(x) = [x, x^2]$

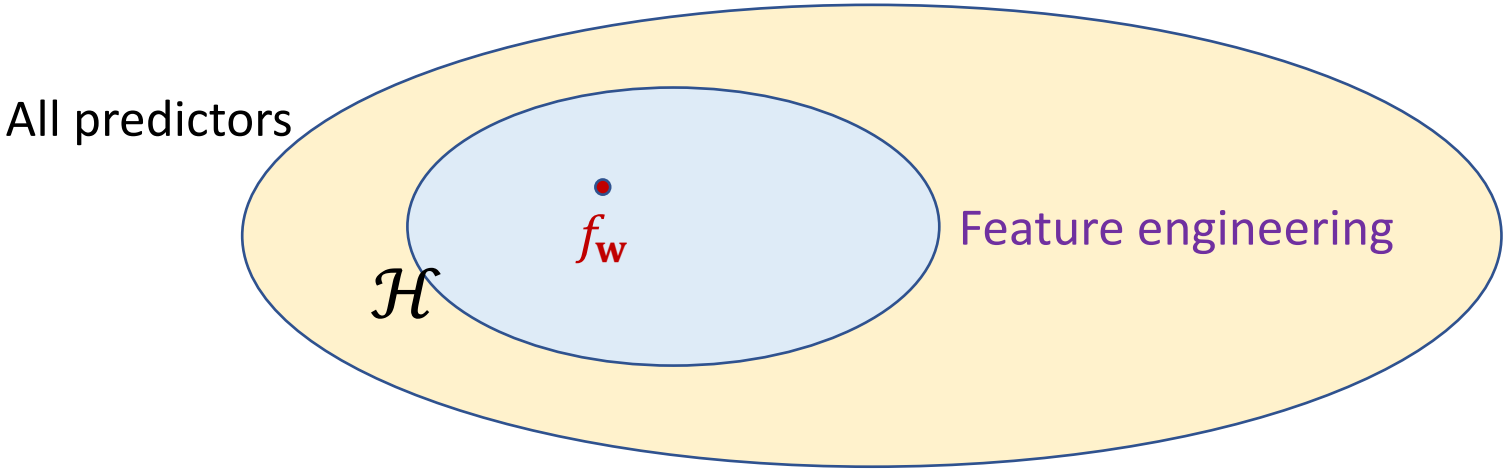
- Predictor $f_{\mathbf{w}}(x)$ can be expressive non-linear functions and decision boundaries of x.

  online demos: https://playground.tensorflow.org/
  https://www.youtube.com/watch?v=3liCbRZPrZA

- Score $\mathbf{w}^T \phi(x)$ is linear function of $\mathbf{w}$, which allows efficient learning.

Feature Engineering + Learning

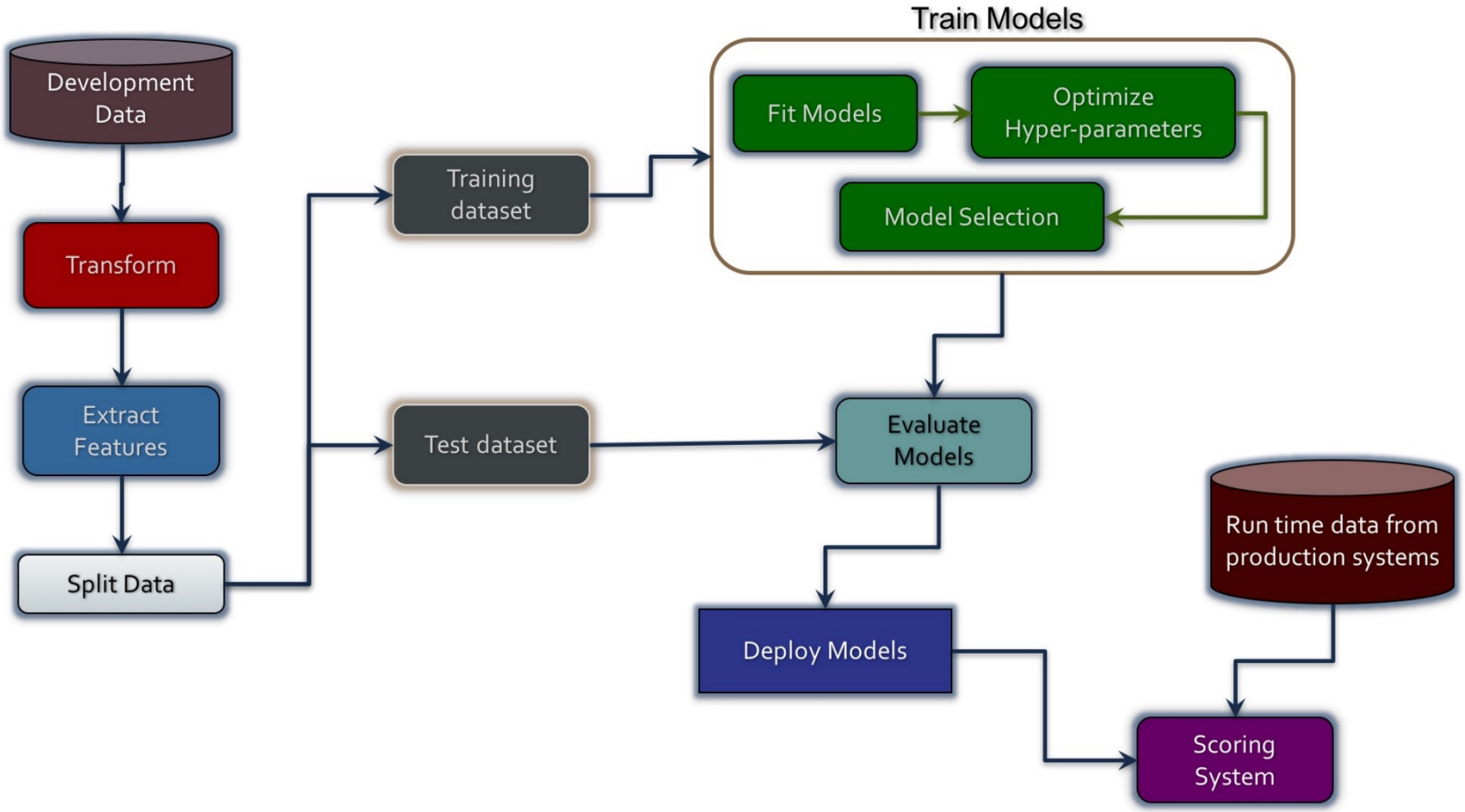
$$\mathcal{H} = \{f_{\mathbf{w}}(x) = \text{sign}(\mathbf{w}^{\mathrm{T}}\phi(x)) : \mathbf{w} \in \mathbb{R}^d\}$$



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

We want $\mathcal{H}$ to contain good predictors but not be too big

Pipeline of ML in practice

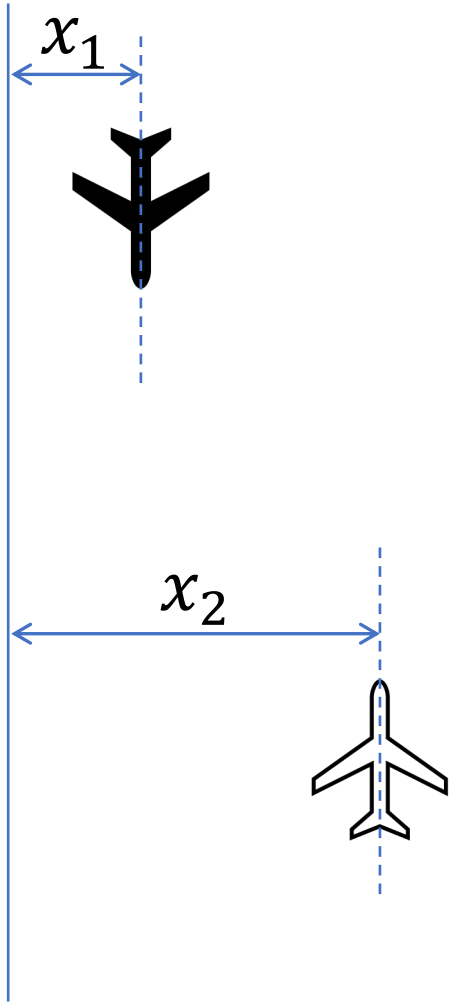


Deep Learning Basics

Motivations I: how to build a system to detect cats?

- What would be a good feature extractor $\phi(x)$ for this task?
- Can we automatically learn a feature extractor?

Motivations II: aircraft collision prediction



- Input: position of two aircrafts $x = [x_1, x_2]$
- Output: saft ($y = +1$) or collision ($y = -1$)

Suppose we know the true function:

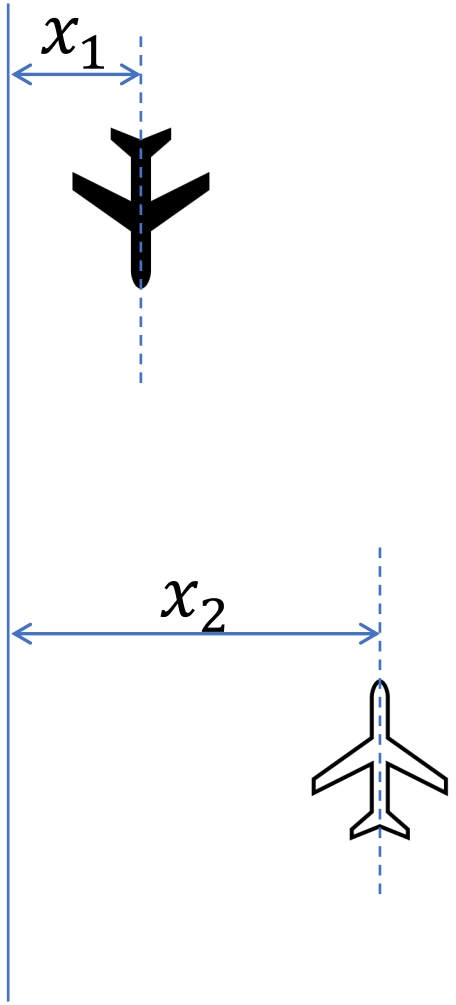
Safe if they are far enough: $y = \text{sign}(|x_1 - x_2| - 1)$

Examples:

$$x = [1, 3] \quad y = +1$$
$$x = [3, 1] \quad y = +1$$
$$x = [1, 1.5] \quad y = -1$$

Motivation II section adapted from https://stanford-cs221.github.io/spring2022-extra/modules/machine-learning/nonlinear-and-neural.pdf

Motivations II: aircraft collision prediction



Decomposing the problem into subproblems:

- Test if aircraft 2 is far right of aircraft 1:

$$h_1 = 1[x_2 - x_1 \geq 1]$$

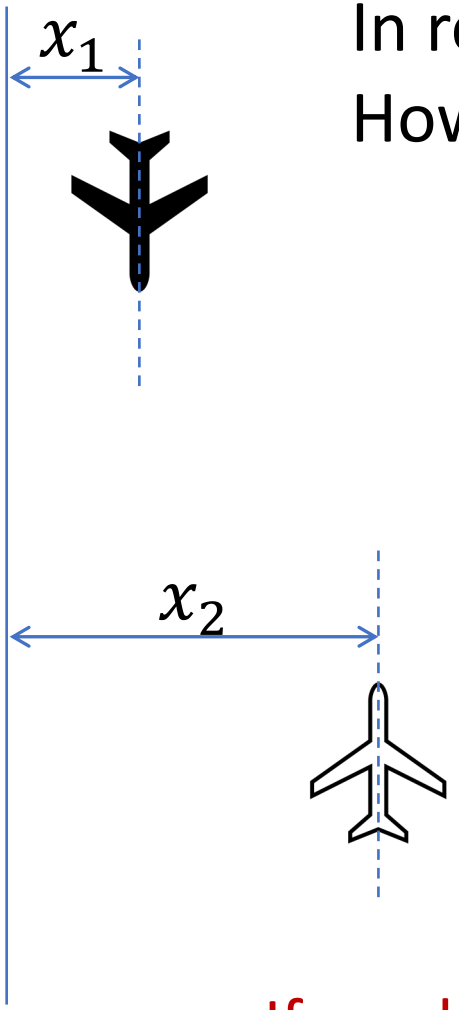
- Test if aircraft 1 is far right of aircraft 2:

$$h_2 = 1[x_1 - x_2 \geq 1]$$

- Safe if at least one is true.

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

Motivations II: aircraft collision prediction



In reality, we do not know the true functions.
How do we learn them from data?

- Define $\phi(x) = [1, x_1, x_2]$
- Test if aircraft 2 is far right of aircraft 1:

  ~~$h_1 = 1[x_2 - x_1 \geq 1]$~~ → $h_1 = 1[\mathbf{v}_1^T \phi(x) \geq 0]$

  If $\mathbf{v_1} = [-1, +1, -1]$, they are equivalent.
- Test if aircraft 1 is far right of aircraft 2:

  ~~$h_2 = 1[x_1 - x_2 \geq 1]$~~ → $h_2 = 1[\mathbf{v}_2^T \phi(x) \geq 0]$

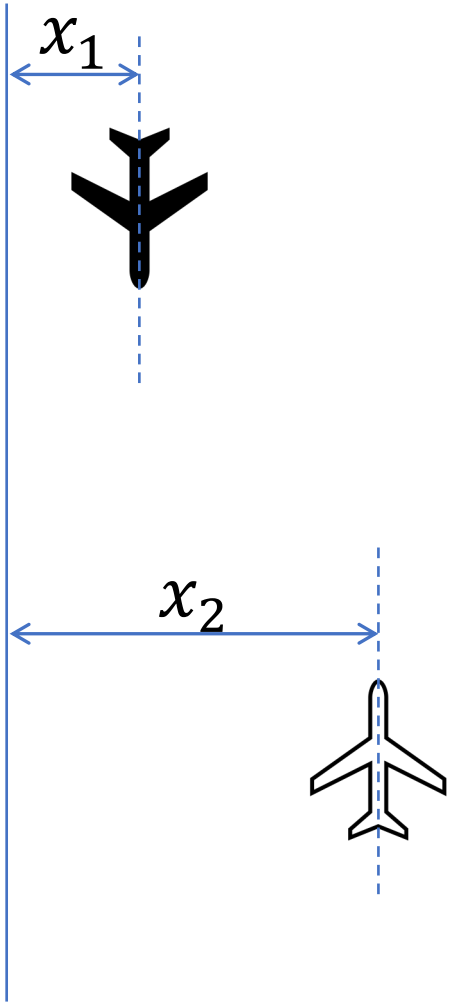
  If $\mathbf{v_2} = [-1, -1, +1]$, they are equivalent.
- Safe if at least one is true.

  ~~$y = \text{sign}(h_1 + h_2)$~~ → $f_{\mathbf{V},\mathbf{w}}(x) = \text{sign}(w_1 h_1 + w_2 h_2)$

  If $\mathbf{w} = [1, 1]$, they are equivalent.

If we do not know the true functions, learn both $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2]$ and $\mathbf{w}$.

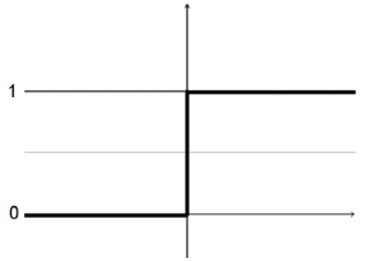
Motivations II: aircraft collision prediction



Learning via gradient descent

- $h_1 = 1[\mathbf{v}_1^T \phi(x) \geq 0]$ gradient with respect to $\mathbf{v}_1$ is 0

Solution: replace the indicator function with an approximation with non-zero gradient.



Indicator function (unit-step)

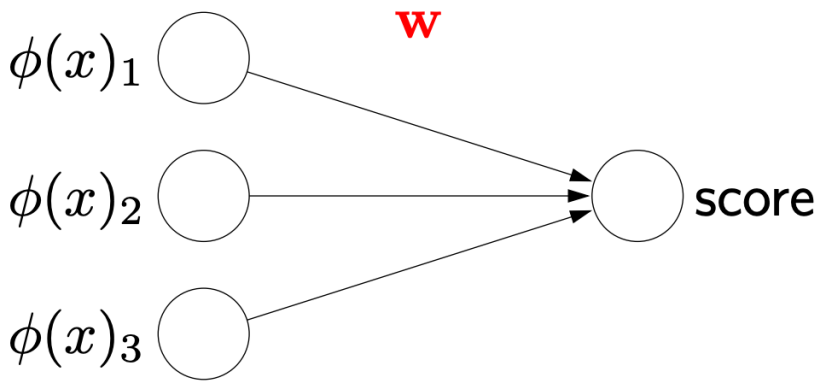
Sigmoid function

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

- $h_1 = \sigma(\mathbf{v}_1^T \phi(x))$

From Linear Functions to Neural Networks

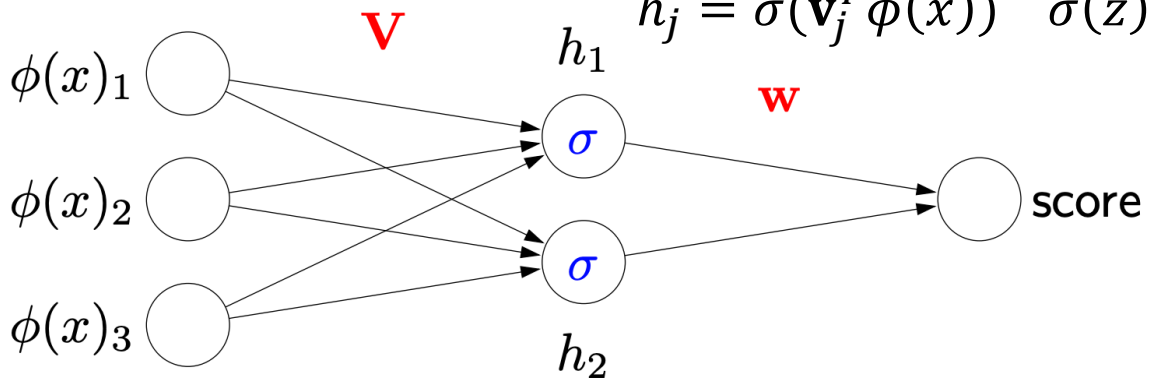
Linear functions:



$$\text{score} = \mathbf{w}_1^T \phi(x)$$

Neural network:
(with 1 hidden layer)

$$h_j = \sigma(\mathbf{v}_j^T \phi(x)) \quad \sigma(z) = (1 + e^{-z})^{-1}$$



$$\text{score} = \mathbf{w}^T \mathbf{h}$$

Intuition: neural networks try to break down the problem into a set of subproblems

Neural Networks: Feature (Representation) Learning

Interpret $h(x)$ as a learned feature representation

Key idea: feature learning

Before: apply linear predictor on manually specify features

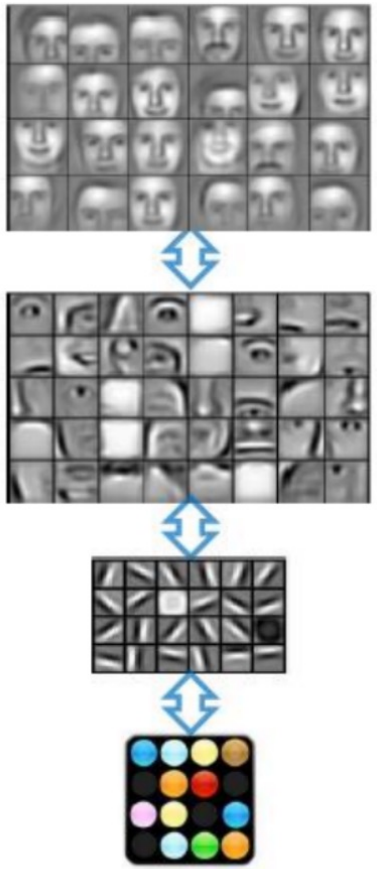
$$\phi(x)$$

Now: apply linear predictor on automatically learned features

$$h(x) = [h_1(x), \ldots, h_k(x)]$$

Neural Networks: Feature (Representation) Learning

multiple levels of abstractions



3rd layer
"Objects"

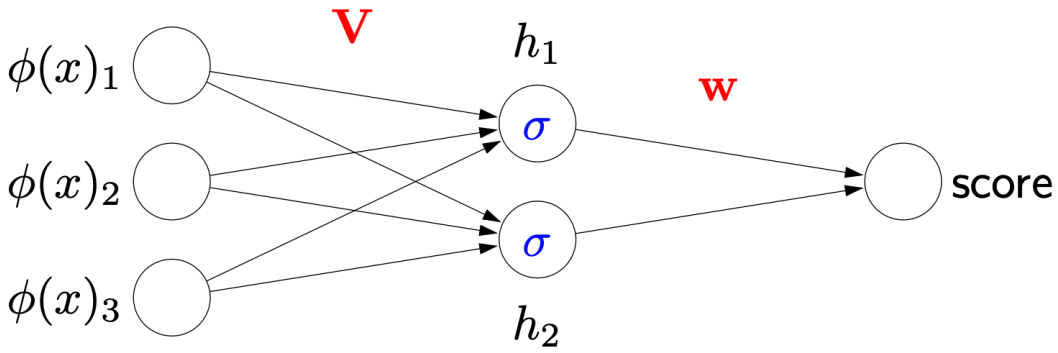
2nd layer
"Object parts"

1st layer
"Edges"

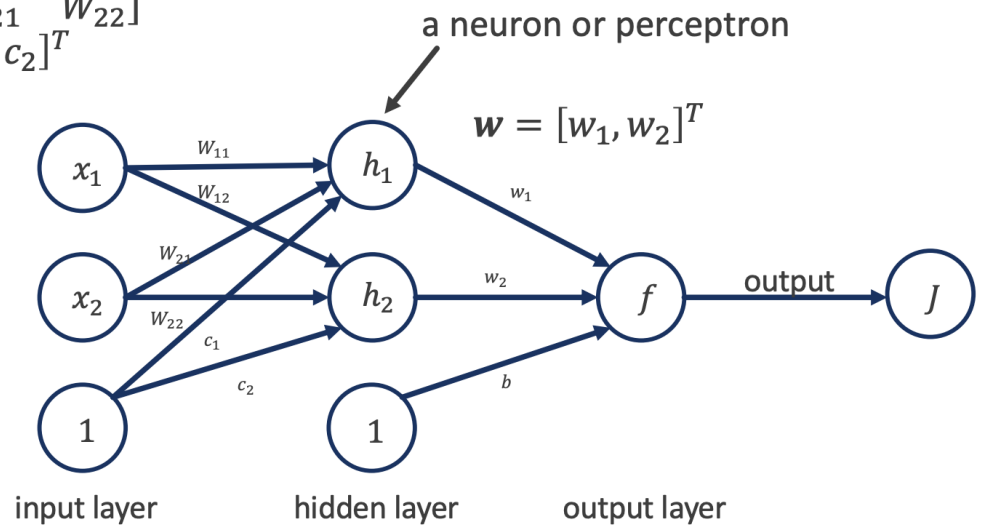
Pixels

Feedforward Neural Networks

- Feedforward: information only travels forward in the network (no loops)
- Neuron: nodes through which data and computations flow.
- Layer: a collection of neurons. (hidden size: number of neurons in hidden layers)
- Activation function: introduces nonlinearity.



$$\boldsymbol{W} = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix}$$
$$\boldsymbol{c} = [c_1, c_2]^T$$

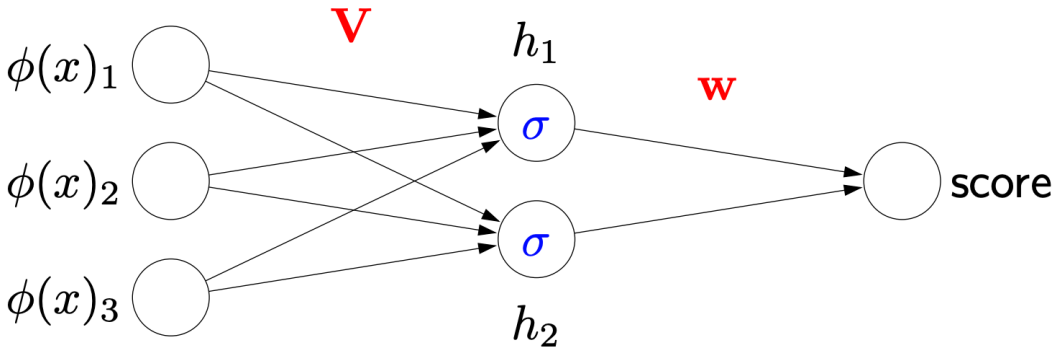


A multi-layer perceptron (MLP)

Feedforward Neural Networks

- What if we remove the activation function?

$\sigma(z) = z$?



$f(\mathbf{x}) = ?$

$$
\begin{aligned}
f(\mathbf{x}) &= w^T\left(\mathbf{V}^T\phi(x)\right) \\
&= w_1 h_1 + w_2 h_2 + b \\
&= w_1(v_{11}\phi(x)_1 + v_{21}\phi(x)_2 + v_{31}\phi(x)_3) \\
&\quad + w_2(v_{12}\phi(x)_1 + v_{22}\phi(x)_2 + v_{32}\phi(x)_3) + b \\
&= (\dots)\phi(x)_1 + (\dots)\phi(x)_2 + (\dots)\phi(x)_3 + b
\end{aligned}
$$

Equivalent to a linear model!

online demos: https://playground.tensorflow.org/

Role of activation functions: introduce nonlinearity (nonlinear in input $\phi(x)$)

for simplicity, we will simply use notation x instead of $\phi(x)$ from now on.

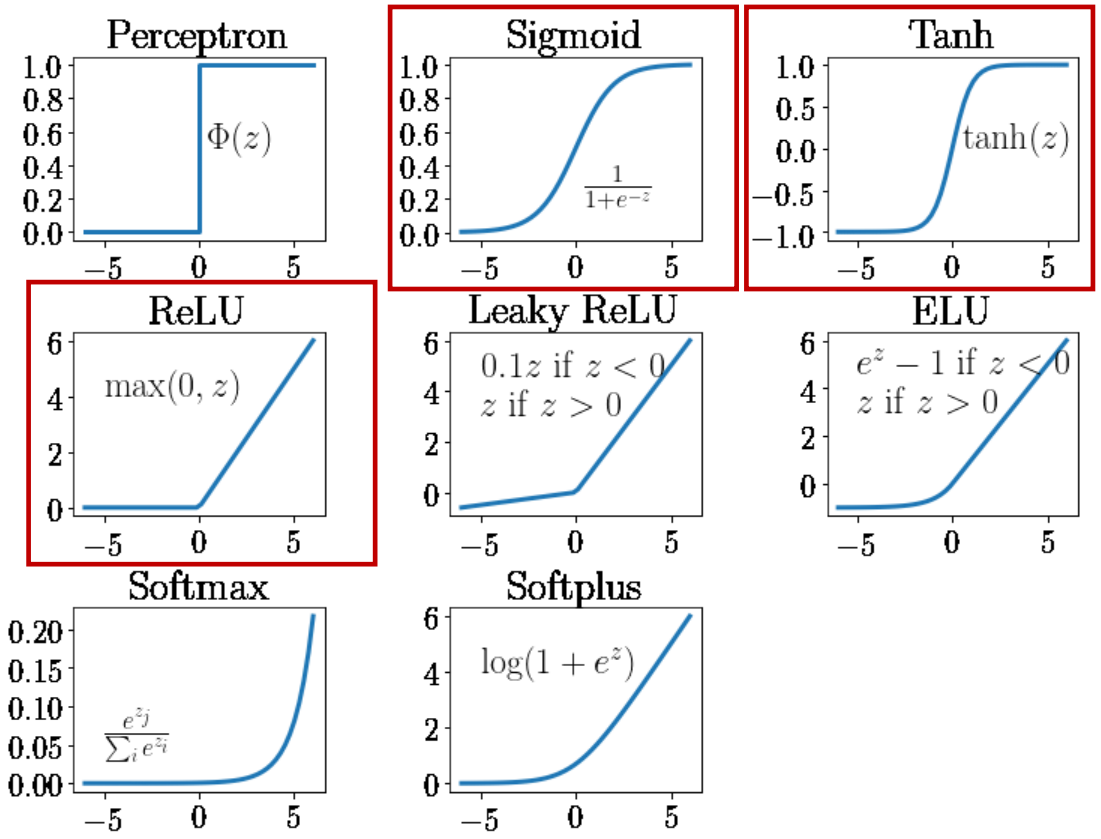
Feedforward Neural Networks

- Commonly used activation functions
  - Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
  - Rectified Linear Unit (ReLU):

$$\sigma(z) = \max\{0, z\}$$

ReLU is recommended for most FNNs

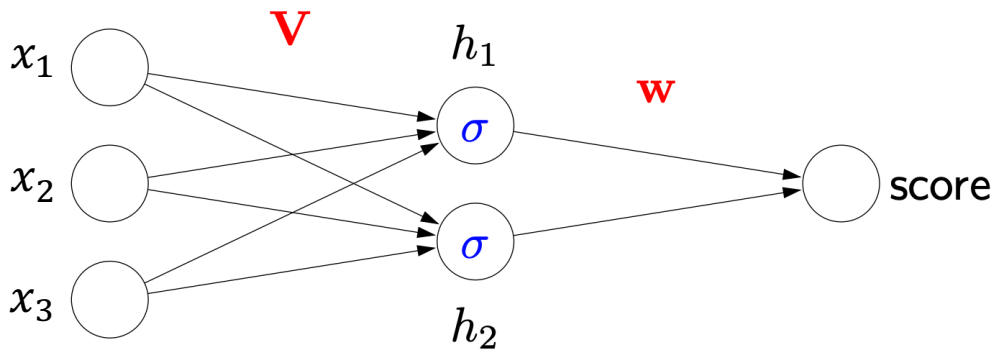
  - Tanh: $\sigma(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$



Feedforward Neural Networks

- What if we we use ReLU activation function?

$$\sigma(z) = \max\{z, 0\}$$



$$\begin{aligned} f(\mathbf{x}) &= w^T(\mathbf{V}^T\mathbf{x}) \\ &= w_1h_1 + w_2h_2 + b \\ &= w_1(v_{11}x_1 + v_{12}x_2 + v_{13}x_3) \\ &\quad + w_2(v_{21}x_1 + v_{22}x_2 + v_{23}x_3) + b \\ &= (\dots)x_1 + (\dots)x_2 + (\dots)x_3 + b \end{aligned}$$

No activation function: Linear in $\mathbf{x}$

$$f(\mathbf{x}) = w^T \max\{0, \mathbf{V}^T\mathbf{x}\}$$

No longer linear in $\mathbf{x}$

Role of activation functions: introduce nonlinearity (nonlinear in input $\mathbf{x}$)

Training Neural Network Models

- Task: regression
- What loss function should we use?

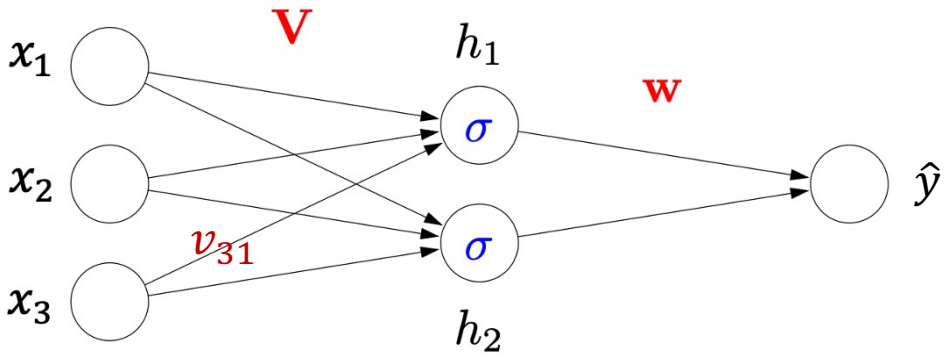
$$\ell = \frac{1}{N}\sum_{n=1}^{N}(y-\hat{y})^2$$

- What are the parameters to be learned?

$$\mathbf{V}, \mathbf{w}$$

- How can we learn them?

Gradient descent



What is the gradient with respect to v_{31}?

$$\nabla_{v_{31}} = \frac{\partial \ell}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h_1}\frac{\partial h_1}{\partial v_{31}} \quad \textit{(chain rule)}$$

$$= \frac{1}{N}\sum_{n=1}^{N} -2(y-\hat{y})\, h_1 \frac{\partial h_1}{\partial v_{31}}$$

$$\frac{\partial h_1}{\partial v_{31}} = \frac{\partial \sigma(\mathbf{v}_1^T\mathbf{x})}{\partial v_{31}} = \frac{\partial \sigma(\mathbf{v}_1^T\mathbf{x})}{\partial(\mathbf{v}_1^T\mathbf{x})}\frac{\partial(\mathbf{v}_1^T\mathbf{x})}{\partial w_{31}} = 1[\mathbf{v}_1^T\mathbf{x} > 0]x_3$$

Training Neural Network Models

- Task: regression
- What loss function should we use?

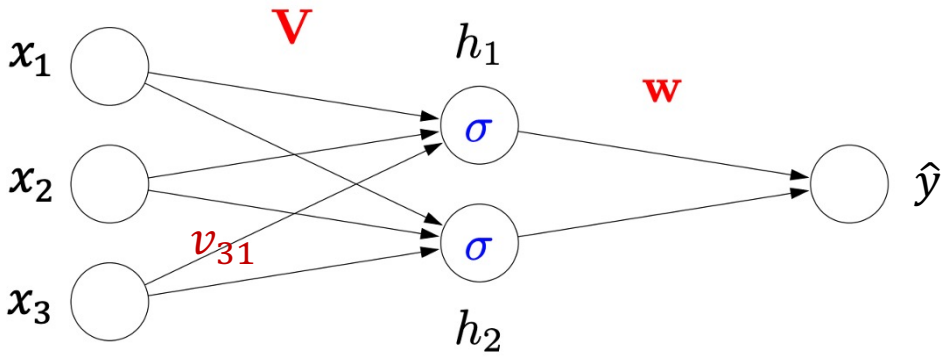
$$\ell = \frac{1}{N}\sum_{n=1}^{N}(y - \hat{y})^2$$

- What are the parameters to be learned?

$$\mathbf{V}, \mathbf{w}$$

- How can we learn them?

Gradient descent



What is the gradient with respect to v_{31}?

What is the gradient with respect to v_{32}?

What is the gradient with respect to v_{21}?

……

Would you like to derive them all?

What if we have 30 layers?

Computational Graphs and Backpropagation

- Mathematically derive gradients w.r.t. all parameters

  Doable by just grind through the chain rules.

- Automate this process?

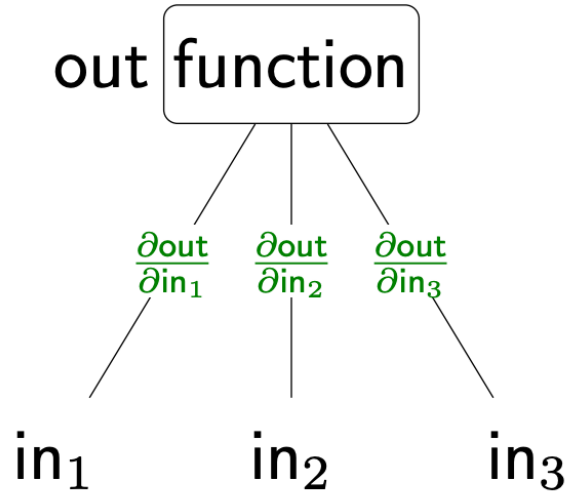
  visualize the computation using a computation graph

- Advantages:
  - Avoid long equations
  - Reveal structure of computations (modularity, efficiency, dependencies)
    — TensorFlow/PyTorch are built on this

This section is partially adapted from https://stanford-cs221.github.io/spring2022-extra/modules/machine-learning/nonlinear-and-neural.pdf

Computational Graphs and Backpropagation

- Think of functions as boxes



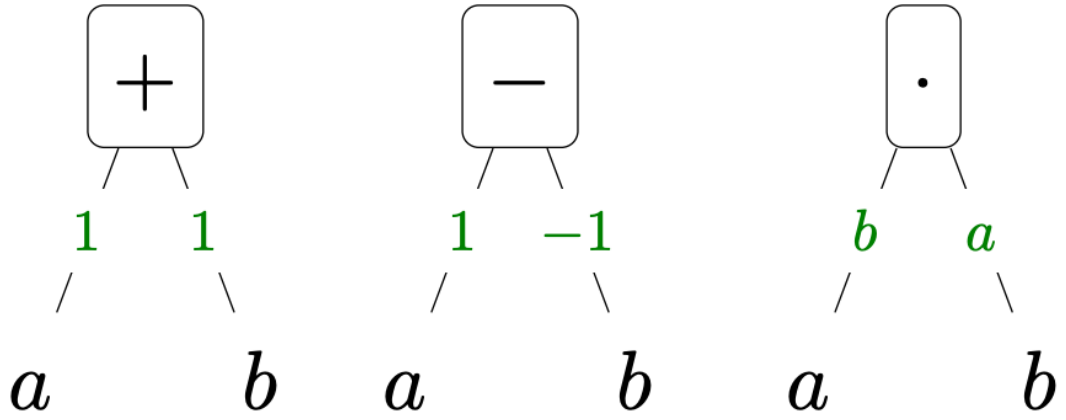
- Partial derivative (gradients):
  how much does the output change if an input slightly changes?

- Example:

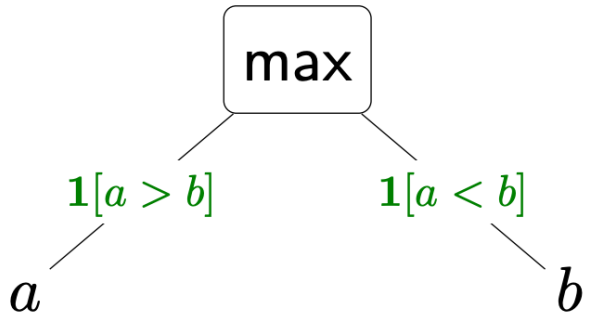
	$2\text{in}_1 + \text{in}_2\text{in}_3 = \text{out}$
(change in_1 slightly)	$2(\text{in}_1 + \epsilon) + \text{in}_2\text{in}_3 = \text{out} + 2\epsilon$
(change in_2 slightly)	$2\text{in}_1 + (\text{in}_2 + \epsilon)\text{in}_3 = \text{out} + \text{in}_3\epsilon$

Computational Graphs and Backpropagation

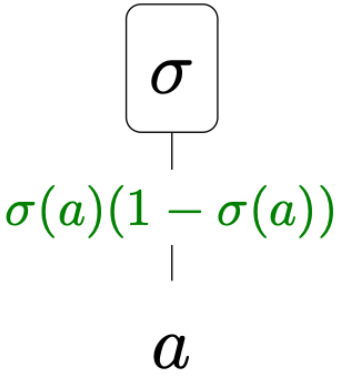
- Building blocks (Five examples)



$a + b$ $a - b$ $a \cdot b$



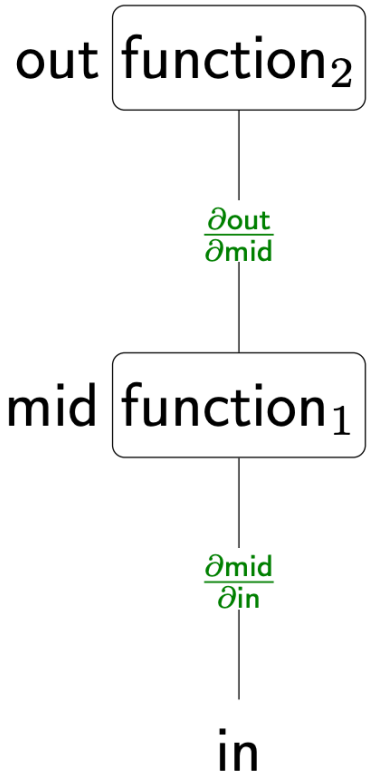
$\max\{a, b\}$



$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

Computational Graphs and Backpropagation

- Composing functions



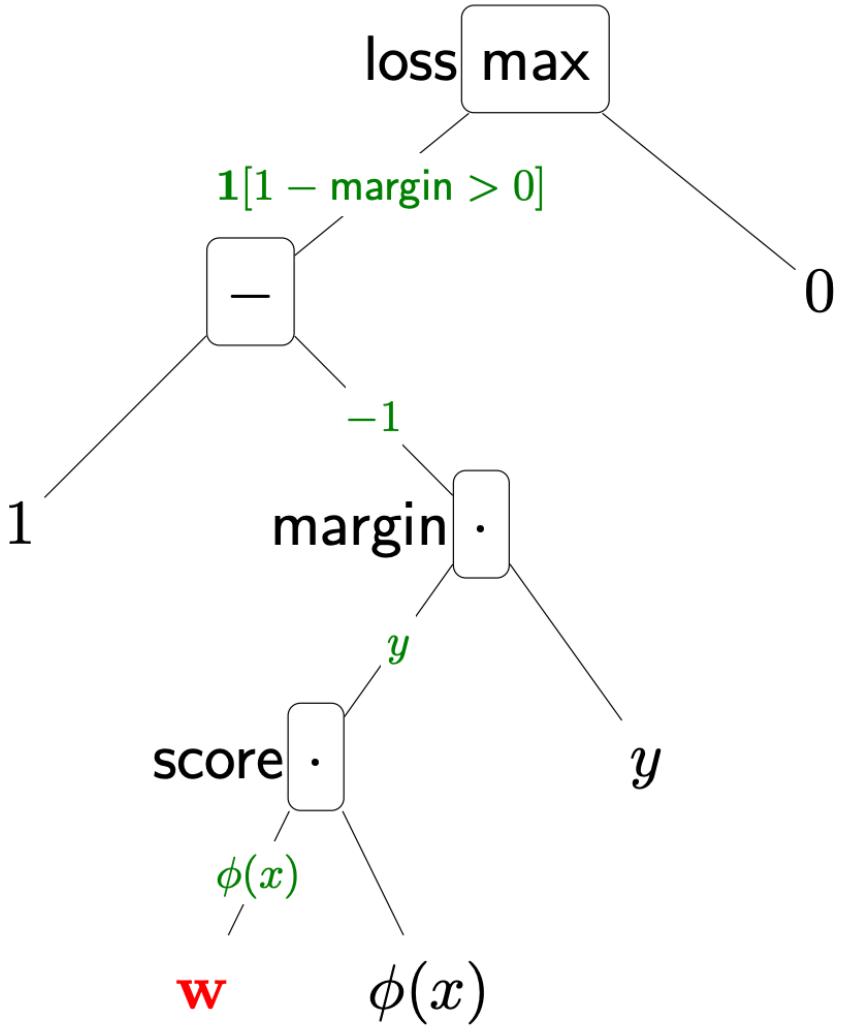
Chain rule: $\frac{\partial \text{out}}{\partial \text{in}} = \frac{\partial \text{out}}{\partial \text{mid}} \frac{\partial \text{mid}}{\partial \text{in}}$

Computational Graphs and Backpropagation

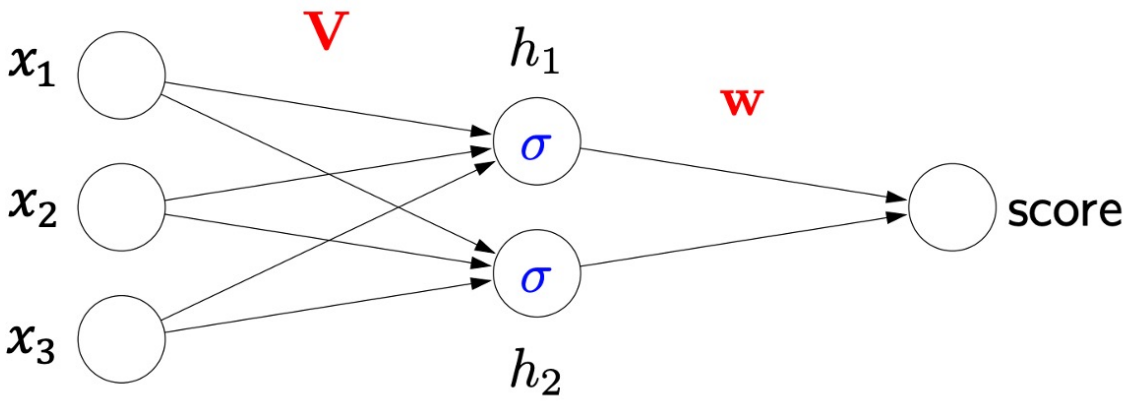
- Example of constructing a computational graph
  - Hinge loss: $\text{Loss}(x, y, \mathbf{w}) = \max\{1 - \mathbf{w} \cdot \phi(x)y, 0\}$
  - Compute:

$$\frac{\partial \text{Loss}(x, y, \mathbf{w})}{\partial \mathbf{w}}$$

  - Gradient:
    multiplying the edges: $-\mathbf{1}[\text{margin} < 1]\phi(x)y$



Computational Graphs and Backpropagation

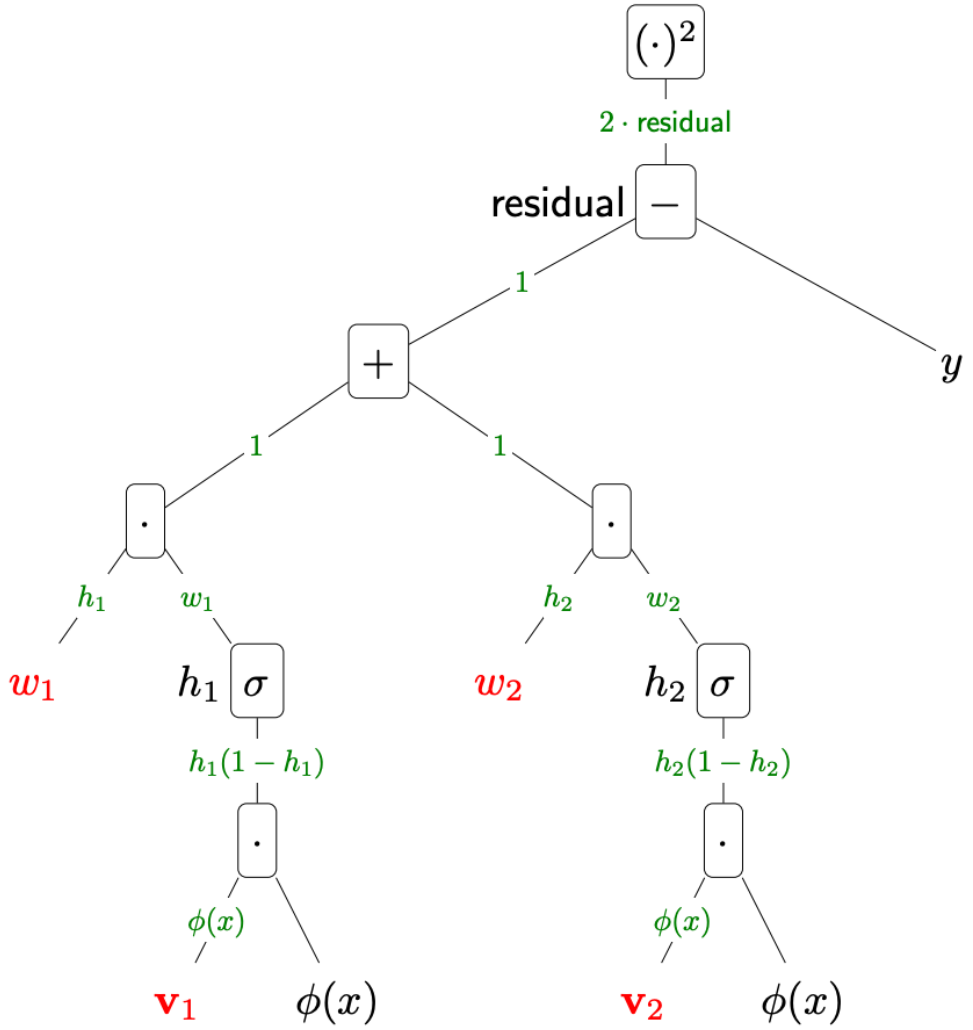


- Exercise:

$$\text{Loss}(x, y, \mathbf{w}) = \left(\sum_{j=1}^{k} w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$

Construct a computational graph for this neural network

Computational Graphs and Backpropagation

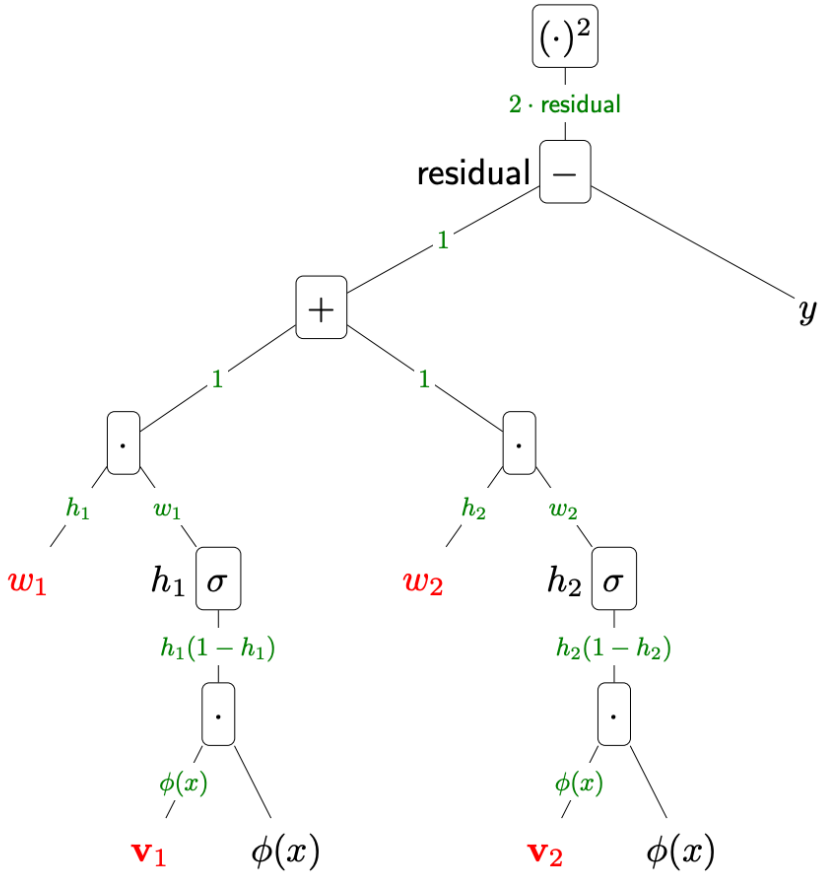


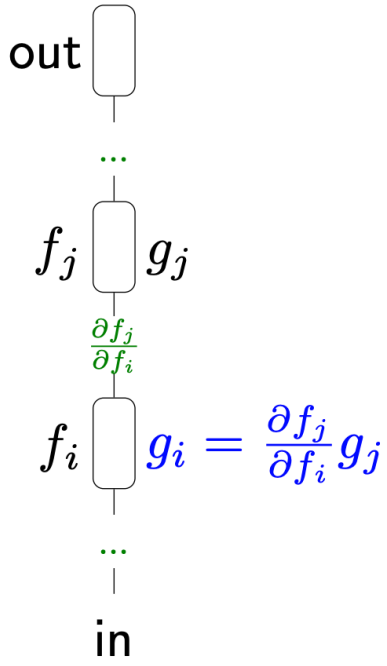
$$\text{Loss}(x, y, \mathbf{w}) = \left(\sum_{j=1}^{k} w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$

$$\frac{\partial \text{Loss}(x, y, \mathbf{w})}{\partial \mathbf{v}_1} = ?$$

$$2 \cdot \text{residual} \cdot 1 \cdot 1 \cdot w_1 \cdot h_1(1 - h_1) \cdot \phi(x)$$

Computational Graphs and Backpropagation





Backpropagation

Definition: Forward/backward values

Forward: f_i is value for subexpression rooted at i

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

Algorithm: backpropagation

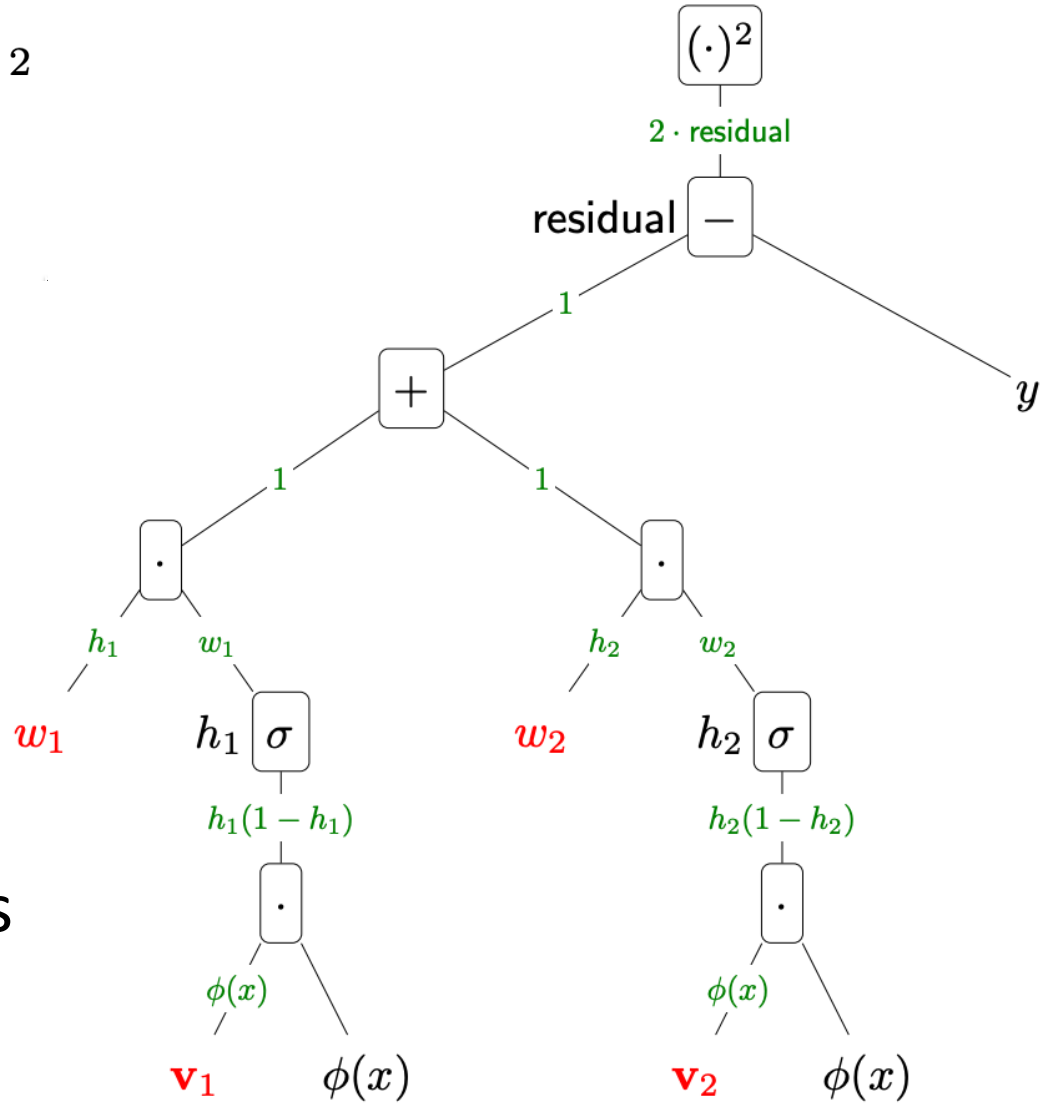
Forward pass: compute each f_i (from leaves to root)

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

Computational Graphs and Backpropagation

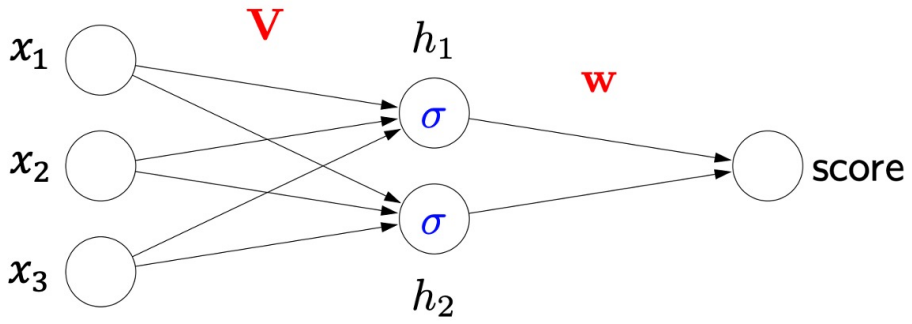
- Exercise: $\text{Loss}(x, y, \mathbf{w}) = \left(\sum_{j=1}^{k} w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$

  - Given a data point:
    $\phi(x) = [1.5, -0.7, 0.6], \quad y = 1.8$
  - Model parameters:
    $\mathbf{v}_1 = [1.1, -1, -0.5], \ \mathbf{v}_2 = [2, 1, 1.2]$
    $\mathbf{w} = [2, 0.6]$
  - Compute the loss function in forward pass and gradients in the backward pass



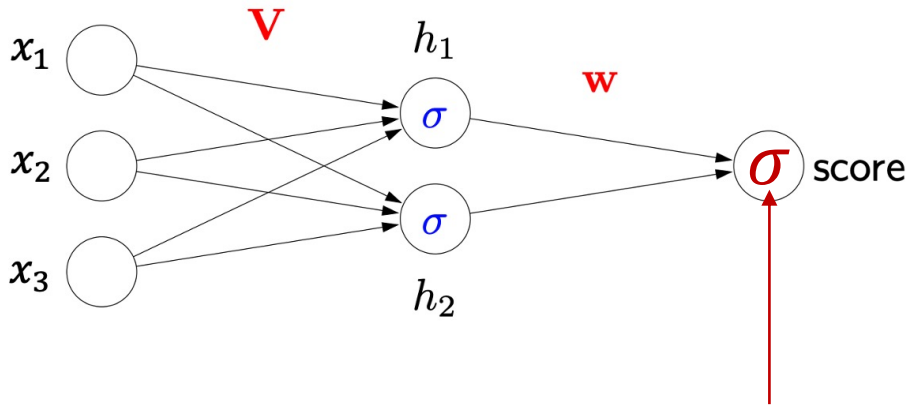
Feedforward Neural Network for Classification

For regression:



$$\text{Loss}(x, y, \mathbf{w}) = \left(\sum_{j=1}^{k} w_j \sigma(\mathbf{v}_j \cdot \phi(x)) - y \right)^2$$

For binary classification:

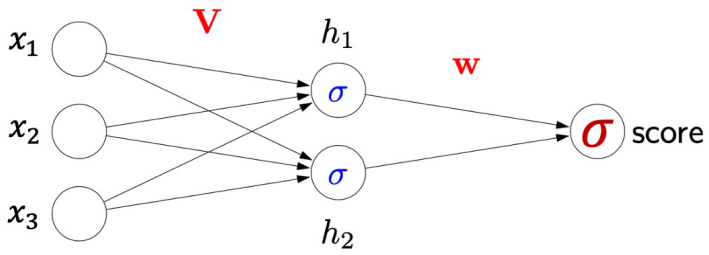


Use a sigmoid function to map the real-valued score to (0, 1)

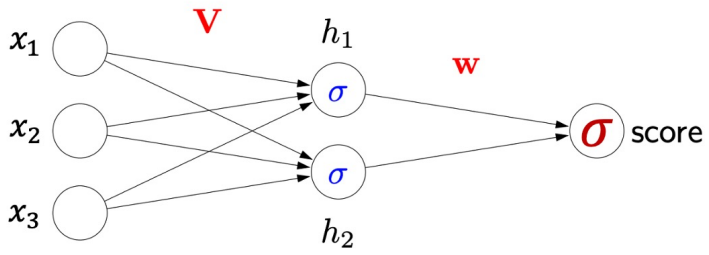
$$\text{Loss}(\mathbf{x}, y, \mathbf{w}) = \log(1 + \exp(-(\mathbf{w}^\top \mathbf{x})y))$$

Feedforward Neural Network for Classification

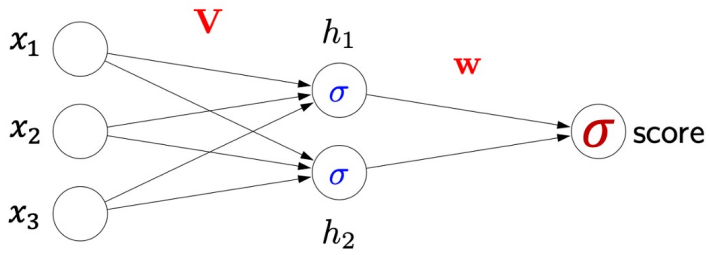
For multiclass classification (OvR):



class 1 vs *not class 1*



class 2 vs *not class 2*

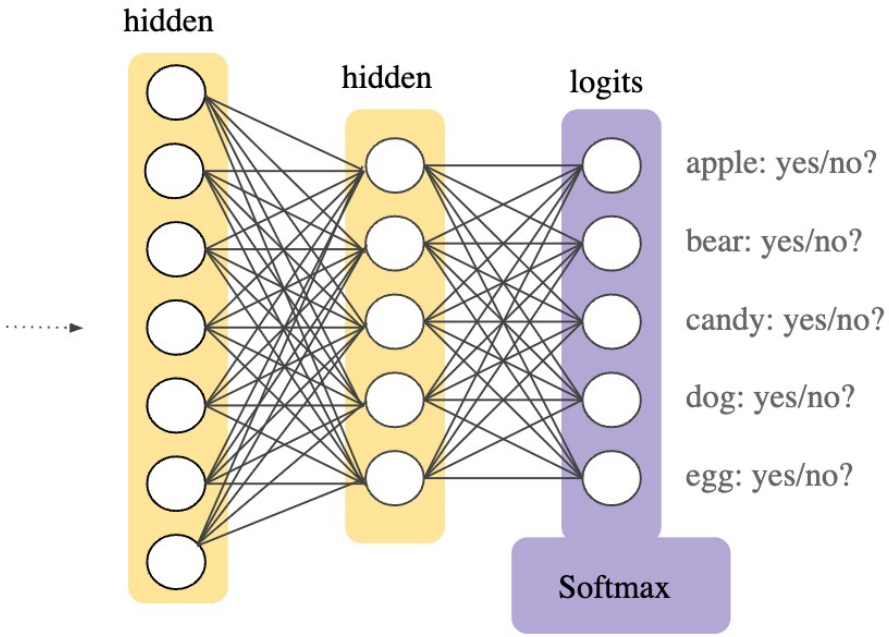


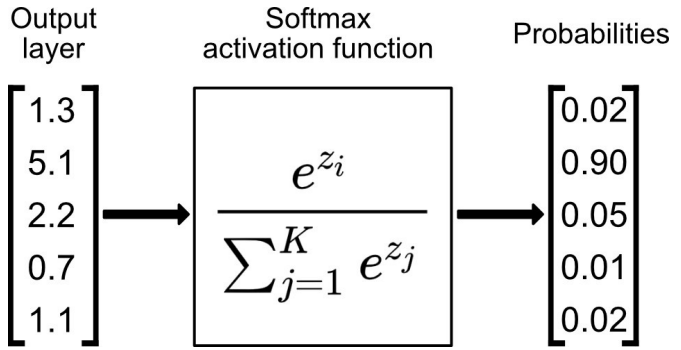
class 3 vs *not class 3*

Not efficient!

Feedforward Neural Network for Classification

Recall: interpret h as learned features for x





Use softmax activation function for the output layer

Maps real-valued scores to a distribution over the classes

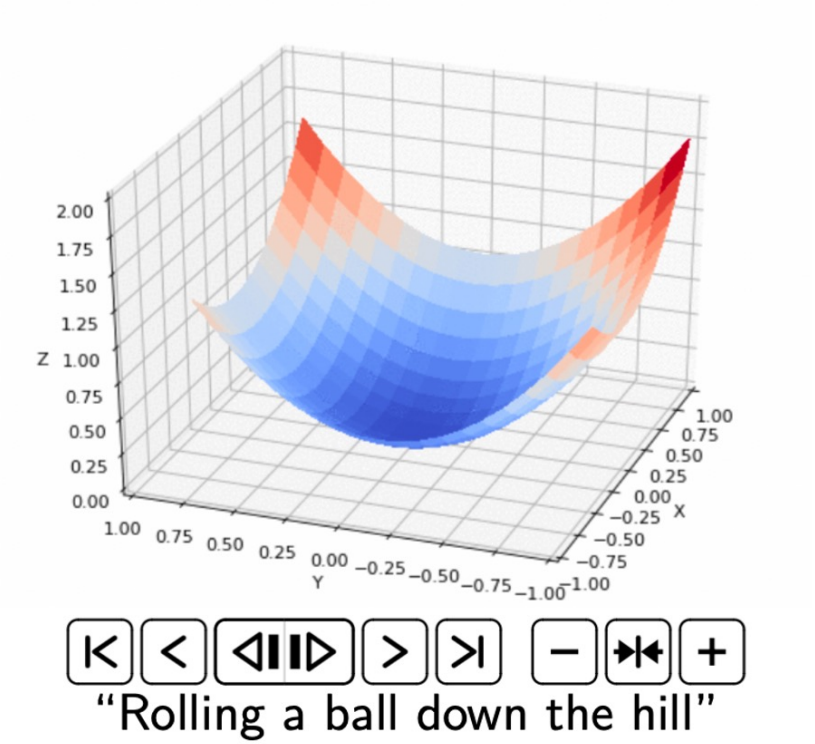
$$\text{Loss}(\mathbf{x}, \mathbf{y}, \Theta) = -\sum_{k=1}^{K} \mathbf{1}[y_k = +1] \log p(y_k = +1 \mid \mathbf{x}) + \mathbf{1}[y_k = -1] \log p(y_k = -1 \mid \mathbf{x})$$

Θ: all model parameters, y_k: the k-th label

Stochastic Gradient Descent

Recall the gradient descent algorithm

- **Graident**: the gradient $\nabla_w f(w)$ is the direction of the greatest increase of $f(w)$.
- Start from an initial location, move a bit along the opposite direction of the gradient.



"Rolling a ball down the hill"

Algorithm: gradient descent

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

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

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$$

Stochastic Gradient Descent

Algorithm: gradient descent

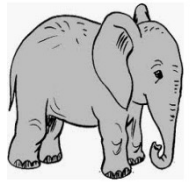
Initialize $\mathbf{w} = [0, \ldots, 0]$

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

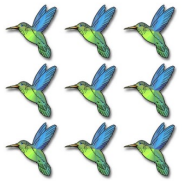
$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_{\mathbf{w}} \text{TrainLoss}(\mathbf{w})}_{\text{gradient}}$$

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

requires iterating through all data points!



Can be expensive



In between: minibatch gradient descent (sample B data points each time)

Most recommended for deep learning B*: batch size*

Algorithm: stochastic gradient descent

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

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

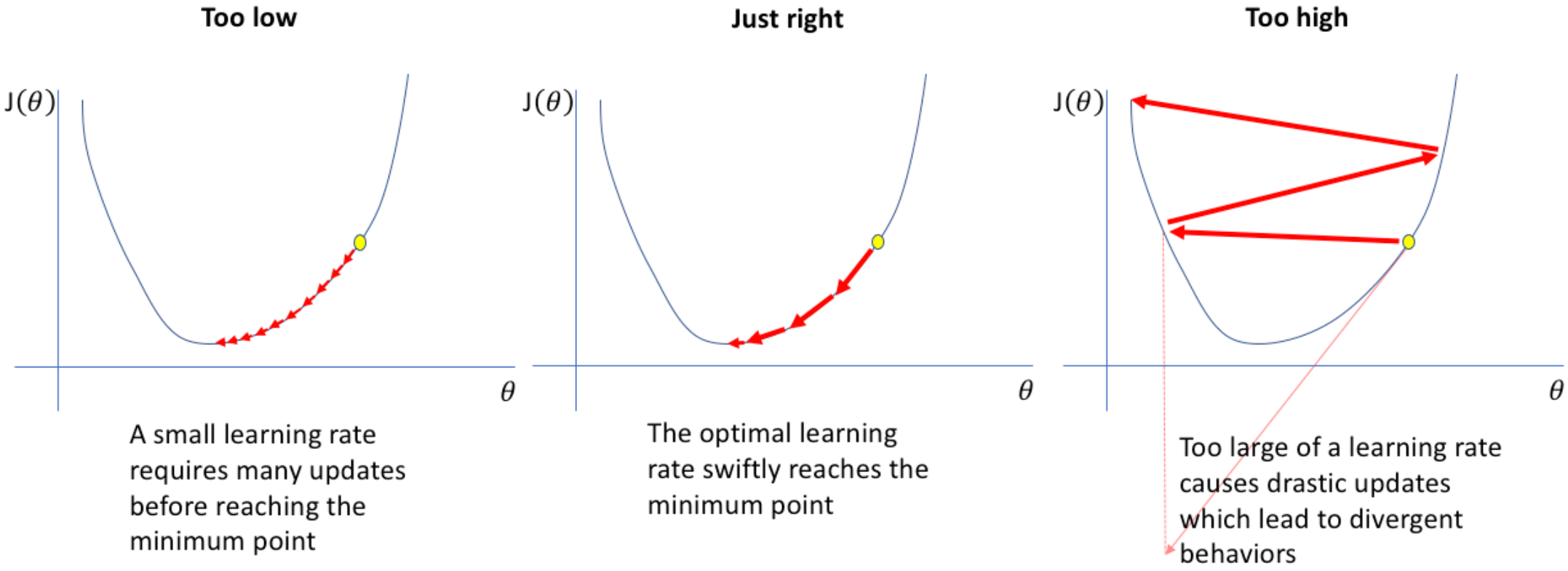
For $(x, y) \in \mathcal{D}_{\text{train}}$:

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

SGD iterates through data samples and update parameters based on each sample

Each update is not as good as GD but we make many more updates.

Impact of Step Size (Learning Rate)



Try this with different learning rates: https://uclaacm.github.io/gradient-descent-visualiser/#playground

Rule of thumb: turn it as a hyperparameter