Classification

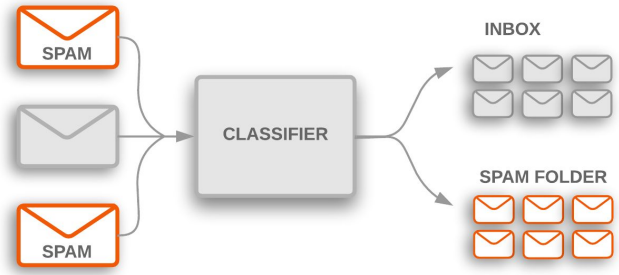
Ryan McDonald[1]

AthNLP

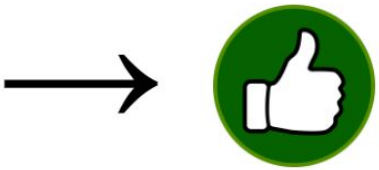
September, 2026

[1]Thanks to Andre Martins and Barbara Plank for use of materials.

Classifiers



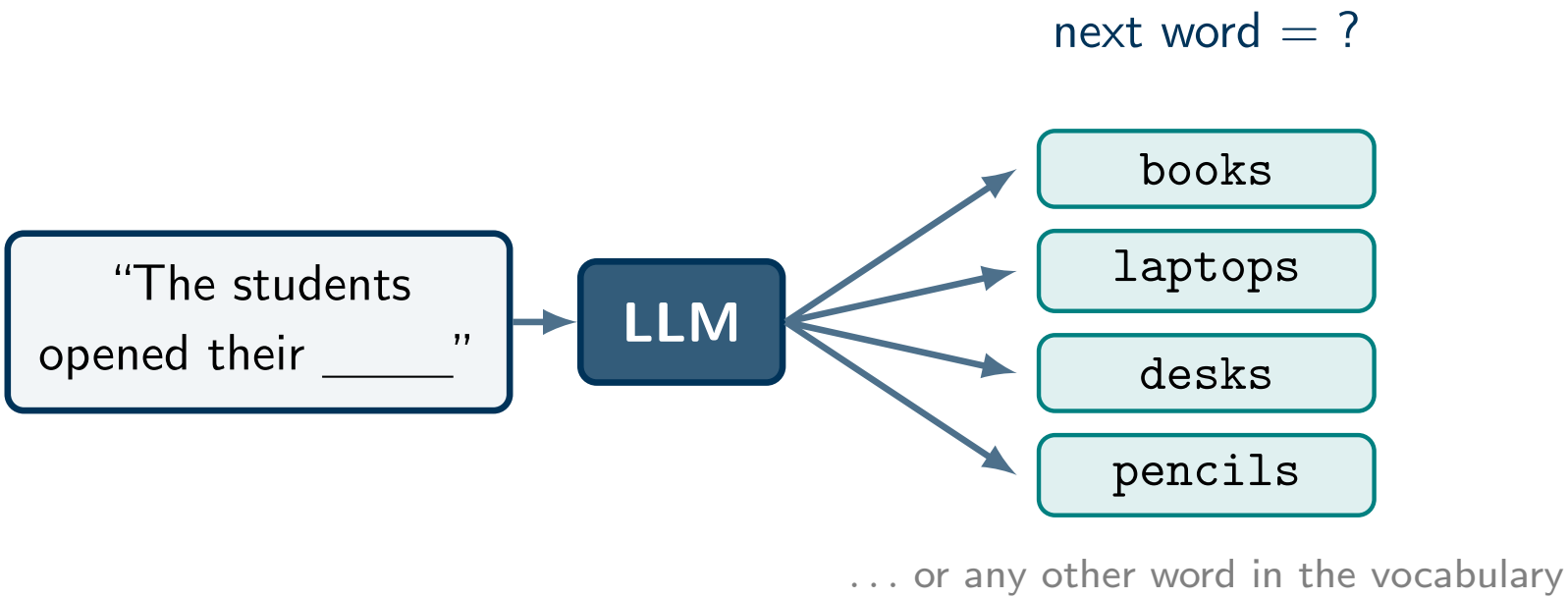
"I love this movie.
I've seen it many times
and it's still awesome."



"This movie is bad.
I don't like it it all.
It's terrible."

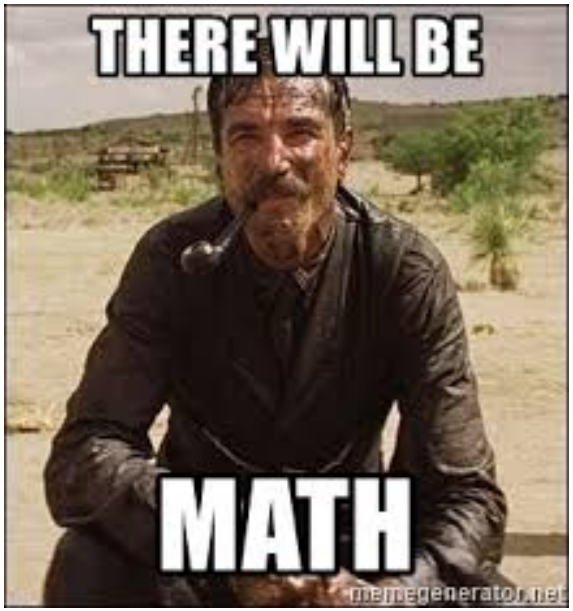
```
Set my alarm tomorrow for 10am   -> Alarm

Quickest way to Boston           -> Navigation

Why is there summer and winter   -> Answer seeking
```

Classifiers



Predicting the next word is **classification** — the label set is the vocabulary.

Warning!



- Focus: machine learning fundamentals
  - Specific to language as input modality
  - Not specific applications
- If you miss a detail, don't worry
- Important to get broad concepts

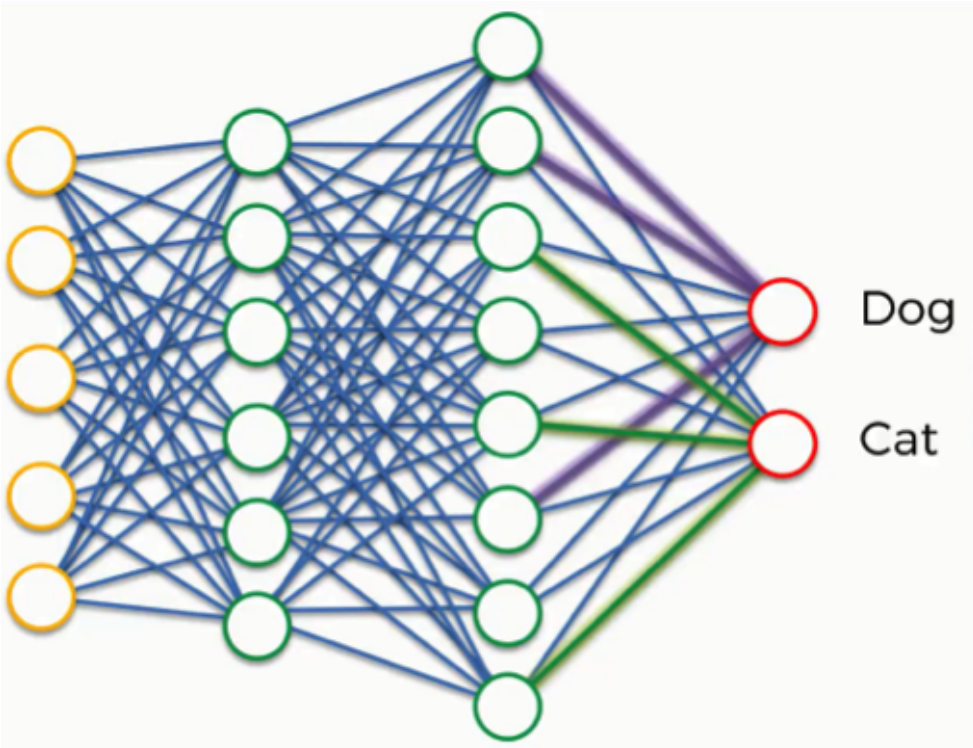
Linear Classifiers

This lecture is 1/2 about linear classifiers!

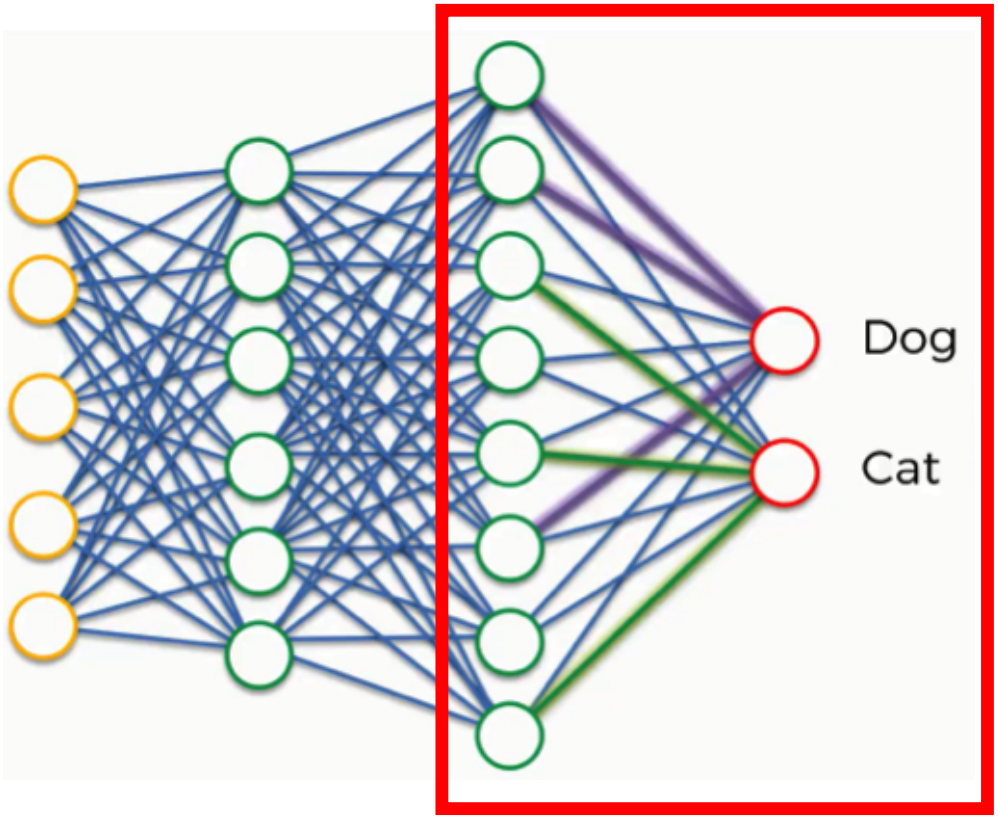
What's a linear classifier? It's 2026 and everybody uses neural networks.

- The underlying machine learning concepts are the same
- The theory (statistics and optimization) are much better understood
- Linear classifiers are **a component of neural networks**.

Linear Classifiers and Neural Networks



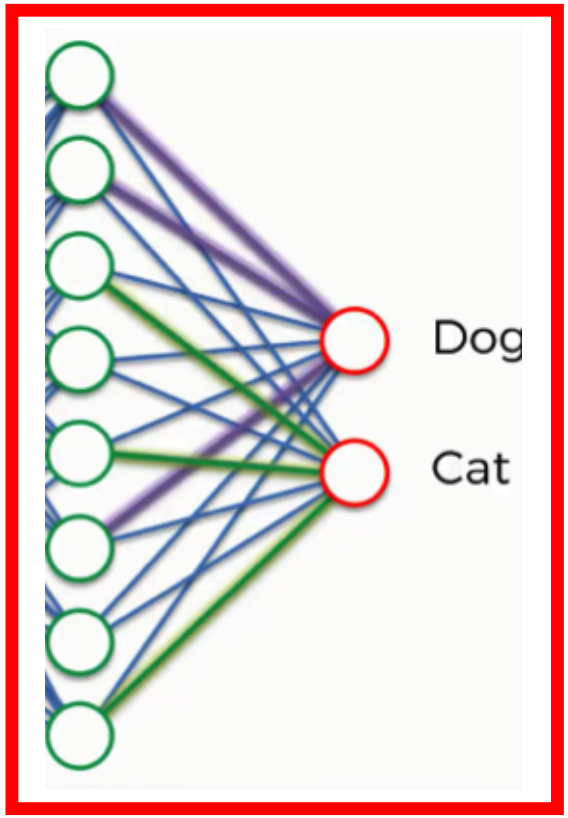
Linear Classifiers and Neural Networks



Linear Classifier

Linear Classifiers and Neural Networks

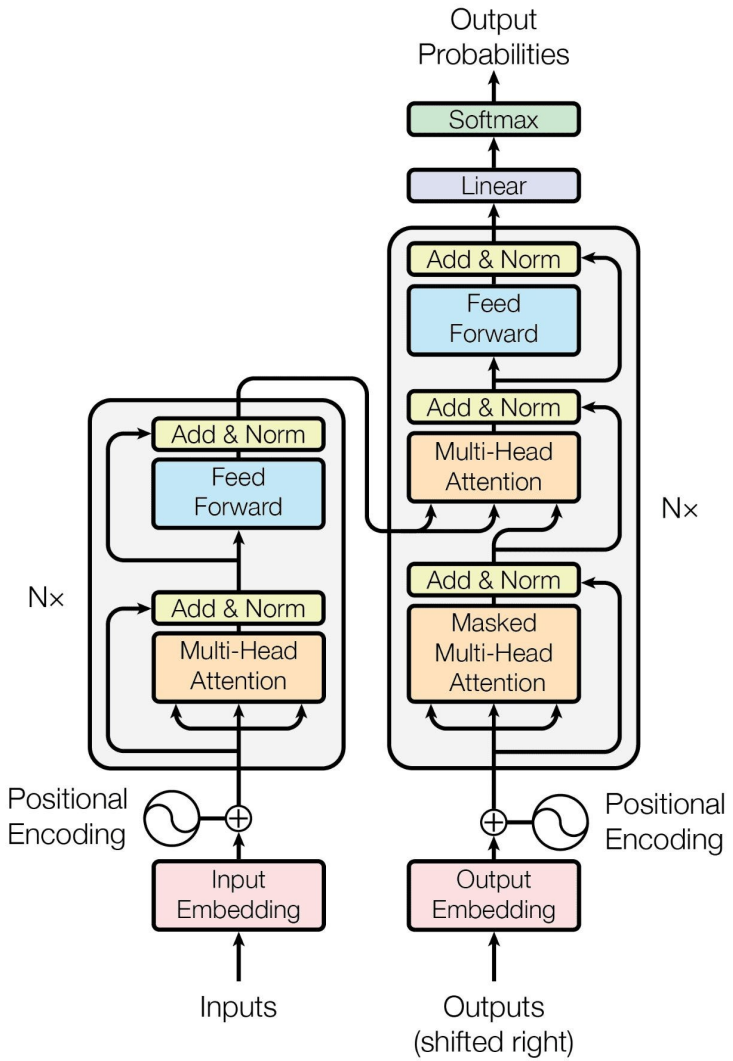
Handcrafted Features



Linear Classifier

Linear Classifiers and Generative AI

- **Transformers**: 99% of LLMs/GenAI
  - ChatGPT; GPT
  - Claude
  - Gemini
  - Llama
  - DeepSeek
  - xAI/Grok
- Last layer = linear classifier
- Last layer predicts next word/token
- I.e., last layer is a classifier!
- Also: Many other linear layers!



Let's Start Simple

- Example 1 – sequence: ⋆ ◇ ○; label: ♣
- Example 2 – sequence: ⋆ ♡ △; label: ♣
- Example 3 – sequence: ⋆ △ ♠; label: □
- Example 4 – sequence: ◇ △ ○; label: □

Let's Start Simple

- Example 1 – sequence: ⋆ ◇ ○; label: ♣
- Example 2 – sequence: ⋆ ♡ △; label: ♣
- Example 3 – sequence: ⋆ △ ♠; label: □
- Example 4 – sequence: ◇ △ ○; label: □

- New sequence: ⋆ ◇ ○; label ?

Let's Start Simple

- Example 1 – sequence: ⋆ ◇ ○; label: ♣
- Example 2 – sequence: ⋆ ♡ △; label: ♣
- Example 3 – sequence: ⋆ △ ♠; label: □
- Example 4 – sequence: ◇ △ ○; label: □

- New sequence: ⋆ ◇ ○; label ♣
- New sequence: ⋆ ◇ ♡; label ?

Let's Start Simple

- Example 1 – sequence: ⋆ ◇ ○; label: ♣
- Example 2 – sequence: ⋆ ♡ △; label: ♣
- Example 3 – sequence: ⋆ △ ♠; label: □
- Example 4 – sequence: ◇ △ ○; label: □

- New sequence: ⋆ ◇ ○; label ♣
- New sequence: ⋆ ◇ ♡; label ♣
- New sequence: ⋆ △ ○; label ?

Let's Start Simple

- Example 1 – sequence: ⋆ ◇ ○; label: ♣
- Example 2 – sequence: ⋆ ♡ △; label: ♣
- Example 3 – sequence: ⋆ △ ♠; label: □
- Example 4 – sequence: ◇ △ ○; label: □

- New sequence: ⋆ ◇ ○; label ♣
- New sequence: ⋆ ◇ ♡; label ♣
- New sequence: ⋆ △ ○; label ?

Why can we do this?

Let's Start Simple: Machine Learning

- Example 1 – sequence: $\star$ $\diamond$ $\circ$; label: $\clubsuit$
- Example 2 – sequence: $\star$ $\heartsuit$ $\triangle$; label: $\clubsuit$
- Example 3 – sequence: $\star$ $\triangle$ $\spadesuit$; label: $\square$
- Example 4 – sequence: $\diamond$ $\triangle$ $\circ$; label: $\square$

- New sequence: $\star$ $\diamond$ $\heartsuit$; label $\clubsuit$

	Label $\clubsuit$		**Label $\square$**
$P(\clubsuit \mid \star)$	$= \frac{\text{count}(\star \text{ and } \clubsuit)}{\text{count}(\star)} = \frac{2}{3} = 0.67$	vs. $P(\square \mid \star)$	$= \frac{\text{count}(\star \text{ and } \square)}{\text{count}(\star)} = \frac{1}{3} = 0.33$
$P(\clubsuit \mid \diamond)$	$= \frac{\text{count}(\diamond \text{ and } \clubsuit)}{\text{count}(\diamond)} = \frac{1}{2} = 0.5$	vs. $P(\square \mid \diamond)$	$= \frac{\text{count}(\diamond \text{ and } \square)}{\text{count}(\diamond)} = \frac{1}{2} = 0.5$
$P(\clubsuit \mid \heartsuit)$	$= \frac{\text{count}(\heartsuit \text{ and } \clubsuit)}{\text{count}(\heartsuit)} = \frac{1}{1} = 1.0$	vs. $P(\square \mid \heartsuit)$	$= \frac{\text{count}(\heartsuit \text{ and } \square)}{\text{count}(\heartsuit)} = \frac{0}{1} = 0.0$

Let's Start Simple: Machine Learning

- Example 1 – sequence: $\star \diamond \circ$; label: ♣
- Example 2 – sequence: $\star \heartsuit \triangle$; label: ♣
- Example 3 – sequence: $\star \triangle \spadesuit$; label: □
- Example 4 – sequence: $\diamond \triangle \circ$; label: □

- New sequence: $\star \triangle \circ$; label ?

	Label ♣		**Label □**
$P(\clubsuit \mid \star)$	$= \frac{\text{count}(\star \text{ and } \clubsuit)}{\text{count}(\star)} = \frac{2}{3} = 0.67$	vs.	$P(\square \mid \star) = \frac{\text{count}(\star \text{ and } \square)}{\text{count}(\star)} = \frac{1}{3} = 0.33$
$P(\clubsuit \mid \triangle)$	$= \frac{\text{count}(\triangle \text{ and } \clubsuit)}{\text{count}(\triangle)} = \frac{1}{3} = 0.33$	vs.	$P(\square \mid \triangle) = \frac{\text{count}(\triangle \text{ and } \square)}{\text{count}(\triangle)} = \frac{2}{3} = 0.67$
$P(\clubsuit \mid \circ)$	$= \frac{\text{count}(\circ \text{ and } \clubsuit)}{\text{count}(\circ)} = \frac{1}{2} = 0.5$	vs.	$P(\square \mid \circ) = \frac{\text{count}(\circ \text{ and } \square)}{\text{count}(\circ)} = \frac{1}{2} = 0.5$

Let's Start Simple – Language Models

Same data, but now treat the label as just the next symbol (a "word") and move it to the rightmost token of the sequence:

- Example 1 – sequence: ⋆ ◇ ○ ♣
- Example 2 – sequence: ⋆ ♡ △ ♣
- Example 3 – sequence: ⋆ △ ♠ □
- Example 4 – sequence: ◇ △ ○ □

"Predict the label" is now "predict the next symbol":

Let's Start Simple – Language Models

Same data, but now treat the label as just the next symbol (a "word") and move it to the rightmost token of the sequence:

- Example 1 – sequence: ⋆ ◇ ○ ♣
- Example 2 – sequence: ⋆ ♡ △ ♣
- Example 3 – sequence: ⋆ △ ♠ □
- Example 4 – sequence: ◇ △ ○ □

"Predict the label" is now "predict the next symbol":

- New sequence: ⋆ ◇ ○ ?

Let's Start Simple – Language Models

Same data, but now treat the label as just the next symbol (a "word") and move it to the rightmost token of the sequence:

- Example 1 – sequence: ⋆ ◇ ○ ♣
- Example 2 – sequence: ⋆ ♡ △ ♣
- Example 3 – sequence: ⋆ △ ♠ □
- Example 4 – sequence: ◇ △ ○ □

"Predict the label" is now "predict the next symbol":

- New sequence: ⋆ ◇ ○ ♣
- New sequence: ⋆ ◇ ♡ ?

Let's Start Simple – Language Models

Same data, but now treat the label as just the next symbol (a "word") and move it to the rightmost token of the sequence:

- Example 1 – sequence: ⋆ ◇ ○ ♣
- Example 2 – sequence: ⋆ ♡ △ ♣
- Example 3 – sequence: ⋆ △ ♠ □
- Example 4 – sequence: ◇ △ ○ □

"Predict the label" is now "predict the next symbol":

- New sequence: ⋆ ◇ ○ ♣
- New sequence: ⋆ ◇ ♡ ♣
- New sequence: ⋆ △ ○ ?

Let's Start Simple – Language Models

Same data, but now treat the label as just the next symbol (a "word") and move it to the rightmost token of the sequence:

- Example 1 – sequence: ⋆ ◇ ○ ♣
- Example 2 – sequence: ⋆ ♡ △ ♣
- Example 3 – sequence: ⋆ △ ♠ □
- Example 4 – sequence: ◇ △ ○ □

"Predict the label" is now "predict the next symbol":

- New sequence: ⋆ ◇ ○ ♣
- New sequence: ⋆ ◇ ♡ ♣
- New sequence: ⋆ △ ○ ?

Same three examples – it's the same problem!

Machine Learning

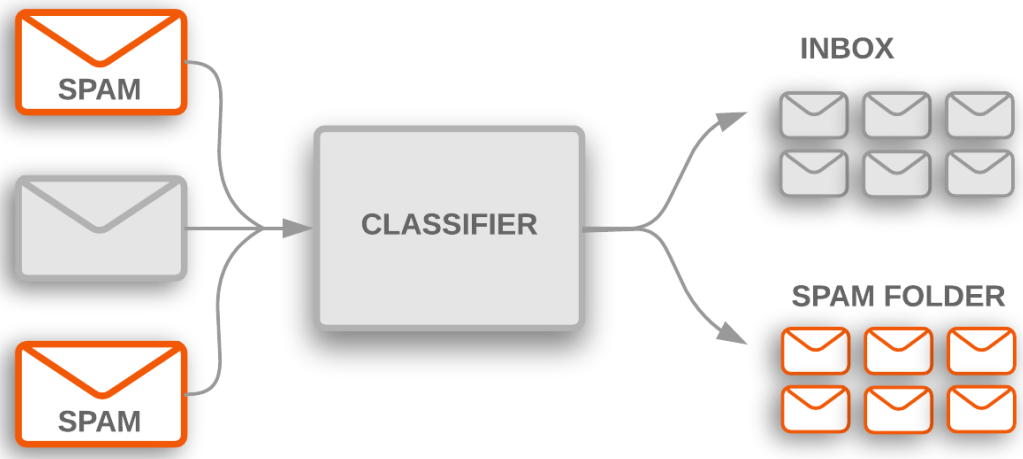
1. Define a model/distribution of interest
2. Make some assumptions if needed
3. Fit the model to the data

Outline

Binary Classification: Spam Detection

Task: Identify if an incoming email/SMS/DM/etc. is spam or not.

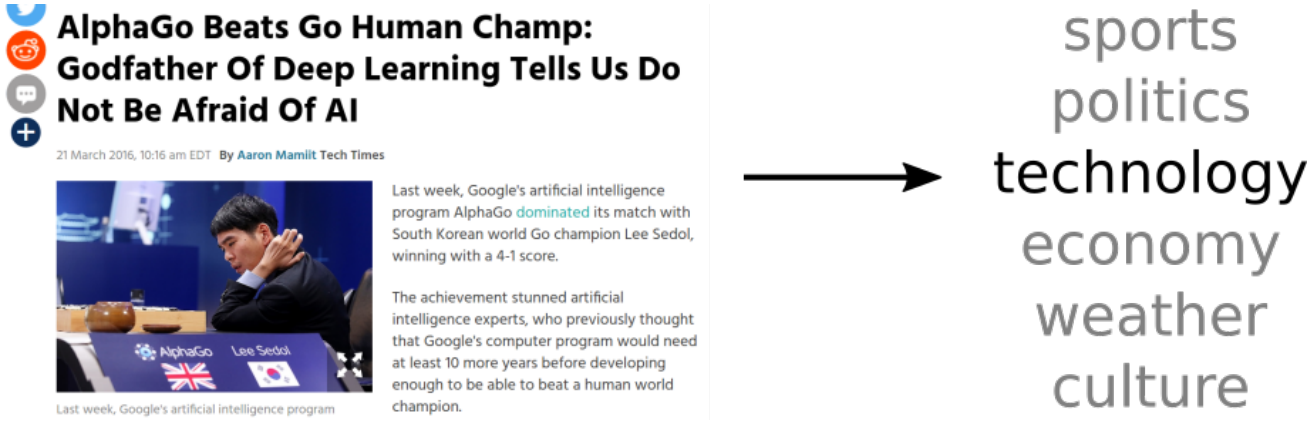
This is a **binary classification problem.**



Multiclass Classification: Topic Labeling

Task: given a news article, determine its topic (politics, sports, etc.)

This is a **multi-class classification problem.**



Inputs and Outputs

- Input $\boldsymbol{x} \in \mathcal{X}$
  - e.g., a news article, a sentence, an image, ...
- Output $\boldsymbol{y} \in \mathcal{Y}$
  - e.g., spam/not spam, a topic, a translation, a next word

- Input/Output pair: $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$
  - e.g., a **news article** together with a **topic**
  - e.g., a **email** together with a **spam/no spam label**
  - e.g., a **sentence prefix** together with a **the next word/token**

(Supervised) Machine Learning

- We are given a **labeled dataset** of input/output pairs:

$$\mathcal{D} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{X}|} \subseteq \mathcal{X} \times \mathcal{Y}$$

- **Goal:** use it to learn a **classifier** $h : \mathcal{X} \rightarrow \mathcal{Y}$ that generalizes well to arbitrary inputs.
- At test time, given $\boldsymbol{x}_t \in \mathcal{X}$, we predict

$$\boldsymbol{y}' = h(\boldsymbol{x}_t).$$

- Hopefully, $\boldsymbol{y}' \approx \boldsymbol{y}_t$ most of the time.

Things can go by different names depending on what $\mathcal{Y}$ is...

Regression

Deals with **continuous** output variables:

- **Regression:** $\mathcal{Y} = \mathbb{R}$
  - e.g., given a news article, how much time a user will spend reading it?
- **Multivariate regression:** $\mathcal{Y} = \mathbb{R}^K$, where $K > 1$
  - e.g., predict the X-Y coordinates in an image where the user will click

Classification

Deals with **discrete** output variables:

- **Binary classification:** $\mathcal{Y} = \{\pm 1\}$
  - e.g., spam detection, positive/negative sentiment
- **Multi-class classification:** $\mathcal{Y} = \{1, 2, \ldots, K\}$
  - e.g., topic classification, positive/negative/neutral sentiment
- **Structured classification:** $\mathcal{Y}$ exponentially large and structured
  - e.g., machine translation, caption generation, image segmentation

What about GenerativeAI?

Our Setup

Let's assume a multi-class classification problem, with $|\mathcal{Y}|$ labels (classes).

Input $\boldsymbol{x}$

What is $\boldsymbol{x}$?

$\boldsymbol{x}$ must be in a compute-friendly (numeric) form

$\boldsymbol{x}$ must be useful for the task

Feature Representations

Typical approach: define a feature map $\boldsymbol{\phi} : \mathcal{X} \rightarrow \mathbb{R}^D$

- $\boldsymbol{\phi}(\boldsymbol{x})$ is a high dimensional feature vector (always a column vector)

We can use feature vectors to encapsulate **Boolean**, **categorical**, and **continuous** features

- To start, we will focus on sparse binary features
- Categorical features can be reduced to a range of one-hot binary values
- We look at continuous (dense) features later

Feature Representations

Feature engineering is an important step in linear classifiers:

- Bag-of-words features for text, also lemmas, parts-of-speech, ...
- Embeddings (e.g., word2vec)
- SIFT features and wavelet representations in computer vision
- External database, APIs and knowledge resources

Later, we will talk about how we can learn these representations

Bag of Words Feature Representation

- $\boldsymbol{x}$ is a (noun) phrase

$$\phi_0(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "George"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_1(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "Washington"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_2(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "Bridge"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_3(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "General"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_4(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains an unknown word} \\ 0 & \text{otherwise} \end{cases}$$

- $\boldsymbol{x}$=General George Washington $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = [1\ 1\ 0\ 1\ 0]^\top$
- $\boldsymbol{x}$=George Washington Bridge $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = [1\ 1\ 1\ 0\ 0]^\top$
- $\boldsymbol{x}$=George Washington University $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = [1\ 1\ 0\ 0\ 1]^\top$
- $\boldsymbol{x}$=George George George of the Jungle $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = [1\ 0\ 0\ 0\ 1]^\top$

Feature Engineering and NLP Pipelines

Classical NLP pipelines consist of stacking together several linear classifiers

Each classifier's predictions are used to handcraft features for other classifiers

Example: Part-of-speech $\rightarrow$ Named Entities $\rightarrow$ Topic Classification

- Part-of-speech: nouns, determiners for Typed Named Entities
  - E.g., Google noun vs. Google verb
- Typed Named Entities: Categories for topic classification
  - E.g., Which George Washington? Person, University/Organization, Bridge/Location?

Outline

A Quick Matrix Primer (1/3): Vectors and Matrices

- A vector is a column of numbers; $\boldsymbol{u} \in \mathbb{R}^3$ means 3 rows, 1 column

$$\boldsymbol{u} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad (3 \times 1) \qquad \boldsymbol{u}^\top = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \quad (1 \times 3)$$

- The transpose $^\top$ turns a column into a row (and back)
- A matrix $\mathbf{A} \in \mathbb{R}^{2 \times 3}$ has 2 rows and 3 columns; $\mathbf{A}_{i,j}$ is row i, column j

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \qquad \mathbf{A}_{2,3} = 6$$

- Addition (same shape only) and scaling work entry by entry

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix} \qquad 2 \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

A Quick Matrix Primer (2/3): Inner and Outer Products

To multiply, the inner dimensions must match: $(m \times n)(n \times p) = (m \times p)$

- Inner product (row $\times$ column) gives a single number

$$\underbrace{\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}}_{1\times 3} \underbrace{\begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}}_{3\times 1} = (1)(4) + (2)(5) + (3)(6) = 32$$

- Outer product (column $\times$ row) gives a matrix

$$\underbrace{\begin{bmatrix} 1 \\ 2 \end{bmatrix}}_{2\times 1} \underbrace{\begin{bmatrix} 3 & 4 \end{bmatrix}}_{1\times 2} = \begin{bmatrix} 3 & 4 \\ 6 & 8 \end{bmatrix} \quad (2 \times 2)$$

A Quick Matrix Primer (3/3): Matrix × Vector

Each entry of the result is the inner product of one row of the matrix with the vector:

$$\underbrace{\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}}_{3\times 2} \underbrace{\begin{bmatrix} 1 \\ 2 \end{bmatrix}}_{2\times 1} = \begin{bmatrix} (1)(1)+(2)(2) \\ (3)(1)+(4)(2) \\ (5)(1)+(6)(2) \end{bmatrix} = \underbrace{\begin{bmatrix} 5 \\ 11 \\ 17 \end{bmatrix}}_{3\times 1}$$

- The inner dimensions must match – here both are 2
  - each row of the matrix has 2 entries, and so does the vector
- The outer dimensions give the shape of the result
  - $(3 \times 2)(2 \times 1) = (3 \times 1)$
- One row in, one number out: a matrix with 3 rows gives 3 numbers

Our Setup

Reminder:

Multi-class classification problem, with $|\mathcal{Y}|$ labels (classes).

Input $\boldsymbol{x}$, represented by a feature vector $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^D$ (a column vector).

Model Parameter Notation – Weights

- Weight matrix $\mathbf{W} \in \mathbb{R}^{|\mathcal{Y}| \times D}$
- Row $\boldsymbol{y}$ of $\mathbf{W}$ is written $\boldsymbol{w}_{\boldsymbol{y}}^{\top}$, with $\boldsymbol{w}_{\boldsymbol{y}} \in \mathbb{R}^{D}$ the weight vector of label $\boldsymbol{y}$

$$\mathbf{W} = \begin{bmatrix} \boldsymbol{w}_1^{\top} \\ \vdots \\ \boldsymbol{w}_{|\mathcal{Y}|}^{\top} \end{bmatrix}$$

- one row per label in $\mathcal{Y}$ and one column per dimension in $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^{D}$

  Intuition: $\mathbf{W}_{\boldsymbol{y},i}$ (entry i of $\boldsymbol{w}_{\boldsymbol{y}}$) says how much feature i counts as *evidence* for label $\boldsymbol{y}$ – positive weights push the score of $\boldsymbol{y}$ up, negative weights push it down.

Model Parameter Notation – Bias

- Bias vector $\boldsymbol{b} \in \mathbb{R}^{|\mathcal{Y}|}$: one bias per label, $b_{\boldsymbol{y}}$ is entry $\boldsymbol{y}$ of $\boldsymbol{b}$

$$\boldsymbol{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_{|\mathcal{Y}|} \end{bmatrix}$$

Intuition: $b_{\boldsymbol{y}}$ is the score label $\boldsymbol{y}$ gets *before looking at any features* – it lets the model prefer labels that are simply more common.

Linear Classifiers: Scores and Predictions

- The score (a.k.a. logit) of a single label $\boldsymbol{y}$:

$$z_{\boldsymbol{y}} = \boldsymbol{w_y}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_{\boldsymbol{y}}$$

$$\boldsymbol{w_y}^\top \boldsymbol{\phi}(\boldsymbol{x}) = \sum_{i=1}^{D} \boldsymbol{w}_{\boldsymbol{y},i}\, \boldsymbol{\phi}_i(\boldsymbol{x}) = \boldsymbol{w}_{\boldsymbol{y},1}\boldsymbol{\phi}_1(\boldsymbol{x}) + \boldsymbol{w}_{\boldsymbol{y},2}\boldsymbol{\phi}_2(\boldsymbol{x}) + \cdots + \boldsymbol{w}_{\boldsymbol{y},D}\boldsymbol{\phi}_D(\boldsymbol{x})$$

- All labels at once – one matrix-vector product:

$$\underbrace{\boldsymbol{z}}_{|\mathcal{Y}| \times 1} = \underbrace{\mathbf{W}}_{|\mathcal{Y}| \times D} \underbrace{\boldsymbol{\phi}(\boldsymbol{x})}_{D \times 1} + \underbrace{\boldsymbol{b}}_{|\mathcal{Y}| \times 1}$$

- Prediction: pick the label with the highest score

$$\boldsymbol{y}' = \textbf{argmax}_{\boldsymbol{y} \in \mathcal{Y}}\ z_{\boldsymbol{y}} = \textbf{argmax}\,(\mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b})$$

Linear Classifiers as Matrix Multiplication

Let $|\mathcal{Y}| = 3$, $D = 3$, $\boldsymbol{w}_{\boldsymbol{y}} = [w_1^{\boldsymbol{y}}, w_2^{\boldsymbol{y}}, w_3^{\boldsymbol{y}}]^\top$, and write ϕ_i for $\phi_i(\boldsymbol{x})$

$$\mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b} = \begin{bmatrix} w_1^1 & w_2^1 & w_3^1 \\ w_1^2 & w_2^2 & w_3^2 \\ w_1^3 & w_2^3 & w_3^3 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$= \begin{bmatrix} (w_1^1 \times \phi_1) + (w_2^1 \times \phi_2) + (w_3^1 \times \phi_3) + b_1 \\ (w_1^2 \times \phi_1) + (w_2^2 \times \phi_2) + (w_3^2 \times \phi_3) + b_2 \\ (w_1^3 \times \phi_1) + (w_2^3 \times \phi_2) + (w_3^3 \times \phi_3) + b_3 \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{w}_1{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_1 \\ \boldsymbol{w}_2{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_2 \\ \boldsymbol{w}_3{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_3 \end{bmatrix} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \boldsymbol{z}$$

One row of $\mathbf{W}$ = one label = one score.

Linear Classifiers – Example

- $D = 5$, $\mathcal{Y} = \{\text{Location (loc), Person (per)}\}$, so $\mathbf{W} \in \mathbb{R}^{2 \times 5}$
- $\boldsymbol{x}$ =George Washington Bridge $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = [1, 1, 1, 0, 0]^\top$

$$
\begin{aligned}
\boldsymbol{z} &= \mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b} \\
&= \begin{bmatrix} -0.6 & 2.4 & 4.0 & -2.1 & 0.1 \\ 0.3 & 1.2 & -5.4 & 3.8 & -0.09 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} (-0.6 \times 1) + (2.4 \times 1) + (4.0 \times 1) + (-2.1 \times 0) + (0.1 \times 0) + 0 \\ (0.3 \times 1) + (1.2 \times 1) + (-5.4 \times 1) + (3.8 \times 0) + (-0.09 \times 0) + 0 \end{bmatrix} \\
&= \begin{bmatrix} 5.8 \\ -3.9 \end{bmatrix} \begin{matrix} \text{loc} \\ \text{per} \end{matrix}
\end{aligned}
$$

$$\boldsymbol{y}' = \textbf{argmax}\left(\mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b}\right) = \textbf{argmax}\{5.8_{\text{loc}}, -3.9_{\text{per}}\} = \text{loc}$$

Model Parameters Θ

$$\Theta = (\mathbf{W}, b)$$

- Each possible Θ is an instance of a model
- Where does Θ come from?
- We will learn Θ from data
- Different ways of learning it give different linear classifiers
  - perceptron, logistic regression, SVMs, ...

Binary Linear Classifier

With binary labels ($\mathcal{Y} = \{\pm 1\}$), $\mathbf{W}$ has just two rows, $\boldsymbol{w}_{+1}^\top$ and $\boldsymbol{w}_{-1}^\top$. We often use a minimal parametrization:

$$\boldsymbol{y}' = \arg\max_{\boldsymbol{y} \in \{\pm 1\}} \boldsymbol{w}_{\boldsymbol{y}}{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_{\boldsymbol{y}}$$

Binary Linear Classifier

With binary labels ($\mathcal{Y} = \{\pm 1\}$), $\mathbf{W}$ has just two rows, $\boldsymbol{w}_{+1}^\top$ and $\boldsymbol{w}_{-1}^\top$. We often use a minimal parametrization:

$$\begin{aligned}
\boldsymbol{y}' &= \arg\max_{\boldsymbol{y} \in \{\pm 1\}} \boldsymbol{w}_{\boldsymbol{y}}{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_{\boldsymbol{y}} \\
&= \begin{cases} +1 & \text{if } \boldsymbol{w}_{+1}{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_{+1} > \boldsymbol{w}_{-1}{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_{-1} \\ -1 & \text{otherwise} \end{cases}
\end{aligned}$$

Binary Linear Classifier

With binary labels ($\mathcal{Y} = \{\pm 1\}$), $\mathbf{W}$ has just two rows, $\boldsymbol{w}_{+1}^\top$ and $\boldsymbol{w}_{-1}^\top$. We often use a minimal parametrization:

$$
\begin{aligned}
\boldsymbol{y}' &= \arg\max_{\boldsymbol{y} \in \{\pm 1\}} \boldsymbol{w}_{\boldsymbol{y}}{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_{\boldsymbol{y}} \\
&= \begin{cases} +1 & \text{if } \boldsymbol{w}_{+1}{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_{+1} > \boldsymbol{w}_{-1}{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_{-1} \\ -1 & \text{otherwise} \end{cases} \\
&= \begin{cases} +1 & \text{if } (\boldsymbol{w}_{+1} - \boldsymbol{w}_{-1})^\top \boldsymbol{\phi}(\boldsymbol{x}) + (b_{+1} - b_{-1}) > 0 \\ -1 & \text{otherwise} \end{cases}
\end{aligned}
$$

Binary Linear Classifier

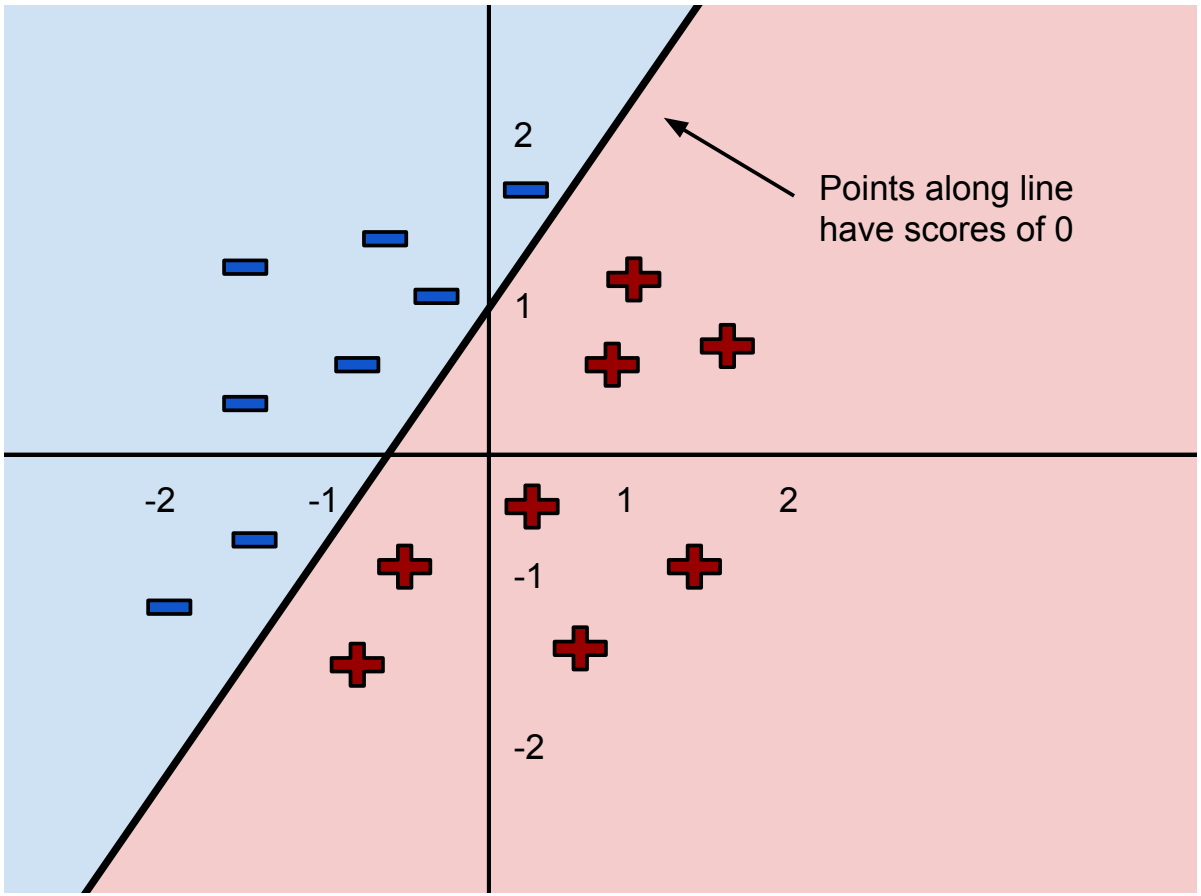
With binary labels ($\mathcal{Y} = \{\pm 1\}$), $\mathbf{W}$ has just two rows, $\boldsymbol{w}_{+1}^\top$ and $\boldsymbol{w}_{-1}^\top$. We often use a minimal parametrization:

$$
\begin{aligned}
\boldsymbol{y}' &= \arg\max_{\boldsymbol{y} \in \{\pm 1\}} \boldsymbol{w}_{\boldsymbol{y}}{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_{\boldsymbol{y}} \\
&= \begin{cases} +1 & \text{if } \boldsymbol{w}_{+1}{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_{+1} > \boldsymbol{w}_{-1}{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_{-1} \\ -1 & \text{otherwise} \end{cases} \\
&= \begin{cases} +1 & \text{if } (\boldsymbol{w}_{+1} - \boldsymbol{w}_{-1})^\top \boldsymbol{\phi}(\boldsymbol{x}) + (b_{+1} - b_{-1}) > 0 \\ -1 & \text{otherwise} \end{cases} \\
&= \text{sign}(\underbrace{(\boldsymbol{w}_{+1} - \boldsymbol{w}_{-1})}_{\boldsymbol{v}}{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + \underbrace{(b_{+1} - b_{-1})}_{c}).
\end{aligned}
$$

That is: only half of the parameters are needed.

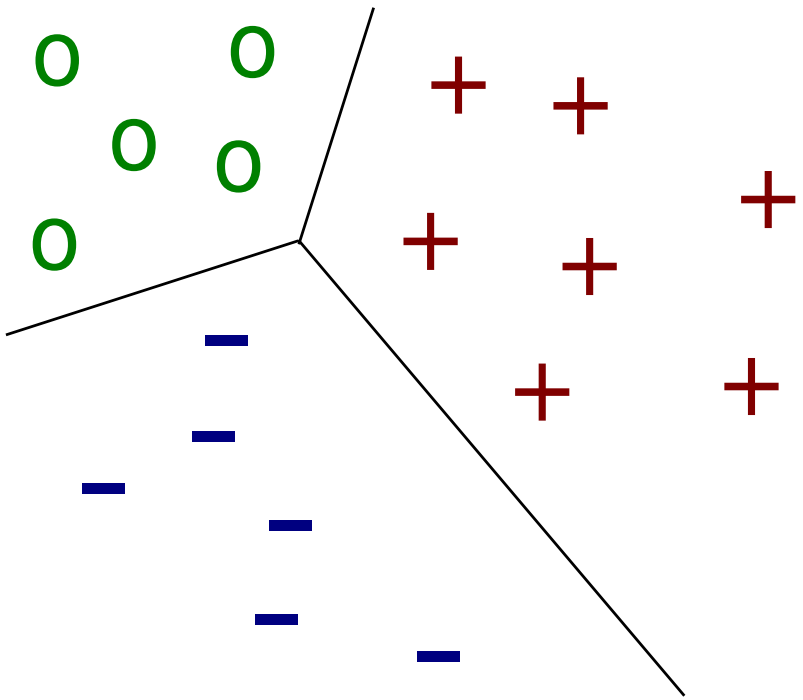
Binary Linear Classifier

Then $(\boldsymbol{v}, c)$ is an hyperplane that divides all points:



Multiclass Linear Classifier

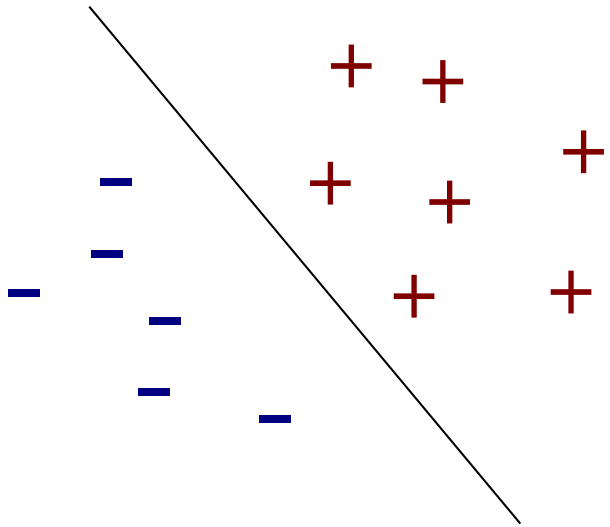
Defines regions of space.



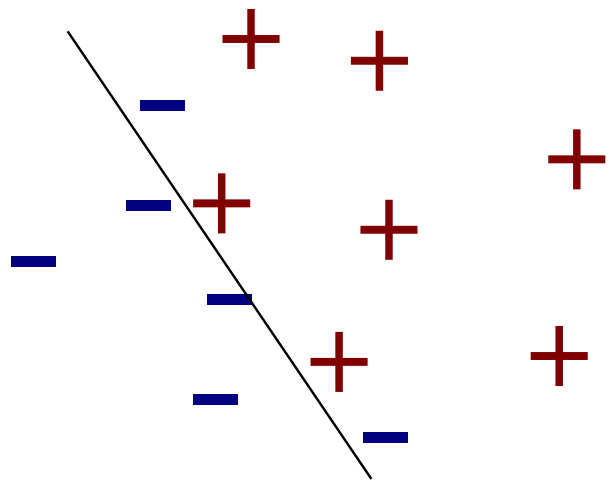
Linear Separability

- A set of points is linearly separable if there exists a $(\mathbf{W}, \boldsymbol{b})$ such that classification is perfect

Separable



Not Separable

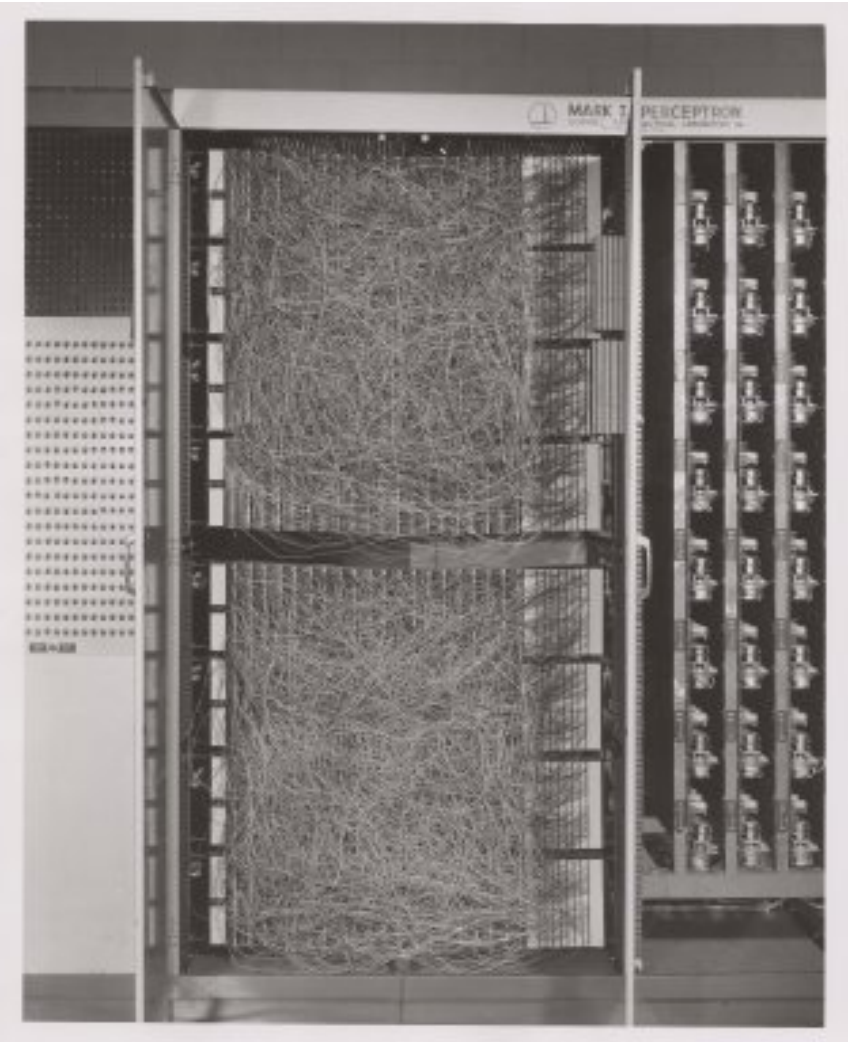


Learning

- Machine Learning = finding the parameters $\boldsymbol{\Theta} = (\mathbf{W}, \boldsymbol{b})$
- Using data! Specifically $\mathcal{D} = \{\boldsymbol{x}_t, \boldsymbol{y}_t\}_{t=1}$
- There are many algorithms for doing this

Outline

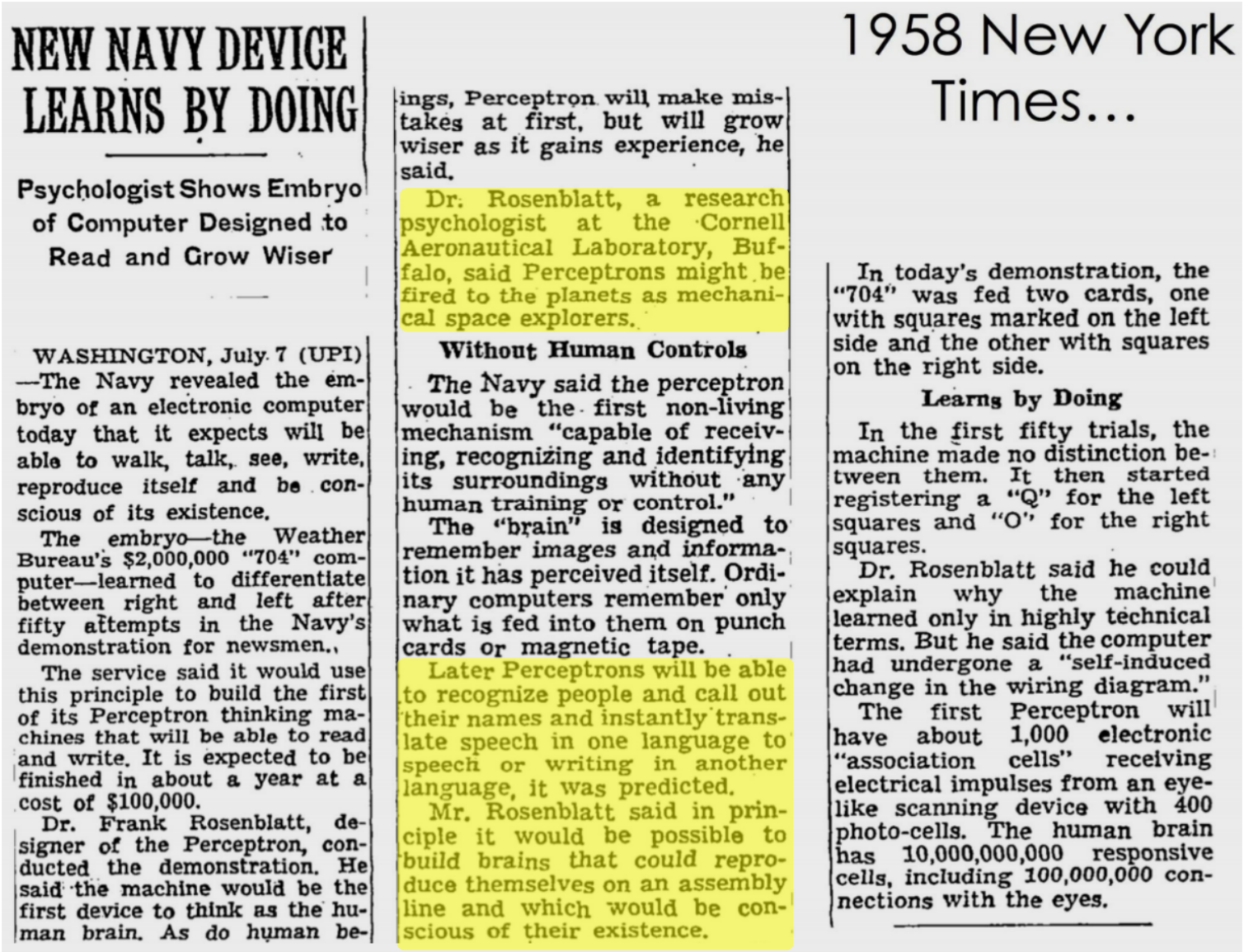
Perceptron (Rosenblatt, 1958)



(Extracted from Wikipedia)

- Invented in 1957 at the Cornell Aeronautical Laboratory by Frank Rosenblatt
- Implemented in custom-built hardware as the "Mark 1 perceptron," designed for image recognition
- 400 photocells, randomly connected to the "neurons." Weights were encoded in potentiometers
- Weight updates during learning were performed by electric motors.

Perceptron in the News...



NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

1958 New York Times...

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

Perceptron Algorithm

- Online algorithm: process one data point $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ at time
  - Take $\boldsymbol{x}_t$; apply the current model to make a prediction for it

$$\boldsymbol{y}' = \textbf{argmax}_{\boldsymbol{y} \in \mathcal{Y}} \; z_{\boldsymbol{y}} \; = \; \textbf{argmax} \left(\mathbf{W} \boldsymbol{\phi}(\boldsymbol{x}_t) + \boldsymbol{b} \right)$$

  - If prediction is correct, proceed
  - Else, correct model: add the feature vector to the row of the correct output & subtract it from the row of the predicted (wrong) output

$$\boldsymbol{w}_{\boldsymbol{y}_t} = \boldsymbol{w}_{\boldsymbol{y}_t} + \boldsymbol{\phi}(\boldsymbol{x}_t) \qquad \boldsymbol{w}_{\boldsymbol{y}'} = \boldsymbol{w}_{\boldsymbol{y}'} - \boldsymbol{\phi}(\boldsymbol{x}_t)$$

  - Same for the corresponding biases

$$b_{\boldsymbol{y}_t} = b_{\boldsymbol{y}_t} + 1 \qquad b_{\boldsymbol{y}'} = b_{\boldsymbol{y}'} - 1$$

Perceptron Update as a Matrix Update

- Let $\boldsymbol{e}_{\boldsymbol{y}} \in \mathbb{R}^{|\mathcal{Y}|}$ be the one-hot vector of label $\boldsymbol{y}$: a 1 in position $\boldsymbol{y}$, zeros elsewhere
- *Add $\phi(\boldsymbol{x}_t)$ to row $\boldsymbol{y}_t$, subtract it from row $\boldsymbol{y}'$* is exactly

$$\mathbf{W} = \mathbf{W} + (\boldsymbol{e}_{\boldsymbol{y}_t} - \boldsymbol{e}_{\boldsymbol{y}'})\,\phi(\boldsymbol{x}_t)^\top, \qquad \boldsymbol{b} = \boldsymbol{b} + (\boldsymbol{e}_{\boldsymbol{y}_t} - \boldsymbol{e}_{\boldsymbol{y}'})$$

- $(\boldsymbol{e}_{\boldsymbol{y}_t} - \boldsymbol{e}_{\boldsymbol{y}'})\,\phi(\boldsymbol{x}_t)^\top$ is an outer product: an $|\mathcal{Y}| \times D$ matrix
  - row $\boldsymbol{y}_t$ is $+\phi(\boldsymbol{x}_t)^\top$, row $\boldsymbol{y}'$ is $-\phi(\boldsymbol{x}_t)^\top$, all other rows are $\mathbf{0}$

$$(\boldsymbol{e}_{\boldsymbol{y}_t} - \boldsymbol{e}_{\boldsymbol{y}'})\,\phi(\boldsymbol{x}_t)^\top = \begin{bmatrix} \mathbf{0}^\top \\ +\phi(\boldsymbol{x}_t)^\top \\ \mathbf{0}^\top \\ -\phi(\boldsymbol{x}_t)^\top \\ \mathbf{0}^\top \end{bmatrix} \begin{matrix} \\ \text{row } \boldsymbol{y}_t \\ \\ \text{row } \boldsymbol{y}' \\ \\ \end{matrix}$$

Remember this shape – every update we see has it.

Perceptron Update – A Small Example

$|\mathcal{Y}| = 3$ labels, $D = 2$ features. Current parameters and a training example:

$$\mathbf{W} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{matrix} \text{label 1} \\ \text{label 2} \\ \text{label 3} \end{matrix} \qquad \boldsymbol{b} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \boldsymbol{\phi}(\boldsymbol{x}_t) = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \boldsymbol{y}_t = 3$$

Predict:

$$\boldsymbol{z} = \mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}_t) + \boldsymbol{b} = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} \implies \boldsymbol{y}' = 1 \neq \boldsymbol{y}_t = 3 \quad \text{mistake!}$$

Update: add $\boldsymbol{\phi}(\boldsymbol{x}_t)$ to row 3, subtract it from row 1

$$\left(\underbrace{\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}}_{\boldsymbol{e}_3} - \underbrace{\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}}_{\boldsymbol{e}_1} \right) \underbrace{\begin{bmatrix} 2 & 1 \end{bmatrix}}_{\boldsymbol{\phi}(\boldsymbol{x}_t)^\top} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} 2 & 1 \end{bmatrix} = \begin{bmatrix} -2 & -1 \\ 0 & 0 \\ 2 & 1 \end{bmatrix}$$

Perceptron Update – A Small Example

Apply it to the weights and the biases:

$$\mathbf{W} \leftarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} -2 & -1 \\ 0 & 0 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} -1 & -1 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \qquad \boldsymbol{b} \leftarrow \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

Only the rows of the true and the predicted label changed. Score the same example again:

$$\boldsymbol{z} = \begin{bmatrix} -1 & -1 \\ 0 & 1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -4 \\ 1 \\ 6 \end{bmatrix} \implies \boldsymbol{y}' = 3 \quad \text{correct!}$$

The update pushes the true label's score up and the wrong one's down.

Perceptron Algorithm

input: labeled data $\mathcal{D}$
initialize $\mathbf{W} = \mathbf{0}$ and $\boldsymbol{b} = \mathbf{0}$
repeat
  observe example $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$
  predict $\boldsymbol{y}' = \textbf{argmax}\,(\mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}_t) + \boldsymbol{b})$
  if $\boldsymbol{y}' \neq \boldsymbol{y}_t$ **then**
    update $\mathbf{W} = \mathbf{W} + (\boldsymbol{e}_{\boldsymbol{y}_t} - \boldsymbol{e}_{\boldsymbol{y}'})\,\boldsymbol{\phi}(\boldsymbol{x}_t)^\top$
    update $\boldsymbol{b} = \boldsymbol{b} + (\boldsymbol{e}_{\boldsymbol{y}_t} - \boldsymbol{e}_{\boldsymbol{y}'})$
  end if
until stopping criterion[2]
output: model parameters $\boldsymbol{\Theta} = (\mathbf{W}, \boldsymbol{b})$

[2] E.g., max iterations; zero/ϵ errors

Perceptron's Mistake Bound

A couple definitions:

- the training data is linearly separable with margin $\gamma > 0$ iff there is a matrix $\mathbf{U}$ whose rows $\boldsymbol{u}_{\boldsymbol{y}}^\top$ have $\|\boldsymbol{u}_{\boldsymbol{y}}\| = 1$ such that

$$\boldsymbol{u}_{\boldsymbol{y}_t}{}^\top \boldsymbol{\phi}(\boldsymbol{x}_t) \geq \boldsymbol{u}_{\boldsymbol{y}'}{}^\top \boldsymbol{\phi}(\boldsymbol{x}_t) + \gamma, \quad \forall t,\ \forall \boldsymbol{y}' \neq \boldsymbol{y}_t.$$

- radius of the data: $R = \max_t \|\boldsymbol{\phi}(\boldsymbol{x}_t)\|$.

Perceptron's Mistake Bound

A couple definitions:

- the training data is linearly separable with margin $\gamma > 0$ iff there is a matrix $\mathbf{U}$ whose rows $\boldsymbol{u}_{\boldsymbol{y}}^\top$ have $\|\boldsymbol{u}_{\boldsymbol{y}}\| = 1$ such that

$$\boldsymbol{u}_{\boldsymbol{y}_t}{}^\top \boldsymbol{\phi}(\boldsymbol{x}_t) \geq \boldsymbol{u}_{\boldsymbol{y}'}{}^\top \boldsymbol{\phi}(\boldsymbol{x}_t) + \gamma, \quad \forall t,\ \forall \boldsymbol{y}' \neq \boldsymbol{y}_t.$$

- radius of the data: $R = \max_t \|\boldsymbol{\phi}(\boldsymbol{x}_t)\|$.

Then we have the following bound of the number of mistakes:

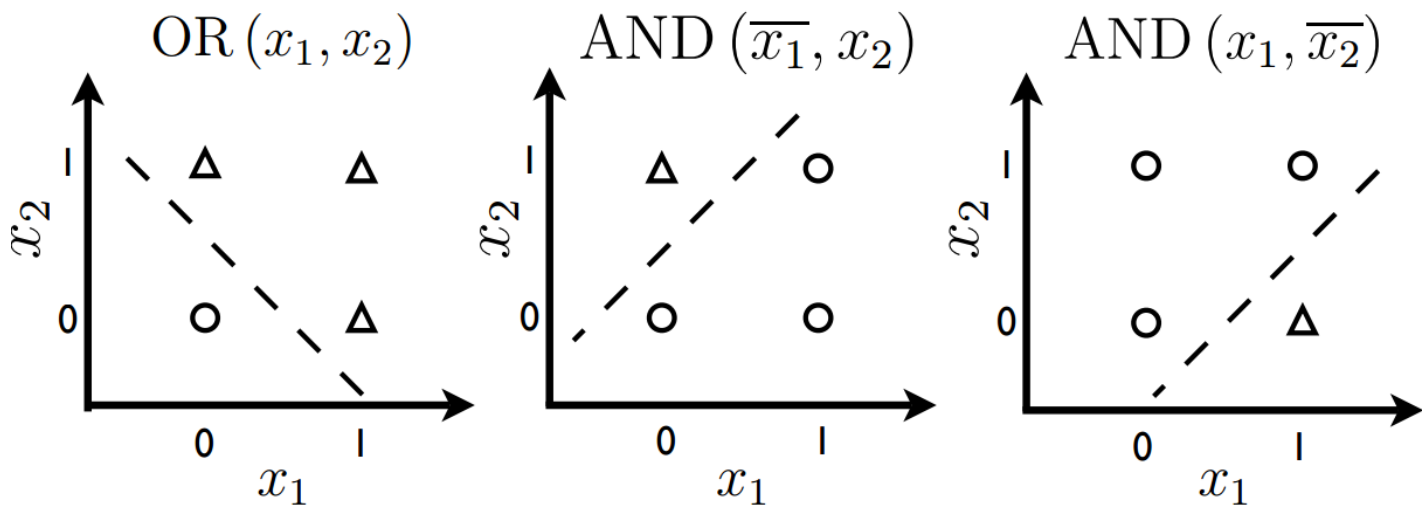
Theorem (Novikoff, 1962)

The perceptron algorithm is guaranteed to find a separating hyperplane after at most $2\frac{R^2}{\gamma^2}$ *mistakes.*

Proof: `https://proceedings.mlr.press/v97/beygelzimer19a/beygelzimer19a-supp.pdf`

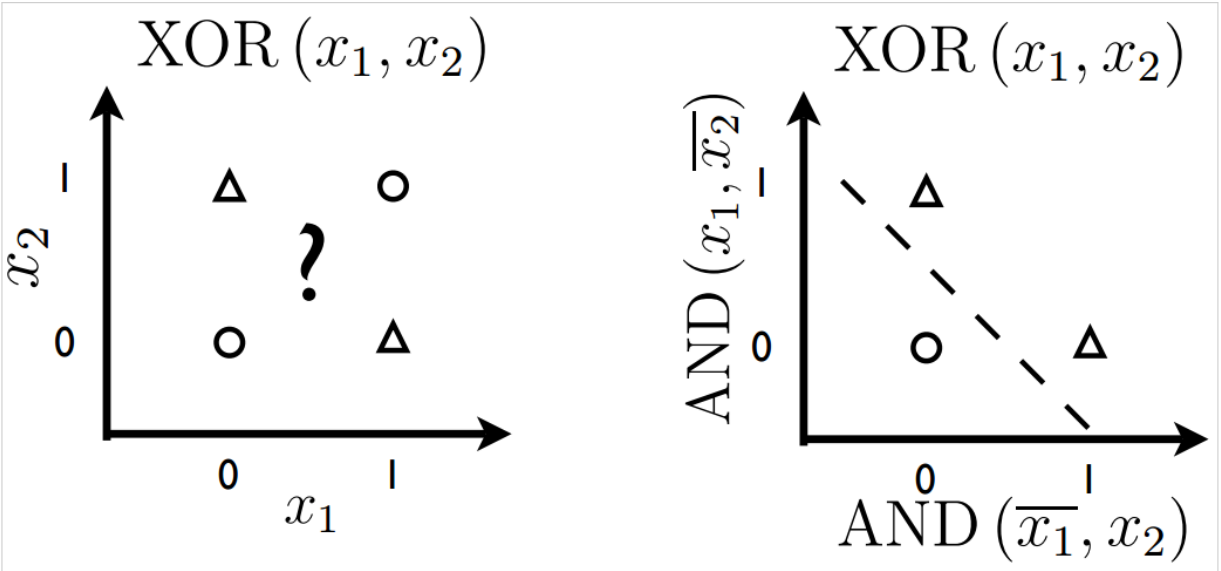
What a Simple Perceptron Can and Can't Do

- Remember: the decision boundary is linear (linear classifier)
- It **can** solve linearly separable problems (OR, AND)



What a Simple Perceptron Can and Can't Do

- ... but it **can't** solve non-linearly separable problems such as simple XOR (unless input is transformed into a better representation):



$$XOR(x_1, x_2) = XOR(AND(x_1, \overline{x_2}), AND(\overline{x_1}, x_2))$$

- Result attributed to Minsky and Papert (1969) but was known before.

Outline

Logistic Regression

Turn the logits $\boldsymbol{z} = \mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b}$ into a conditional probability:

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(z_{\boldsymbol{y}})}{Z_{\boldsymbol{x}}}, \quad Z_{\boldsymbol{x}} = \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp(z_{\boldsymbol{y}'}), \quad \forall z_{\boldsymbol{y}} \in \boldsymbol{z}$$

This is called a **softmax** transformation

In vector form $\boldsymbol{p} = \textbf{softmax}(\boldsymbol{z}) = [\frac{\exp(z_1)}{Z_{\boldsymbol{x}}}, \ldots, \frac{\exp(z_{|\mathcal{Y}|})}{Z_{\boldsymbol{x}}}]^\top$

I.e., $\boldsymbol{p} = [p_1, \ldots, p_{|\mathcal{Y}|}]^\top$, where $p_{\boldsymbol{y}} = P(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(z_{\boldsymbol{y}})}{Z_{\boldsymbol{x}}}$

Probability distribution: $\sum_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) = 1$ and $P(\boldsymbol{y}|\boldsymbol{x}) \geq 0, \forall \boldsymbol{y}$

Logistic Regression

$$P_{\Theta}(y|x) = \frac{\exp(z_y)}{Z_x}, \qquad z = \mathbf{W}\phi(x) + b$$

- The parameters are the matrix and the bias: $\Theta = (\mathbf{W}, b)$

How do we learn Θ?

- Set Θ to minimize the negative conditional log-likelihood:

$$\begin{aligned}\Theta &= \arg\min_{\Theta} -\log\left(\prod_{t=1} P_{\Theta}(y_t|x_t)\right) = \arg\min_{\Theta} -\sum_{t=1} \log P_{\Theta}(y_t|x_t) \\ &= \arg\min_{\Theta} \sum_{t=1}\left(\log\sum_{y'}\exp(z_{y'}) - z_{y_t}\right),\end{aligned}$$

i.e., assign as much probability mass as possible to the correct labels!

Logistic Regression

- This objective function is convex
- Therefore any local minimum is a global minimum
- No closed form solution, but lots of numerical techniques
  - Gradient methods (gradient descent, conjugate gradient)
  - Quasi-Newton methods (L-BFGS, ...)

Recap: Convex functions

Pro: Guarantee of a global minima ✓

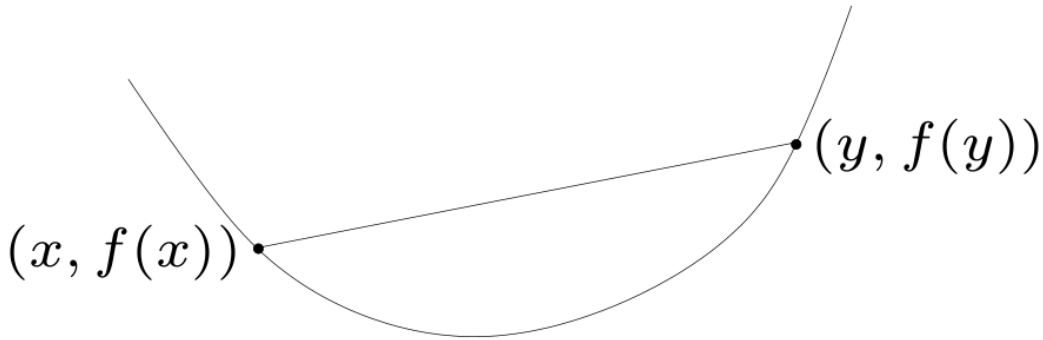


Figure: Illustration of a convex function. The line segment between any two points on the graph lies entirely above the curve.

Recap: Gradients

A gradient of a function $f(\boldsymbol{\Theta})$ wrt parameters $\boldsymbol{\Theta} = [w_1, \ldots, w_P]$ is:

$$\nabla_{\boldsymbol{\Theta}} f(\boldsymbol{\Theta}) = \left[\frac{\partial}{\partial w_1} f, \ldots, \frac{\partial}{\partial w_P} f\right]$$

I.e., the vector of partial derivatives of f, which is the derivative of f wrt to each variable w_i

Our parameters are a matrix $\mathbf{W}$ and a vector $\boldsymbol{b}$, so the gradient has the same shape as the parameters: $\nabla_{\mathbf{W}} f$ is $|\mathcal{Y}| \times D$ and $\nabla_{\boldsymbol{b}} f$ is $|\mathcal{Y}| \times 1$[3]

Gradient = direction and fastest rate of increase of f at point $\boldsymbol{\Theta}$

When a gradient is zero we are at a stationary point of f. For convex functions that means global minima.

[3] Nothing changes if you stack all the entries of $\mathbf{W}$ and $\boldsymbol{b}$ into one long vector.

Recap: Iterative Descent Methods

Goal: find the minimum/minimizer of $f : \mathbb{R}^d \rightarrow \mathbb{R}$

- Proceed in **small steps** in the **optimal direction** till a **stopping criterion** is met (usually norm of gradient is small)
- **Gradient descent (GD)** updates: $\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta} - \eta \nabla_{\boldsymbol{\Theta}} f(\boldsymbol{\Theta})$
- η is the step size / learning rate

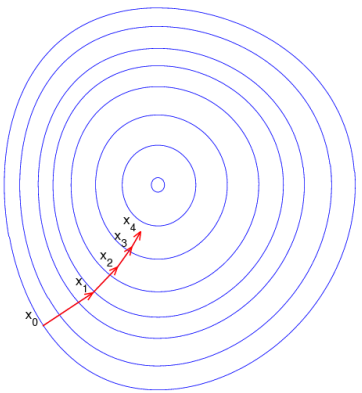


Figure: Illustration of gradient descent. The red lines correspond to steps taken in the negative gradient direction.

Logistic Regression: Gradient Descent (GD)

- Let $L(\boldsymbol{\Theta}; (\boldsymbol{x}, \boldsymbol{y})) = \left(\log \sum_{\boldsymbol{y}'} \exp(z_{\boldsymbol{y}'}) - z_{\boldsymbol{y}}\right)$, with $\boldsymbol{z} = \mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b}$
- Call this our loss function for instance $\boldsymbol{x}, \boldsymbol{y}$
  - We want to minimize over $\mathcal{D} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}$ with GD
  - I.e., Find $\arg\min_{\boldsymbol{\Theta}} \sum_{t=1} L(\boldsymbol{\Theta}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$
  - Logistic-regressions loss function often called log-loss or cross-entropy
- GD update will look like

$$\begin{aligned} \boldsymbol{\Theta} &= \boldsymbol{\Theta} - \eta \nabla_{\boldsymbol{\Theta}} \left(\sum_{t=1} L(\boldsymbol{\Theta}; (\boldsymbol{x}_t, \boldsymbol{y}_t))\right) \\ &= \boldsymbol{\Theta} - \eta \sum_{t=1} \nabla_{\boldsymbol{\Theta}} L(\boldsymbol{\Theta}; (\boldsymbol{x}_t, \boldsymbol{y}_t)) \end{aligned}$$

- Need to calculate $\nabla_{\boldsymbol{\Theta}} L(\boldsymbol{\Theta}; (\boldsymbol{x}, \boldsymbol{y}))$: gradient of L w.r.t. $\boldsymbol{\Theta}$
- This is a batch optimization: updates are over whole dataset

Stochastic Gradient Descent (SGD)

SGD is like perceptron – update every instance:

- Pick $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ randomly
- Update $\boldsymbol{\Theta} = \boldsymbol{\Theta} - \eta \nabla_{\boldsymbol{\Theta}} L(\boldsymbol{\Theta}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$

- i.e. we approximate the true gradient with a noisy, unbiased, gradient, based on a single sample
- Variants exist in-between (mini-batches)
- GD and SGD guaranteed to find the optimal $\boldsymbol{\Theta}$ (for suitable step sizes)

Logistic Regression: Simple SGD Algorithm

input: labeled data $\mathcal{D}$, step size η

initialize: $\mathbf{W} = \mathbf{0}$, $\boldsymbol{b} = \mathbf{0}$

repeat

observe example $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$

Update $\boldsymbol{\Theta} = \boldsymbol{\Theta} - \eta \nabla_{\boldsymbol{\Theta}} L(\boldsymbol{\Theta}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$

until stopping criterion

output: model parameters $\boldsymbol{\Theta} = (\mathbf{W}, \boldsymbol{b})$

- Picking step sizes example of hyperparameter tuning
- Stopping criterion usually gradient is small: $\|\nabla_{\boldsymbol{\Theta}} L(\boldsymbol{\Theta}; (\boldsymbol{x}_t, \boldsymbol{y}_t))\| < \epsilon$, $\forall t$
- Small (or zero) gradient is stationary point – global minimum

Computing the Gradient: $\nabla_{\Theta} L(\Theta; (x, y))$

- We need $\nabla_{\mathbf{W}} L$ and $\nabla_{\boldsymbol{b}} L$, where

$$L(\Theta; (x, y)) = \log \sum_{y'} \exp(z_{y'}) - z_y, \qquad \boldsymbol{z} = \mathbf{W}\phi(x) + \boldsymbol{b}$$

- Θ only touches L through the logits $\boldsymbol{z}$, so use the chain rule in two steps:
  1. $\nabla_{\boldsymbol{z}} L$ (how does the loss change with the scores?)
  2. $\boldsymbol{z} = \mathbf{W}\phi(x) + \boldsymbol{b}$ (how do the scores change with $\mathbf{W}$ and $\boldsymbol{b}$?)

- Some reminders:
  1. $\frac{\partial}{\partial u} \log F(u) = \frac{1}{F(u)} \frac{\partial}{\partial u} F(u)$
  2. $\frac{\partial}{\partial u} \exp F(u) = \exp(F(u)) \frac{\partial}{\partial u} F(u)$

Step 1: Gradient w.r.t. the Logits z

Take the partial derivative w.r.t. one logit z_k:

$$\begin{aligned}
\frac{\partial L}{\partial z_k} &= \frac{\partial}{\partial z_k}\left(\log \sum_{y'} \exp(z_{y'}) - z_y\right) \\
&= \frac{1}{\sum_{y'} \exp(z_{y'})} \cdot \frac{\partial}{\partial z_k} \sum_{y'} \exp(z_{y'}) - \frac{\partial z_y}{\partial z_k} \\
&= \frac{1}{Z_x} \sum_{y'} \exp(z_{y'}) \frac{\partial z_{y'}}{\partial z_k} - \frac{\partial z_y}{\partial z_k} \qquad \frac{\partial z_{y'}}{\partial z_k} = [\![y' = k]\!]\text{: only one term survives} \\
&= \frac{\exp(z_k)}{Z_x} - [\![k = y]\!] \;=\; P_\Theta(k|x) - [\![k = y]\!]
\end{aligned}$$

Stacking these for all k gives one clean vector:

$$\nabla_z L = p - e_y \qquad \text{(predicted distribution} - \text{the truth)}$$

where $p = \textbf{softmax}(z)$ and e_y is the one-hot vector of the correct label.

Step 2: From Logits to W and b

Each logit is $z_k = \boldsymbol{w}_k{}^\top \boldsymbol{\phi}(\boldsymbol{x}) + b_k$, so

$$\frac{\partial z_k}{\partial \mathbf{W}_{k,i}} = \boldsymbol{\phi}_i(\boldsymbol{x}), \qquad \frac{\partial z_k}{\partial b_k} = 1$$

Chain rule, entry by entry:

$$\frac{\partial L}{\partial \mathbf{W}_{k,i}} = \underbrace{\frac{\partial L}{\partial z_k}}_{p_k - [\![k=\boldsymbol{y}]\!]} \cdot \underbrace{\frac{\partial z_k}{\partial \mathbf{W}_{k,i}}}_{\boldsymbol{\phi}_i(\boldsymbol{x})} = (p_k - [\![k = \boldsymbol{y}]\!])\, \boldsymbol{\phi}_i(\boldsymbol{x})$$

Every entry at once is an outer product:

$$\nabla_{\mathbf{W}} L = (\boldsymbol{p} - \boldsymbol{e}_{\boldsymbol{y}})\, \boldsymbol{\phi}(\boldsymbol{x})^\top, \qquad \nabla_{\boldsymbol{b}} L = \boldsymbol{p} - \boldsymbol{e}_{\boldsymbol{y}}$$

- $\underbrace{(\boldsymbol{p} - \boldsymbol{e}_{\boldsymbol{y}})}_{|\mathcal{Y}| \times 1} \underbrace{\boldsymbol{\phi}(\boldsymbol{x})^\top}_{1 \times D}$ is $|\mathcal{Y}| \times D$ – the same shape as $\mathbf{W}$, as it must be

What does the update look like?

A single SGD step on example $(\boldsymbol{x}, \boldsymbol{y})$ is one outer product:

$$\mathbf{W} = \mathbf{W} - \eta\,(\boldsymbol{p} - \boldsymbol{e_y})\,\boldsymbol{\phi}(\boldsymbol{x})^\top, \qquad \boldsymbol{b} = \boldsymbol{b} - \eta\,(\boldsymbol{p} - \boldsymbol{e_y})$$

Written out, the outer product is an $|\mathcal{Y}| \times D$ matrix – one row per label:

$$(\boldsymbol{p} - \boldsymbol{e_y})\,\boldsymbol{\phi}(\boldsymbol{x})^\top = \begin{bmatrix} p_1\,\boldsymbol{\phi}(\boldsymbol{x})^\top \\ \vdots \\ (p_{\boldsymbol{y}} - 1)\,\boldsymbol{\phi}(\boldsymbol{x})^\top \\ \vdots \\ p_{|\mathcal{Y}|}\,\boldsymbol{\phi}(\boldsymbol{x})^\top \end{bmatrix}$$

row $\boldsymbol{y}$: the true label

- Row $\boldsymbol{y}$: $p_{\boldsymbol{y}} - 1 \leq 0$, so it moves towards $\boldsymbol{\phi}(\boldsymbol{x})$
- Every other row: $p_{\boldsymbol{y}'} \geq 0$, so it moves away from $\boldsymbol{\phi}(\boldsymbol{x})$
- Unlike the perceptron, every row moves – each by its own probability

SGD for Logistic Regression

input: labeled data $\mathcal{D}$, step size η

initialize: $\mathbf{W} = \mathbf{0}$, $\boldsymbol{b} = \mathbf{0}$

repeat

observe example $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$

$\boldsymbol{p} = \textbf{softmax}\left(\mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}_t) + \boldsymbol{b}\right)$

$\mathbf{W} = \mathbf{W} - \eta\left(\boldsymbol{p} - \boldsymbol{e}_{\boldsymbol{y}_t}\right)\boldsymbol{\phi}(\boldsymbol{x}_t)^\top$

$\boldsymbol{b} = \boldsymbol{b} - \eta\left(\boldsymbol{p} - \boldsymbol{e}_{\boldsymbol{y}_t}\right)$

until stopping criterion

output: model parameters $\boldsymbol{\Theta} = (\mathbf{W}, \boldsymbol{b})$

Logistic Regression Summary

- Define conditional probability

$$\boldsymbol{p} = \textbf{softmax}\left(\mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b}\right), \qquad P_{\boldsymbol{\Theta}}(\boldsymbol{y}|\boldsymbol{x}) = p_{\boldsymbol{y}}$$

- Set parameters to minimize negative conditional log-likelihood:

$$\boldsymbol{\Theta} = \arg\min_{\boldsymbol{\Theta}} \sum_t -\log P_{\boldsymbol{\Theta}}(\boldsymbol{y}_t|\boldsymbol{x}_t) = \arg\min_{\boldsymbol{\Theta}} \sum_t L(\boldsymbol{\Theta}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$$

- Gradient is an outer product: $\nabla_{\mathbf{W}} L = (\boldsymbol{p} - \boldsymbol{e}_{\boldsymbol{y}})\, \boldsymbol{\phi}(\boldsymbol{x})^\top$
- Can run gradient descent (or any gradient-based optimization algorithm)

The Story So Far

- Logistic regression is probabilistic: minimizes neg log-likelihood
  - also called log-linear model and max-entropy classifier
  - no closed form solution
  - For training instance $(\boldsymbol{x}, \boldsymbol{y})$, the SGD update is

$$\mathbf{W} = \mathbf{W} - \eta \left(\boldsymbol{p} - \boldsymbol{e}_{\boldsymbol{y}} \right) \boldsymbol{\phi}(\boldsymbol{x})^\top$$

- Perceptron is non-probabilistic; minimizes training errors
  - Assumes linearly separable, though works well anyways
  - For training instance $(\boldsymbol{x}, \boldsymbol{y})$, the update is

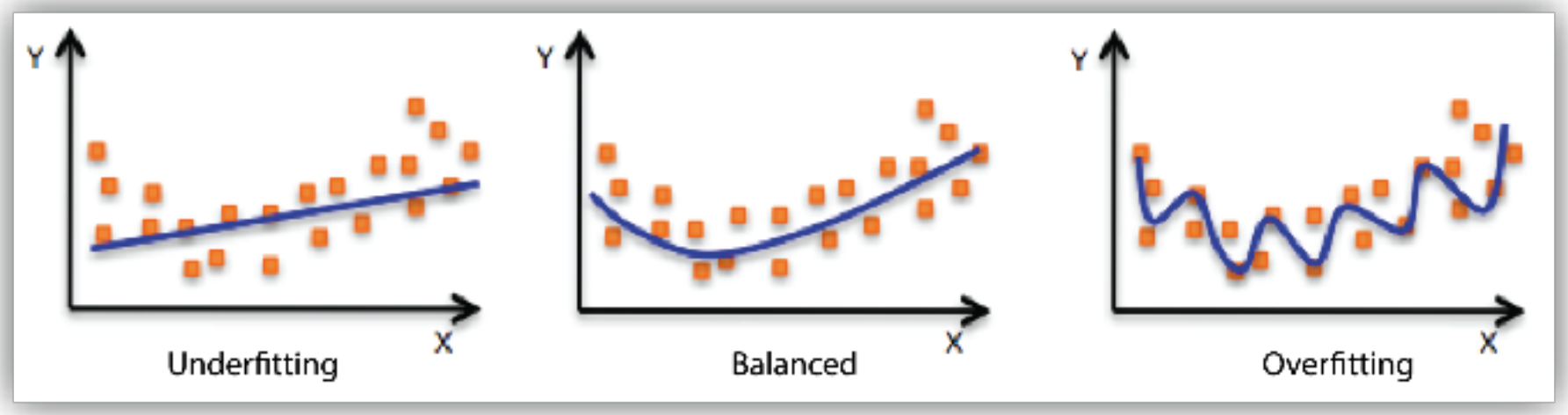
$$\mathbf{W} = \mathbf{W} - \left(\boldsymbol{e}_{\boldsymbol{y}'} - \boldsymbol{e}_{\boldsymbol{y}} \right) \boldsymbol{\phi}(\boldsymbol{x})^\top$$

Same outer product! The only difference: logistic regression uses the soft distribution $\boldsymbol{p}$ where the perceptron uses the hard prediction $\boldsymbol{e}_{\boldsymbol{y}'}$.

Outline

Overfitting

If the model is too complex (too many parameters) and the data is scarce, we run the risk of overfitting:



Regularization

In practice, we regularize models to prevent overfitting

$$\arg\min_{\boldsymbol{\Theta}} \sum_{t=1} L(\boldsymbol{\Theta}; (\boldsymbol{x}_t, \boldsymbol{y}_t)) + \lambda\Omega(\boldsymbol{\Theta}),$$

where $\Omega(\boldsymbol{\Theta})$ is the regularization function, and λ controls how much to regularize.

- Gaussian prior (ℓ_2), promotes smaller weights[4]:

$$\Omega(\boldsymbol{\Theta}) = \|\mathbf{W}\|_F^2 = \sum_{\boldsymbol{y}} \sum_{i} \mathbf{W}_{\boldsymbol{y},i}^2.$$

- Laplacian prior (ℓ_1), promotes sparse weights!

$$\Omega(\boldsymbol{\Theta}) = \|\mathbf{W}\|_1 = \sum_{\boldsymbol{y}} \sum_{i} |\mathbf{W}_{\boldsymbol{y},i}|$$

[4] $\|\cdot\|_F$ is the Frobenius norm: the sum of squares over both indices of the matrix.

Empirical Risk Minimization

$$\arg\min_{\Theta} \sum_{t=1} L(\Theta; (x_t, y_t)) + \lambda\Omega(\Theta),$$

- This formulation is generally called *Empirical Risk Minimization*.
- It consists of a Loss/Risk function that we want to minimize;
- And (optionally) a regularization function that stops us from overfitting to the data.
- Useful abstraction that covers most modern ML algorithms.
- Linear classifiers: different choices of L and Ω invoke specific models

Logistic Regression with ℓ_2 Regularization

- Still optimize with GD or SGD
- What is the new gradient?

$$\sum_{t=1} \nabla_{\mathbf{W}} L(\boldsymbol{\Theta}; (\boldsymbol{x}_t, \boldsymbol{y}_t)) + \nabla_{\mathbf{W}} \lambda \Omega(\boldsymbol{\Theta})$$

$$= \sum_{t=1} \nabla_{\mathbf{W}} L(\boldsymbol{\Theta}; (\boldsymbol{x}_t, \boldsymbol{y}_t)) + \nabla_{\mathbf{W}} \frac{\lambda}{2} \|\mathbf{W}\|_F^2$$

- We know $\nabla_{\mathbf{W}} L(\boldsymbol{\Theta}; (\boldsymbol{x}_t, \boldsymbol{y}_t)) = (\boldsymbol{p} - \boldsymbol{e}_{\boldsymbol{y}_t}) \, \boldsymbol{\phi}(\boldsymbol{x}_t)^\top$
- Just need $\nabla_{\mathbf{W}} \frac{\lambda}{2} \|\mathbf{W}\|_F^2 = \lambda \mathbf{W}$ – again the same shape as $\mathbf{W}$

Loss Function

Should match as much as possible the metric we want to optimize at test time

Should be well-behaved (continuous, maybe smooth) to be amenable to optimization (this rules out the 0/1 loss)

Some examples:

- Squared loss for regression
- Negative log-likelihood (cross-entropy): multinomial logistic regression
- Hinge loss: support vector machines
- A bunch more ...

Linear Classifier

Could not possible cover everything.
Please look at Andre Martins excellent lecture for LXMLS:

- `http://lxmls.it.pt/2019/LINEAR_LEARNERS.pdf`
- Also covers
  - Naive Bayes
  - Support Vector Machines (SVMs)
  - SVMs v Perceptron v Log Reg
  - Non-Linear Classifiers $\neq$ Neural Networks
    - K-Nearest neighbors (we'll do this)
    - Kernel methods (SVMs)

Outline

Recap: Sparse Feature Representations

- $\boldsymbol{x}$ is a name

$$\phi_0(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "George"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_1(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "Washington"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_2(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "Bridge"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_3(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "General"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_4(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains an unknown word} \\ 0 & \text{otherwise} \end{cases}$$

- $\boldsymbol{x}$=General George Washington $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = [1\ 1\ 0\ 1\ 0]^\top$
- $\boldsymbol{x}$=George Washington Bridge $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = [1\ 1\ 1\ 0\ 0]^\top$
- $\boldsymbol{x}$=George Washington University $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = [1\ 1\ 0\ 0\ 1]^\top$
- $\boldsymbol{x}$=George George George of the Jungle $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = [1\ 0\ 0\ 0\ 1]^\top$

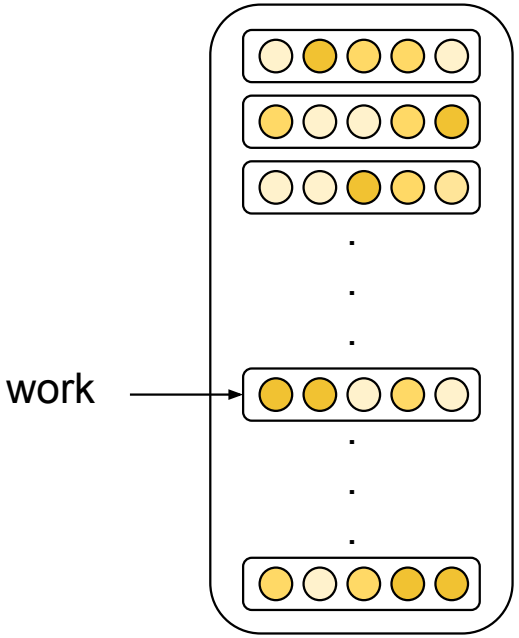
Dense Feature Representations

- $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^D$
- But $\boldsymbol{\phi}$ is dense real valued vector, i.e., zero values not frequent
- E.g.,

$$\boldsymbol{\phi}(\boldsymbol{x}) = [0.123, 0.439, -0.213, 0.692, -0.002]^\top$$

- But where does $\boldsymbol{\phi}(\boldsymbol{x})$ come from?
- We learn it from data!
- Long history: tf-idf, vector space models, ..., Word2Vec, Glove, ...

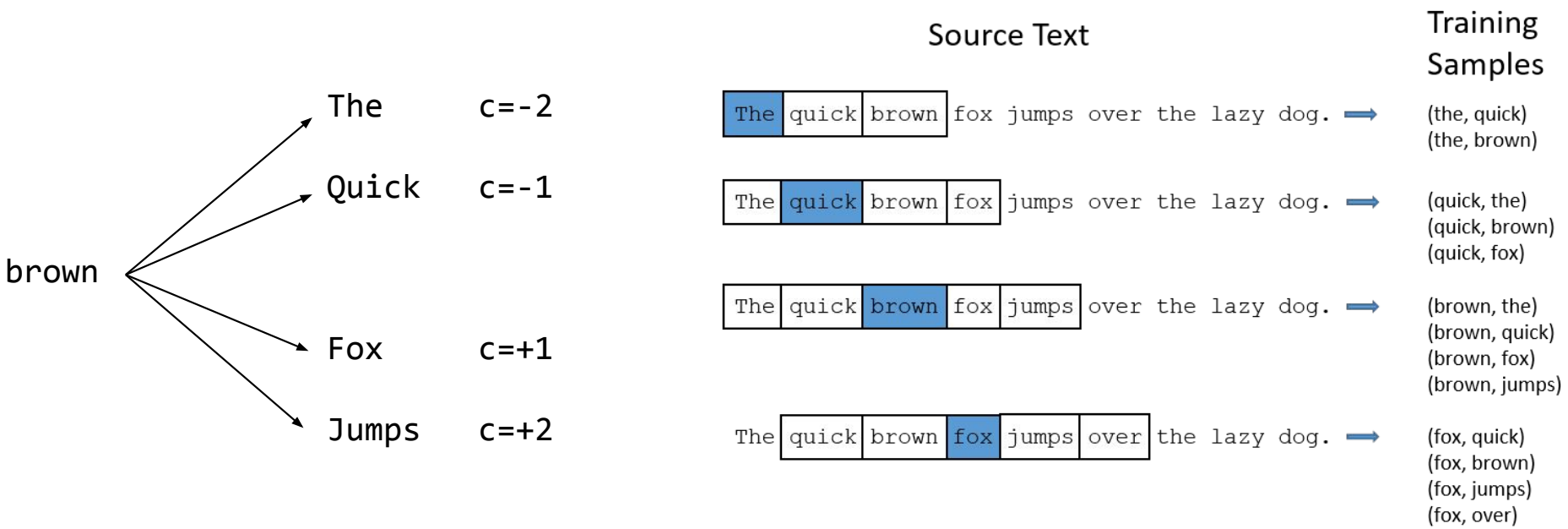
Embedding / Lookup Table



- Input is a word x with embedding $\phi(x) \in \mathbb{R}^D$, for all $x \in \mathcal{V}$
- $\mathcal{V}$ is a fixed vocabulary of words
- We store these in an embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times D}$ – one row per word
  - These are the model *word embeddings*
  - AKA embedding layer; word look-up table; ...
  - A lookup is just a matrix product: $\phi(x) = \mathbf{E}^\top \boldsymbol{e}_x$, with $\boldsymbol{e}_x$ the one-hot vector of x

Word2Vec

"You shall know a word by the company it keeps"



Example from McCormick `http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/`

Word2Vec

- Corpus $\mathcal{C} = \{\mathcal{X}_1, \ldots, \mathcal{X}_{|\mathcal{C}|}\}$
- With sentences $\mathcal{X} = x_1, \ldots, x_{|\mathcal{X}|}$
- Vocab $\mathcal{V} = \{x_i | x_i \in \mathcal{X} \text{ and } \mathcal{X} \in \mathcal{C}\}$
- Goal: learn vector/embedding for all $x_i \in \mathcal{V}$ (embedding table)

- **word2vec** (Mikolov et al., 2013)
  - Define vector/embedding per word: $\phi(x_i)$[5]
  - word2vec optimizes/maximizes (SkipGram model):

$$\sum_j^{|\mathcal{C}|} \sum_i^{|\mathcal{X}_j|} \sum_{-c \leq k \leq c, k \neq 0} \log p(x_{i+k} | x_i) = \sum_j^{|\mathcal{C}|} \sum_i^{|\mathcal{X}_j|} \sum_{-c \leq k \leq c, k \neq 0} \log \frac{e^{\phi(x_i)^\top \phi(x_{i+k})}}{\sum_{\boldsymbol{x}_l \in \mathcal{V}} e^{\phi(x_i)^\top \phi(x_l)}}$$

Maximize the probability word embedding can predict neighbours in some context window (of size c)

[5] Usually two embeddings used: word and word as context. Simplified here.

Word2Vec

Re-writing the equation:

$$\left(\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0} \log e^{\phi(x_j)^\top \phi(x_{j+k})}\right) - \left(\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0} \log \sum_{x_l\in\mathcal{V}} e^{\phi(x_j)^\top \phi(x_l)}\right)$$

- On the left: Sum over positive contexts
- On the right: Sum over negative contexts
  - Not feasible to sum over entire vocabulary
- Solution: negative sampling

$$\left(\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0} \log e^{\phi(x_j)^\top \phi(x_{j+k})}\right) - \left(\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0} \log \sum_{x_l\in\mathcal{V}_s} e^{\phi(x_j)^\top \phi(x_l)}\right)$$

- $\mathcal{V}_s$ is randomly sampled, i.e., $\mathcal{V}_s \subset \mathcal{V}$ and $|\mathcal{V}_s| << |\mathcal{V}|$ (often 1)

Word2Vec

$$\left(\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0}\log e^{\phi(x_j)^\top \phi(x_{j+k})}\right) - \left(\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0}\log \sum_{x_l\in\mathcal{V}_s} e^{\phi(x_j)^\top \phi(x_l)}\right)$$

- $\phi(x_i)$ are used as final word embeddings
- Usually optimized with SGD[6]

[6] Negate function to make it a loss and minimize.

Varibable Length Inputs

- What is input $\boldsymbol{x}$ is not just a single word?
- Or $\boldsymbol{x}$ is not of fixed length, e.g., sentences or documents?
- E.g., $\boldsymbol{x} = x_1 \ldots x_n$

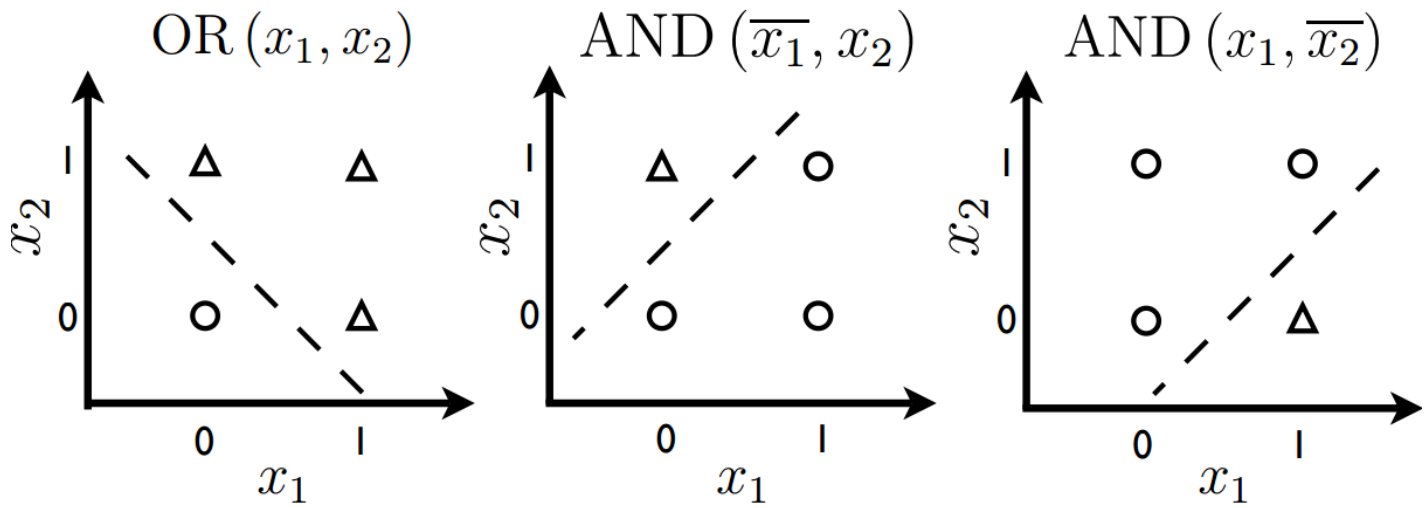
COMMON SOLUTIONS

- Average: $\boldsymbol{\phi}(\boldsymbol{x}) = \frac{1}{|\boldsymbol{x}|} \sum_{x \in \boldsymbol{x}} \boldsymbol{\phi}(x)$
  - Other pointwise operations, e.g., max, sum, ...
- Truncation+concatentation: $\boldsymbol{\phi}(\boldsymbol{x}) = [\boldsymbol{\phi}(x_1); \ldots; \boldsymbol{\phi}(x_k)] \in \mathbb{R}^{kD}$ for a fixed k (stack the column vectors)
- Sparse-dense: Whole look-up table is input, but
  - Zero out rows of words that are not present
  - Usually not practical

Outline

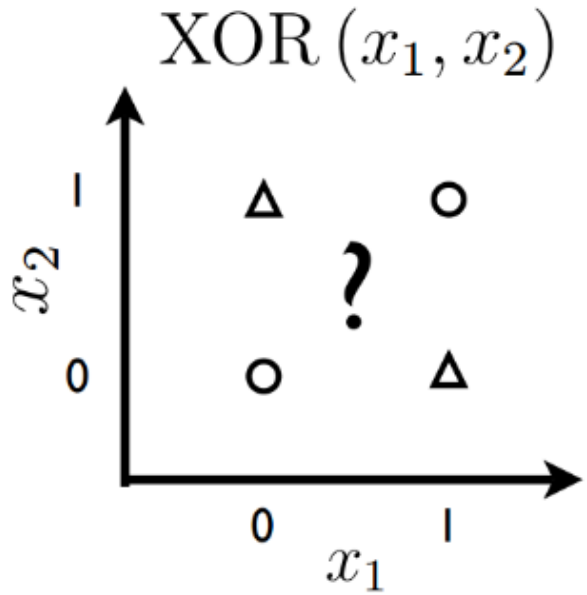
Recap: What a Linear Classifier Can Do

- It **can** solve linearly separable problems (OR, AND)



Recap: What a Linear Classifier Can't Do

- ... but it **can't** solve non-linearly separable problems such as simple XOR:



What If Data Are Not Linearly Separable?

What If Data Are Not Linearly Separable?

Engineer better features (often works!)

What If Data Are Not Linearly Separable?

Engineer better features (often works!)

Similarity-based / Kernel methods:

- define a similarity / kernel function between points
- use it to classify new instances; need a good function

What If Data Are Not Linearly Separable?

Engineer better features (often works!)

Similarity-based / Kernel methods:

- define a similarity / kernel function between points
- use it to classify new instances; need a good function

Neural networks (up next)

- embrace non-convexity and local minima

Two Views of Machine Learning

There's two big ways of building machine learning systems:

1. Feature-based: describe objects' properties (features) and build models that manipulate them
   - everything that we have seen so far.
2. Similarity-based: don't describe objects by their properties; rather, build systems based on **comparing** objects to each other
   - k-th nearest neighbors; kernel methods; Gaussian processes.

Sometimes the two are equivalent!

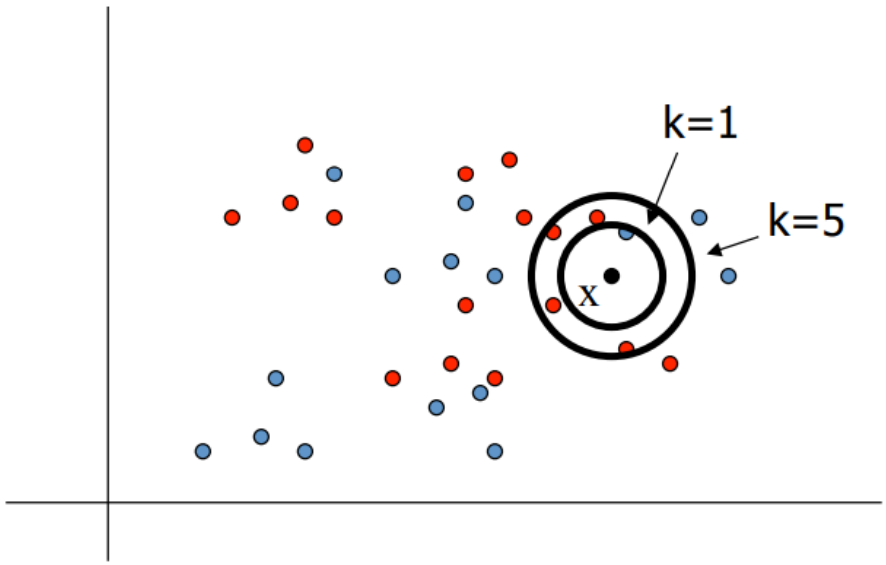
Parametric vs. Non-parametric

Another way to classify machine learning systems:

1. Parametric: Fix the number of parameters, model structure and hypothesis space
   - Goal: find parameters to optimize objective in the hypothesis space
   - Everything so far plus NNs
2. Non-parametric: No/little assumptions about form of solution; size of parameters not fixed
   - Goal: find the function/solution to best fit data
   - Similarity methods (k-th nearest neighbors); kernel methods; decision trees, random forests, gaussian processes.

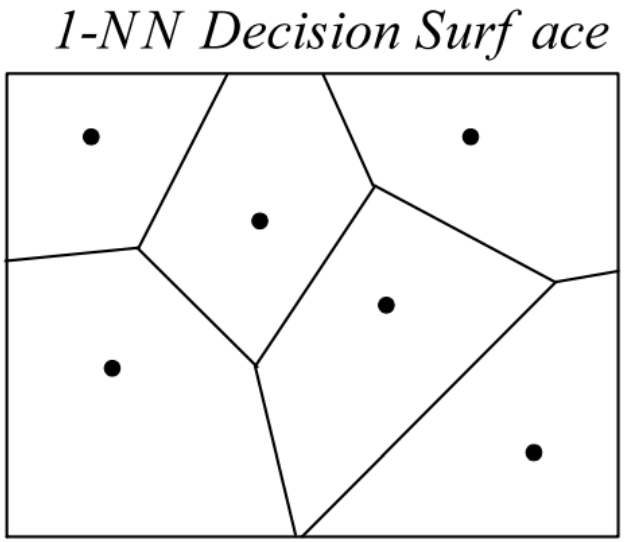
(K-th) Nearest Neighbor Classifier

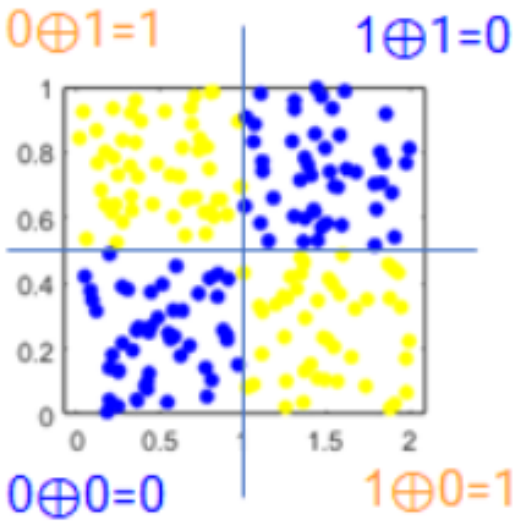
- Memorize (training) data $\mathcal{D} = \{\boldsymbol{x}_t, \boldsymbol{y}_t\}_{t=1}$
- To classify a new datapoint $\boldsymbol{x}$
  - Find the k closest data points in the data (e.g., training set)
  - Assign the most frequent class in those k points



(K-th) Nearest Neighbor Classifier

- We are not learning any parameters
- Hypothesis space can change by adding more data points
- Power is greater than linear classifier (xor on right)





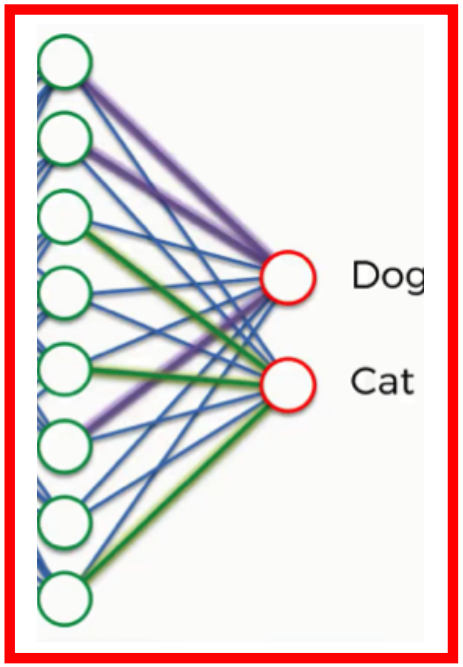
Similarity Functions and Inference

- Common Similarity functions
  - Euclidean: $\|\boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\phi}(\boldsymbol{x}')\| = \sqrt{\sum_i (\boldsymbol{\phi}_i(\boldsymbol{x}) - \boldsymbol{\phi}_i(\boldsymbol{x}'))^2}$
  - Inner product: $\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')$
  - Cosine: $\frac{\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')}{||\boldsymbol{\phi}(\boldsymbol{x})||\ ||\boldsymbol{\phi}(\boldsymbol{x}')||}$
  - Function can also be learned!

- Searching a K-NN database
  - Dense retrieval! Used in RAG, search, etc.
  - Brute-force
  - Branch and bound / k-d tree
  - Approx: Greedy proximity graph; locality sensitive hashing

Outline

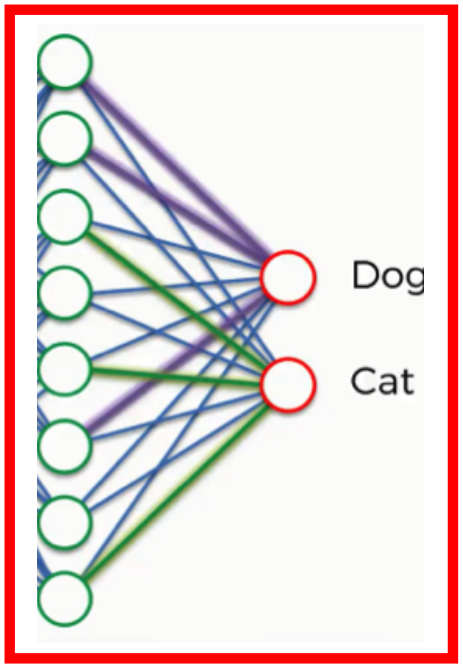
Recap: Weights and Biases



Linear Classifier

$$\boldsymbol{z} = \mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b}, \quad \boldsymbol{y}' = \textbf{argmax}(\boldsymbol{z}), \; \mathbf{W} = \begin{bmatrix} \vdots \\ \boldsymbol{w}_{\boldsymbol{y}}^{\top} \\ \vdots \end{bmatrix}, \; \boldsymbol{b} = \begin{bmatrix} \vdots \\ \boldsymbol{b}_{\boldsymbol{y}} \\ \vdots \end{bmatrix}$$

Recap: Weights and Biases

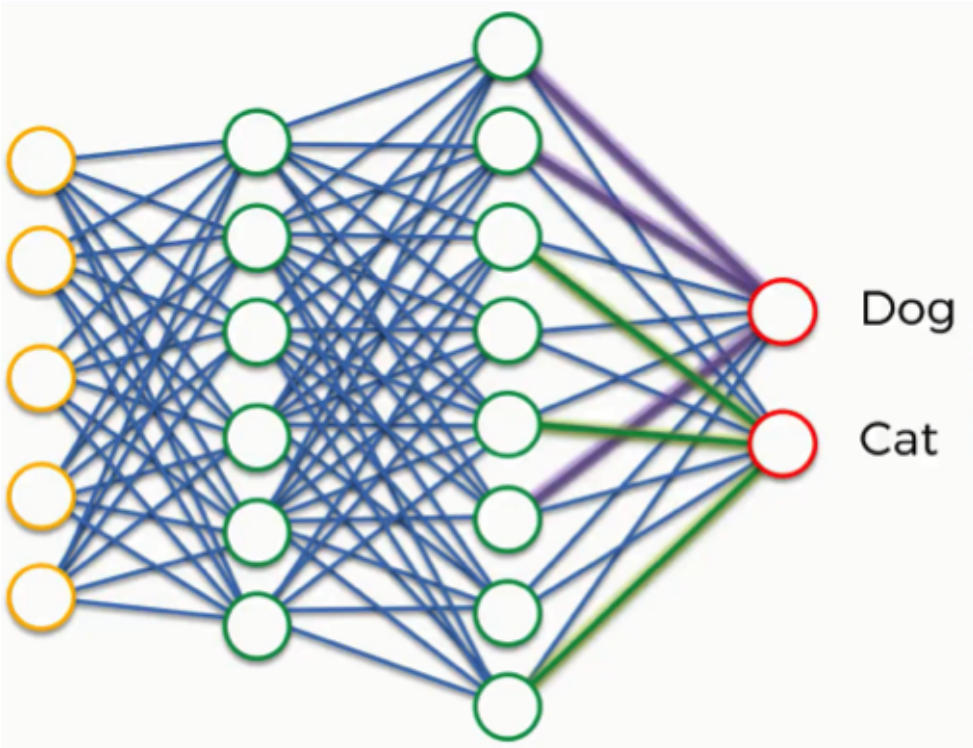


Linear Classifier

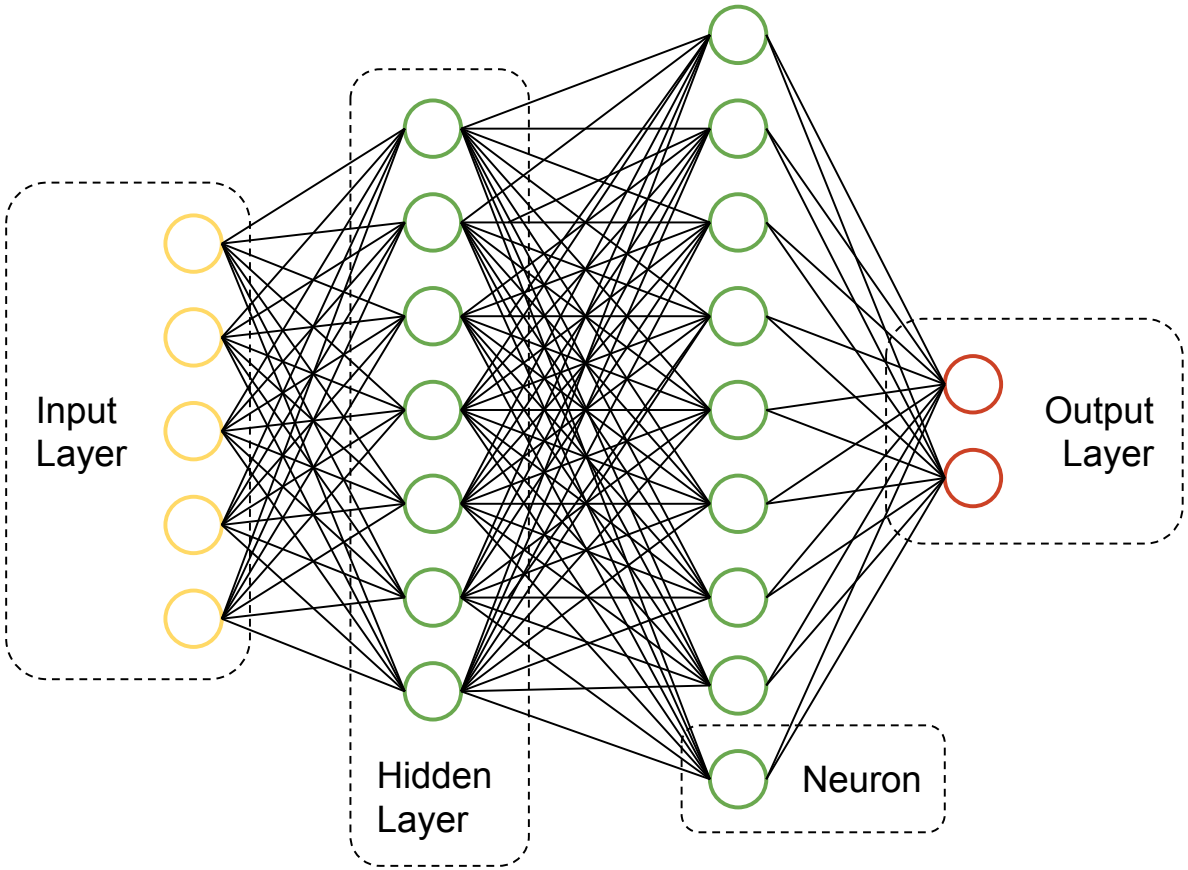
$$\boldsymbol{z} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}, \quad \boldsymbol{y}' = \textbf{argmax}(\boldsymbol{z}), \; \mathbf{W} = \begin{bmatrix} \vdots \\ \boldsymbol{w}_{\boldsymbol{y}}^{\top} \\ \vdots \end{bmatrix}, \; \boldsymbol{b} = \begin{bmatrix} \vdots \\ b_{\boldsymbol{y}} \\ \vdots \end{bmatrix}$$

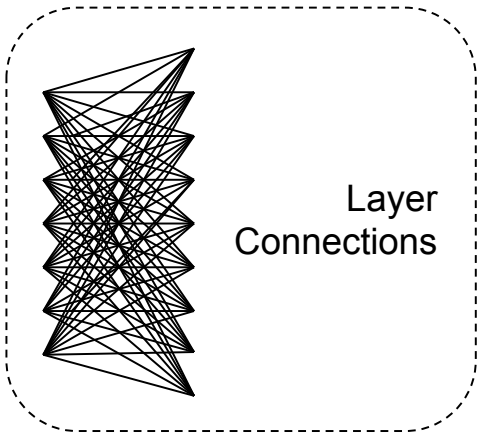
Notation: input is now a dense vector $\boldsymbol{x}$ instead of sparse features $\boldsymbol{\phi}(\boldsymbol{x})$.

On to neural networks!



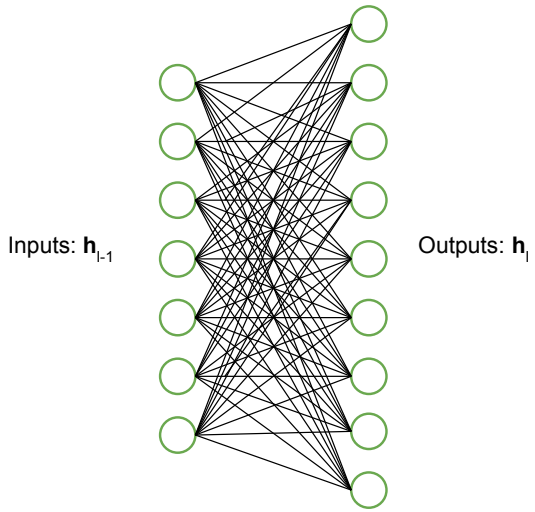
Neurons, Layers and Connections





- A (dense / fully-connected) feed-forward neural network (FF-NN)
  - AKA fully connected network (FCN) / multilayer perceptron (MLP)
- Input and output layers are special (more on this)
- However connections between layers take a similar form

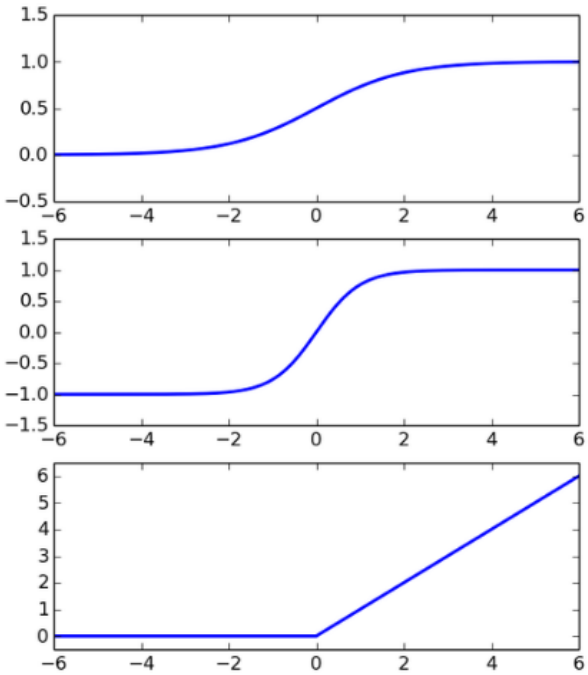
Hidden Layer Connections



- Let $\boldsymbol{h}_l \in \mathbb{R}^{D_l}$ be the l^{th} hidden layer with D_l dimensions/neurons
- $\boldsymbol{h}_0 = \boldsymbol{x}$ the input
- $\boldsymbol{h}_l = \sigma_l(\mathbf{W}_l \boldsymbol{h}_{l-1} + \boldsymbol{b}_l)$ $\leftarrow$ same form as the linear classifier
- $\mathbf{W}_l \in \mathbb{R}^{D_l \times D_{l-1}}$ and $\boldsymbol{b}_l \in \mathbb{R}^{D_l}$ are layer parameters
- σ_l is the layer's (non-linear) activation function
- Row j of $\mathbf{W}_l$ = the weights of one neuron (instead of one label)

Activation Functions

- Non-linearity by transforming/projecting the data
- Squashes output to finite range
- Examples ...



Sigmoid

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

Hyperbolic Tangent

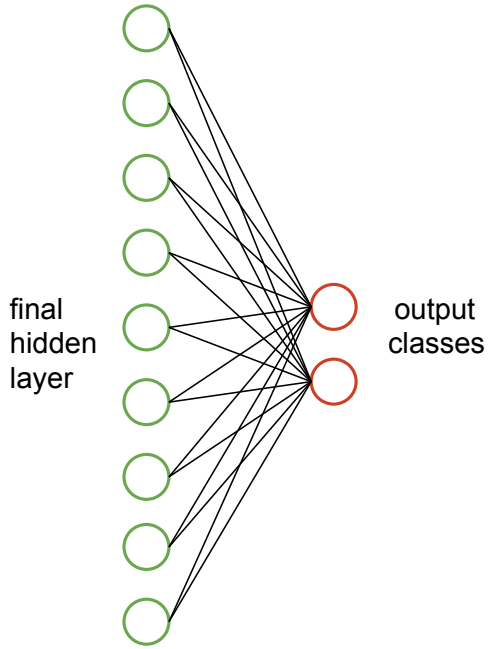
$$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Rectified Linear

$$\phi(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

From Hughes and Correll 2016

Output Layer



- This was in the last lecture
- $\boldsymbol{z} = \mathbf{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L; \quad \boldsymbol{y}' = \mathbf{argmax}(\boldsymbol{z})$
- $z_{\boldsymbol{y}}$ is the logit: the raw score of label $\boldsymbol{y}$

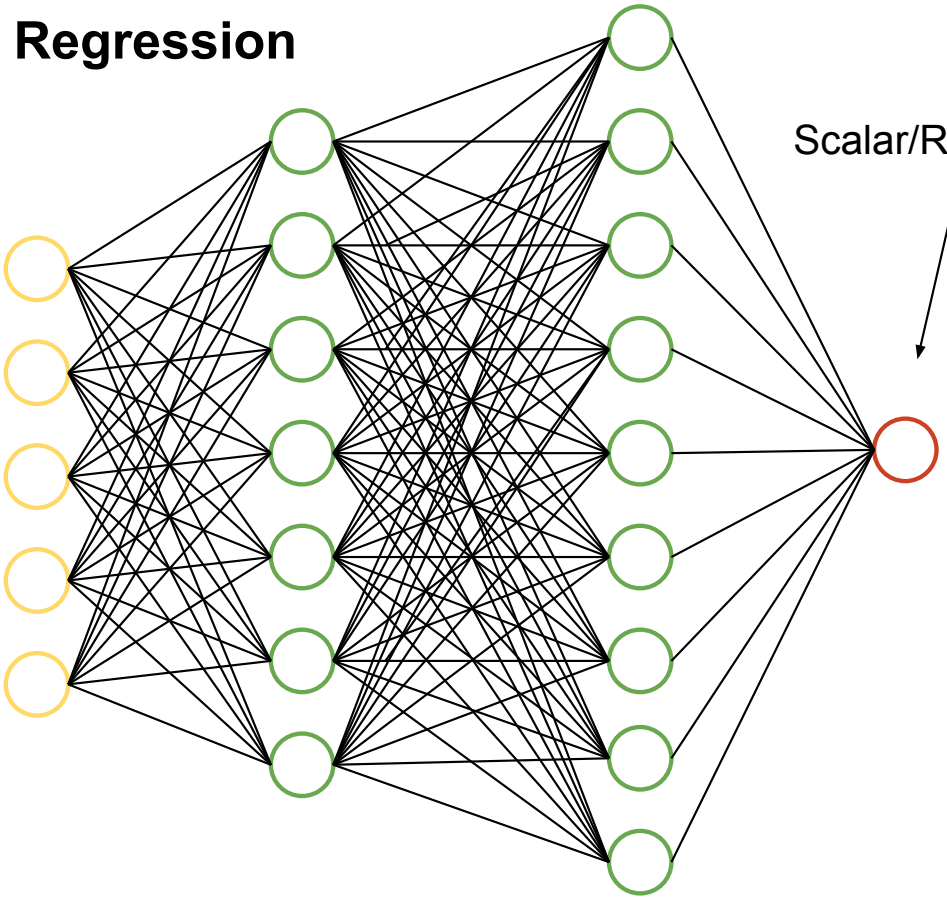
- Training?
  - Common paradigm: take the softmax of the logits, $\boldsymbol{p} = \mathbf{softmax}(\boldsymbol{z})$
  - Gives probability distribution $P(\boldsymbol{y}|\boldsymbol{x}) = p_{\boldsymbol{y}}, \forall p_{\boldsymbol{y}} \in \boldsymbol{p}$
  - Minimize some loss

Example NN Loss: Cross-Entropy

- Can use any differentiable loss function.
- Let $\Theta = \{(\mathbf{W}_l, \boldsymbol{b}_l)\}$, i.e., parameters for all layers l
- Goal: Find Θ to maximize $P(\boldsymbol{y}|\boldsymbol{x}; \Theta)$ for all $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}$
- Cross-Entropy (CE)
  - Intuition: Avg. bits needed to distinguish two distributions
  - Let $\overline{P}$ be the truth: $\overline{P}(\boldsymbol{y}'|\boldsymbol{x}_t) = 1$ if $\boldsymbol{y}' = \boldsymbol{y}_t$ and 0 otherwise
  - CE $= -E_{\overline{P}} \log P = -\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \overline{P}(\boldsymbol{y}|\boldsymbol{x}) \log P(\boldsymbol{y}|\boldsymbol{x}; \Theta)$
  - Therefore: $CE = -\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \log P(\boldsymbol{y}|\boldsymbol{x}; \Theta)$
- Min of $CE = -\sum_{(\boldsymbol{x},\boldsymbol{y})} \log P(\boldsymbol{y}|\boldsymbol{x}; \Theta)$
- Cross-entropy = Min Neg Log-likelihood / log-loss
- In other words: logistic regression!

Examples: NN for non-classification tasks

Regression

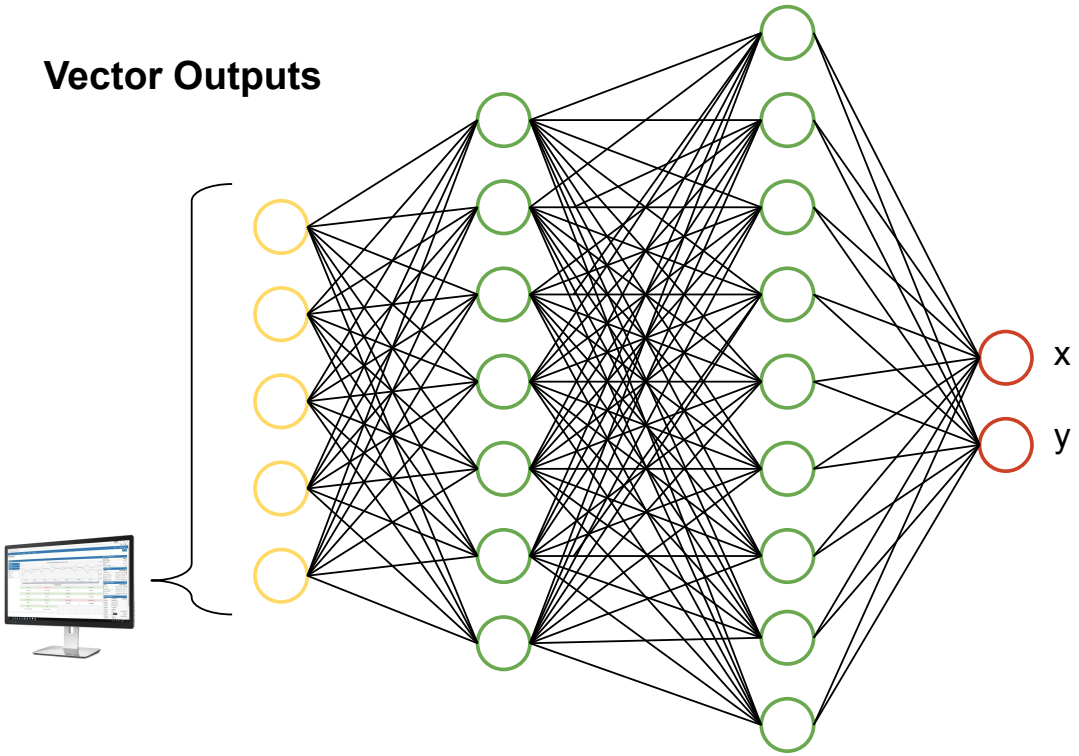


Scalar/Real-value

Loss Function
Mean-Squared Error (MSE)

$$MSE - \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Examples: NN for non-classification tasks



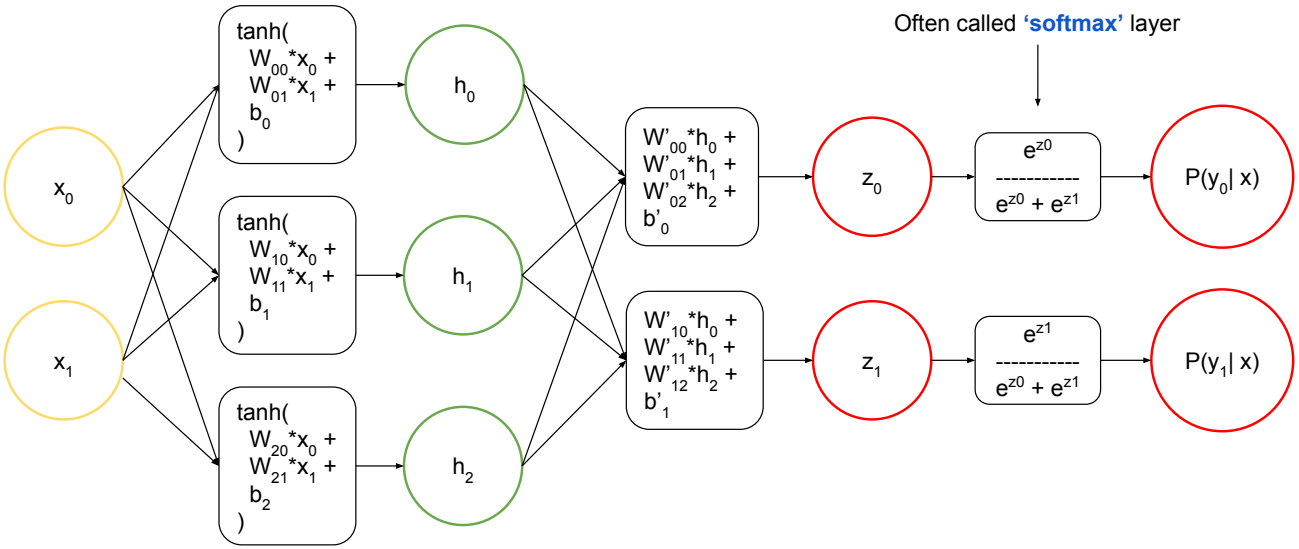
Example: Next Mouse Click

Predict: x-y coordinates

Loss: Euclidean Distance

A Wee Example

- $\boldsymbol{x} \in \mathbb{R}^2$
- $\boldsymbol{h} = \tanh(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$ with $\mathbf{W} \in \mathbb{R}^{3\times 2}$ and $\boldsymbol{b} \in \mathbb{R}^3$
- $|\mathcal{Y}| = 2$ with $\boldsymbol{z} = \mathbf{W}'\boldsymbol{h} + \boldsymbol{b}'$, $\mathbf{W}' \in \mathbb{R}^{2\times 3}$ and $\boldsymbol{b}' \in \mathbb{R}^2$
- Cross-entropy:
  - $L(\boldsymbol{\Theta}; (\boldsymbol{x}, \boldsymbol{y})) = -\log(P(\boldsymbol{y}|\boldsymbol{x})) = -\log \frac{e^{z_{\boldsymbol{y}}}}{\sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{z_{\boldsymbol{y}'}}}$



Neural Networks So Far

- Neural network structure (FF-NN; FCN; MLP)
  - Input layer: for now, assume given to us $\boldsymbol{x} \in \mathbb{R}^D$
  - Outputs: $\boldsymbol{y} \in \mathcal{Y}$
  - Hidden layers: $\boldsymbol{h}_l \in \mathbb{R}^{D_l}$; with $\boldsymbol{h}_l = \sigma_l(\mathbf{W}_l \boldsymbol{h}_{l-1} + \boldsymbol{b}_l)$
    - Thus, model parameters $\boldsymbol{\Theta} = \{\mathbf{W}_l, \boldsymbol{b}_l \mid \forall l\}$
    - Including the last (output) layer, $\boldsymbol{z} = \mathbf{W}_L \boldsymbol{h}_{L-1} + \boldsymbol{b}_L$
  - Loss function: $L(\boldsymbol{\Theta}; (\boldsymbol{x}, \boldsymbol{y}))$ – usually log-loss/cross-entropy

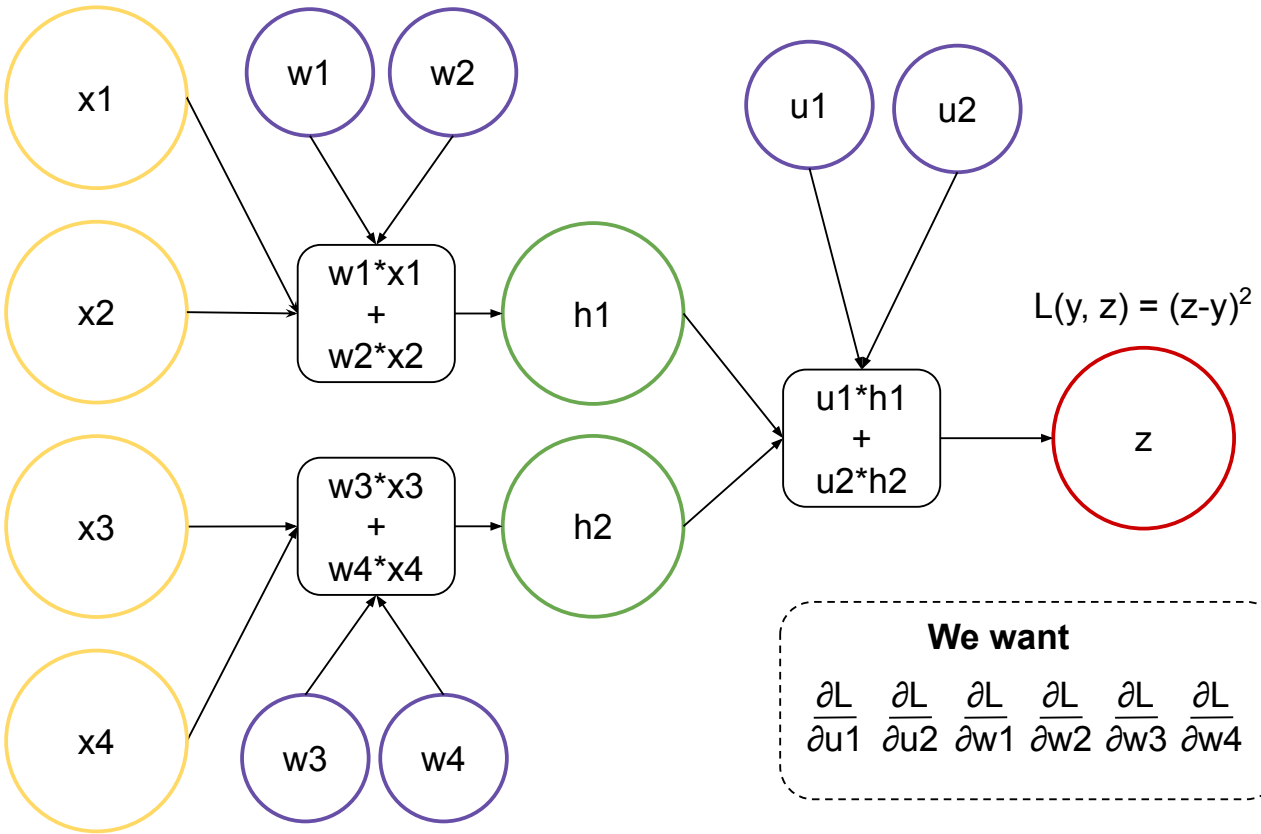
Neural Networks: Optimization

- Hidden layers make model non-convex!
- No single global optimum. Must settle for a local one.
- If loss function and activation functions are differentiable, then can be optimized with gradient-based techniques (e.g., gradient descent)
- Gradient computation a little trickier
  - Solution: backpropagation (Rumelhart et al., 1988)

Backpropagation and the Chain Rule

- We need to compute $\nabla_{\boldsymbol{\Theta}} L(\boldsymbol{\Theta}; \mathcal{D}) = [\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots], \forall w_i \in \boldsymbol{\Theta}$
  - For linear classifiers, $\boldsymbol{\Theta}$ was a single $(\mathbf{W}, \boldsymbol{b})$
  - For NNs, $\boldsymbol{\Theta}$ is one $(\mathbf{W}_l, \boldsymbol{b}_l)$ per layer: $\boldsymbol{\Theta} = \{\mathbf{W}_l, \boldsymbol{b}_l \mid \forall l\}$
  - Each $\nabla_{\mathbf{W}_l} L$ has the same shape as $\mathbf{W}_l$
- Chain rule: $\frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$

Toy Example: Analytical Partial Derivatives



We want

$$\frac{\partial L}{\partial u1} \quad \frac{\partial L}{\partial u2} \quad \frac{\partial L}{\partial w1} \quad \frac{\partial L}{\partial w2} \quad \frac{\partial L}{\partial w3} \quad \frac{\partial L}{\partial w4}$$

All base derivatives

$$\frac{\partial L}{\partial z} = 2(z-y)$$

$$\frac{\partial z}{\partial h1} = u1 \qquad \frac{\partial z}{\partial h2} = u2$$

$$\frac{\partial z}{\partial u1} = h1 \qquad \frac{\partial z}{\partial u2} = h2$$

$$\frac{\partial h1}{\partial w1} = x1 \qquad \frac{\partial h1}{\partial w2} = x2$$

$$\frac{\partial h1}{\partial x1} = w1 \qquad \frac{\partial h1}{\partial x2} = w2$$

$$\frac{\partial h2}{\partial w3} = x3 \qquad \frac{\partial h2}{\partial w4} = x4$$

$$\frac{\partial h2}{\partial x3} = w3 \qquad \frac{\partial h2}{\partial x4} = w4$$

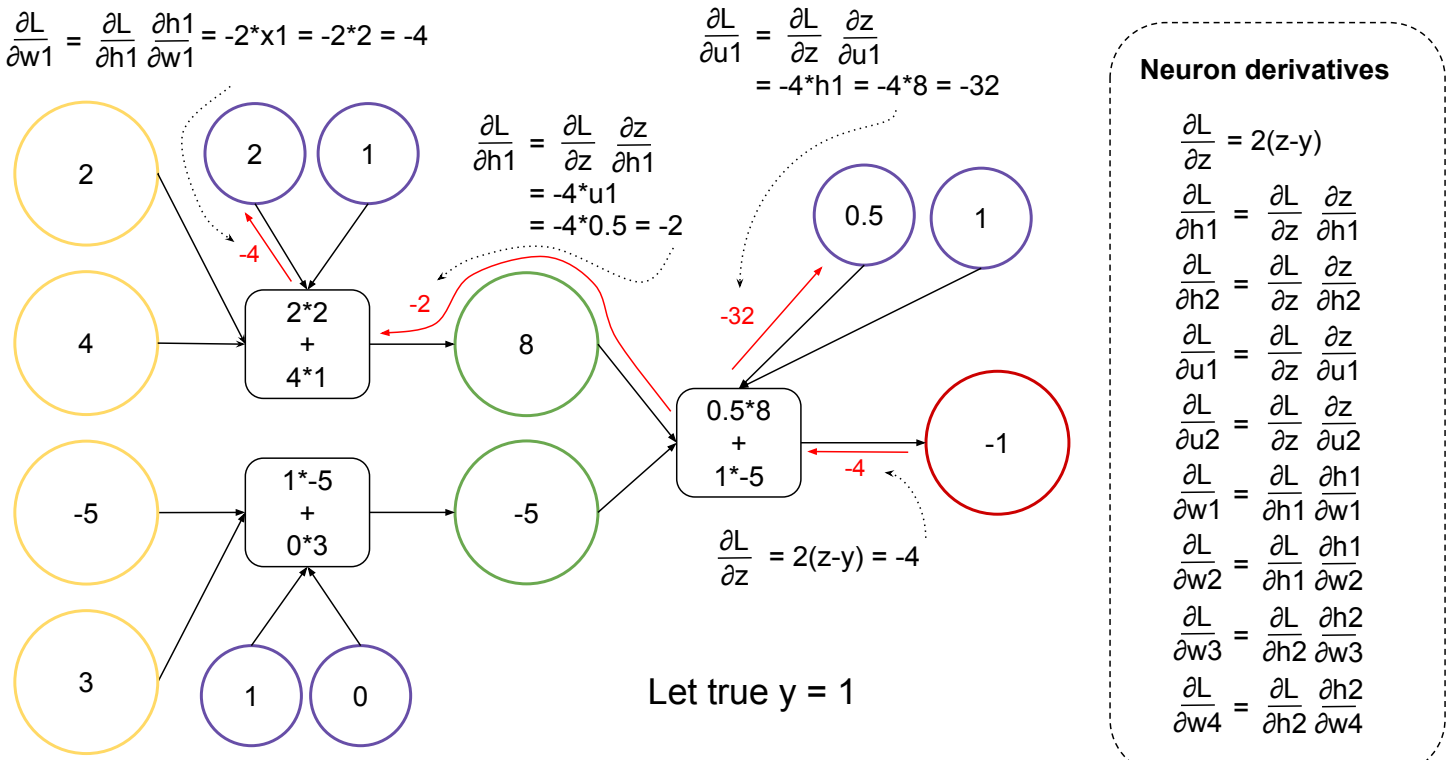
Full derivation examples

$$\frac{\partial L}{\partial u1} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial u1} = 2(z-y)*h1$$

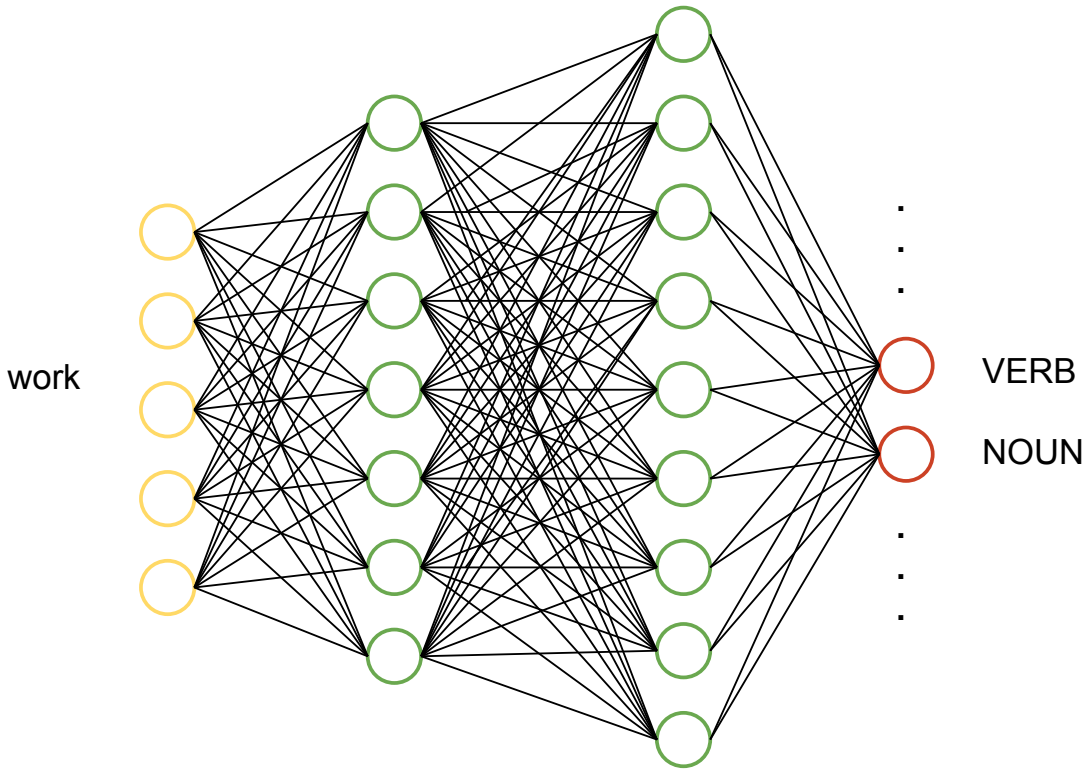
$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial h1}\frac{\partial h1}{\partial w1} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial h1}\frac{\partial h1}{\partial w1} = 2(z-y)*u1*x1$$

Toy Example: Backpropagation at Work

- Analytically computing chain rule in deep networks is onerous
- Backpropagation
  - Forward pass: compute values at neurons and final loss
  - Backward pass: compute $\frac{\partial L}{\partial w_i}$ at each neuron
  - $\frac{\partial L}{\partial w_i}$ of parameter neurons form gradient



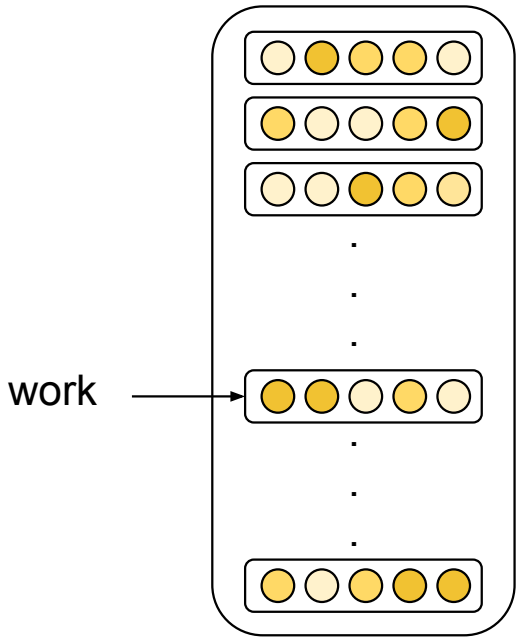
Input layer



- Consider classifying a word in isolation with a part-of-speech tag[7]
- The input is a word from a fixed, finite vocabulary $\mathcal{V}$
- But the network needs a vector: $\boldsymbol{h}_0 = \boldsymbol{x} \in \mathbb{R}^D$

[7] This is contrived. We usually use context.

Input layer = Embedding layer

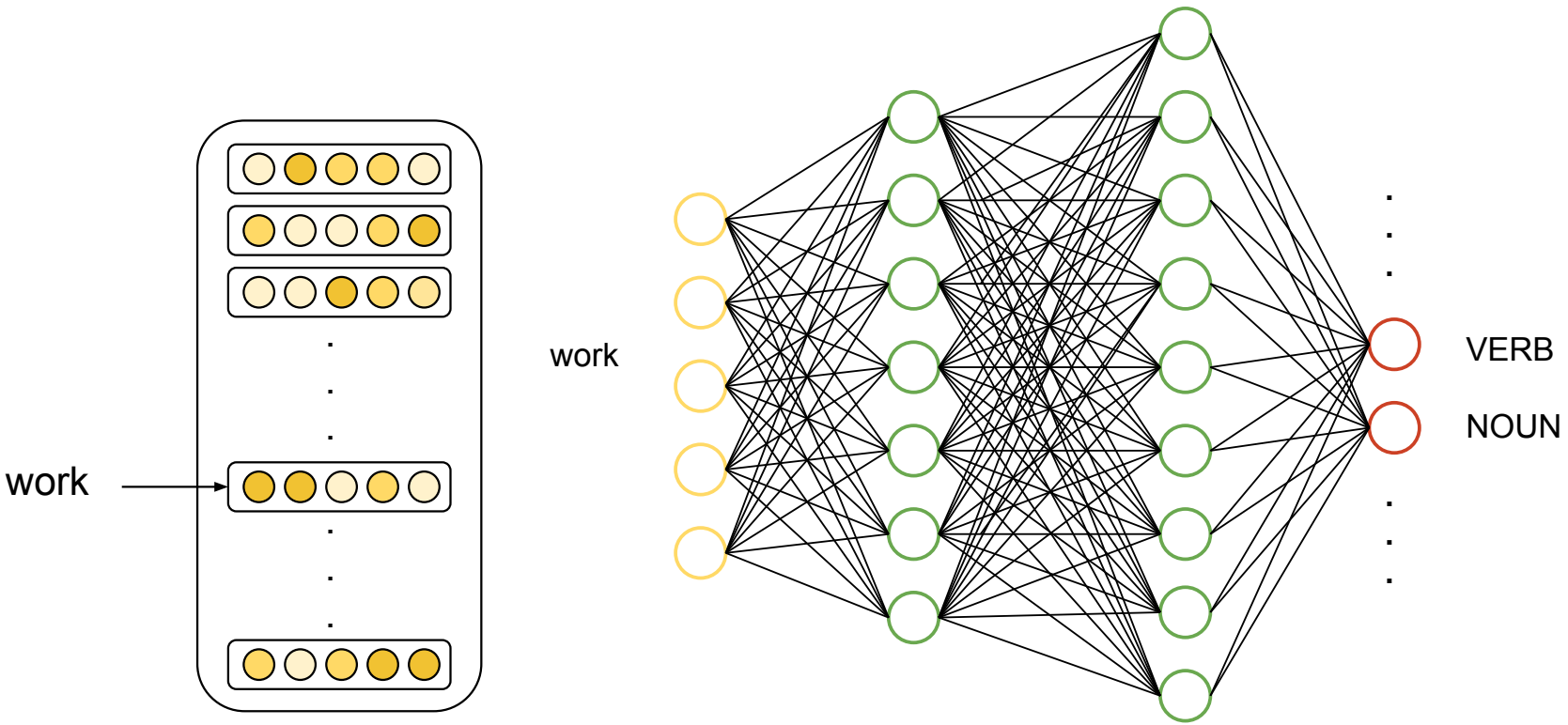


- Each word in $\mathcal{V}$ gets a vector in $\mathbb{R}^D$
- We store these in the embedding matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{V}| \times D}$ – one row per word
  - These are the model *word embeddings*
  - AKA embedding layer; word look-up table; ...
  - The lookup is again a matrix product: $\boldsymbol{h}_0 = \mathbf{E}^\top \boldsymbol{e}_x$

Input layer = Embedding layer

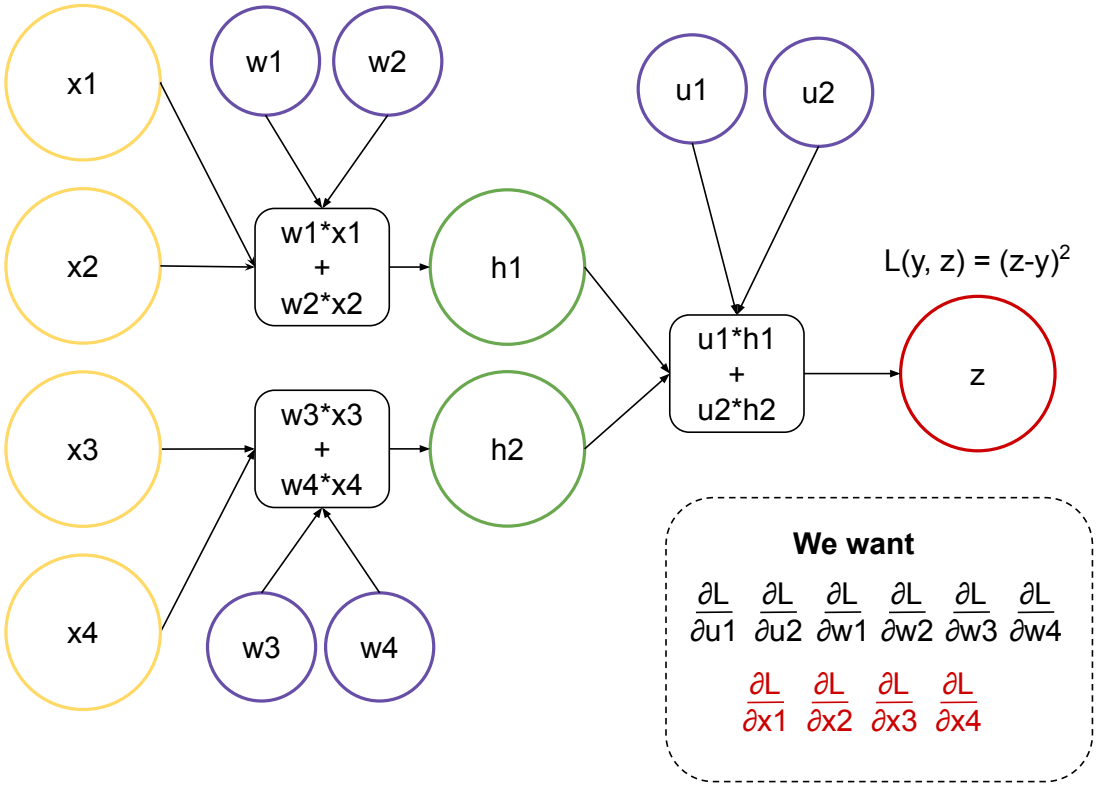
- Static embedding layer
  - Fixed word embeddings; not updated during training
  - Examples: SVD; word2vec; glove; ...
- Dynamic embedding layer
  - Randomly initialize word embeddings
  - Learn during training of the full network
  - Updated like any other layer during backpropagation
- Static + Dynamic
  - Initialize model with static embeddings; update dynamically
  - Combination: part of embedding layer is static; part is learned

Input layer



- Static (e.g., word2vec) or dynamic word embeddings give us input layer

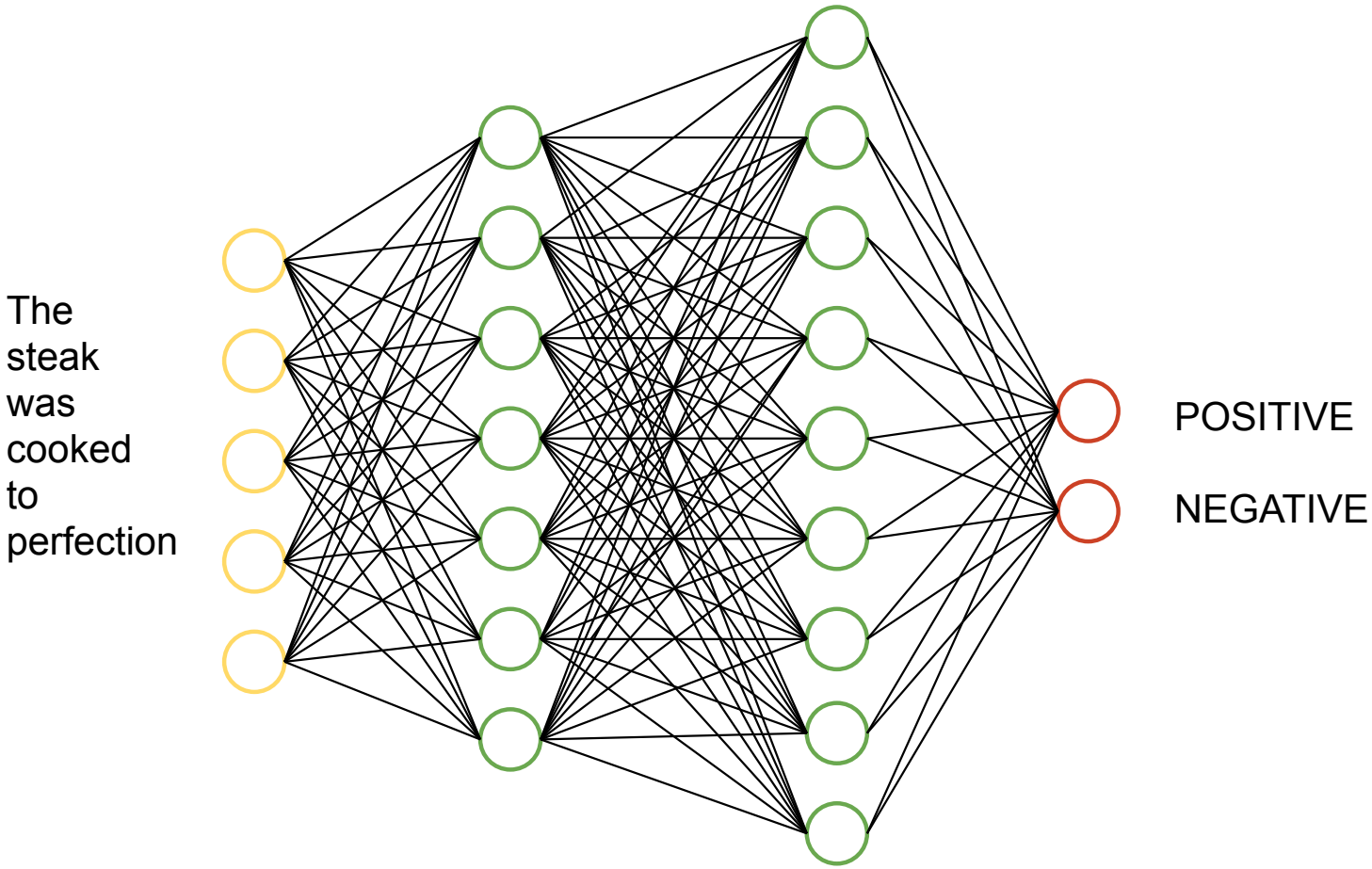
Dynamic Input layer



- Gradient now includes the input neurons, $\frac{\partial L}{\partial \boldsymbol{x}_i}$, and hence $\nabla_{\mathbf{E}} L$
- Every value in the entire lookup table is a parameter: $\boldsymbol{\Theta} = \{\mathbf{E}\} \cup \{\mathbf{W}_l, \boldsymbol{b}_l \mid \forall l\}$

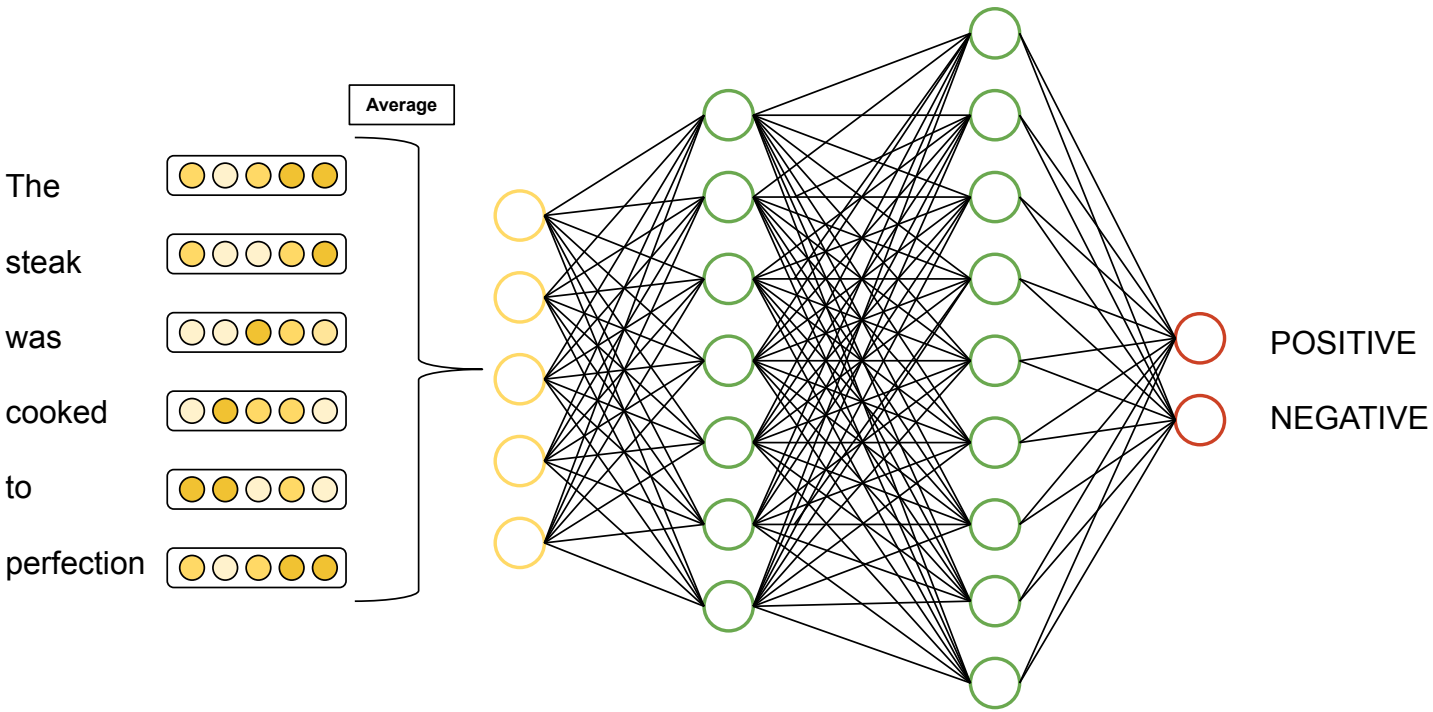
Variable Length Inputs

- But what if input is a whole document and not just a single word?
- Feed-forward neural networks assume a fixed-length input, $\boldsymbol{x} \in \mathbb{R}^D$
- Documents are not fixed length



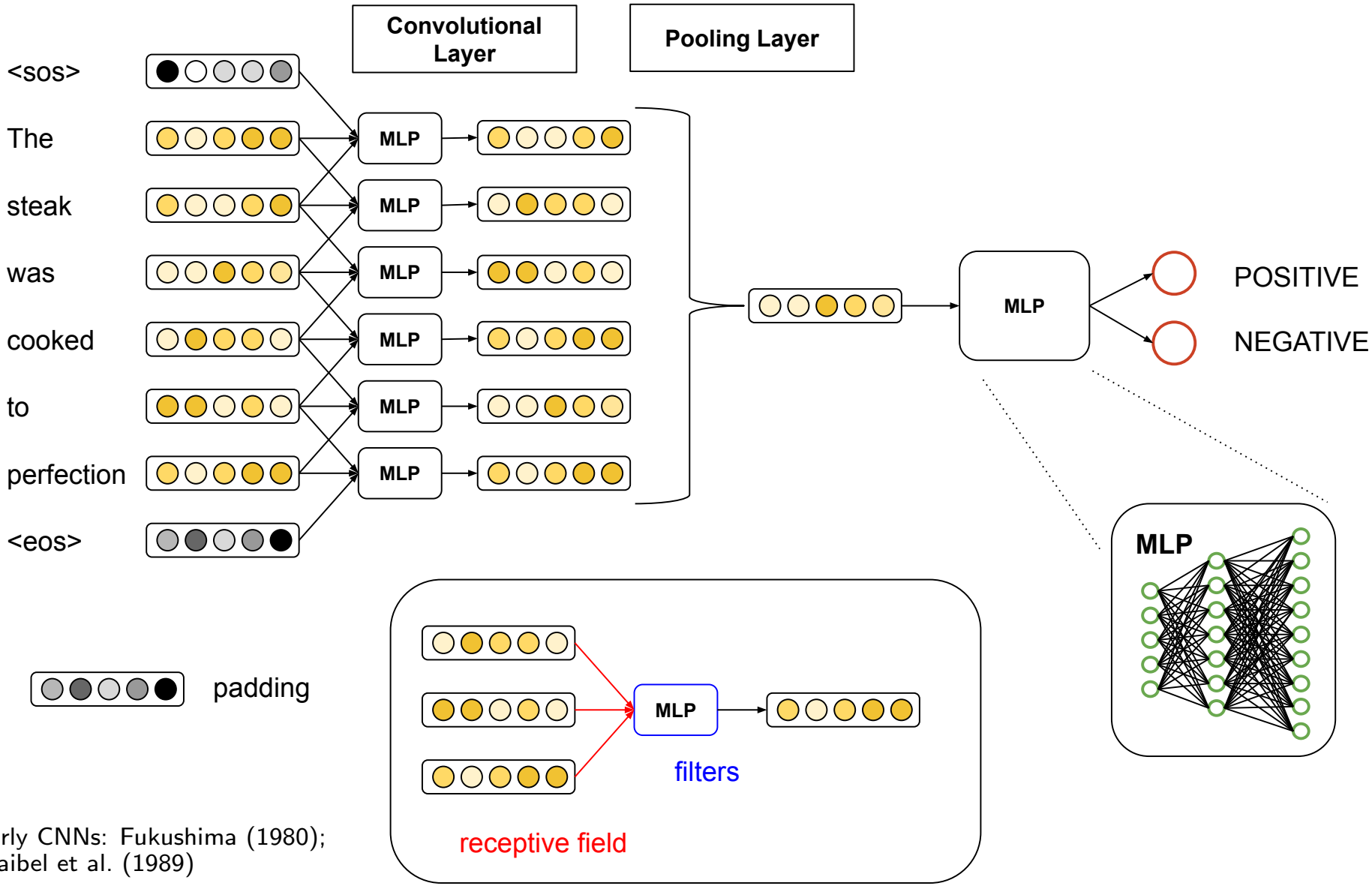
Variable Length Inputs: Options

1. Truncate document at fixed length K and stack the embeddings, $\boldsymbol{x} \in \mathbb{R}^{KD}$
2. Average embeddings (below), $\boldsymbol{x} \in \mathbb{R}^{D}$
3. convolutional (CNNs) and recurrent neural networks (RNNs)[8]



[8] RNNs not covered. See https://athnlp.github.io/2024/ppts/Day02AthNLP2024-Lec2-BPlank-handout.pdf

Convolutional Neural Networks

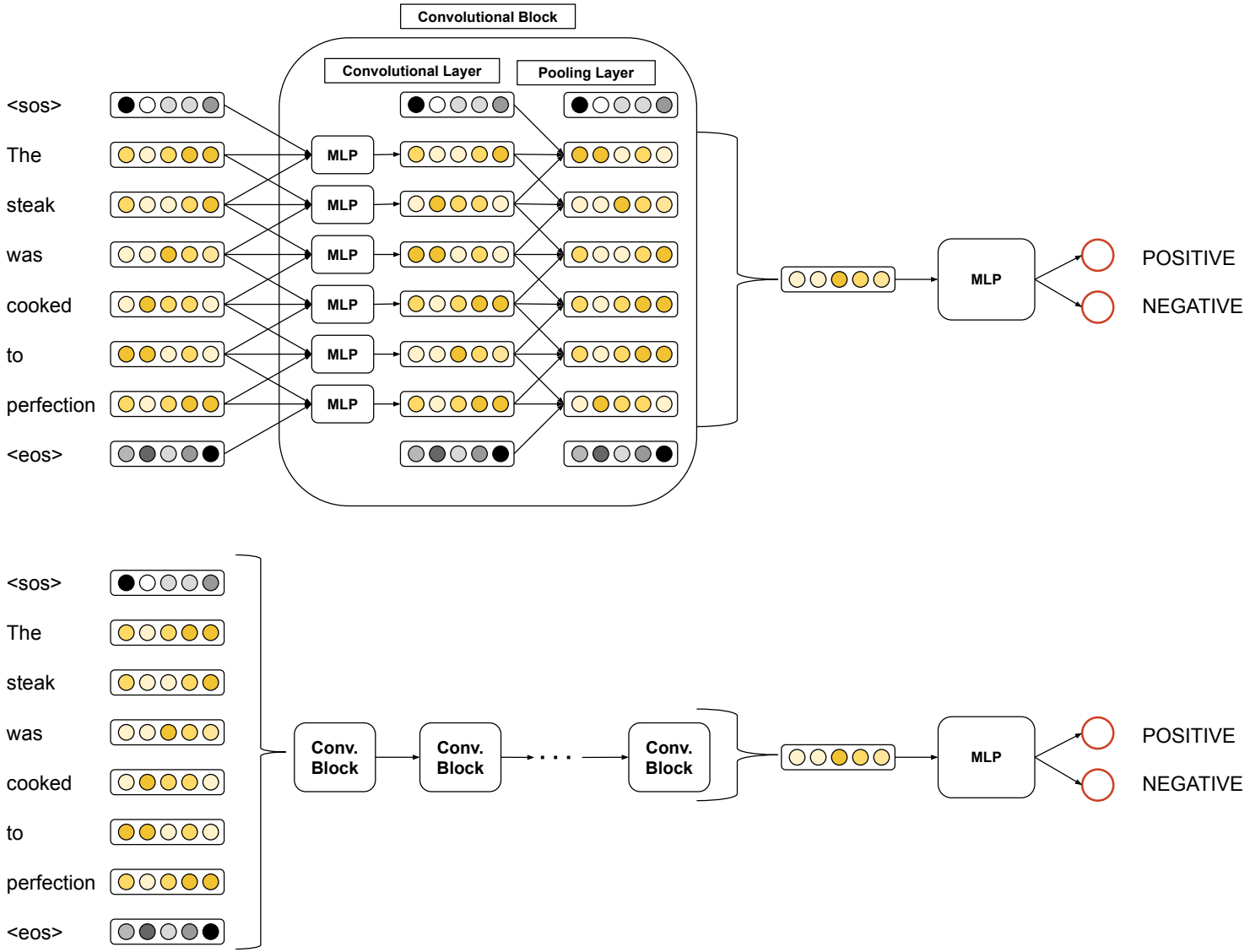


Early CNNs: Fukushima (1980); Waibel et al. (1989)

Convolutional Neural Networks

- Convolutional layer
  - A NN sub-architecture
  - Slides over input at a fixed stride, usually 1
  - Receptive field: fixed size input (e.g., *n*-gram)
  - Filter: MLP that creates a single vector output per position
  - Can be multiple filters: Almost always shared positionally; sometimes even per layer
- Pooling layer
  - Converts convolutional output to a single fixed-length vector
  - Average pooling: average outputs of convolutional layers
  - Max pooling: position-wise max over outputs of convolutional layers
  - NN: Can also learn this, e.g., attention.

Deep Convolutional Neural Networks



Core Neural Network Summary

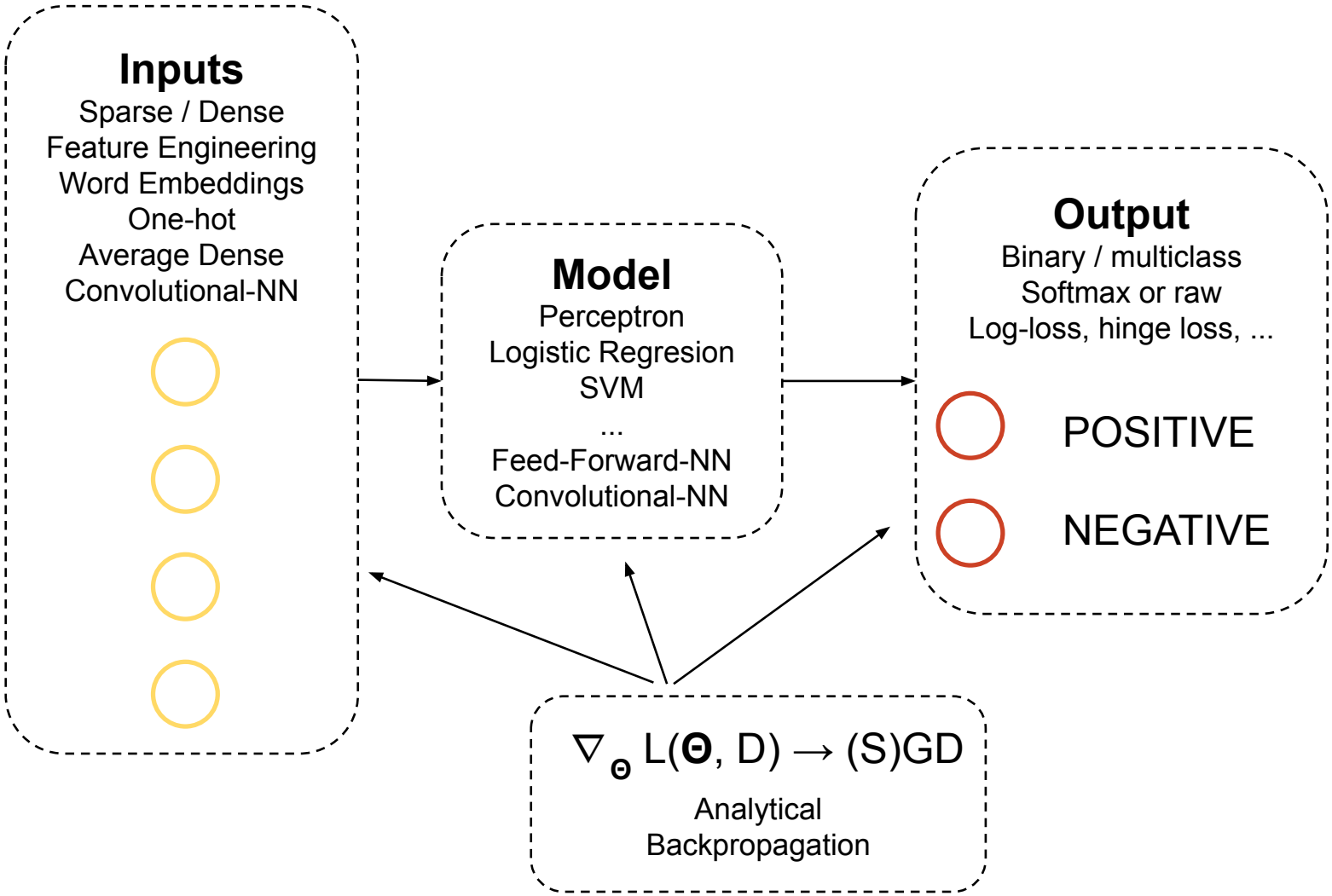
- Fully-connected Feed-forward Neural Networks
- Neurons, layers and connections
- Output layers (linear) and losses
- Back propagation
- Input layers
  - Static vs dynamic vs mixed
- Variable length inputs: CNNs
- Deep NNs – keep adding layers

Where Does Network Structure Come From?

- Hyperparamters: input/hidden dimensions; activation functions; ...
  - Usually empirical but becoming standardized (e.g., transformers)
- Deep Learning = lot's of layers. How many? Empirical accuracy vs. resources.
- Fully-connected/dense required?
  - No!
  - Sparse layers / chunks. Especially in LLMs
  - But for FF-NN components usually full-connected
  - Any efficiency concerns lessened by modern architectures (GPU, TPU)

Main Points (Parametric ML)

The steak was cooked to perfection



... in Words

- Sparse (binary) vs. dense (embeddings) features
- Optimization: Use gradient-based techniques
- Linear Classifiers
  - Scores are a matrix-vector product: $\boldsymbol{z} = \mathbf{W}\boldsymbol{\phi}(\boldsymbol{x}) + \boldsymbol{b}$, one row of $\mathbf{W}$ per label
  - Usually sparse features
  - Loss functions define model
  - Regularization necessary for good performance
- Sparse vs. Dense representations
- Parameteric (e.g., linear cls) vs. Non-parametric (e.g., knn) ML
- Neural Networks
  - Final layer = linear classifier: same $\boldsymbol{z} = \mathbf{W}_L\boldsymbol{h}_{L-1} + \boldsymbol{b}_L$
  - Hidden layers = linear + non-lin activation
  - Compute gradient with backpropagation
  - Input layer: static (e.g., word2vec) vs. dynamic (backprop)
  - (Deep) CNNs for variable length inputs

References I

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Minsky, M. and Papert, S. (1969). Perceptrons.

Novikoff, A. B. (1962). On convergence proofs for perceptrons. In *Symposium on the Mathematical Theory of Automata*.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1.

Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., and Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339.