Classification

Ryan McDonald [1]

ASAPP

AthNLP, September, 2024

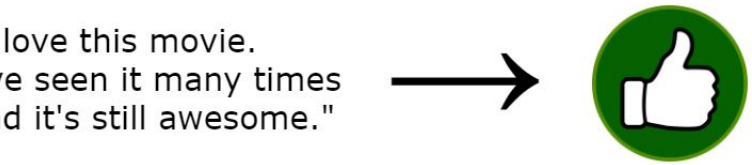
[1] Slide provenance: Ryan McDonald → Shay Cohen → Stefan Riezler → André Martins → Ryan McDonald

Classifiers

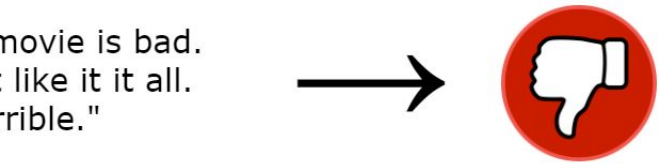
NOUN or VERB

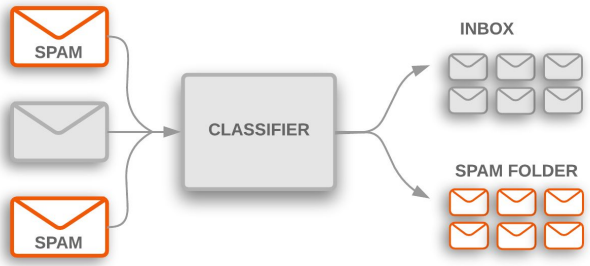
How does sodium bicarbonate work ?

"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."



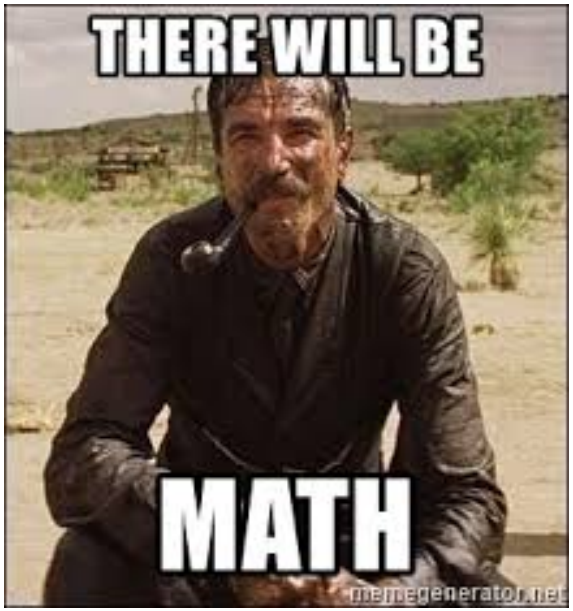



```
Set my alarm tomorrow for 10am   -> Alarm

Quickest way to Boston           -> Navigation

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

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

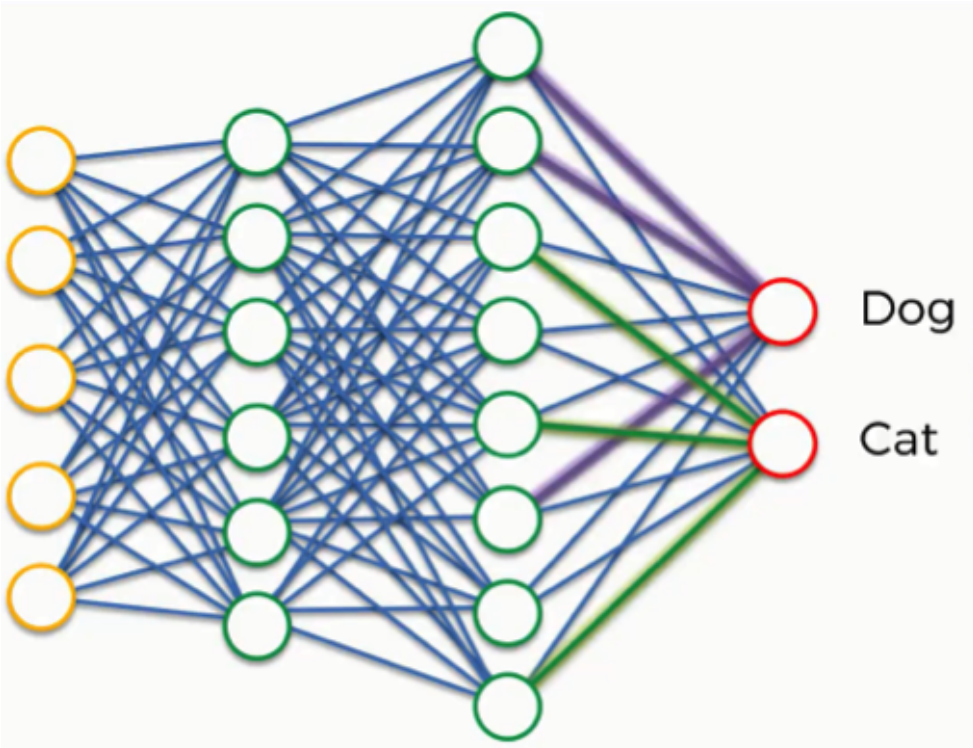
Linear Classifiers

This lecture is 2/3 about linear classifiers!

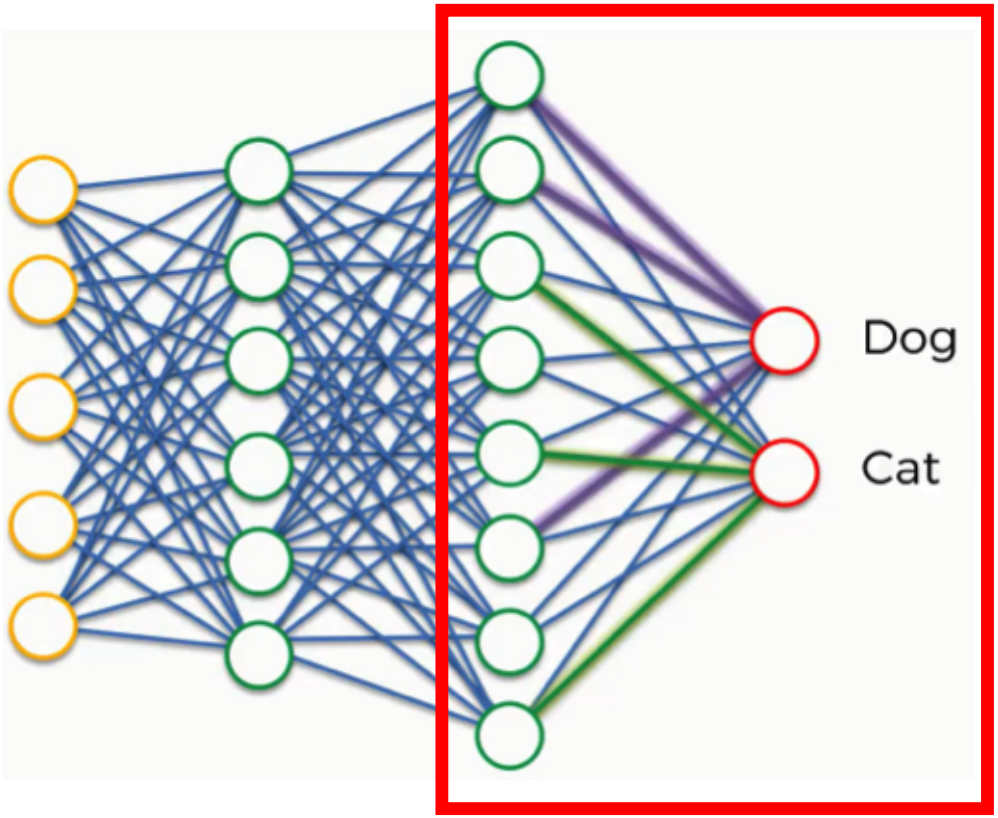
Why? It's 2024 and everybody uses neural networks.

- 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



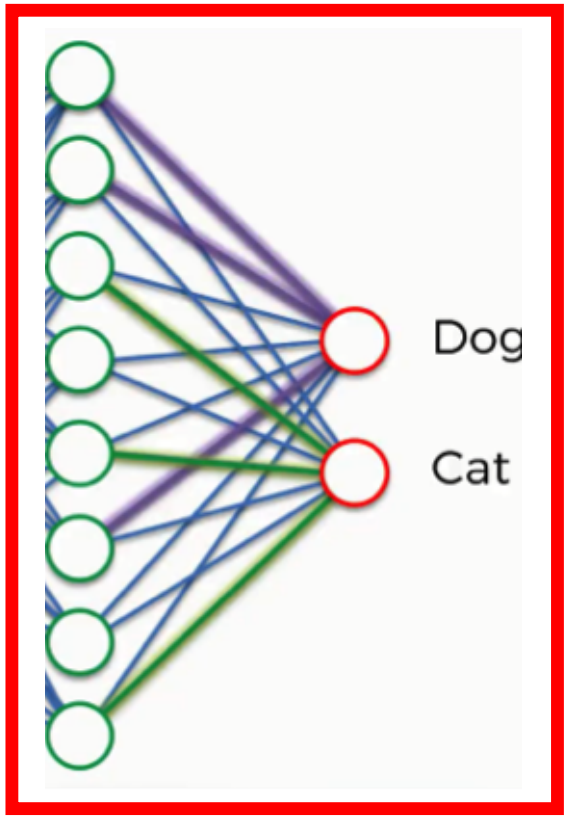
Linear Classifiers and Neural Networks



Linear Classifier

Linear Classifiers and Neural Networks

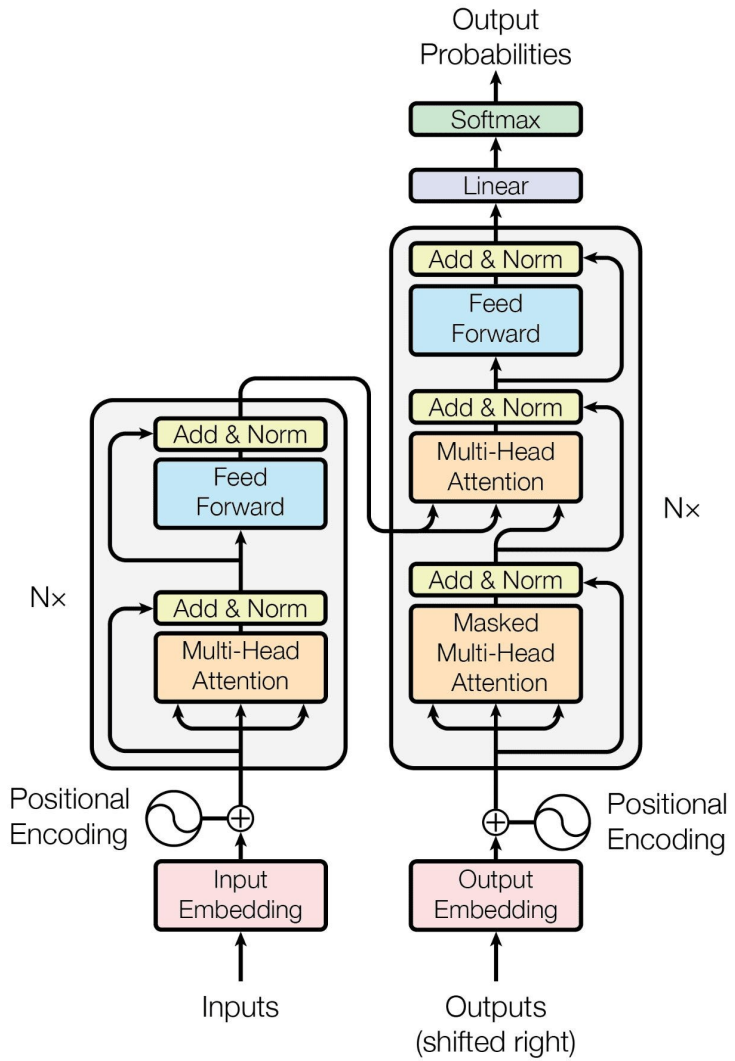
Handcrafted Features



Linear Classifier

Linear Classifiers and Generative AI

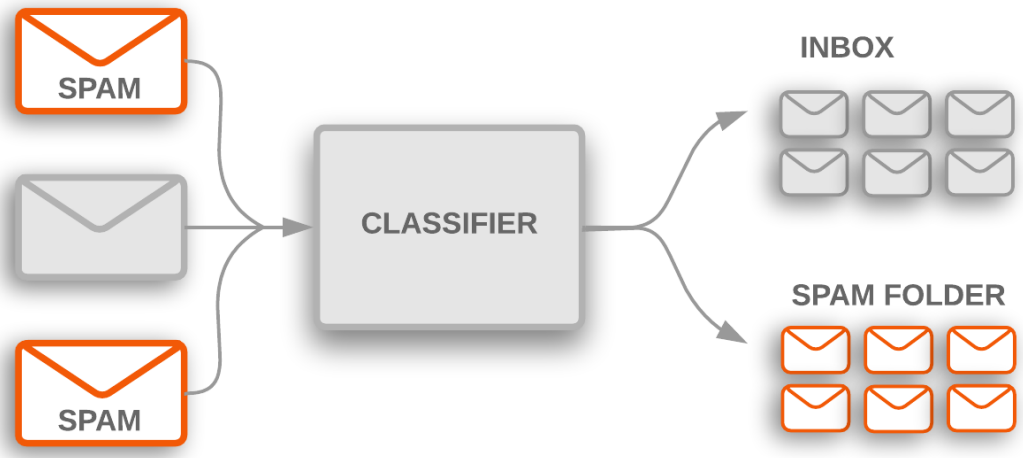
- **Transformers**: 99% of LLMs/GenAI
  - ChatGPT; GPT4*
  - Claude
  - Gemini
  - Llama*
- Last layer = linear classifier
- Last layer predicts next word/token
- I.e., last layer is a classifier!



Binary Classification: Spam Detection

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

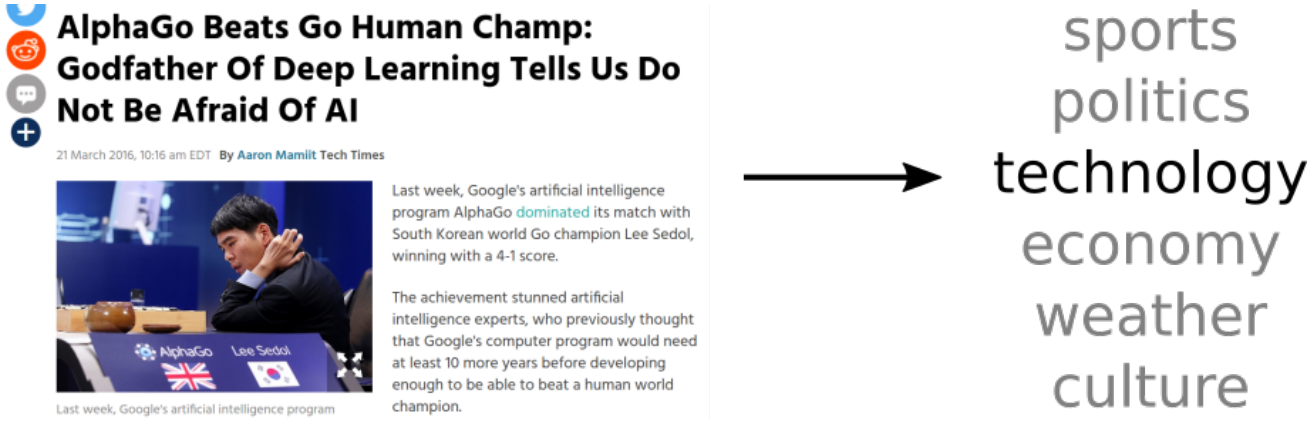
This is a **binary classification problem.**



Multiclass Classification: Topic Labeling

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

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



Let's Start Simple

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

Let's Start Simple

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

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

Let's Start Simple

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

- New sequence: $\star \diamond \circ$; label -1
- New sequence: $\star \diamond \heartsuit$; label ?

Let's Start Simple

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

- New sequence: $\star \diamond \circ$; label -1
- New sequence: $\star \diamond \heartsuit$; label -1
- New sequence: $\star \triangle \circ$; label ?

Let's Start Simple

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

- New sequence: $\star \diamond \circ$; label -1
- New sequence: $\star \diamond \heartsuit$; label -1
- New sequence: $\star \triangle \circ$; label ?

Why can we do this?

Let's Start Simple: Machine Learning

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

- New sequence: $\star \diamond \heartsuit$; label -1

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

Let's Start Simple: Machine Learning

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

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

Label -1 **Label $+1$**

$$P(-1|\star) = \frac{\text{count}(\star \text{ and } -1)}{\text{count}(\star)} = \frac{2}{3} = 0.67 \text{ vs. } P(+1|\star) = \frac{\text{count}(\star \text{ and } +1)}{\text{count}(\star)} = \frac{1}{3} = 0.33$$

$$P(-1|\triangle) = \frac{\text{count}(\triangle \text{ and } -1)}{\text{count}(\triangle)} = \frac{1}{3} = 0.33 \text{ vs. } P(+1|\triangle) = \frac{\text{count}(\triangle \text{ and } +1)}{\text{count}(\triangle)} = \frac{2}{3} = 0.67$$

$$P(-1|\circ) = \frac{\text{count}(\circ \text{ and } -1)}{\text{count}(\circ)} = \frac{1}{2} = 0.5 \text{ vs. } P(+1|\circ) = \frac{\text{count}(\circ \text{ and } +1)}{\text{count}(\circ)} = \frac{1}{2} = 0.5$$

Machine Learning

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

Outline

Some Notation: 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, an image segmentation

- 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., an **image** partitioned into **segmentation regions**

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?

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

Feature Representations

We need to represent information about $\boldsymbol{x}$

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

- $\boldsymbol{\phi}(\boldsymbol{x})$ is a high dimensional feature 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 in neural networks

Examples

- $\boldsymbol{x}$ is a document and $\boldsymbol{y}$ is a topic

$$\boldsymbol{\phi}_j(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains the word "interest"} \\ 0 & \text{otherwise} \end{cases}$$

$$\boldsymbol{\phi}_j(\boldsymbol{x}) = \% \text{ of words in } \boldsymbol{x} \text{ with punctuation}$$

- $\boldsymbol{x}$ is a word and $\boldsymbol{y}$ is a part-of-speech tag

$$\boldsymbol{\phi}_j(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ ends in "ed"} \\ 0 & \text{otherwise} \end{cases}$$

Bag of Words Feature Representation

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

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?

Our Setup

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

Linear Classifiers – Weights/Parameters

- Parametrized by a weight vector $\boldsymbol{w} \in \mathbb{R}^D$ (one weight per feature)
- E.g., $D = 5$, $\boldsymbol{w} = [0.3, 1.2, -5.4, 3.8, -0.09]$
- $\boldsymbol{\phi}(\boldsymbol{x})$ and $\boldsymbol{w}$ are vectors of same length – D
- We actually need $|\mathcal{Y}|$ weight vectors $\boldsymbol{w_y} \in \mathbb{R}^D$
  - i.e., one weight vector per output label $\boldsymbol{y}$

Linear Classifiers – Weights/Parameters

- ! Important Concept !
- $\boldsymbol{w}_{\boldsymbol{y}}$ is weight/parameter vector for output label $\boldsymbol{y}$
- Let $\mathcal{W} = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|\mathcal{Y}|}]$
- $\mathcal{W}$ is a concatenation of all $\boldsymbol{w}_{\boldsymbol{y}}$
- Example
  - $\boldsymbol{w}_1 = [1, 1]$, $\boldsymbol{w}_2 = [2, 2]$, $\boldsymbol{w}_3 = [3, 3]$ for $|\mathcal{Y}| = 3$
  - Then $\mathcal{W} = [1, 1, 2, 2, 3, 3]$

Linear Classifiers – Predictions

- The score (or probability) of a particular label is based on a linear combination of features and their weights

- At test time, predict the class $\boldsymbol{y}'$ which maximizes this score:

$$\boldsymbol{y}' = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}) = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \sum_i \boldsymbol{w}_{i,\boldsymbol{y}} \cdot \boldsymbol{\phi}_i(\boldsymbol{x})$$

- At training time, different strategies to learn $\boldsymbol{w}_{\boldsymbol{y}}$'s yield different linear classifiers: perceptron, logistic regression, SVMs, ...

Linear Classifiers – Example

- $D = 5$, $\mathcal{Y} = \{\text{Person (per), Location (loc)}\}$
- $\boldsymbol{w}_{\text{per}} = [0.3, 1.2, -5.4, 3.8, -0.09]$
- $\boldsymbol{w}_{\text{loc}} = [-0.6, 2.4, 4.0, -2.1, 0.1]$
- $\boldsymbol{x}$ =George Washington Bridge $\rightarrow \boldsymbol{\phi}(x) = [1, 1, 1, 0, 0]$

$$
\begin{aligned}
\boldsymbol{y}' &= \arg \max_{\boldsymbol{y} \in \{\text{loc,per}\}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}) \\
&= \arg \max_{\boldsymbol{y} \in \{\text{loc,per}\}} \{\ [-0.6, 2.4, 4.0, -2.1, 0.1]_{\text{loc}} \cdot [1, 1, 1, 0, 0], \\
&\qquad\qquad [0.3, 1.2, -5.4, 3.8, -0.09]_{\text{per}} \cdot [1, 1, 1, 0, 0]\ \} \\
&= \arg \max_{\boldsymbol{y} \in \{\text{loc,per}\}} \{5.8_{\text{loc}}, -3.9_{\text{per}}\} \\
&= \text{loc}
\end{aligned}
$$

Linear Classifiers – Bias Terms

- Often linear classifiers are presented as

$$\boldsymbol{y}' = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}) + b_{\boldsymbol{y}}$$

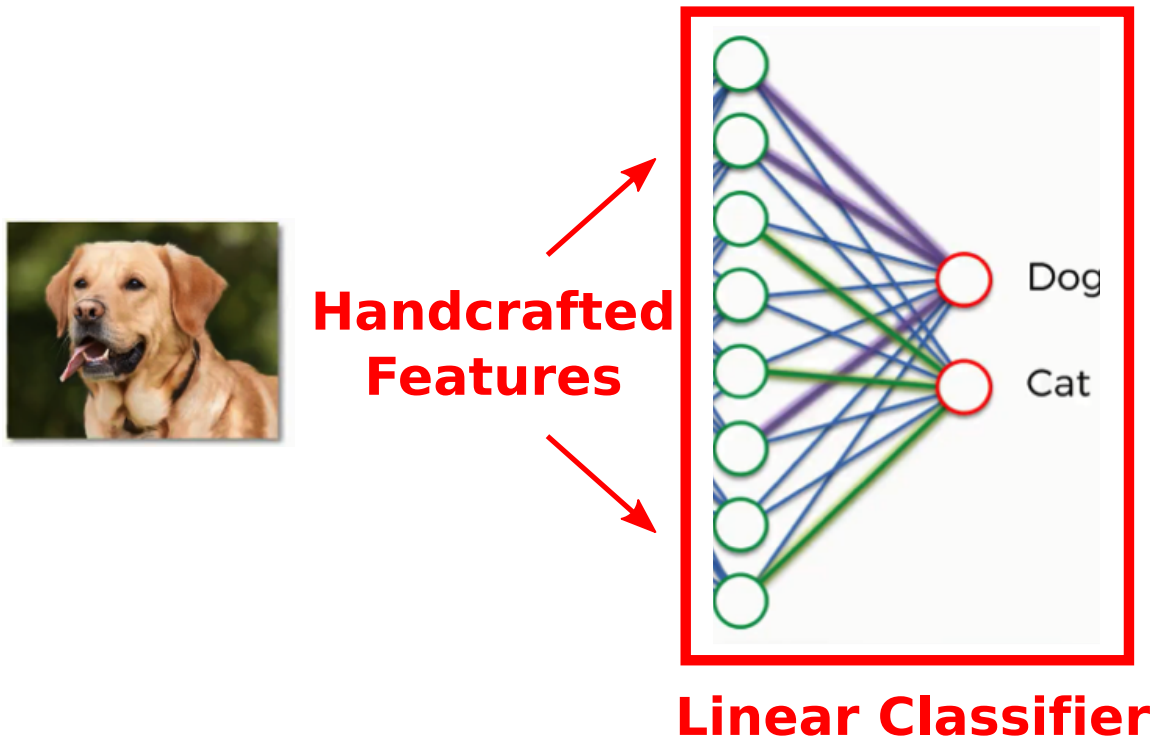
  where $b_{\boldsymbol{y}}$ is a bias or offset term

- This can be folded into $\boldsymbol{\phi}(\boldsymbol{x})$ via a constant feature

- I.e., $\boldsymbol{\phi}(\boldsymbol{x}) = [\boldsymbol{\phi}(\boldsymbol{x}), 1]$

- For now, we assume this for simplicity

Commonly Used Notation in Neural Networks



$$y' = \textbf{argmax}\left(\mathbf{W}\phi(x)^\top + \boldsymbol{b}\right), \quad \mathbf{W} = \begin{bmatrix} \vdots \\ \boldsymbol{w_y} \\ \vdots \end{bmatrix}, \; \boldsymbol{b} = \begin{bmatrix} \vdots \\ b_y \\ \vdots \end{bmatrix}.$$

Binary Linear Classifier

With binary labels ($\mathcal{Y} = \{\pm 1\}$) we often use a minimal parametrization:

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

Binary Linear Classifier

With binary labels ($\mathcal{Y} = \{\pm 1\}$) we often use a minimal parametrization:

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

Binary Linear Classifier

With binary labels ($\mathcal{Y} = \{\pm 1\}$) we often use a minimal parametrization:

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

Binary Linear Classifier

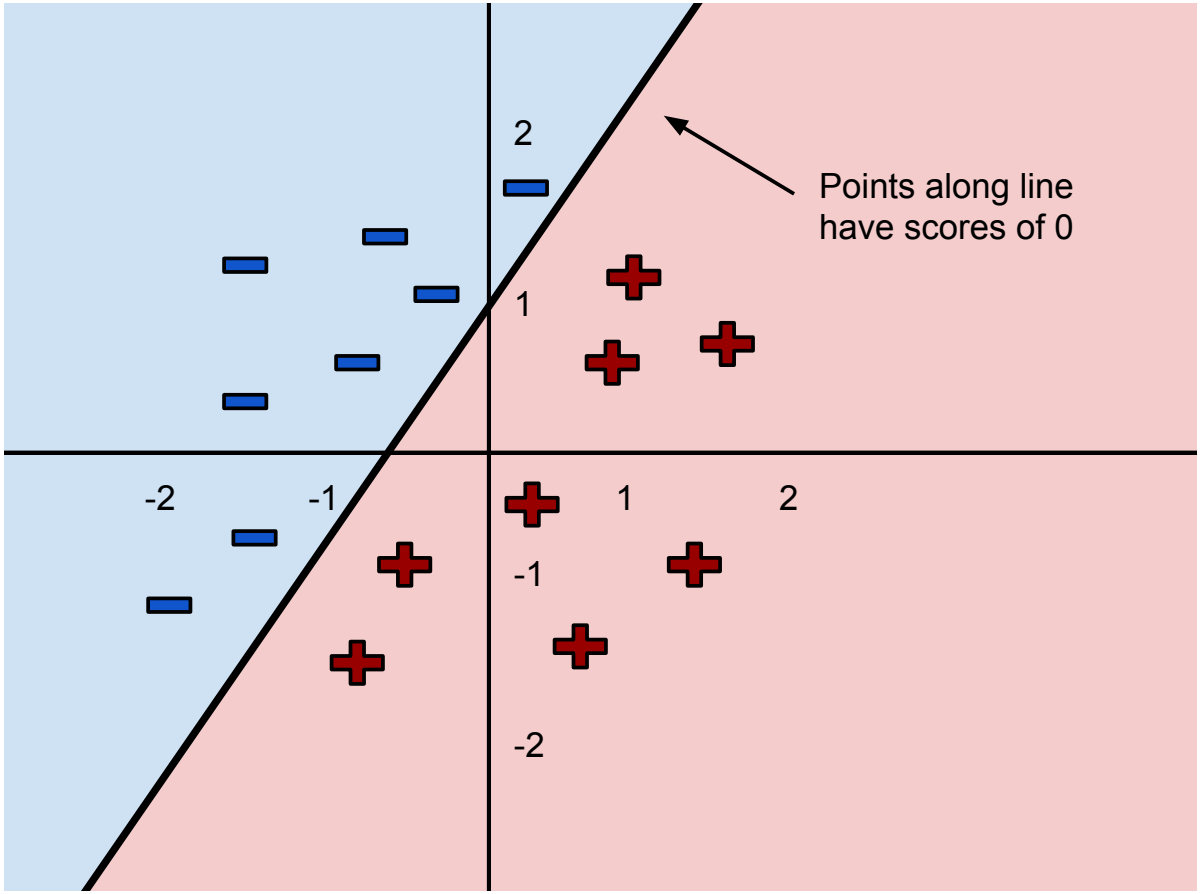
With binary labels ($\mathcal{Y} = \{\pm 1\}$) we often use a minimal parametrization:

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

That is: only half of the parameters are needed.

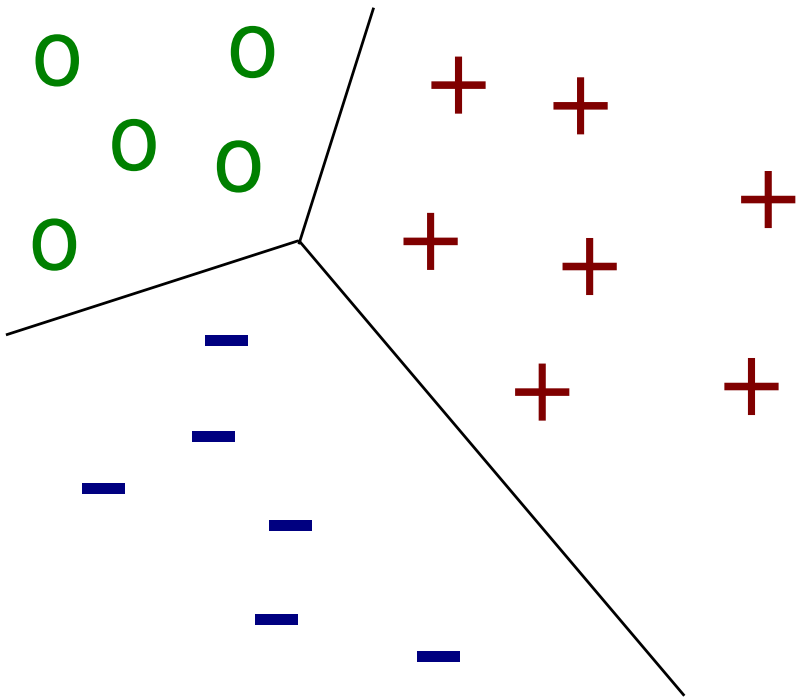
Binary Linear Classifier

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



Multiclass Linear Classifier

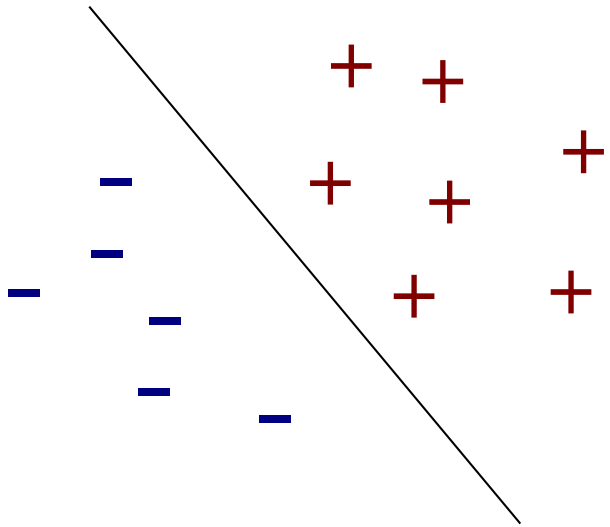
Defines regions of space.



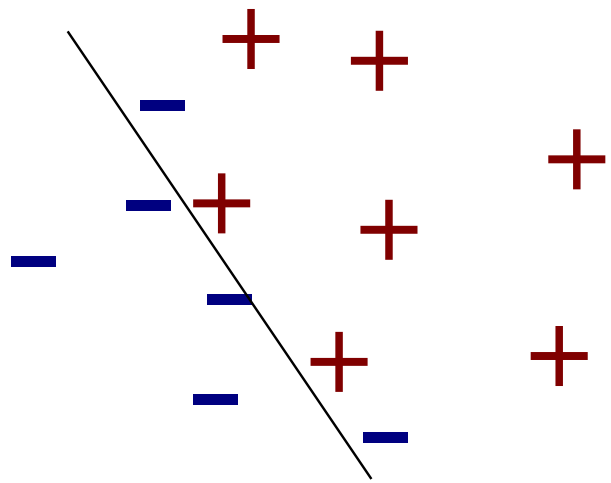
Linear Separability

- A set of points is linearly separable if there exists a $\boldsymbol{w}$ such that classification is perfect

Separable



Not Separable

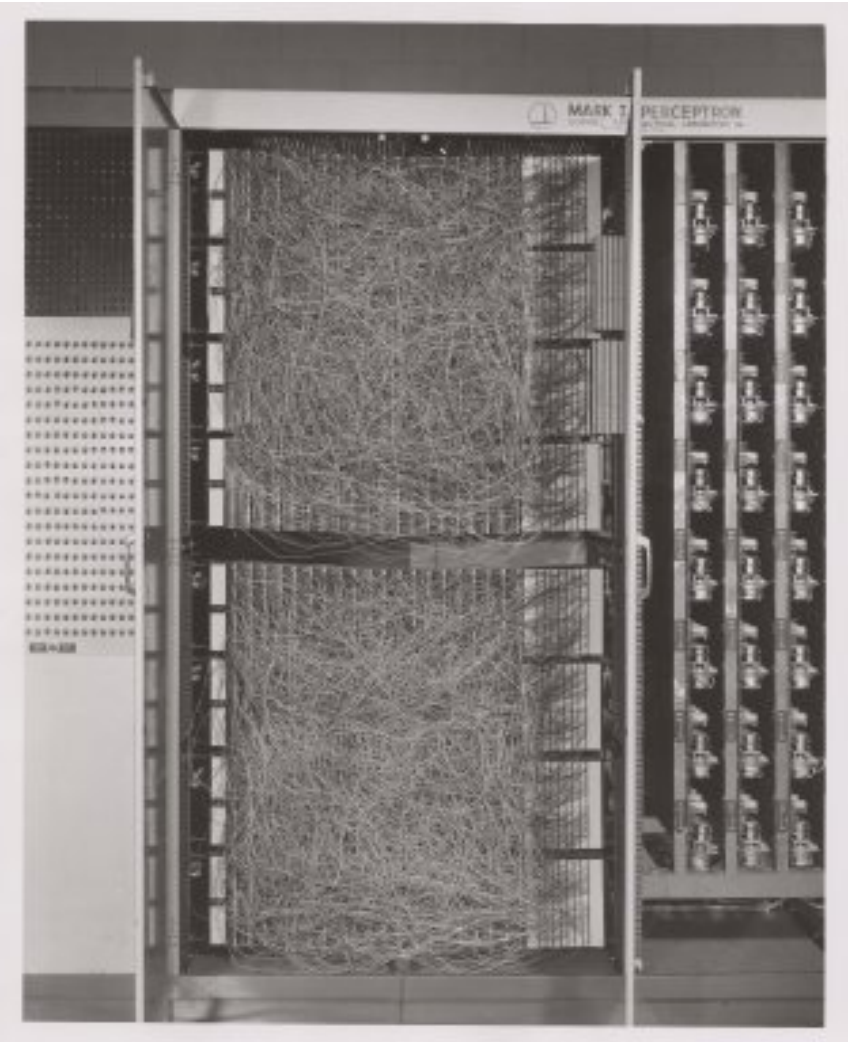


Learning

- Machine Learning = finding weights/parameters $\mathcal{W}/\boldsymbol{w}$
- Using data! Specifically $\mathcal{D} = \{\boldsymbol{x}_t, \boldsymbol{y}_t\}_{t=1}$
- There are many algorithms for doing this

Outline

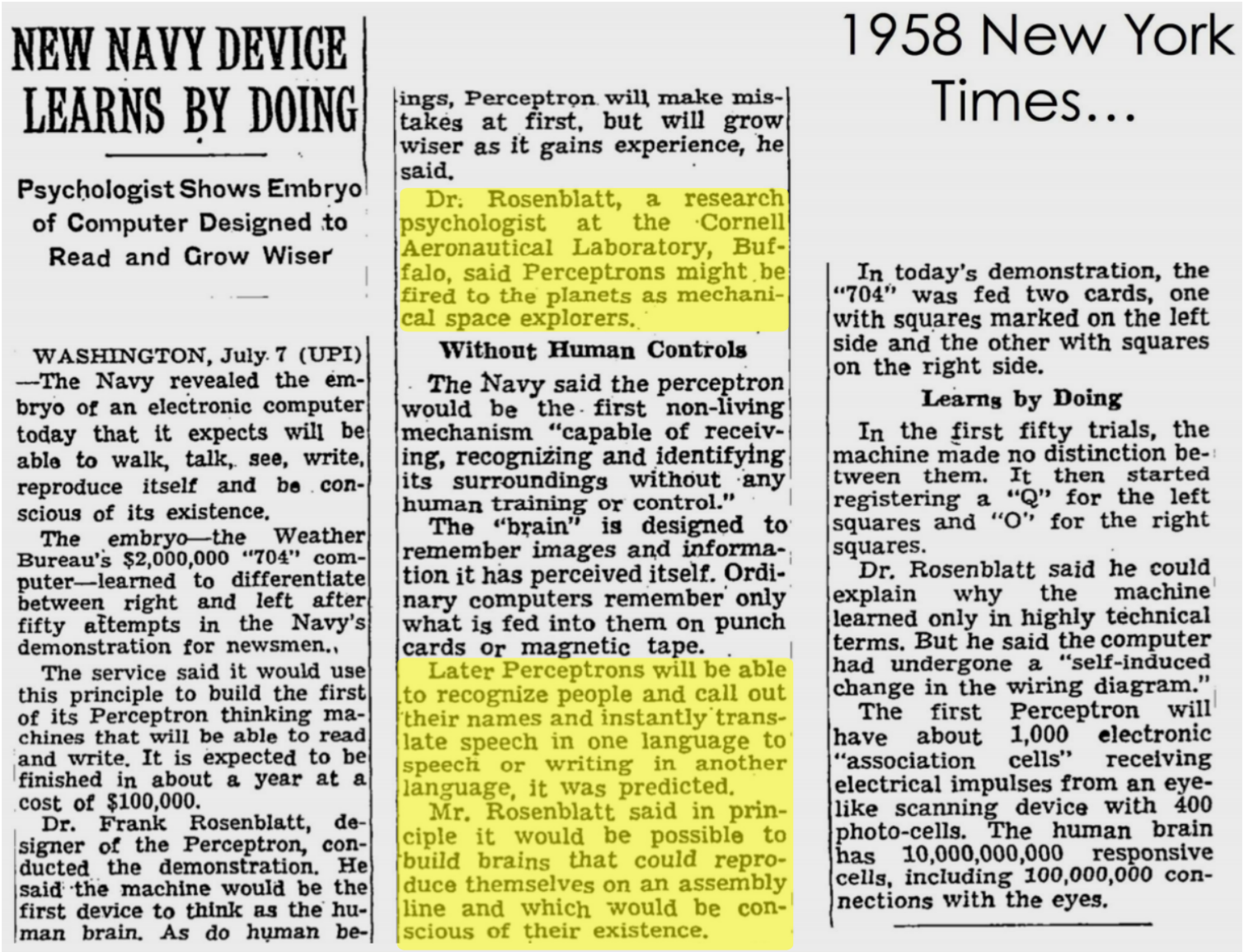
Perceptron (Rosenblatt, 1958)



(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 at each round
  - Take $\boldsymbol{x}_t$; apply the current model to make a prediction for it
  - If prediction is correct, proceed
  - Else, correct model: add feature vector w.r.t. correct output & subtract feature vector w.r.t. predicted (wrong) output

Perceptron Algorithm

input: labeled data $\mathcal{D}$
initialize $\mathcal{W}^0 = \mathbf{0}$, i.e., $\boldsymbol{w}_{\boldsymbol{y}}^{(0)} = \mathbf{0}$, $\forall \boldsymbol{y}$
initialize $k = 0$
repeat
  update $\boldsymbol{w}_{\boldsymbol{y}}^{(k+1)} = \boldsymbol{w}_{\boldsymbol{y}}^{(k)}$, $\forall \boldsymbol{y}$
  observe example $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$
  predict $\boldsymbol{y}' = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{w}_{\boldsymbol{y}}^{(k)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)$
  if $\boldsymbol{y}' \neq \boldsymbol{y}_t$ **then**
    update $\boldsymbol{w}_{\boldsymbol{y}_t}^{(k+1)} = \boldsymbol{w}_{\boldsymbol{y}_t}^{(k)} + \boldsymbol{\phi}(\boldsymbol{x}_t)$
    update $\boldsymbol{w}_{\boldsymbol{y}'}^{(k+1)} = \boldsymbol{w}_{\boldsymbol{y}'}^{(k)} - \boldsymbol{\phi}(\boldsymbol{x}_t)$
  end if
  increment k
until maximum number of epochs
output: model weights $\boldsymbol{w}$

Perceptron's Mistake Bound

A couple definitions:

- the training data is linearly separable with margin $\gamma > 0$ iff there is a weight vectors $\boldsymbol{u_y}$ with $\|\boldsymbol{u_y}\| = 1$ such that

$$\boldsymbol{u}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq \boldsymbol{u}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) + \gamma, \quad \forall i,\ \forall \boldsymbol{y}' \neq \boldsymbol{y}_t.$$

- 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 weight vectors $\boldsymbol{u_y}$ with $\|\boldsymbol{u_y}\| = 1$ such that

$$\boldsymbol{u}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq \boldsymbol{u}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) + \gamma, \quad \forall i,\ \forall \boldsymbol{y}' \neq \boldsymbol{y}_t.$$

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

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

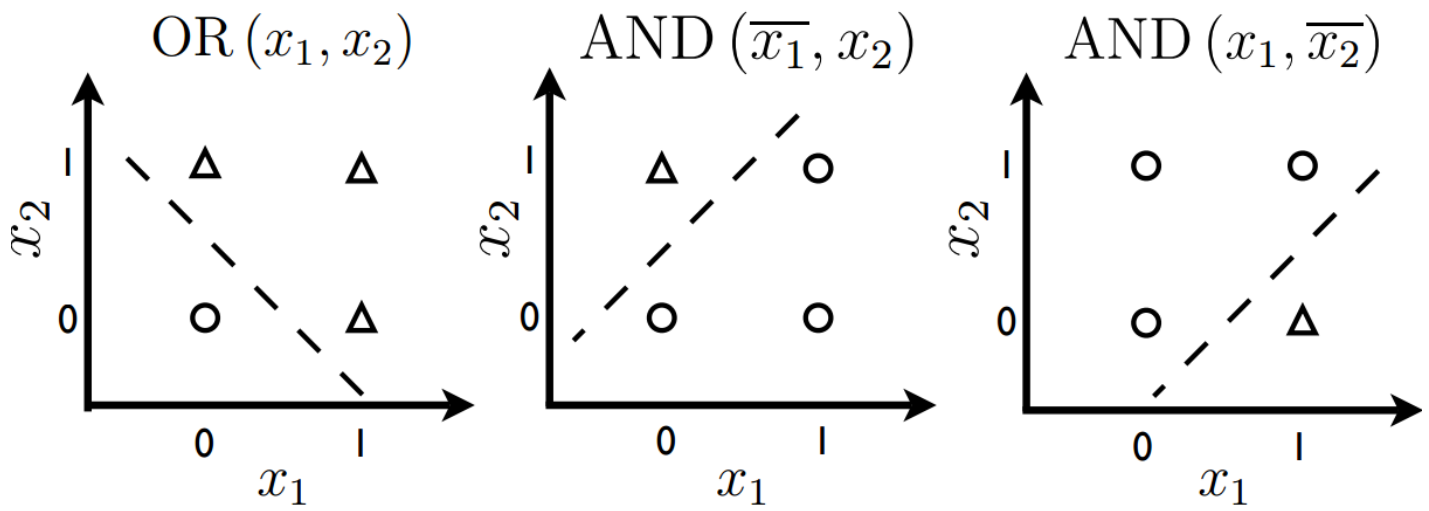
Theorem (Novikoff (1962))

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`

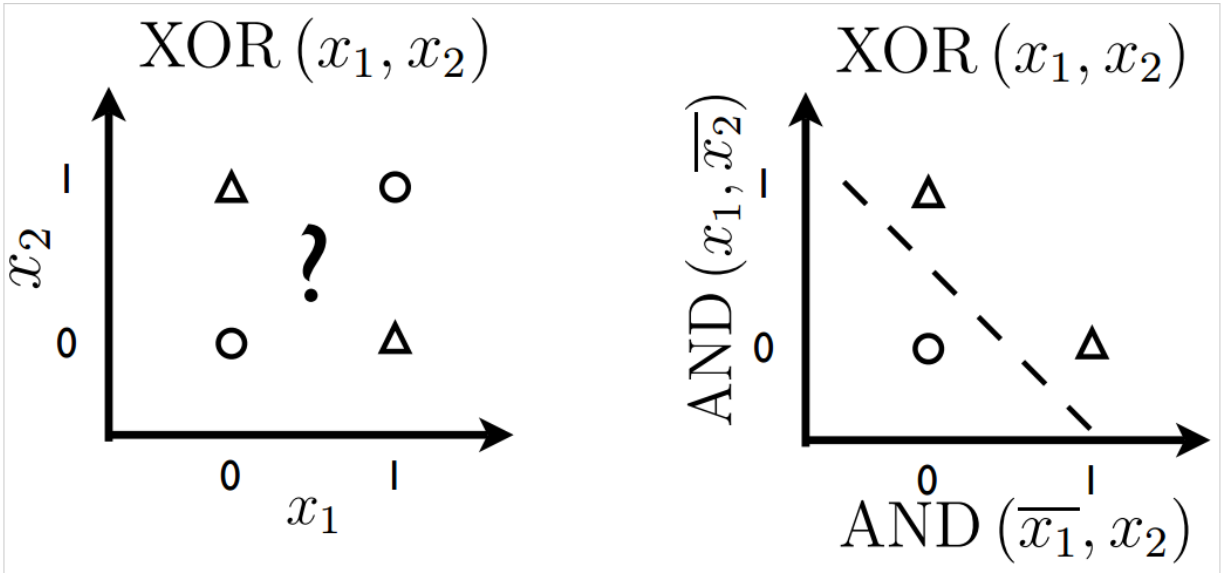
What a Simple Perceptron Can and Can't Do

- 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):



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

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

Is it any good in practice?

Until 2013/2014, perceptron variants were pretty close to state-of-the-art

- Hall et al. 2012: Named-entity recognition
- Huang et al. 2012: Part-of-speech tagging
- Li et al. 2013: Event/relation extraction
- Yu et al. 2013: Machine Translation
- Bohnet et al. 2016: Syntactic parsing

We are going to cover more complex and principled linear classifiers

However, they rarely were significantly better than perceptron variants in practice.

Outline

Logistic Regression

Define a conditional probability:

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}))}{Z_{\boldsymbol{x}}}, \qquad \text{where } Z_{\boldsymbol{x}} = \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp(\boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}))$$

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

Exponentiating and normalizing is called the softmax transformation[2]

Note: still a linear classifier

$$\begin{aligned}
\arg\max_{\boldsymbol{y}} \; P(\boldsymbol{y}|\boldsymbol{x}) &= \arg\max_{\boldsymbol{y}} \; \frac{\exp(\boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}))}{Z_{\boldsymbol{x}}} \\
&= \arg\max_{\boldsymbol{y}} \; \exp(\boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x})) \\
&= \arg\max_{\boldsymbol{y}} \; \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x})
\end{aligned}$$

[2] More later during neural networks!

Logistic Regression

$$P_{\mathcal{W}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}))}{Z_{\boldsymbol{x}}}$$

- Let $\mathcal{W} = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|\mathcal{Y}|}]$ be a vector concatenating all weights $\boldsymbol{w}_{\boldsymbol{y}}$

How do we learn $\mathcal{W}$?

- Set $\mathcal{W}$ to minimize the negative conditional log-likelihood:

$$\begin{aligned}
\mathcal{W} &= \arg\min_{\mathcal{W}} -\log\left(\prod_{t=1} P_{\mathcal{W}}(\boldsymbol{y}_t|\boldsymbol{x}_t)\right) = \arg\min_{\mathcal{W}} -\sum_{t=1} \log P_{\mathcal{W}}(\boldsymbol{y}_t|\boldsymbol{x}_t) \\
&= \arg\min_{\mathcal{W}} \sum_{t=1} \left(\log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)\right),
\end{aligned}$$

i.e., set weights to 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 ✓

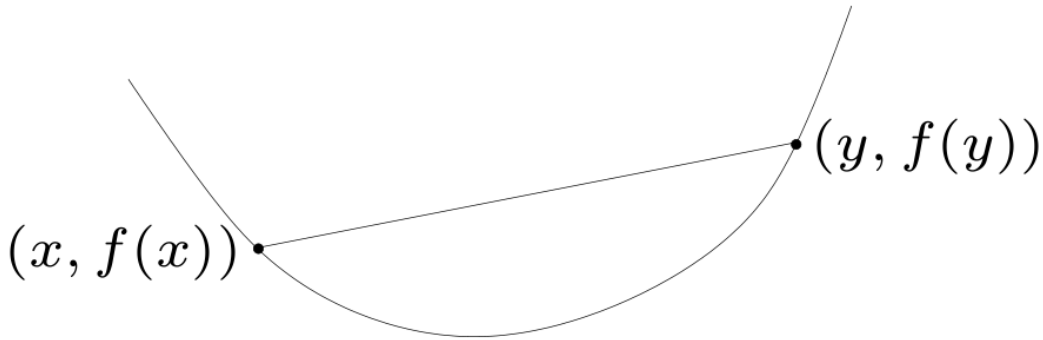


Figure: Illustration of a convex function. The line segment between any two points on the graph lies entirely above the curve.

Recap: Gradients

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

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

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

The gradient gives the direction and fastest rate of increase of f at point $\mathcal{W}$

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

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: $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} - \eta_k \nabla f(\boldsymbol{w}^{(k)})$

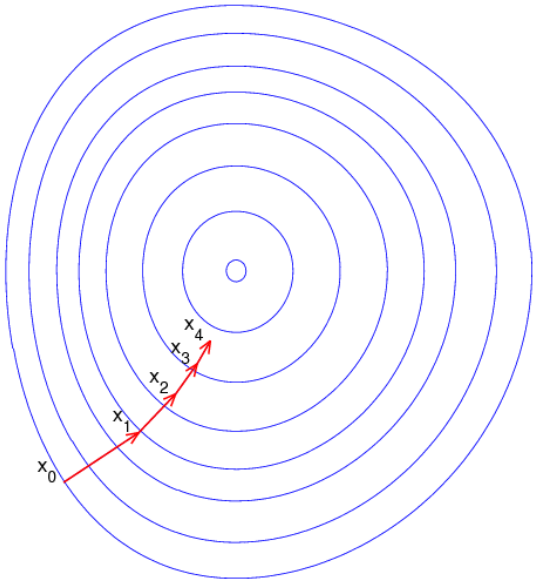


Figure: Illustration of gradient descent. The red lines correspond to steps taken in the negative gradient direction.

Logistic Regression: Gradient Descent (GD)

- Let $L(\mathcal{W}; (\boldsymbol{x}, \boldsymbol{y})) = \left(\log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w}'_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x})) - \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x})\right)$
- 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 $\arg\min_{\mathcal{W}} \sum_{t=1} L(\mathcal{W}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$
  - Logistic-regressions loss function often called log-loss or cross-entropy
- GD update will look like

$$\begin{aligned} \mathcal{W}^{k+1} &= \mathcal{W}^k - \eta_k \nabla_{\mathcal{W}} \left(\sum_{t=1} L(\mathcal{W}; (\boldsymbol{x}_t, \boldsymbol{y}_t))\right) \\ &= \mathcal{W}^k - \eta_k \sum_{t=1} \nabla_{\mathcal{W}} L(\mathcal{W}; (\boldsymbol{x}_t, \boldsymbol{y}_t)) \end{aligned}$$

- Need to calculate $\nabla_{\mathcal{W}} L(\mathcal{W}; (\boldsymbol{x}, \boldsymbol{y}))$: gradient of L w.r.t. $\mathcal{W}$
- 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 $\mathcal{W}^{k+1} = \mathcal{W}^k - \eta_k \nabla_{\mathcal{W}} L(\mathcal{W}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$

- 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 $\mathcal{W}$ (for suitable step sizes)

Logistic Regression: Simple SGD Algorithm

input: labeled data $\mathcal{D}$, step sizes $\eta_0, \eta_1, \ldots$
initialize $\mathcal{W} = \mathbf{0}$, i.e., $\boldsymbol{w}_{\boldsymbol{y}}^{(0)} = \mathbf{0}$, $\forall \boldsymbol{y}$
initialize $k = 0$
repeat
  observe example $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$
  Update $\mathcal{W}^{k+1} = \mathcal{W}^k - \eta_k \nabla_{\mathcal{W}} L(\mathcal{W}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$
  increment k
until stopping criterion
output: model weights $\mathcal{W}$

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

Computing the Gradient: $\nabla_{\mathcal{W}} L(\mathcal{W}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$

- We need $\nabla_{\mathcal{W}} L(\mathcal{W}; (\boldsymbol{x}, \boldsymbol{y}))$, where

$$L(\mathcal{W}; (\boldsymbol{x}, \boldsymbol{y})) = \log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x})) - \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x})$$

$$\mathcal{W} = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{\boldsymbol{y}'}, \ldots, \boldsymbol{w}_{\boldsymbol{y}}, \ldots, \boldsymbol{w}_{|\mathcal{Y}|}]$$

Some reminders:

1. $\nabla_{\boldsymbol{w}} \log F(\boldsymbol{w}) = \frac{1}{F(\boldsymbol{w})} \nabla_{\boldsymbol{w}} F(\boldsymbol{w})$
2. $\nabla_{\boldsymbol{w}} \exp F(\boldsymbol{w}) = \exp(F(\boldsymbol{w})) \nabla_{\boldsymbol{w}} F(\boldsymbol{w})$

Computing the Gradient

$$
\begin{aligned}
\nabla_{\mathcal{W}} L(\mathcal{W}; (\boldsymbol{x}, \boldsymbol{y})) &= \nabla_{\mathcal{W}} \left(\log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x})) - \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}) \right) \\
&= \nabla_{\mathcal{W}} \log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x})) - \nabla_{\mathcal{W}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}) \\
&= \frac{1}{\sum_{\boldsymbol{y}'} \exp(\boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}))} \sum_{\boldsymbol{y}'} \nabla_{\mathcal{W}} \exp(\boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x})) - \nabla_{\mathcal{W}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}) \\
&= \frac{1}{Z_{\boldsymbol{x}}} \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x})) \nabla_{\mathcal{W}} \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}) - \nabla_{\mathcal{W}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}) \\
&= \sum_{\boldsymbol{y}'} \frac{\exp(\boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}))}{Z_{\boldsymbol{x}}} \nabla_{\mathcal{W}} \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}) - \nabla_{\mathcal{W}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}) \\
&= \sum_{\boldsymbol{y}'} P_{\mathcal{W}}(\boldsymbol{y}' | \boldsymbol{x}) \nabla_{\mathcal{W}} \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}) - \nabla_{\mathcal{W}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}).
\end{aligned}
$$

Computing the Gradient

$\nabla_{\mathcal{W}} L(\mathcal{W}; (\boldsymbol{x}, \boldsymbol{y})) = \sum_{\boldsymbol{y}'} P_{\mathcal{W}}(\boldsymbol{y}'|\boldsymbol{x}) \nabla_{\mathcal{W}} \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}) - \nabla_{\mathcal{W}} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x})$

Let's look at the partial derivative wrt to a variable i: $\frac{\partial}{\partial w_i} L(\mathcal{W}; (\boldsymbol{x}, \boldsymbol{y}))$

Remember that $\mathcal{W} = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{\boldsymbol{y}'}, \ldots, \boldsymbol{w}_{\boldsymbol{y}}, \ldots, \boldsymbol{w}_{|\mathcal{Y}|}]$

Cases:

1. i indexes a weight w_i in $\mathcal{W}$ that is in $\boldsymbol{w}_{\boldsymbol{y}}$

$$\sum_{\boldsymbol{y}'} P_{\mathcal{W}}(\boldsymbol{y}'|\boldsymbol{x}) \frac{\partial}{\partial w_i} \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}) - \frac{\partial}{\partial w_i} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}) = P_{\mathcal{W}}(\boldsymbol{y}|\boldsymbol{x}) \phi_i(\boldsymbol{x}) - \phi_i(\boldsymbol{x})$$

2. i indexes a weight w_i in $\mathcal{W}$ that is in $\boldsymbol{w}_{\boldsymbol{y}'}$ where $\boldsymbol{y}' \neq \boldsymbol{y}$

$$\sum_{\boldsymbol{y}'} P_{\mathcal{W}}(\boldsymbol{y}'|\boldsymbol{x}) \frac{\partial}{\partial w_i} \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}) - \frac{\partial}{\partial w_i} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}) = P_{\mathcal{W}}(\boldsymbol{y}'|\boldsymbol{x}) \phi_i(\boldsymbol{x})$$

What does the update look like?

Cases:

1. For true output $\boldsymbol{y}$

$$\boldsymbol{w}_{\boldsymbol{y}}^{k+1} = \boldsymbol{w}_{\boldsymbol{y}}^{k} - \eta \left(P_{\mathcal{W}}(\boldsymbol{y}|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\phi}(\boldsymbol{x}) \right)$$

2. For $\boldsymbol{y}' \neq \boldsymbol{y}$

$$\boldsymbol{w}_{\boldsymbol{y}'}^{k+1} = \boldsymbol{w}_{\boldsymbol{y}'}^{k} - \eta \left(P_{\mathcal{W}}(\boldsymbol{y}'|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}) \right)$$

SGD for Logistic Regression

input: labeled data $\mathcal{D}$, step sizes $\eta_0, \eta_1, \ldots$

initialize $\mathcal{W} = \mathbf{0}$, i.e., $\boldsymbol{w}_{\boldsymbol{y}}^{(0)} = \mathbf{0}$, $\forall \boldsymbol{y}$

initialize $k = 0$

repeat

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

$\boldsymbol{w}_{\boldsymbol{y}_t}^{k+1} = \boldsymbol{w}_{\boldsymbol{y}_t}^{k} - \eta_k \left(P_{\mathcal{W}}(\boldsymbol{y}_t|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\phi}(\boldsymbol{x})\right)$

$\boldsymbol{w}_{\boldsymbol{y}'}^{k+1} = \boldsymbol{w}_{\boldsymbol{y}'}^{k} - \eta_k \left(P_{\mathcal{W}}(\boldsymbol{y}'|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})\right)$ for $\boldsymbol{y}' \neq \boldsymbol{y}_t$

increment k

until stopping criterion

output: model weights $\mathcal{W}$

Logistic Regression Summary

- Define conditional probability

$$P_{\mathcal{W}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}))}{Z_{\boldsymbol{x}}}$$

- Set weights to minimize negative conditional log-likelihood:

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

- Can find the gradient and run gradient descent (or any gradient-based optimization algorithm)

The Story So Far

- Logistic regression is discriminative: maximizes conditional likelihood
  - also called log-linear model and max-entropy classifier
  - no closed form solution
  - For training instance $(\boldsymbol{x}, \boldsymbol{y})$, SDG updates look like

$$\boldsymbol{w}_{\boldsymbol{y}}^{k+1} = \boldsymbol{w}_{\boldsymbol{y}}^{k} + \eta \left(\boldsymbol{\phi}(\boldsymbol{x}) - P_{\mathcal{W}}(\boldsymbol{y}|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}) \right)$$

$$\boldsymbol{w}_{\boldsymbol{y}'}^{k+1} = \boldsymbol{w}_{\boldsymbol{y}'}^{k} - \eta \left(P_{\mathcal{W}}(\boldsymbol{y}'|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}) \right) \text{ for } \boldsymbol{y}' \neq \boldsymbol{y}$$

- Perceptron is a discriminative, non-probabilistic classifier
  - For training instance $(\boldsymbol{x}, \boldsymbol{y})$, updates look like

$$\boldsymbol{w}_{\boldsymbol{y}}^{k+1} = \boldsymbol{w}_{\boldsymbol{y}}^{k} + \boldsymbol{\phi}(\boldsymbol{x})$$

$$\boldsymbol{w}_{\boldsymbol{y}'}^{k+1} = \boldsymbol{w}_{\boldsymbol{y}'}^{k} - \boldsymbol{\phi}(x) \text{ for } \boldsymbol{y}' \neq \boldsymbol{y}$$

SGD updates for logistic regression and perceptron's updates look similar!

Classification Margin

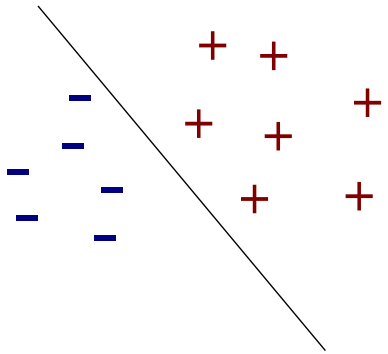
- For a training set $\mathcal{D}$
- Margin of a weight vector $\mathcal{W}$ is largest γ such that

$$\boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq \gamma$$

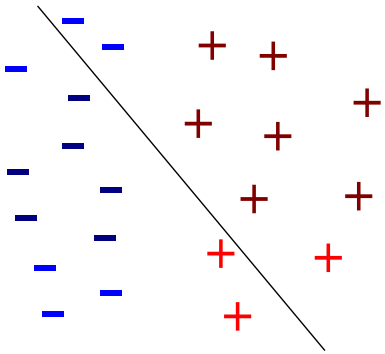
- for every training instance $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$, $\boldsymbol{y}' \neq \boldsymbol{y} \in \mathcal{Y}$

Classification Margin

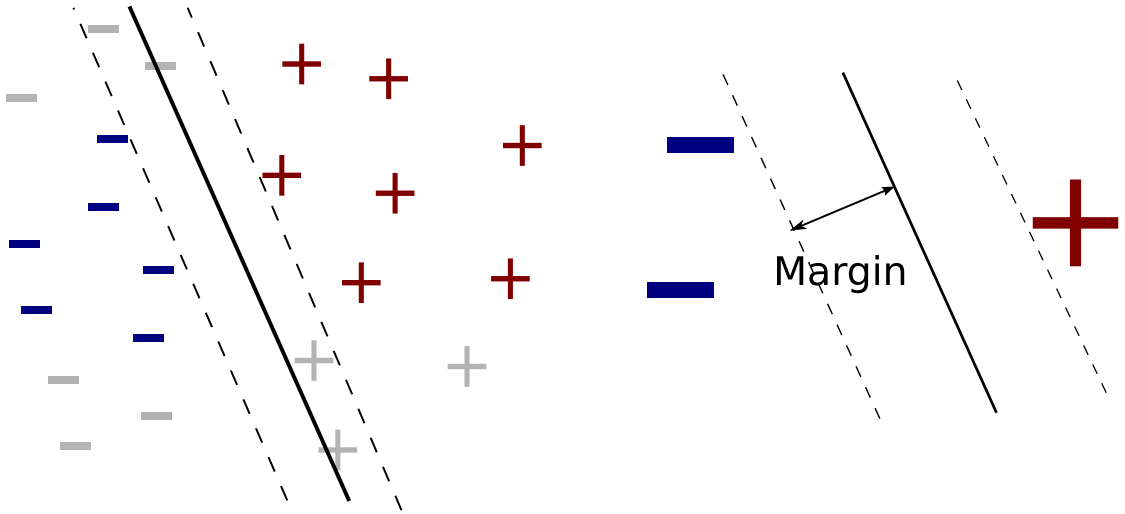
Training



Testing



Denote the value of the margin by γ



Maximizing Margin

- Intuitively maximizing margin makes sense
- More importantly, generalization error to unseen test data is proportional to the inverse of the margin

$$\epsilon \propto \frac{R^2}{\gamma^2 \times |\mathcal{D}|}$$

- **Perceptron**:
  - If a training set is separable by some margin, the perceptron will find a $\boldsymbol{w}$ that separates the data
  - However, the perceptron does not pick $\boldsymbol{w}$ to maximize the margin!
- **Logistic Regression**:
  - Not guaranteed to even separate data
  - softmax & log-loss is a margin-like optimization

Outline

Maximizing Margin

Let $\gamma > 0$

$$\max_{||\mathcal{W}|| \leq 1} \quad \gamma$$

such that:

$$\boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq \gamma$$

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

$$\text{and } \boldsymbol{y}' \in \mathcal{Y}, \ \boldsymbol{y}' \neq \boldsymbol{y}_t$$

- Note: algorithm still minimizes error if data is separable
- $||\mathcal{W}||$ is bound since scaling trivially produces larger margin
- $\mathcal{W} = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{|\mathcal{Y}|}]$

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{||\mathcal{W}|| \leq 1} \quad \gamma$$

such that:

$$\boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq \gamma$$

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

and $\boldsymbol{y}' \in \mathcal{Y}, \ \boldsymbol{y}' \neq \boldsymbol{y}_t$

=

Min Norm:

$$\min_{\mathcal{W}} \quad \frac{1}{2} ||\mathcal{W}||^2$$

such that:

$$\boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq 1$$

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

and $\boldsymbol{y}' \in \mathcal{Y}, \ \boldsymbol{y}' \neq \boldsymbol{y}_t$

- Instead of fixing $||\mathcal{W}||$ we fix the margin $\gamma = 1$
- Re-parameterize $\mathcal{W}' = \mathcal{W}/\gamma \rightarrow \|\mathcal{W}'\| = \|\mathcal{W}\|/\gamma \rightarrow \gamma = \frac{\|\mathcal{W}\|}{\|\mathcal{W}'\|} = \frac{1}{\|\mathcal{W}'\|}$.

Support Vector Machines

$$\mathcal{W} = \arg\min_{\mathcal{W}} \frac{1}{2}||\mathcal{W}||^2$$

such that:

$$\boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq 1$$

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D} \text{ and } \boldsymbol{y}' \in \mathcal{Y},\ \boldsymbol{y}' \neq \boldsymbol{y}_t$$

- Quadratic programming problem – a well known convex optimization problem
- Can be solved with many techniques.

Support Vector Machines

What if data is not separable?

$$\mathcal{W} = \arg\min_{\mathcal{W},\xi} \ \frac{1}{2}||\mathcal{W}||^2 + C\sum_{t=1}\xi_t$$

such that:

$$\boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq 1 - \xi_t \text{ and } \xi_t \geq 0$$

$$\forall(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D} \text{ and } \boldsymbol{y}' \in \mathcal{Y},\ \boldsymbol{y}' \neq \boldsymbol{y}_t$$

ξ_t: trade-off between margin per example and $\|\mathcal{W}\|$
Larger C = more examples correctly classified

Support Vector Machines

$$\mathcal{W} = \arg\min_{\mathcal{W},\xi} \ \frac{\lambda}{2}||\mathcal{W}||^2 + \sum_{t=1} \xi_t \qquad \lambda = \frac{1}{C}$$

such that:

$$\boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq 1 - \xi_t, \ \ \forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D} \text{ and } \boldsymbol{y}' \in \mathcal{Y}, \ \boldsymbol{y}' \neq \boldsymbol{y}_t$$

Support Vector Machines

$$\mathcal{W} = \arg\min_{\mathcal{W},\xi} \ \frac{\lambda}{2}||\mathcal{W}||^2 + \sum_{t=1} \xi_t \qquad \lambda = \frac{1}{C}$$

such that:

$$\boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq 1 - \xi_t, \ \ \forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D} \text{ and } \boldsymbol{y}' \in \mathcal{Y}, \ \boldsymbol{y}' \neq \boldsymbol{y}_t$$

$$= \ \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq 1 - \xi_t, \ \ \forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$$

Support Vector Machines

$$\mathcal{W} = \arg\min_{\mathcal{W},\xi} \ \frac{\lambda}{2}||\mathcal{W}||^2 + \sum_{t=1} \xi_t \qquad \lambda = \frac{1}{C}$$

such that:

$$\boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq 1 - \xi_t, \ \ \forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D} \text{ and } \boldsymbol{y}' \in \mathcal{Y}, \ \boldsymbol{y}' \neq \boldsymbol{y}_t$$

$$= \ \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) \geq 1 - \xi_t, \ \ \forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$$

$$= \ \xi_t \geq 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t), \ \ \forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$$

Support Vector Machines

$$\mathcal{W} = \arg\min_{\mathcal{W},\xi} \ \frac{\lambda}{2}||\mathcal{W}||^2 + \sum_{t=1} \xi_t$$

such that:

$$\xi_t \geq 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t), \ \ \forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$$

Hinge loss

If $\mathcal{W}$ classifies $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ with margin 1, penalty $\xi_t = 0$ (by def'n $\xi_t \geq 0$)
Otherwise penalty $\xi_t = 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)$

$$L(\mathcal{W}; (\boldsymbol{x}_t, \boldsymbol{y}_t)) = \max\ (0, 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t))$$

Support Vector Machines

$$\mathcal{W} = \arg\min_{\mathcal{W},\xi} \ \frac{\lambda}{2}||\mathcal{W}||^2 + \sum_{t=1} \xi_t$$

such that:

$$\xi_t \geq 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t), \ \ \forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$$

Hinge loss equivalent

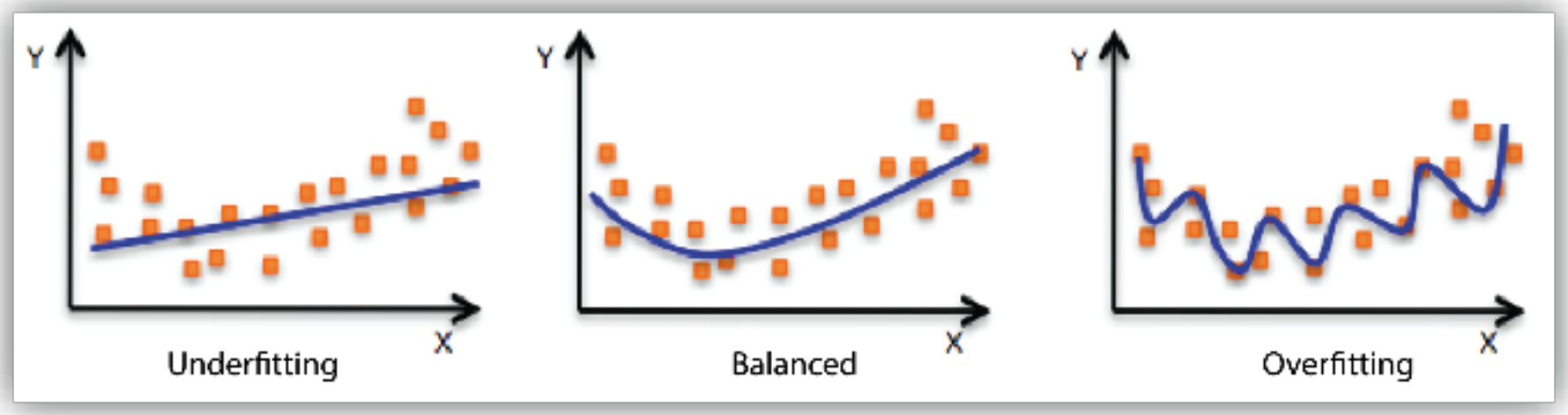
$$\mathcal{W} = \arg\min_{\mathcal{W}} \ \sum_{t=1} L((\boldsymbol{x}_t, \boldsymbol{y}_t); \mathcal{W}) \ + \ \frac{\lambda}{2}||\mathcal{W}||^2$$

$$= \arg\min_{\mathcal{W}} \ \left(\sum_{t=1} \max\left(0, 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{w}_{\boldsymbol{y}'} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)\right) \right) \ + \ \frac{\lambda}{2}||\mathcal{W}||^2$$

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

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

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

- Gaussian prior (ℓ_2), promotes smaller weights:

$$\Omega(\mathcal{W}) = \|\mathcal{W}\|_2^2 = \sum_i \mathcal{W}_i^2.$$

- Laplacian prior (ℓ_1), promotes sparse weights!

$$\Omega(\mathcal{W}) = \|\mathcal{W}\|_1 = \sum_i |\mathcal{W}_i|$$

Logistic Regression with ℓ_2 Regularization

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

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

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

- We know $\nabla_{\mathcal{W}} L(\mathcal{W}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$
- Just need $\nabla_{\mathcal{W}} \frac{\lambda}{2} \|\mathcal{W}\|^2 = \lambda \mathcal{W}$

Support Vector Machines

Hinge-loss formulation: ℓ_2 regularization already happening!

$$
\begin{aligned}
\mathcal{W} &= \arg\min_{\mathcal{W}} \sum_{t=1} L(\mathcal{W}; (\boldsymbol{x}_t, \boldsymbol{y}_t)) + \lambda\Omega(\mathcal{W}) \\
&= \arg\min_{\mathcal{W}} \sum_{t=1} \max\left(0, 1 + \max_{\boldsymbol{y} \neq \boldsymbol{y}_t} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)\right) + \lambda\Omega(\mathcal{W}) \\
&= \arg\min_{\mathcal{W}} \sum_{t=1} \max\left(0, 1 + \max_{\boldsymbol{y} \neq \boldsymbol{y}_t} \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)\right) + \frac{\lambda}{2}\|\mathcal{W}\|^2
\end{aligned}
$$

$\uparrow$ SVM optimization $\uparrow$

SVMs vs. Logistic Regression

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

SVMs vs. Logistic Regression

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

SVMs/hinge-loss: $\max\ (0, 1 + \max_{\boldsymbol{y} \neq \boldsymbol{y}_t}\ (\boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)))$

$$\mathcal{W} = \arg\min_{\mathcal{W}} \sum_{t=1} \max\ (0, 1 + \max_{\boldsymbol{y} \neq \boldsymbol{y}_t}\ \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)) + \frac{\lambda}{2}\|\mathcal{W}\|^2$$

SVMs vs. Logistic Regression

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

SVMs/hinge-loss: $\max\ (0, 1 + \max_{\boldsymbol{y} \neq \boldsymbol{y}_t}\ (\boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)))$

$$\mathcal{W} = \arg\min_{\mathcal{W}} \sum_{t=1} \max\ (0, 1 + \max_{\boldsymbol{y} \neq \boldsymbol{y}_t}\ \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)) + \frac{\lambda}{2}\|\mathcal{W}\|^2$$

Logistic Regression/log-loss: $\log \sum_{\boldsymbol{y}'_t} \exp(\boldsymbol{w}_{\boldsymbol{y}'_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)$

$$\mathcal{W} = \arg\min_{\mathcal{W}} \sum_{t=1} \left(\log \sum_{\boldsymbol{y}'_t} \exp(\boldsymbol{w}_{\boldsymbol{y}'_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t)) - \boldsymbol{w}_{\boldsymbol{y}_t} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) \right) + \frac{\lambda}{2}\|\mathcal{W}\|^2$$

$$\mathcal{W} = \arg\min_{\mathcal{W}} \sum_{t=1} \left(\sum_{\boldsymbol{y}'_t} P(\boldsymbol{y}'_t|\boldsymbol{x}) - P(\boldsymbol{y}_t|\boldsymbol{x}) \right) + \frac{\lambda}{2}\|\mathcal{W}\|^2$$

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 ...

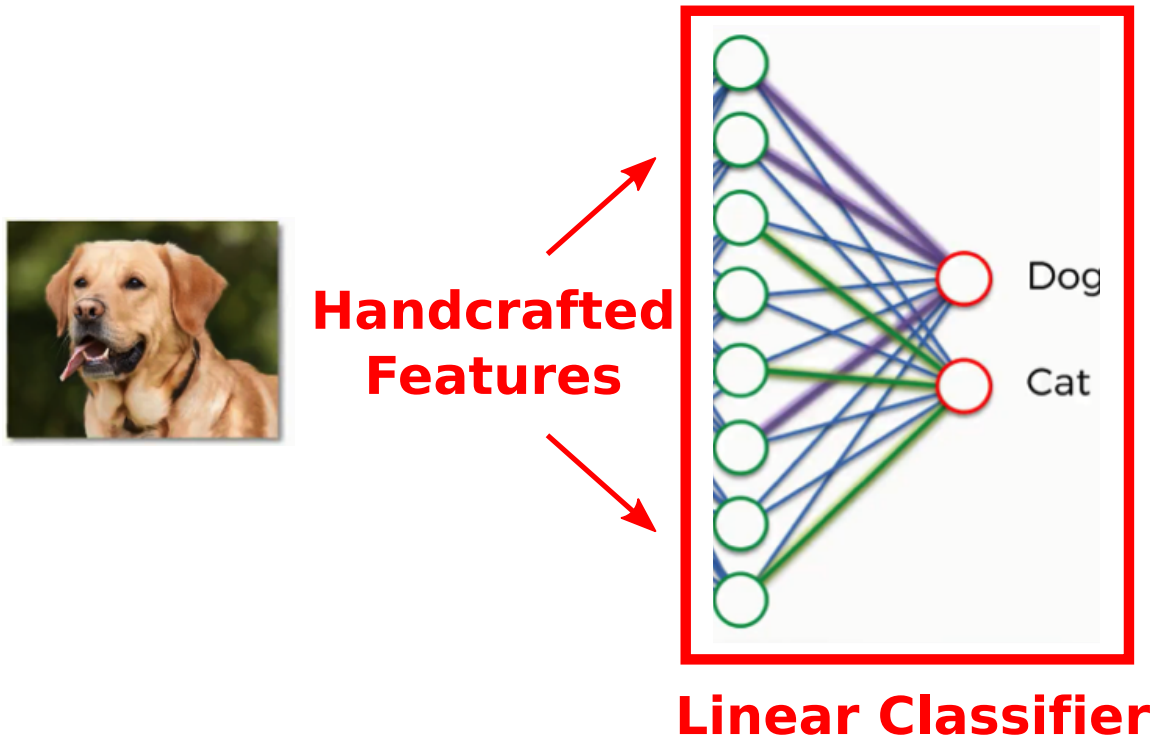
Linear Classifier

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
  - Sub-gradient descent
    - Needed for SVMs
    - Perceptron update is sub-gradient with no margin
  - Non-Linear Classifiers $\neq$ Neural Networks
    - K-Nearest neighbors
    - Kernel methods

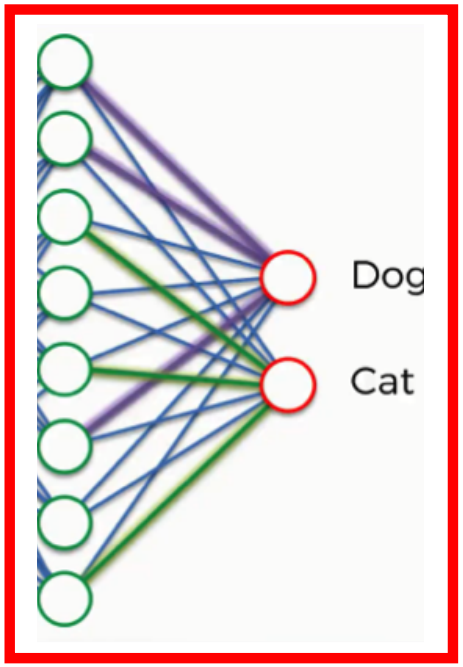
Outline

Reminder



$$y' = \textbf{argmax}\left(\mathbf{W}\phi(x)^\top + \boldsymbol{b}\right), \quad \mathbf{W} = \begin{bmatrix} \vdots \\ \boldsymbol{w_y} \\ \vdots \end{bmatrix}, \; \boldsymbol{b} = \begin{bmatrix} \vdots \\ \boldsymbol{b_y} \\ \vdots \end{bmatrix}.$$

No more ϕ



Linear Classifier

$$y' = \textbf{argmax}\left(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}\right), \; \boldsymbol{x} = \boldsymbol{\phi}(\boldsymbol{x})^\top, \; \mathbf{W} = \begin{bmatrix} \vdots \\ \boldsymbol{w_y} \\ \vdots \end{bmatrix}, \; \boldsymbol{b} = \begin{bmatrix} \vdots \\ \boldsymbol{b_y} \\ \vdots \end{bmatrix}$$

Linear classifiers as Matrix Multiplication

Let $\boldsymbol{w}_1 = [w_1^1, w_2^1, w_3^1]$, $\boldsymbol{w}_2 = [w_1^2, w_2^2, w_3^2]$, $\boldsymbol{w}_3 = [w_1^3, w_2^3, w_3^3]$ and $\boldsymbol{x} = [x_1, x_2, x_3]$

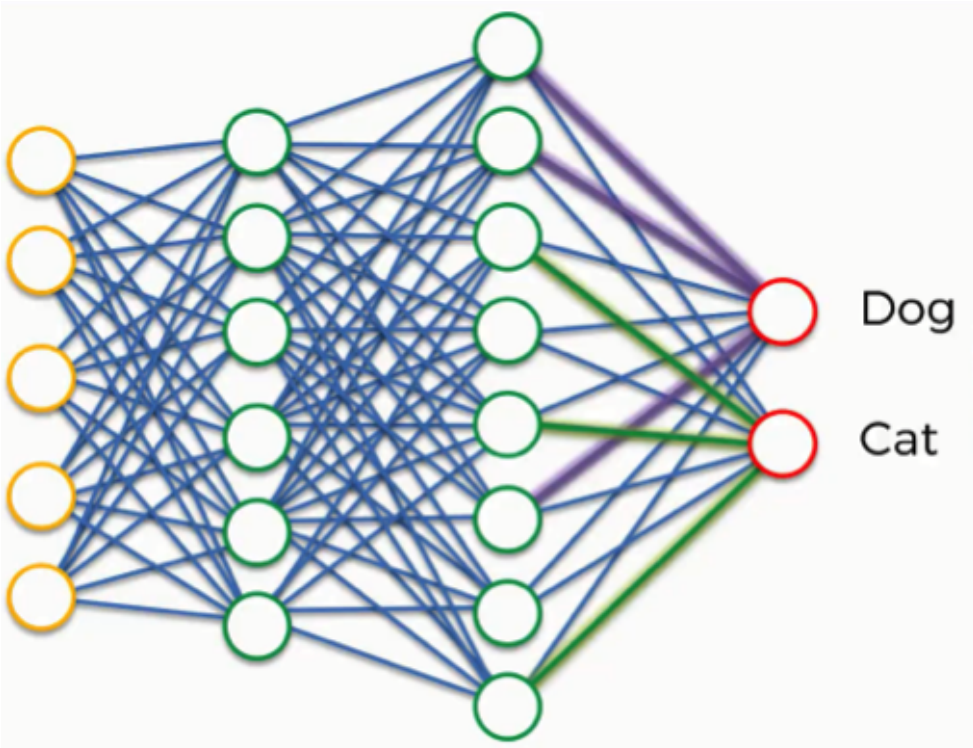
$$\mathbf{W}\boldsymbol{x} + \boldsymbol{b} = \begin{bmatrix} w_1^1 & w_2^1 & w_3^1 \\ w_1^2 & w_2^2 & w_3^2 \\ w_1^3 & w_2^3 & w_3^3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$= \begin{bmatrix} (w_1^1 \times x_1) + (w_2^1 \times x_2) + (w_3^1 \times x_3) \\ (w_1^2 \times x_1) + (w_2^2 \times x_2) + (w_3^2 \times x_3) \\ (w_1^3 \times x_1) + (w_2^3 \times x_2) + (w_3^3 \times x_3) \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

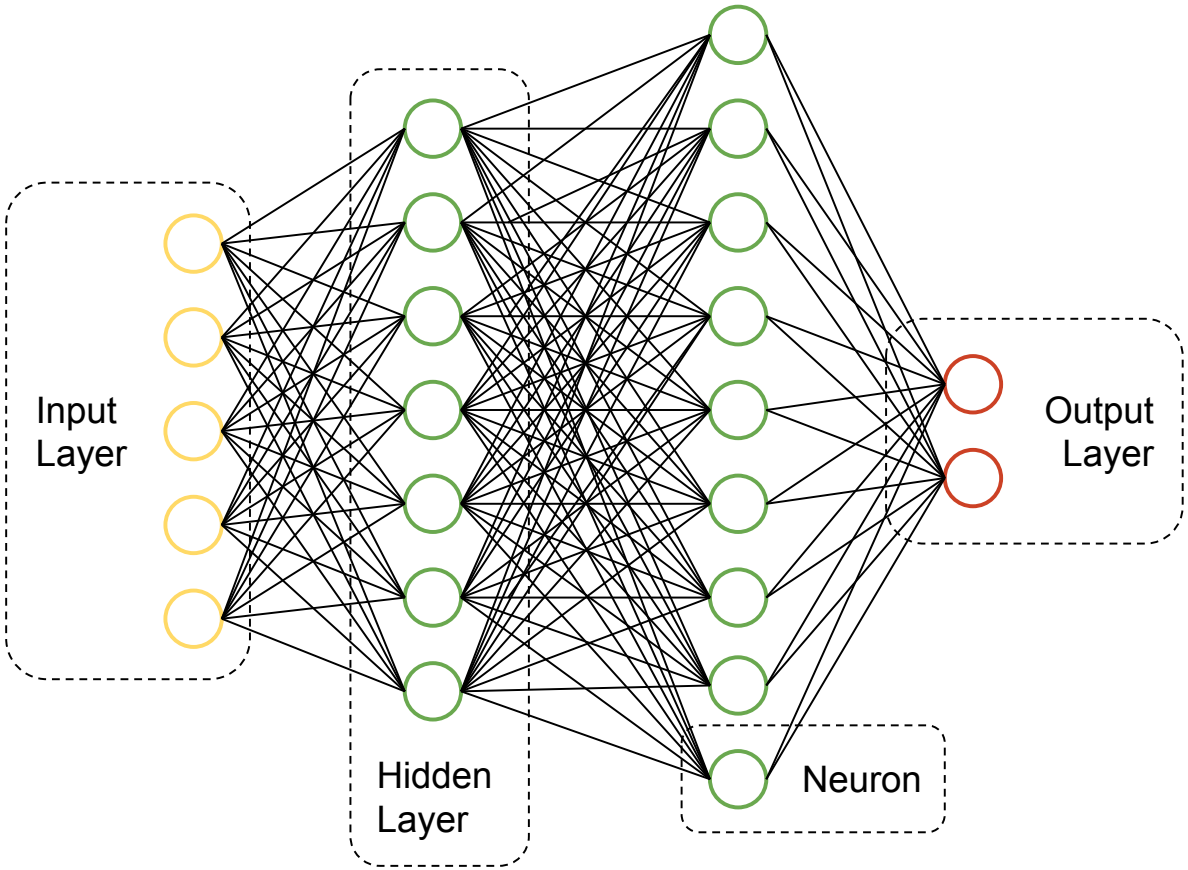
$$= \begin{bmatrix} (w_1^1 \times x_1) + (w_2^1 \times x_2) + (w_3^1 \times x_3) + b_1 \\ (w_1^2 \times x_1) + (w_2^2 \times x_2) + (w_3^2 \times x_3) + b_2 \\ (w_1^3 \times x_1) + (w_2^3 \times x_2) + (w_3^3 \times x_3) + b_3 \end{bmatrix}$$

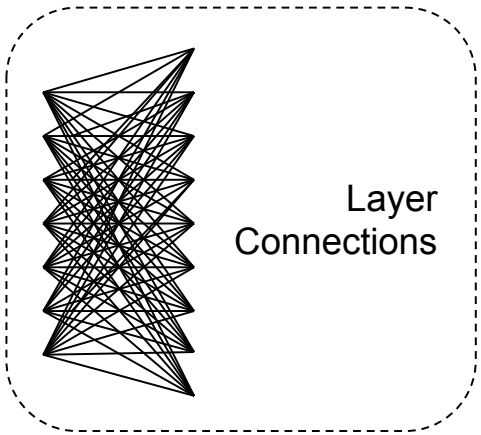
$$= \begin{bmatrix} \boldsymbol{w}_1 \cdot \boldsymbol{x} + b_1 \\ \boldsymbol{w}_2 \cdot \boldsymbol{x} + b_2 \\ \boldsymbol{w}_3 \cdot \boldsymbol{x} + b_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{y}_2 \\ \boldsymbol{y}_3 \end{bmatrix}$$

On to neural networks!



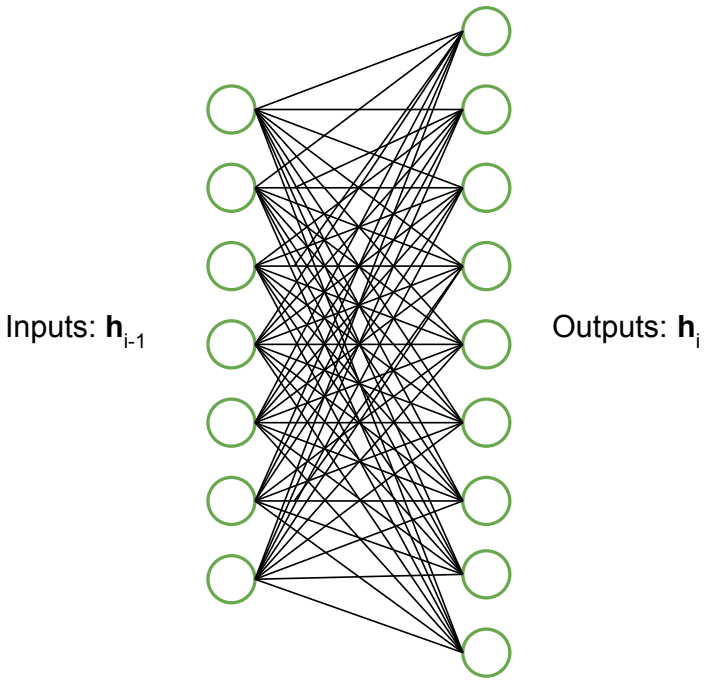
Neurons, Layers and Connections





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

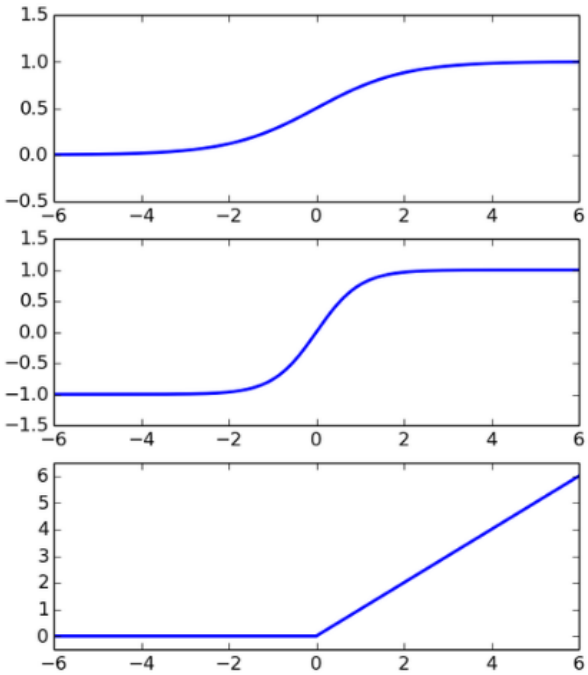
Hidden Layer Connections



- Let $\boldsymbol{h}_i \in \mathbb{R}^{D_i}$ be the i^{th} hidden layer with D_i dimensions/neurons
- $\boldsymbol{h}_i = f_i(\mathbf{W}_i \boldsymbol{h}_{i-1} + \boldsymbol{b}_i)$
- $\mathbf{W}_i \in \mathbb{R}^{D_i \times D_{i-1}}$ and $\boldsymbol{b}_i \in D_i$ are layer parameters
- f_i is the layer's (non-linear) activation function

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

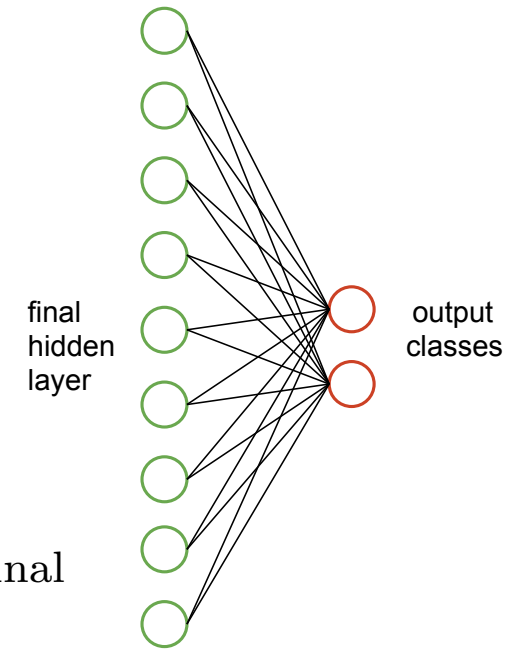
$$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

Rectified Linear

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

From Hughes and Correll 2016

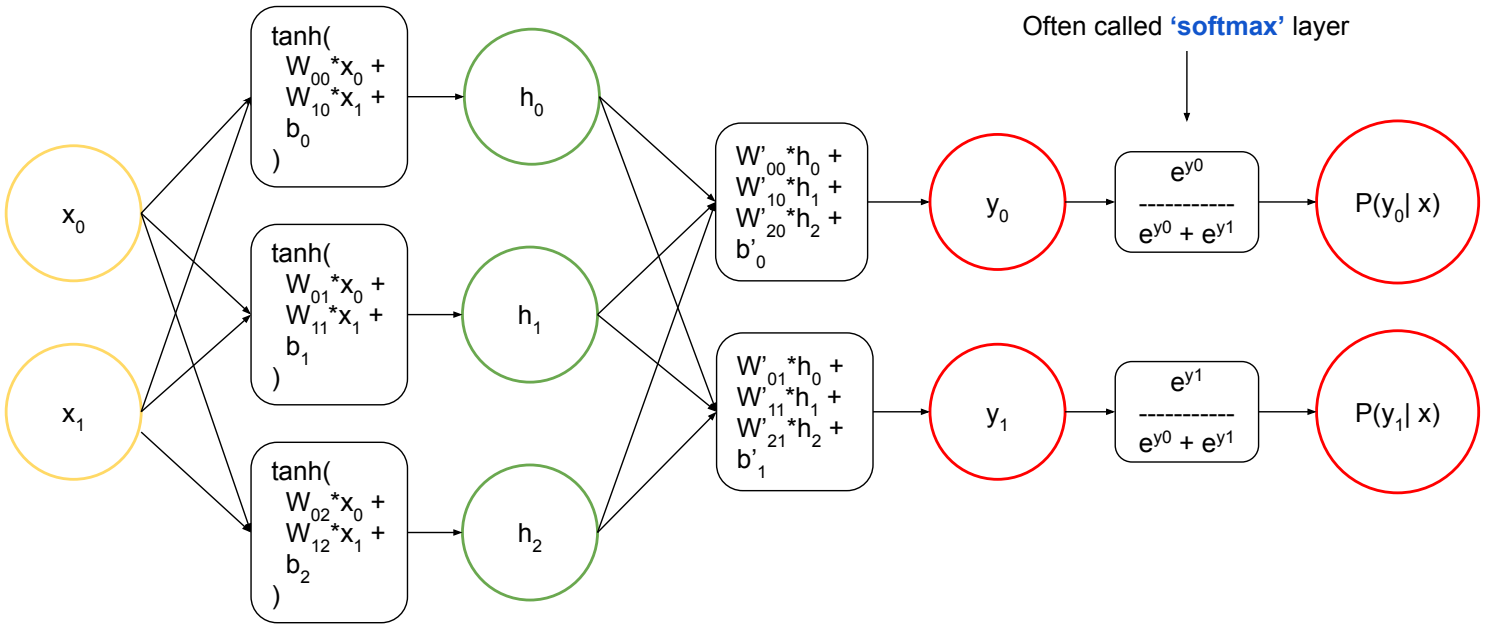
Output Layer



- This was first 2/3 of the lecture!
- $\boldsymbol{y}' = \textbf{argmax}\, \boldsymbol{y}$; where $\boldsymbol{y} = \mathbf{W}_{\text{final}} \boldsymbol{h}_{\text{final}-1} + \boldsymbol{b}_{\text{final}}$
- I.e., $\mathbf{W}_{\text{final}} = \mathcal{W}$ and $\boldsymbol{h}_{\text{final}-1} = \boldsymbol{\phi}(\boldsymbol{x})$
- $\boldsymbol{y}_i$ often called the logit of $\boldsymbol{y}_i$ or written $\text{logit}(\boldsymbol{y})$
- Various models correspond to different loss functions $L(\mathcal{W}; \mathcal{D})$
  - Logistic regression: log-loss/cross-entropy via softmax $\frac{e^{\text{logit}(\boldsymbol{y})}}{\sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\text{logit}(\boldsymbol{y}')}}$
  - SVMs: hinge-loss

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 $\boldsymbol{y} = \mathbf{W}'\boldsymbol{h} + \boldsymbol{b}'$ with $\mathbf{W}' \in \mathbb{R}^{2\times 3}$ and $\boldsymbol{b}' \in \mathbb{R}^2$
- Log-loss (cross-entropy):
  - $L(\mathcal{W}; (\boldsymbol{x}, \boldsymbol{y})) = -\log(P(\boldsymbol{y}|\boldsymbol{x})) = -\log \frac{e^{\text{logit}(\boldsymbol{y})}}{\sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\text{logit}(\boldsymbol{y}')}}$



Neural Networks So Far

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

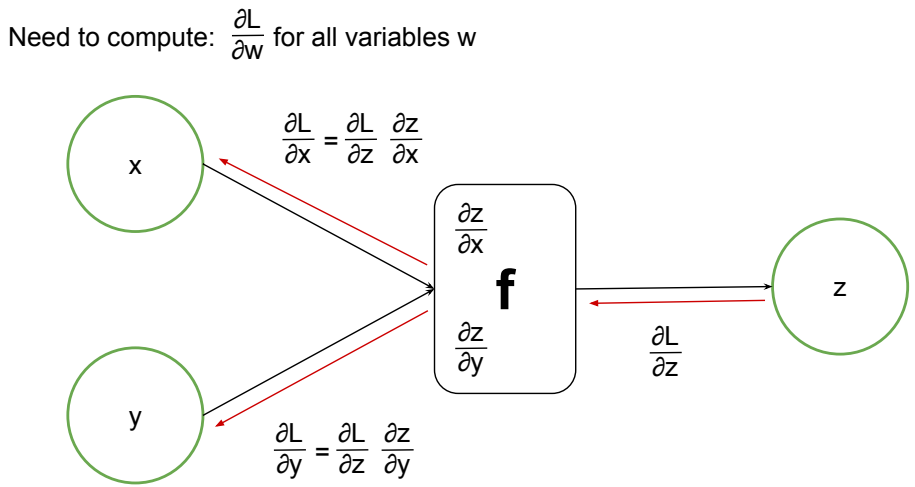
Neural Networks: Optimization

- 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))

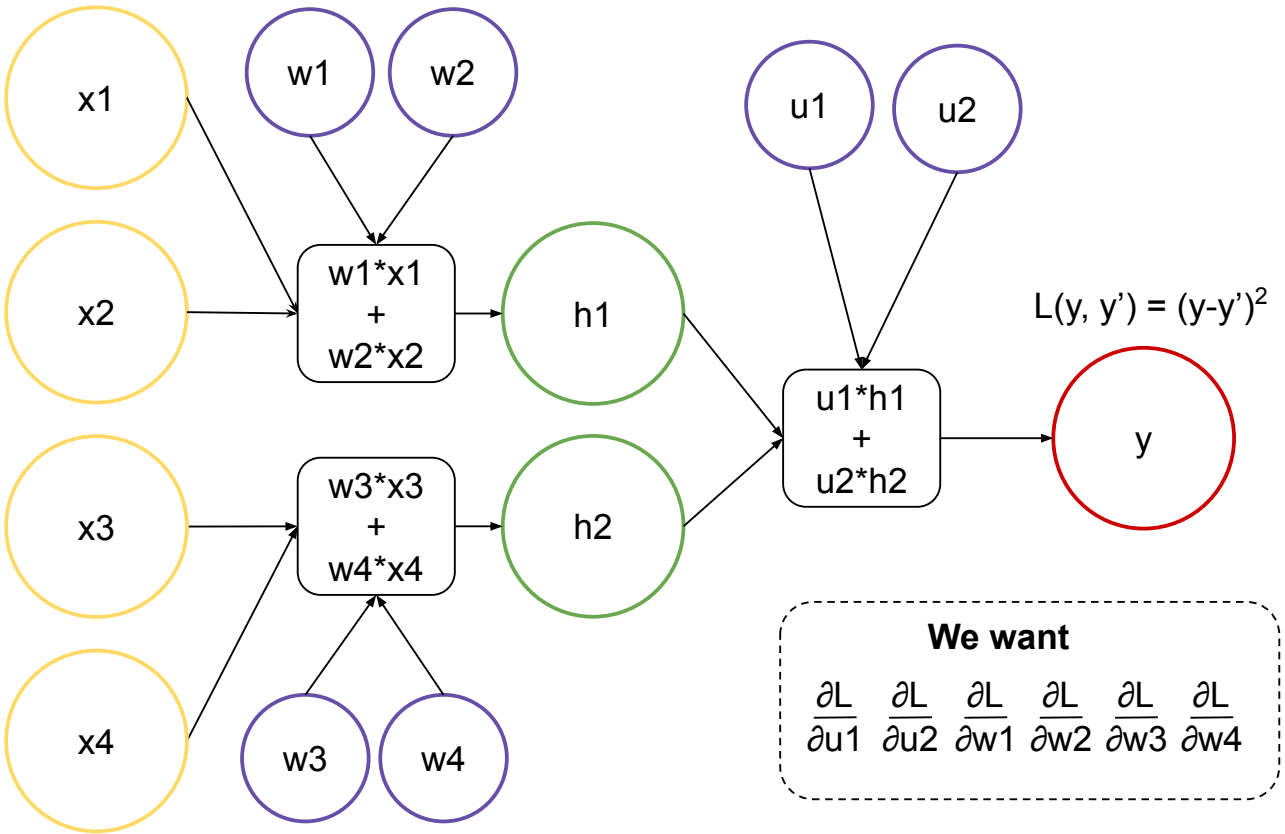
Backpropagation and the Chain Rule

- We need to compute $\nabla_{\mathcal{W}} L(\mathcal{W}; \mathcal{D}) = [\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots], \forall w_i \in \mathcal{W}$
  - For linear classifiers, $\mathcal{W}$ were feature weights
  - For NNs, $\mathcal{W}$ is the set of all weights, e.g., $\mathcal{W} = \{\mathbf{W}_i, \boldsymbol{b}_i \mid \forall i\}$
- Chain rule: $z = g(y)$ and $y = f(x)$, then $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$

- Example: Lets say $L = g(z)$ and $z = f(x, y)$
- Compute all partial derivatives for L, $\frac{\partial L}{\partial z}$, $\frac{\partial L}{\partial x}$, $\frac{\partial L}{\partial y}$



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

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

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

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

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

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

$$\frac{\partial h2}{\partial w3} = x3 \qquad \frac{\partial h1}{\partial w4} = x4$$

$$\frac{\partial h1}{\partial x3} = w3 \qquad \frac{\partial h1}{\partial x4} = w4$$

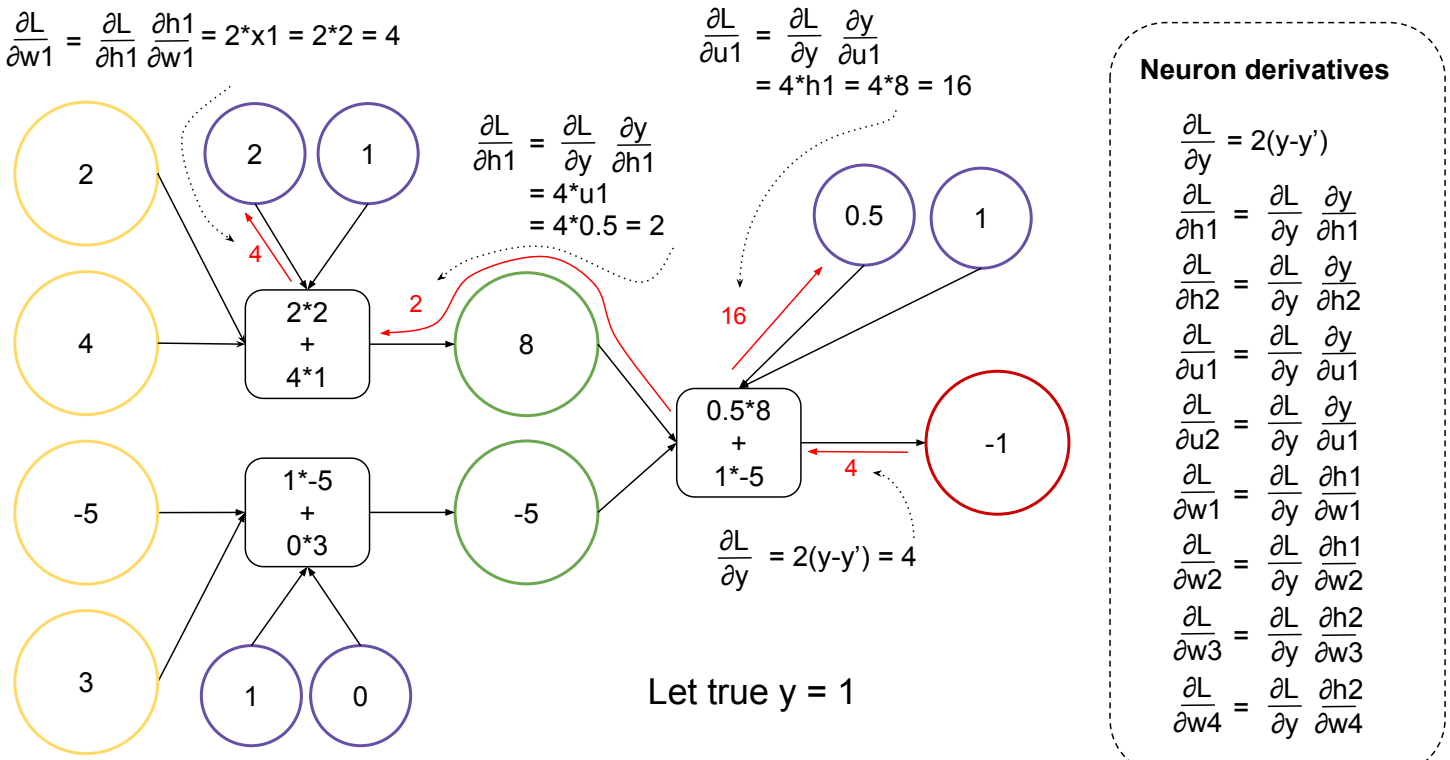
Full derivation examples

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

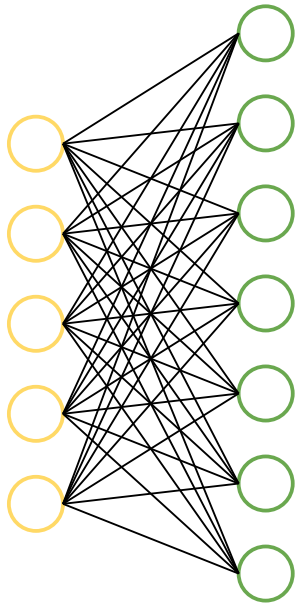
$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial h1}\frac{\partial h1}{\partial w1} = 2(y-y')*u1*x1$$

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

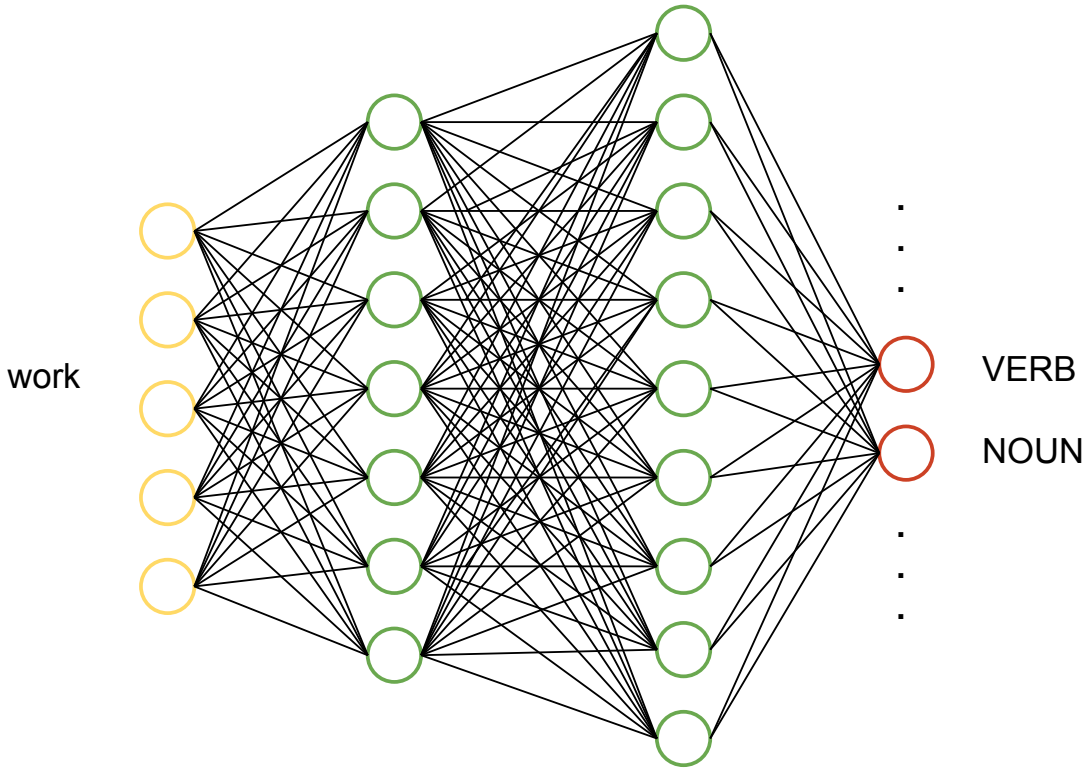


Input layer



- $\boldsymbol{x} \in \mathbb{R}^D$
- What is this for language?
- Words are discreet
- Sparse or one-hot vectors used in linear classifiers?
  - Parameter sparsity and computational bottlenecks
  - Does not leverage flexibility of NNs
- Solution: Embrace the vector!

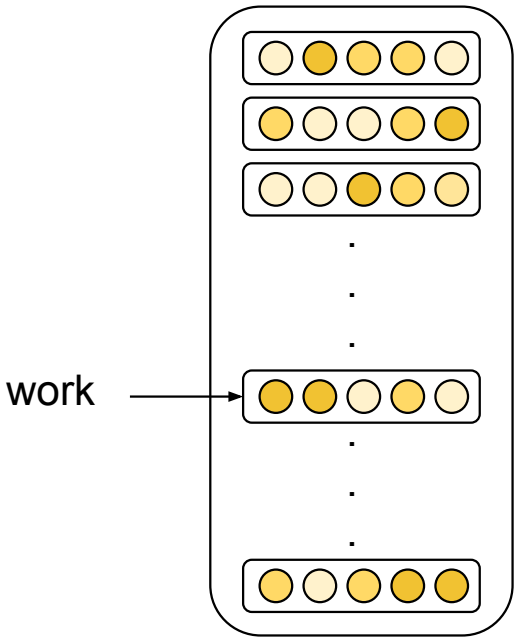
Input layer



- Consider classifying a word in isolation with a part-of-speech tag[3]
- Input is a word $\boldsymbol{x} \in \mathbb{R}^D$
- There is a fixed a finite vocabulary $\mathcal{V}$, i.e., $\boldsymbol{x} \in \mathcal{V}$

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

Input layer = Embedding layer



- Input is a word $\boldsymbol{x} \in \mathbb{R}^D$ for all $\boldsymbol{x} \in \mathcal{V}$
- We store these in a $|\mathcal{V}| \times D$ look up table
  - These are the model *word embeddings*
  - AKA embedding layer; word look-up table; ...

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

Example Static Embedding Layer: Word2Vec

- Corpus $\mathcal{C} = \{\mathcal{X}_1, \ldots, \mathcal{X}_{|\mathcal{C}|}\}$
- With sentences $\mathcal{X} = \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{|\mathcal{X}|}$
- Vocab $\mathcal{V} = \{\boldsymbol{x}_i | \boldsymbol{x}_i \in \mathcal{X} \text{ and } \mathcal{X} \in \mathcal{C}\}$
- Goal: learn vector/embedding $\boldsymbol{x}_i$ for all $\boldsymbol{x}_i \in \mathcal{V}$

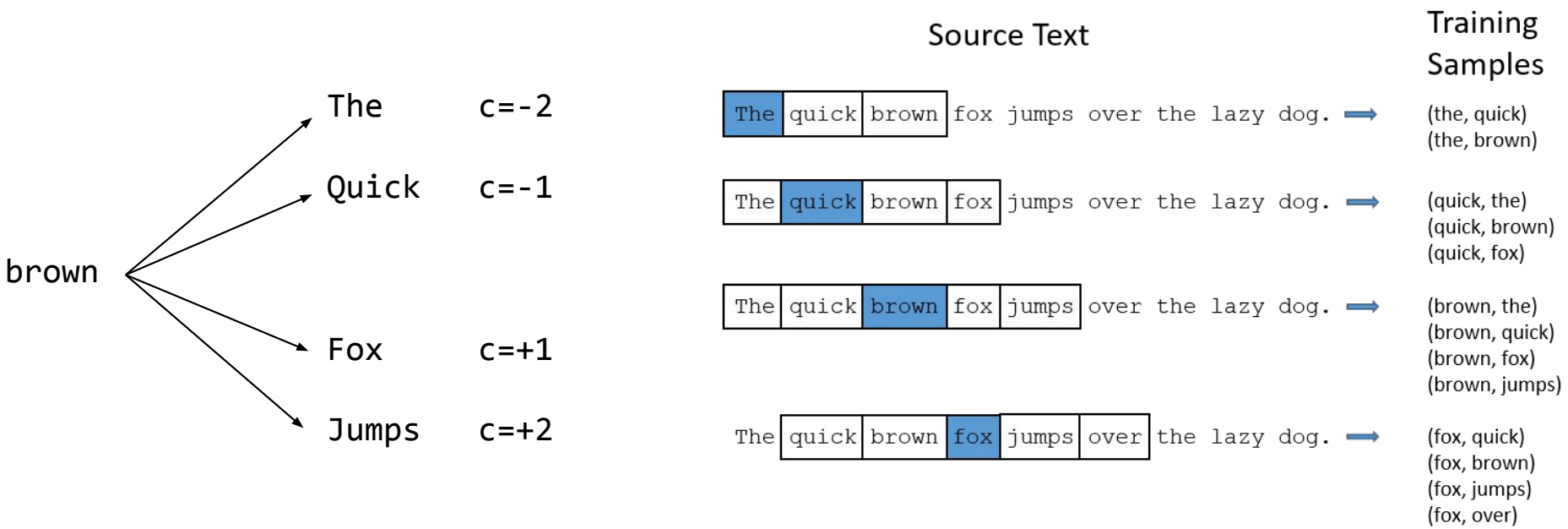
- word2vec (Mikolov et al. (2013))
  - Define two embeddings per word: $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$
  - $\boldsymbol{x}_i$ represents word as focus; $\boldsymbol{x}'_i$ as context
  - word2vec optimizes (SkipGram model):

$$\sum_{i}^{|\mathcal{C}|} \sum_{j}^{|\mathcal{X}|} \sum_{-c \leq k \leq c, k \neq 0} \log p(x_{j+k} | x_j) = \sum_{i}^{|\mathcal{C}|} \sum_{j}^{|\mathcal{X}|} \sum_{-c \leq k \leq c, k \neq 0} \log \frac{e^{\boldsymbol{x}_j \cdot \boldsymbol{x}'_{j+k}}}{\sum_{\boldsymbol{x}_l \in \mathcal{V}} e^{\boldsymbol{x}_j \cdot \boldsymbol{x}'_l}}$$

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

Example Static Embedding Layer: Word2Vec

$$\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0}\log p(x_{j+k}|x_j) = \sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0}\log \frac{e^{\boldsymbol{x}_j\cdot\boldsymbol{x}'_{j+k}}}{\sum_{\boldsymbol{x}_l\in\mathcal{V}} e^{\boldsymbol{x}_j\cdot\boldsymbol{x}'_l}}$$



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

Example Static Embedding Layer: Word2Vec

Re-writing the equation:

$$\left(\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0}\log e^{\boldsymbol{x}_j\cdot\boldsymbol{x}'_{j+k}}\right) - \left(\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0}\log \sum_{\boldsymbol{x}_l\in\mathcal{V}} e^{\boldsymbol{x}_j\cdot\boldsymbol{x}'_l}\right)$$

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

$$\left(\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0}\log e^{\boldsymbol{x}_j\cdot\boldsymbol{x}'_{j+k}}\right) - \left(\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0}\log \sum_{\boldsymbol{x}_l\in\mathcal{V}_s} e^{\boldsymbol{x}_j\cdot\boldsymbol{x}'_l}\right)$$

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

Example Static Embedding Layer: Word2Vec

$$\left(\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0}\log e^{\boldsymbol{x}_j\cdot\boldsymbol{x}'_{j+k}}\right) - \left(\sum_{i}^{|\mathcal{C}|}\sum_{j}^{|\mathcal{X}|}\sum_{-c\leq k\leq c, k\neq 0}\log \sum_{\boldsymbol{x}_l\in\mathcal{V}_s} e^{\boldsymbol{x}_j\cdot\boldsymbol{x}'_l}\right)$$

- Parameters of the model are $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$
- $\boldsymbol{x}_i$ are used as final word embeddings ($\boldsymbol{x}'_i$ usually discarded)
- Usually optimized with SGD

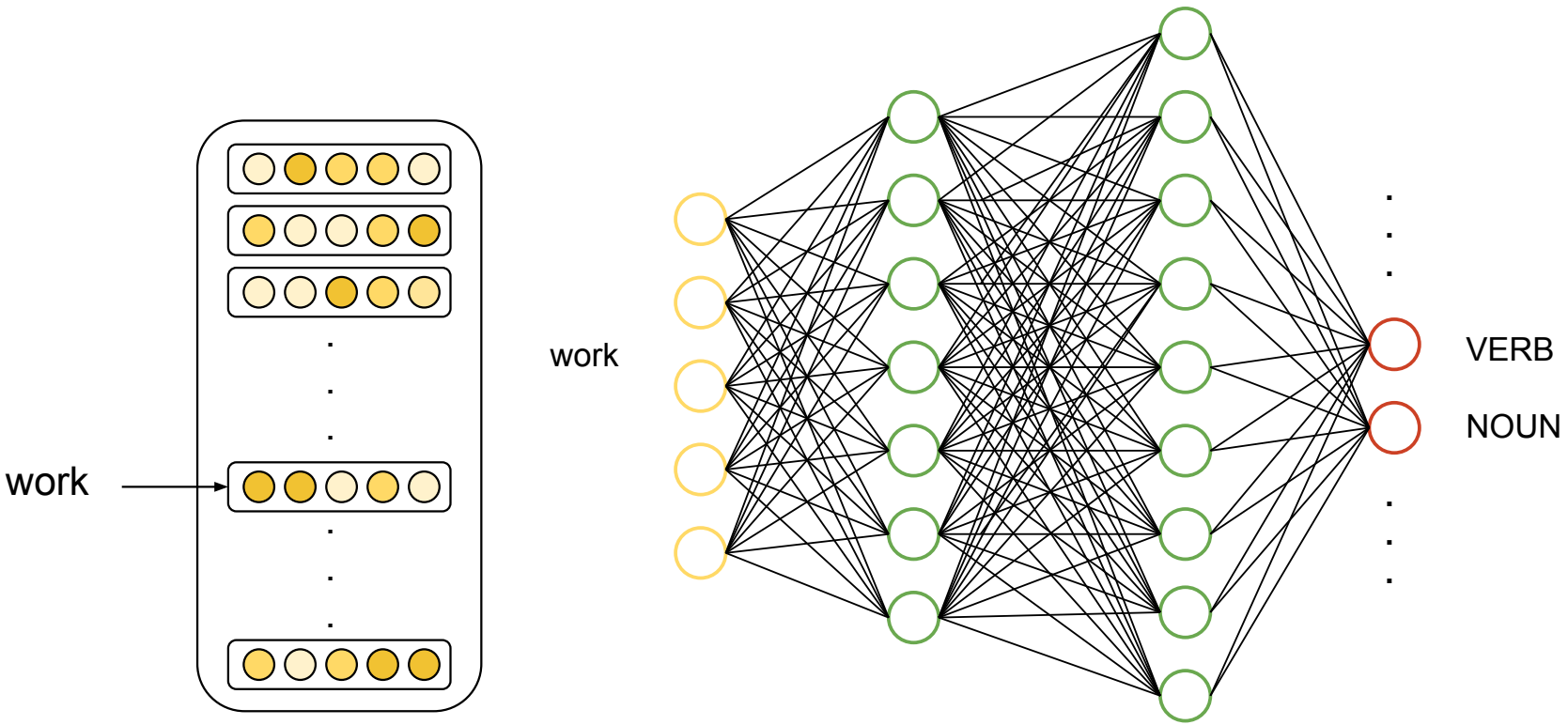
Fun word arithmetic artifact:

$$\boldsymbol{x}_{\text{Greece}} - (\boldsymbol{x}_{\text{Canada}} - \boldsymbol{x}_{\text{Ottawa}}) = \boldsymbol{x}_{\text{Athens}}$$

Embeddings via Language Models

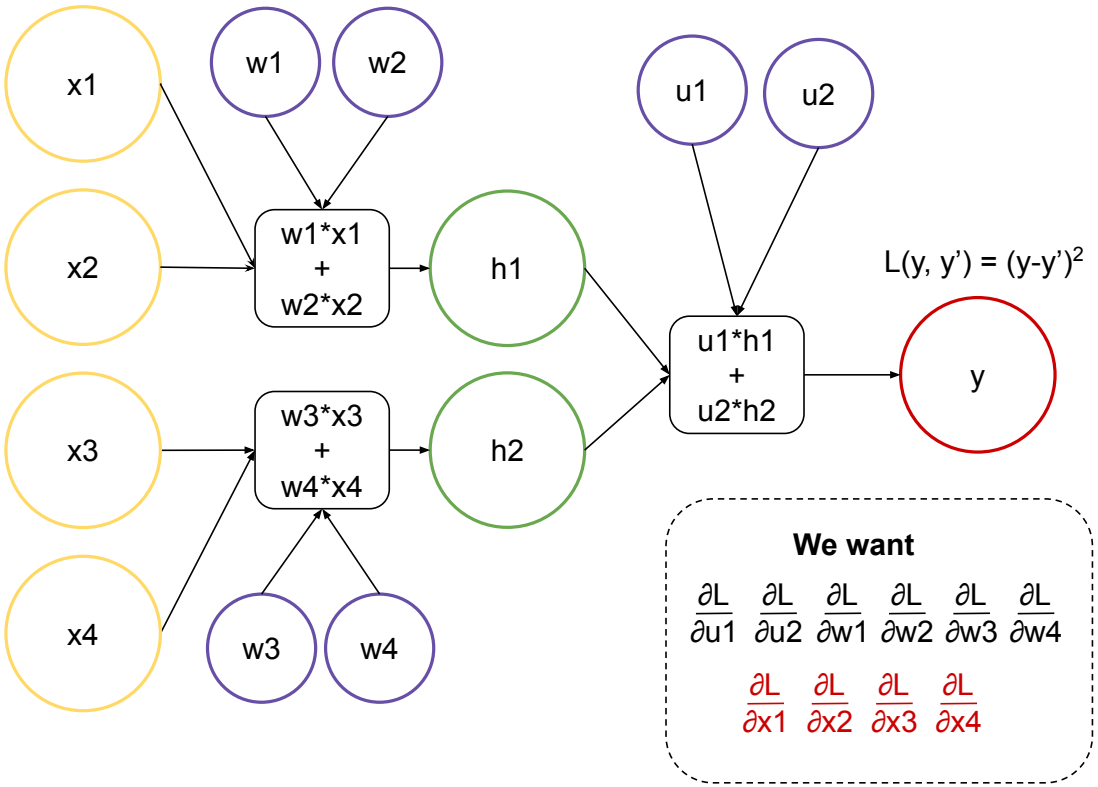
- word2vec is an example of a language model
- It models the probability of a word given a context
- Pre-trained contextual language models dominate NLP: ELMO, BERT, ROBERTA, XLNet, ..., GPT, Claude, Gemini, Llama, ...
- Transformed the field, business and potentially the economy
- Other lectures will cover RNNs, which is main building block of NN language models and LLMs

Input layer



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

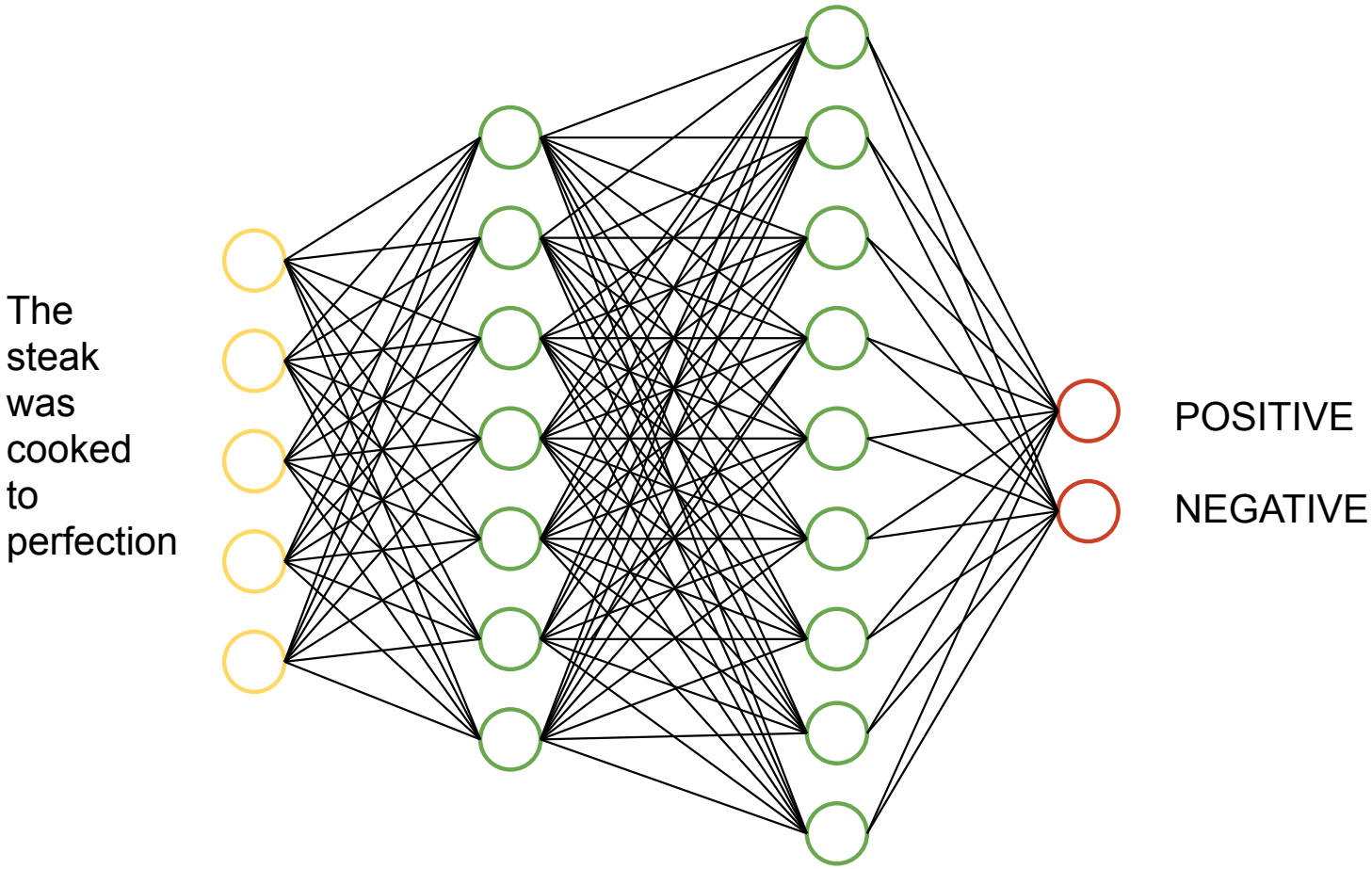
Dynamic Input layer



- Gradient now includes input neurons, $\frac{\partial L}{\partial \boldsymbol{x}_i}$
- Every value in the entire lookup table is a parameter!

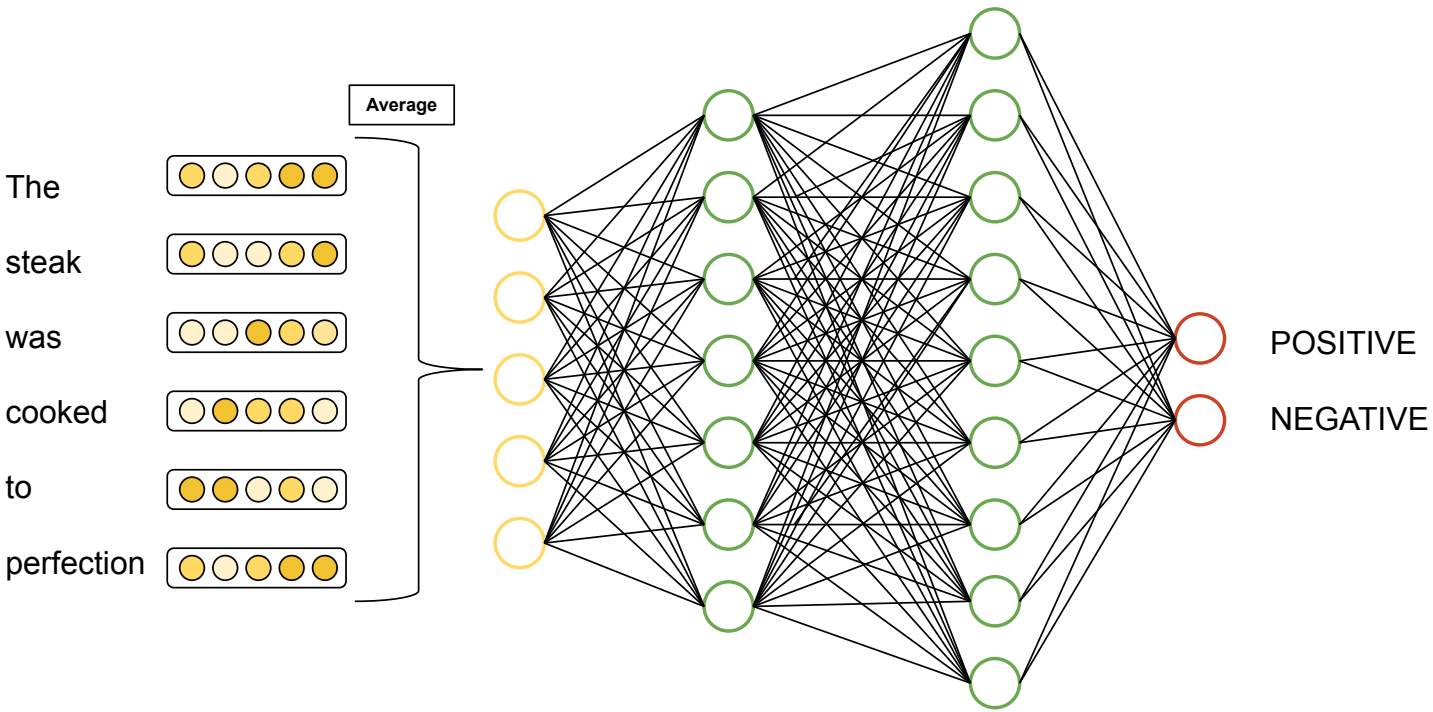
Variable Length Inputs

- 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



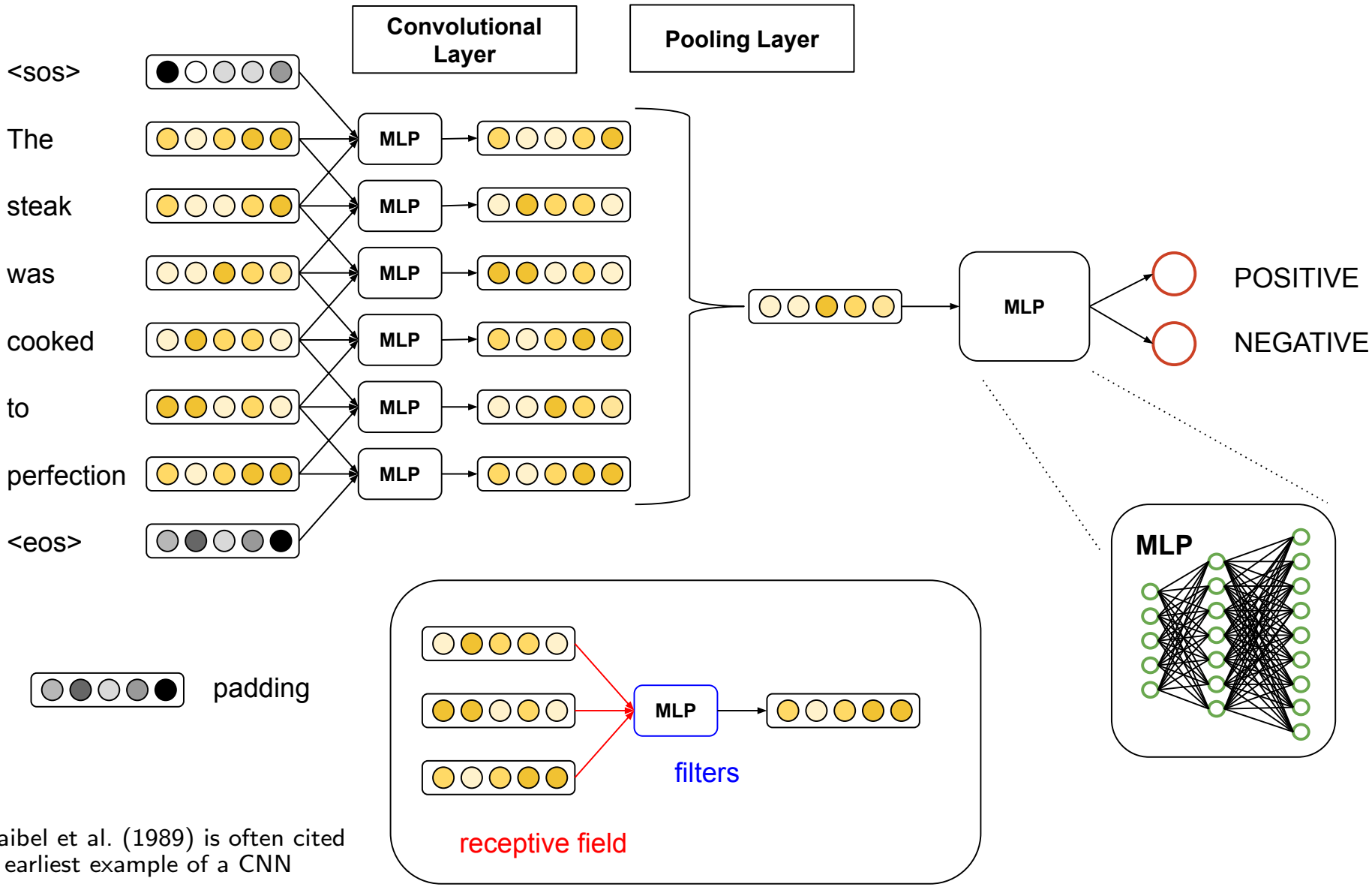
Variable Length Inputs: Options

1. Truncate document at fixed length K, $\boldsymbol{x} \in \mathbb{R}^{K \times D}$
2. Average embeddings (below), $\boldsymbol{x} \in \mathbb{R}^{D}$
3. convolutional and recurrent neural networks[4]



[4] RNNs next lecture

Convolutional Neural Networks

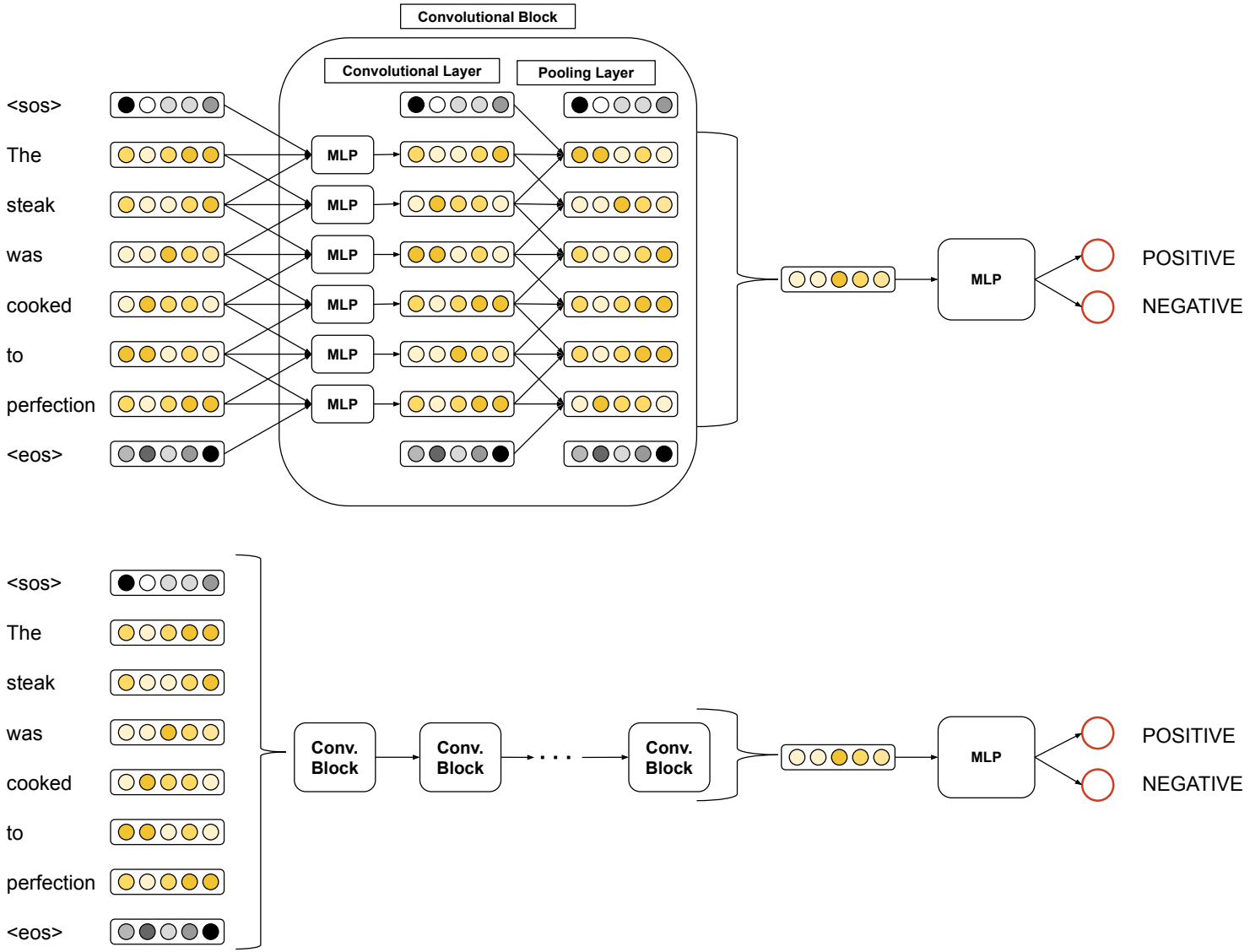


Waibel et al. (1989) is often cited as earliest example of a CNN

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

Deep Convolutional Neural Networks



Neural Network Summary

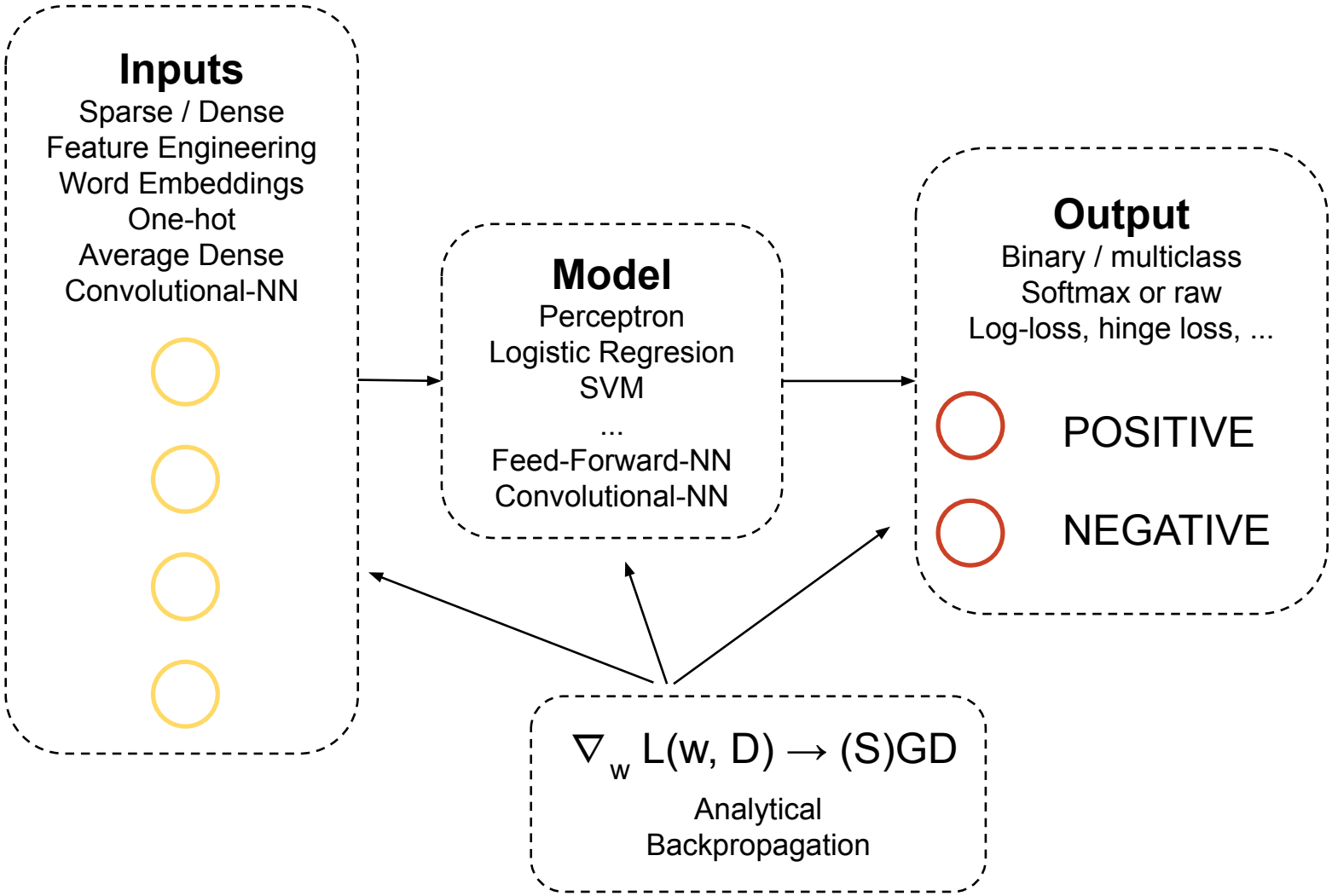
- Feed-forward Neural Networks
- Neurons, layers and connections
- Output layers and losses
- Back propagation
- Input layers
  - Static vs dynamic vs mixed
- High-level questions
  - Where does layer and network structure come from?
  - Why should I use neural networks?

Where Does Network Structure Come From?

- Hyperparamters: input/hidden dimensions; activation functions; ...
  - Usually empirical
  - Can largely be automated
- Deep Learning = lot's of layers
- Fully-connected/dense required?
  - No!
  - However, rarely does more specialized layer connections help
  - Any efficiency concerns lessened by modern architectures (GPU, TPU)

Main Points

The steak was cooked to perfection



Main Points in Words

- Sparse (binary) vs. dense (embeddings) features
- Optimization: Use gradient-based techniques
- Linear Classifiers
  - Usually sparse features with block representations
  - Loss functions define model (Log reg vs. SVMs)
  - Regularization necessary for good performance
- Neural Networks
  - Final layer = linear classifiers
  - Hidden layers = non-convex
  - Compute gradient with backpropagation
  - Input layer: static (e.g., word2vec) vs. dynamic (backprop)
  - Input layer: Usually dense look-up table

References I

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.