Classification

Ryan McDonald[1]

AthNLP

September, 2025

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

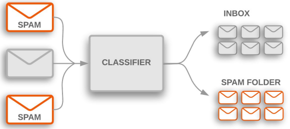
Classifiers

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." → 👎

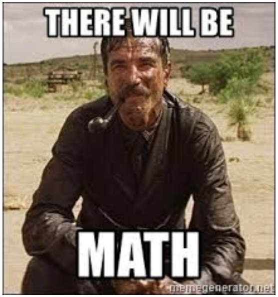


```
Set my alarm tomorrow for 10am   -> Alarm

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

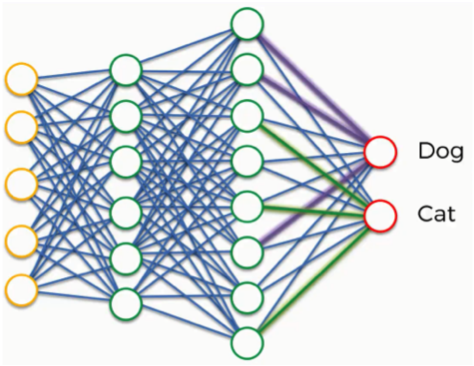
Linear Classifiers

This lecture is 1/2 about linear classifiers!

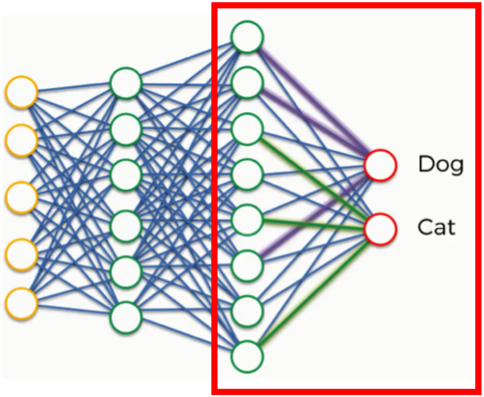
Why? It's 2025 and everybody uses neural networks.

- 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



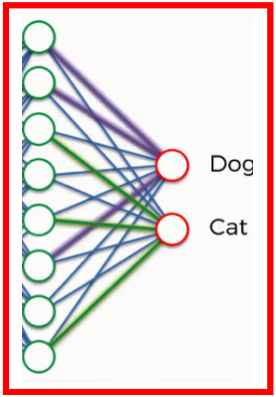
Linear Classifiers and Neural Networks



Linear Classifier

Linear Classifiers and Neural Networks

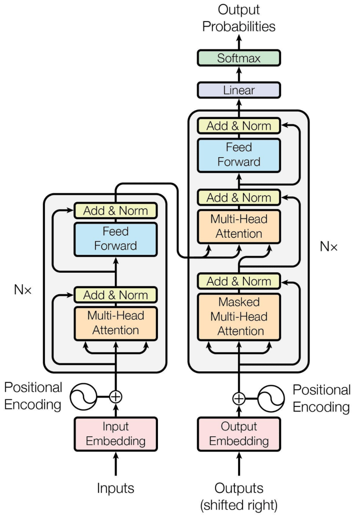
Handcrafted Features



Linear Classifier

Linear Classifiers and Generative AI

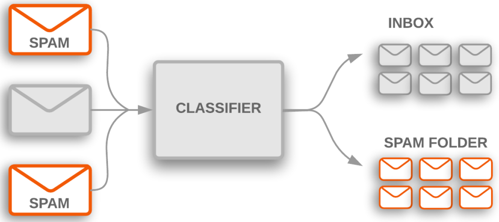
- **Transformers**: 99% of LLMs/GenAI
  - ChatGPT; GPT
  - Claude
  - Gemini
  - Llama
  - DeepSeek
  - xAI/Grok
- Last layer = linear classifier
- Last layer predicts next word/token
- I.e., last layer is a classifier!
- Also: Many other linear layers!



Binary Classification: Spam Detection

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

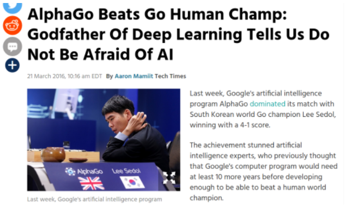
This is a **binary classification problem.**



Multiclass Classification: Topic Labeling

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

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



AlphaGo Beats Go Human Champ: Godfather Of Deep Learning Tells Us Do Not Be Afraid Of AI

21 March 2016, 10:16 am EDT By Aaron Mamiit Tech Times

Last week, Google's artificial intelligence program AlphaGo dominated its match with South Korean world Go champion Lee Sedol, winning with a 4-1 score.

The achievement stunned artificial intelligence experts, who previously thought that Google's computer program would need at least 10 more years before developing enough to be able to beat a human world champion.

Last week, Google's artificial intelligence program

$\longrightarrow$

sports
politics
technology
economy
weather
culture

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 $+1$**

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

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

$$P(-1|\heartsuit) = \frac{\text{count}(\heartsuit \text{ and } -1)}{\text{count}(\heartsuit)} = \frac{1}{1} = 1.0 \text{ vs. } P(+1|\heartsuit) = \frac{\text{count}(\heartsuit \text{ and } +1)}{\text{count}(\heartsuit)} = \frac{0}{1} = 0.0$$

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

1. **Terminology, notation and feature representations**
2. Linear Classifiers
    - Perceptron
    - Logistic Regression
3. Regularization
4. Dense Representations
5. Similarity-based Learning
6. Neural Networks

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** $h : \mathcal{X} \to \mathcal{Y}$ 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).

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 $\boldsymbol{\phi} : \mathcal{X} \to \mathbb{R}^D$

- $\boldsymbol{\phi}(\boldsymbol{x})$ 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 later

Examples

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

$$\phi_j(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains the word “interest”} \\ 0 & \text{otherwise} \end{cases}$$

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

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

$$\phi_j(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ ends in “ed”} \\ 0 & \text{otherwise} \end{cases}$$

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

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?

Outline

1. Terminology, notation and feature representations
2. **Linear Classifiers**
   - Perceptron
   - Logistic Regression
3. Regularization
4. Dense Representations
5. Similarity-based Learning
6. Neural Networks

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 $\boldsymbol{w}_{\boldsymbol{y}} \in \mathbb{R}^D$
    - i.e., one weight vector per output label $\boldsymbol{y}$

Linear Classifiers – Weights/Parameters

- ! Important Concept !
- $\boldsymbol{w_y}$ 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 $\boldsymbol{w_y}$
- 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[2] of a particular label is based on a linear combination of features and their weights, e.g., for each $\boldsymbol{y} \in \mathcal{Y}$

$$\text{score}(\boldsymbol{y}, \boldsymbol{x}) = \boldsymbol{w}_{\boldsymbol{y}} \cdot \boldsymbol{\phi}(\boldsymbol{x}) = \sum_i \boldsymbol{w}_{i,\boldsymbol{y}} \cdot \boldsymbol{\phi}_i(\boldsymbol{x})$$

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

$$\boldsymbol{y}' = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \text{score}(\boldsymbol{y}, \boldsymbol{x})$$

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

[2]Called *logit* in NNs.

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

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

Linear Classifiers – Bias Terms

- Often linear classifiers are presented as

$$\text{score}(\boldsymbol{y}, \boldsymbol{x}) \;\; = \;\; \boldsymbol{w_y} \cdot \phi(\boldsymbol{x}) + b_{\boldsymbol{y}}$$

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

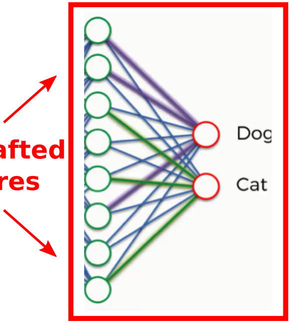
- This can be folded into $\phi(\boldsymbol{x})$ via a constant feature

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

- For now, we assume this for simplicity

Commonly Used Notation in Neural Networks

Handcrafted Features



Linear Classifier

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

Binary Linear Classifier

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

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

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:

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

Binary Linear Classifier

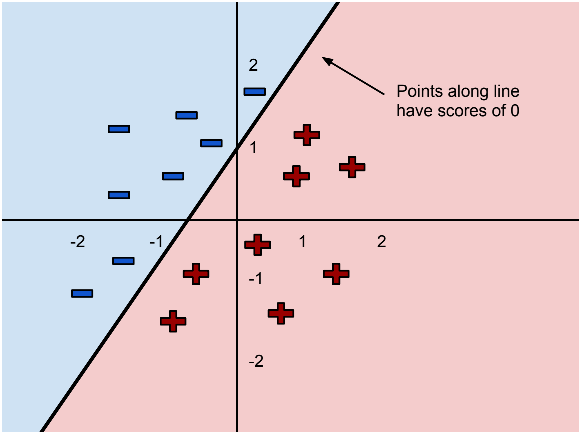
With binary labels ($\mathcal{Y} = \{\pm 1\}$) we often use a minimal parametrization:

$$\begin{aligned}
\boldsymbol{y}' &= \arg\max_{\boldsymbol{y}\in\{\pm1\}} \; \boldsymbol{w_y}\cdot\boldsymbol{\phi}(\boldsymbol{x}) + b_{\boldsymbol{y}} \\
&= \begin{cases} +1 & \text{if } \boldsymbol{w}_{+1}\cdot\boldsymbol{\phi}(\boldsymbol{x}) + b_{+1} > \boldsymbol{w}_{-1}\cdot\boldsymbol{\phi}(\boldsymbol{x}) + b_{-1} \\ -1 & \text{otherwise} \end{cases} \\
&= \begin{cases} +1 & \text{if } (\boldsymbol{w}_{+1} - \boldsymbol{w}_{-1})\cdot\boldsymbol{\phi}(\boldsymbol{x}) + (b_{+1} - b_{-1}) > 0 \\ -1 & \text{otherwise} \end{cases} \\
&= \text{sign}(\underbrace{(\boldsymbol{w}_{+1} - \boldsymbol{w}_{-1})}_{\boldsymbol{v}}\cdot\boldsymbol{\phi}(\boldsymbol{x}) + \underbrace{(b_{+1} - b_{-1})}_{c}).
\end{aligned}$$

That is: only half of the parameters are needed.

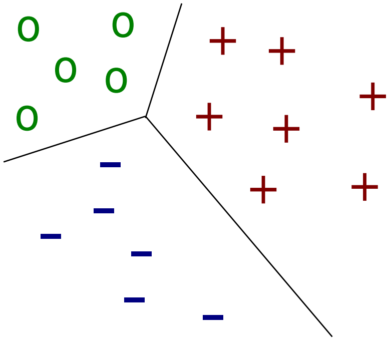
Binary Linear Classifier

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



Multiclass Linear Classifier

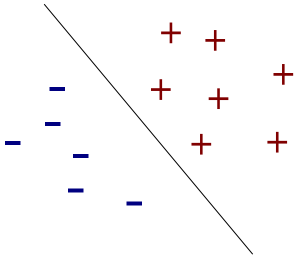
Defines regions of space.



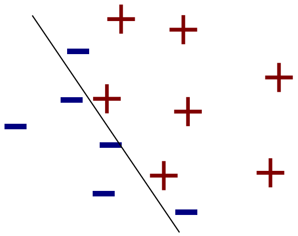
Linear Separability

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

Separable



Not Separable



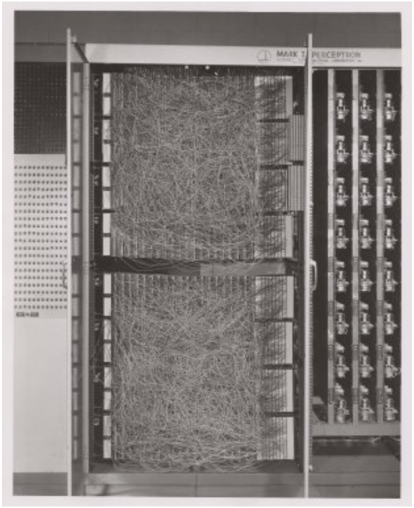
Learning

- 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

1. Terminology, notation and feature representations
2. **Linear Classifiers**
   - Perceptron
   - Logistic Regression
3. Regularization
4. Dense Representations
5. Similarity-based Learning
6. Neural Networks

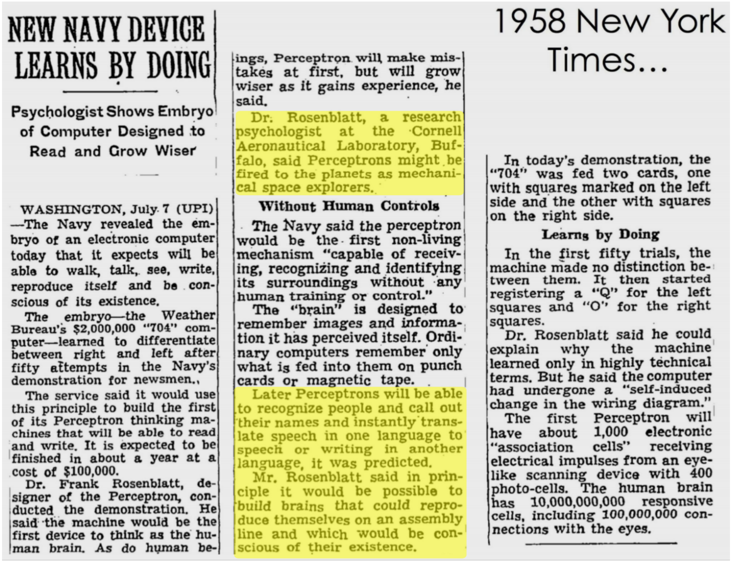
Perceptron (Rosenblatt, 1958)



(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...



1958 New York Times...

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.

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

$$\boldsymbol{y}' = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \ \text{score}(\boldsymbol{y}, \boldsymbol{x}_t)$$

  - 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

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

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

Perceptron Algorithm

```
input: labeled data $\mathcal{D}$
initialize $\mathcal{W} = \mathbf{0}$, i.e., $\boldsymbol{w_y} = \mathbf{0}$, $\forall \boldsymbol{y}$
repeat
    observe example $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$
    predict $\boldsymbol{y}' = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \ \mathrm{score}(\boldsymbol{y}, \boldsymbol{x}_t)$
    if $\boldsymbol{y}' \neq \boldsymbol{y}_t$ then
        update $\boldsymbol{w}_{\boldsymbol{y}_t} = \boldsymbol{w}_{\boldsymbol{y}_t} + \boldsymbol{\phi}(\boldsymbol{x}_t)$
        update $\boldsymbol{w}_{\boldsymbol{y}'} = \boldsymbol{w}_{\boldsymbol{y}'} - \boldsymbol{\phi}(\boldsymbol{x}_t)$
    end if
until stopping criterion[3]
output: model weights $\mathcal{W}$
```

[3] 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 are 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 are 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:

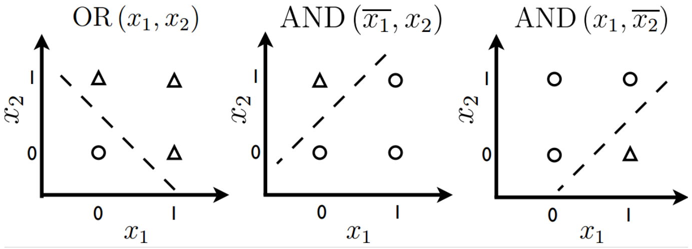
Theorem (Novikoff (1962))

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

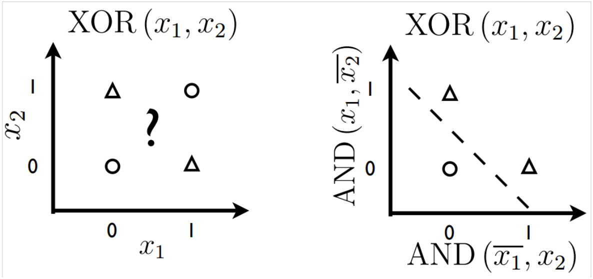
- Remember: the decision boundary is linear (linear classifier)
- It **can** solve linearly separable problems (OR, AND)



$\mathrm{OR}\,(x_1, x_2)$
$\mathrm{AND}\,(\overline{x_1}, x_2)$
$\mathrm{AND}\,(x_1, \overline{x_2})$
x_1
x_2
0
1

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):



$\mathrm{XOR}(x_1, x_2)$
x_2
1
0
?
0
1
x_1
$\mathrm{XOR}(x_1, x_2)$
$\mathrm{AND}(x_1, \overline{x_2})$
1
0
0
1
$\mathrm{AND}(\overline{x_1}, x_2)$

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

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

Outline

1. Terminology, notation and feature representations
2. **Linear Classifiers**
   - Perceptron
   - Logistic Regression
3. Regularization
4. Dense Representations
5. Similarity-based Learning
6. Neural Networks

Logistic Regression

Define a conditional probability:

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp(\mathrm{score}(\boldsymbol{y}, \boldsymbol{x}))}{Z_{\boldsymbol{x}}}, \qquad Z_{\boldsymbol{x}} = \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp(\mathrm{score}(\boldsymbol{y}', \boldsymbol{x}))$$

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

Exponentiating and normalizing score = softmax transformation[4]

Note: still a linear classifier

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

[4] More later during neural networks!

Logistic Regression

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

- 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., 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 ✓

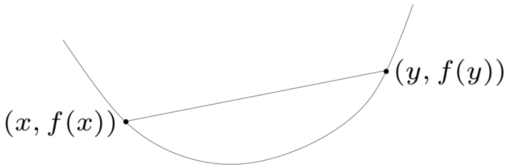


Figure: Illustration of a convex function. The line segment between any two points on the graph lies entirely above the curve.

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 $f : \mathbb{R}^d \to \mathbb{R}$

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

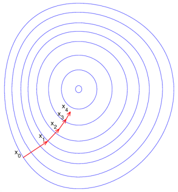


Figure: Illustration of gradient descent. The red lines correspond to steps taken in the negative gradient direction.

Logistic Regression: Gradient Descent (GD)

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

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

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

- 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 size $\eta$

initialize: $\mathcal{W} = \mathbf{0}$, i.e., $\boldsymbol{w_y} = \mathbf{0}$, $\forall \boldsymbol{y}$

repeat
    observe example $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$
    Update $\mathcal{W} = \mathcal{W} - \eta \nabla_{\mathcal{W}} L(\mathcal{W}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$

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

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

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?

Combine all $\frac{\partial}{\partial w_i}$ into vector updates for each $\boldsymbol{w_y}$

Cases:

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

$$\boldsymbol{w_y} = \boldsymbol{w_y} - \eta \left(P_{\mathcal{W}}(\boldsymbol{y}|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\phi}(\boldsymbol{x}) \right)$$

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

$$\boldsymbol{w_{y'}} = \boldsymbol{w_{y'}} - \eta \left(P_{\mathcal{W}}(\boldsymbol{y}'|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}) \right)$$

SGD for Logistic Regression

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

initialize: $\mathcal{W} = \mathbf{0}$, i.e., $\boldsymbol{w_y} = \mathbf{0}$, $\forall \boldsymbol{y}$

repeat
    observe example $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{D}$
    $\boldsymbol{w}_{\boldsymbol{y}_t} = \boldsymbol{w}_{\boldsymbol{y}_t} - \eta\,(P_{\mathcal{W}}(\boldsymbol{y}_t|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\phi}(\boldsymbol{x}))$
    $\boldsymbol{w}_{\boldsymbol{y}'} = \boldsymbol{w}_{\boldsymbol{y}'} - \eta\,(P_{\mathcal{W}}(\boldsymbol{y}'|\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x}))$ for $\boldsymbol{y}' \neq \boldsymbol{y}_t$

until stopping criterion

output: model weights $\mathcal{W}$
```

Logistic Regression Summary

- Define conditional probability

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

- 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 probabilistic: minimizes neg log-likelihood
  - also called log-linear model and max-entropy classifier
  - no closed form solution
  - For training instance $(\boldsymbol{x}, \boldsymbol{y})$, SDG updates look like

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

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

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

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

$$\boldsymbol{w}_{\boldsymbol{y}'} = \boldsymbol{w}_{\boldsymbol{y}'} - \boldsymbol{\phi}(\mathsf{x}) \text{ for } \boldsymbol{y}' \neq \boldsymbol{y}$$

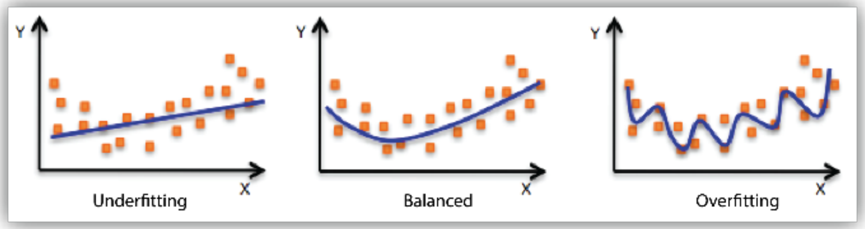
SGD updates for logistic regression and perceptron look similar!

Outline

1. Terminology, notation and feature representations
2. Linear Classifiers
   - Perceptron
   - Logistic Regression
3. **Regularization**
4. Dense Representations
5. Similarity-based Learning
6. Neural Networks

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

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

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!

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

Empirical Risk Minimization

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

- 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?

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

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

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

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

1. Terminology, notation and feature representations
2. Linear Classifiers
   - Perceptron
   - Logistic Regression
3. Regularization
4. **Dense Representations**
5. Similarity-based Learning
6. Neural Networks

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

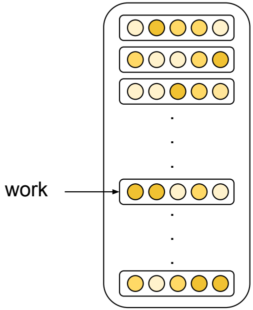
Dense Feature Representations

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

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

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

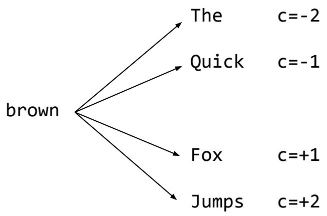
Embedding / Lookup Table

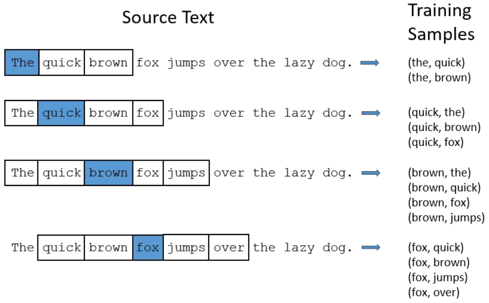


- Input is a word $\phi(x) \in \mathbb{R}^D$ for all $x \in \mathcal{V}$
- $\mathcal{V}$ is a fixed vocabulary of words
- We store these in a $|\mathcal{V}| \times D$ look up table
  - These are the model *word embeddings*
  - AKA embedding layer; word look-up table; ...

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 (SkipGram model):

$$\sum_{j}^{|\mathcal{C}|} \sum_{i}^{|\mathcal{X}_j|} \sum_{-c \le k \le c, k \ne 0} \log p(x_{i+k} | x_i) = \sum_{j}^{|\mathcal{C}|} \sum_{i}^{|\mathcal{X}_j|} \sum_{-c \le k \le c, k \ne 0} \log \frac{e^{\phi(x_i) \cdot \phi(x_{i+k})}}{\sum_{\boldsymbol{x}_l \in \mathcal{V}} e^{\phi(x_i) \cdot \phi(x_l)}}$$

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:

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

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

- Solution: negative sampling

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

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

Word2Vec

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

- $\phi(x_i)$ 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., $\boldsymbol{x} = x_1 \dots x_n$

COMMON SOLUTIONS

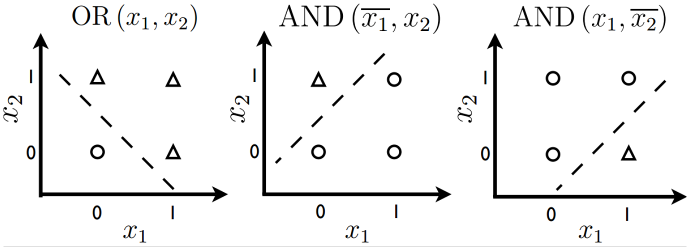
- Average: $\phi(\boldsymbol{x}) = \frac{1}{|\boldsymbol{x}|} \sum_{x \in \boldsymbol{x}} \phi(x)$
  - Other pointwise operations, e.g., max, sum, ...
- Truncation+concatentation: $\phi(\boldsymbol{x}) = [\phi(x_1), \dots, \phi(x_k)]$ for a fixed k
- Sparse-dense: Whole look-up table is input, but
 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

Outline

1. Terminology, notation and feature representations
2. Linear Classifiers
   - Perceptron
   - Logistic Regression
3. Regularization
4. Dense Representations
5. **Similarity-based Learning**
6. Neural Networks

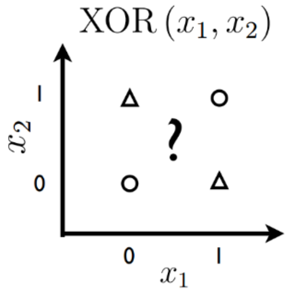
Recap: What a Linear Classifier Can Do

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


$\mathrm{OR}\,(x_1, x_2)$
$\mathrm{AND}\,(\overline{x_1}, x_2)$
$\mathrm{AND}\,(x_1, \overline{x_2})$
x_1
x_2
0
1

Recap: What a Linear Classifier Can't Do

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



Summary: Linear Classifiers

We've seen

- Perceptron
- Logistic regression
- Others: Support vector machines, LDA, ...

All lead to convex optimization problems $\Rightarrow$ no issues with local minima/initialization

All assume the features are well-engineered such that the data is nearly linearly separable

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!

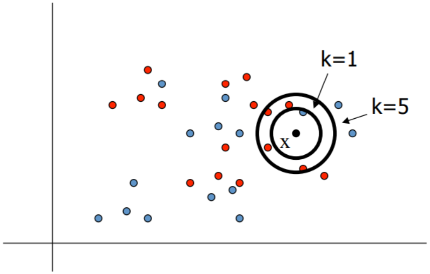
Parametric vs. Non-parametric

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.

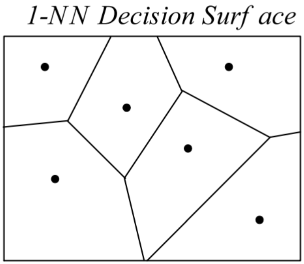
(K-th) Nearest Neighbor Classifier

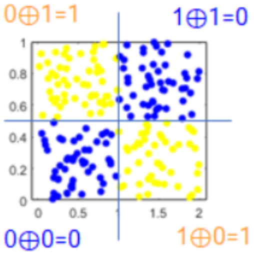
- Memorize (training) data $\mathcal{D} = \{\boldsymbol{x}_t, \boldsymbol{y}_t\}_{t=1}$
- To classify a new datapoint $\boldsymbol{x}$
    - Find the k closest data points in the data (e.g., training set)
    - Assign the most frequent class in those k points



(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)





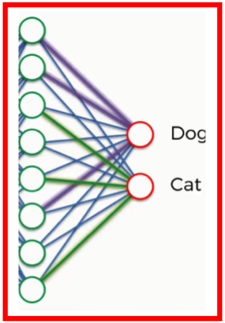
Similarity Functions and Inference

- Common Similarity functions
  - Euclidean: $\sqrt{\sum_i (\boldsymbol{\phi}(\boldsymbol{x}) - \boldsymbol{\phi}(\boldsymbol{x}'))^2}$
  - Inner product: $\boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{x}')$
  - Cosine: $\frac{\boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{x}')}{||\boldsymbol{\phi}(\boldsymbol{x})|| \, ||\boldsymbol{\phi}(\boldsymbol{x}')||}$
  - 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

1. Terminology, notation and feature representations
2. Linear Classifiers
   - Perceptron
   - Logistic Regression
3. Regularization
4. Dense Representations
5. Similarity-based Learning
6. **Neural Networks**

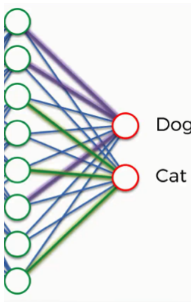
Weights and biases



Linear Classifier

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

Weights and biases



Linear Classifier

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

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 $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$

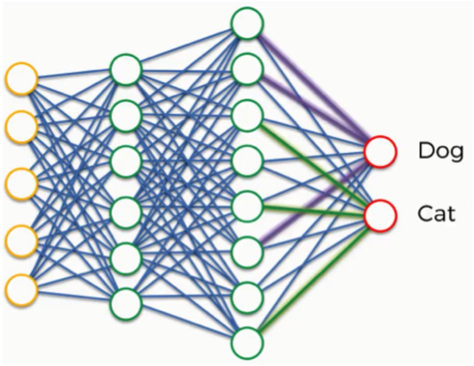
$$\mathbf{W}\boldsymbol{x} + \boldsymbol{b} = \begin{bmatrix} w_1^1 & w_2^1 & w_3^1 \\ w_1^2 & w_2^2 & w_3^2 \\ w_1^3 & w_2^3 & w_3^3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

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

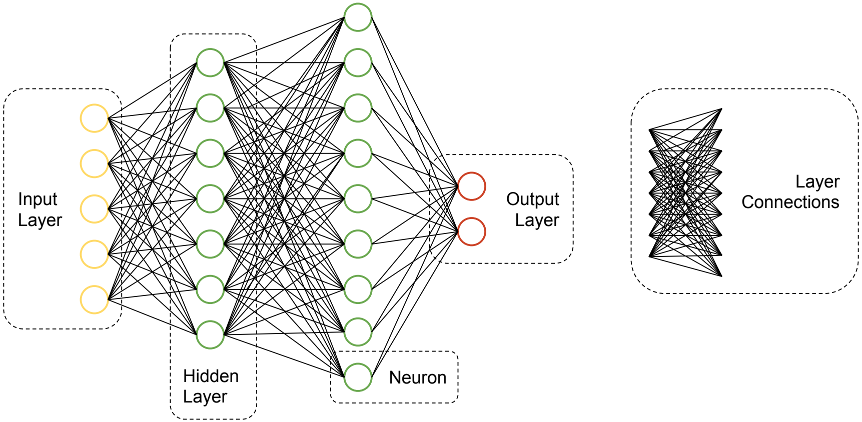
$$= \begin{bmatrix} (w_1^1 \times x_1) + (w_2^1 \times x_2) + (w_3^1 \times x_3) + b_1 \\ (w_1^2 \times x_1) + (w_2^2 \times x_2) + (w_3^2 \times x_3) + b_2 \\ (w_1^3 \times x_1) + (w_2^3 \times x_2) + (w_3^3 \times x_3) + b_3 \end{bmatrix}$$

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

On to neural networks!

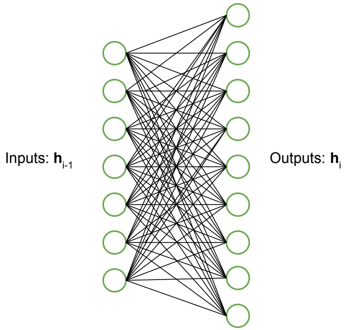


Neurons, Layers and Connections



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

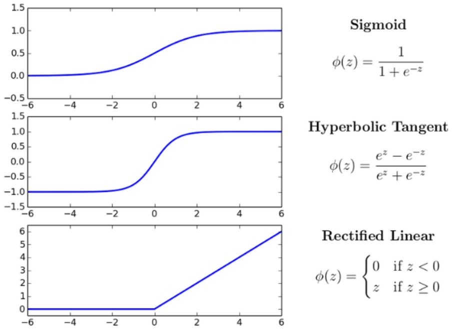
Hidden Layer Connections



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

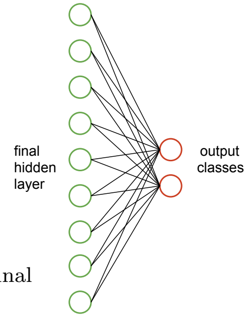
Activation Functions

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



From Hughes and Correll 2016

Output Layer



- This was in the last lecture!
- $y' = \textbf{argmax}\, \boldsymbol{y}$; where $\boldsymbol{y} = \mathbf{W}_{\text{final}} \boldsymbol{h}_{\text{final}-1} + \boldsymbol{b}_{\text{final}}$
- $\boldsymbol{y}_i$ often called the logit: the raw value/score

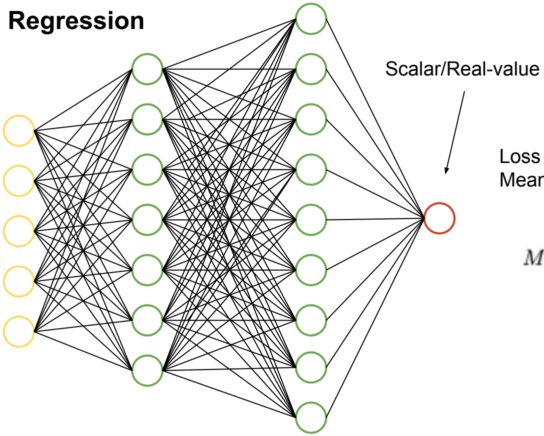
- Training?
  - Common paradigm: take softmax of logits $\frac{e^{\boldsymbol{y}_i}}{\sum_{\boldsymbol{y} \in \mathcal{Y}} e^{\boldsymbol{y}}}$
  - Gives probability distribution $P(\boldsymbol{y}|\boldsymbol{x})$
  - Minimize some loss

Example NN Loss: Cross-Entropy

- Can use any differentiable loss function.
- Let $\mathcal{W} = \{(\mathbf{W}_i, \boldsymbol{b}_i)\}$, i.e., parameters for all layers i
- Goal: Find $\mathcal{W}$ to maximize $P(\boldsymbol{y}|\boldsymbol{x}; \mathcal{W})$ 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
  - $\text{CE} = -E_{\overline{P}} \log P = -\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \overline{P}(\boldsymbol{y}|\boldsymbol{x}) \log P(\boldsymbol{y}|\boldsymbol{x}; \mathcal{W})$
  - Therefore: $CE = -\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \log P(\boldsymbol{y}|\boldsymbol{x}; \mathcal{W})$
- Min of $CE = -\sum_{(\boldsymbol{x},\boldsymbol{y})} \log P(\boldsymbol{y}|\boldsymbol{x}; \mathcal{W})$ = max of $\sum_{(\boldsymbol{x},\boldsymbol{y})} \log P(\boldsymbol{y}|\boldsymbol{x}; \mathcal{W})$
- Cross-entropy = Maximum Likelihood / log-loss.

Examples: NN for non-classification tasks

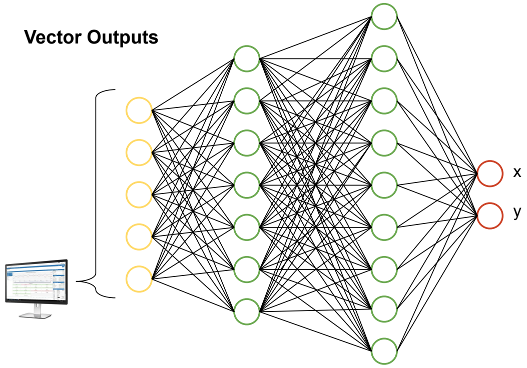
Regression



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



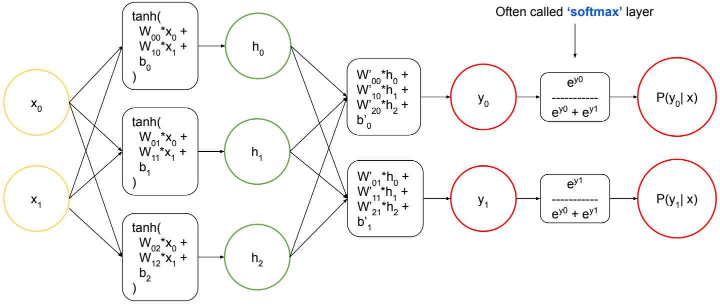
Example: Next Mouse Click

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


Often called 'softmax' layer
x_0
x_1
tanh(W_00*x_0 + W_10*x_1 + b_0)
tanh(W_01*x_0 + W_11*x_1 + b_1)
tanh(W_02*x_0 + W_12*x_1 + b_2)
h_0
h_1
h_2
W'_00*h_0 + W'_10*h_1 + W'_20*h_2 + b'_0
W'_01*h_0 + W'_11*h_1 + W'_21*h_2 + b'_1
y_0
y_1
e^y0 / e^y0 + e^y1
e^y1 / e^y0 + e^y1
P(y_0| x)
P(y_1| x)

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: $\boldsymbol{h}_i \in \mathbb{R}^{D_i}$; with $\boldsymbol{h}_i = \sigma_i(\mathbf{W}_i \boldsymbol{h}_{i-1} + \boldsymbol{b}_i)$
    - 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

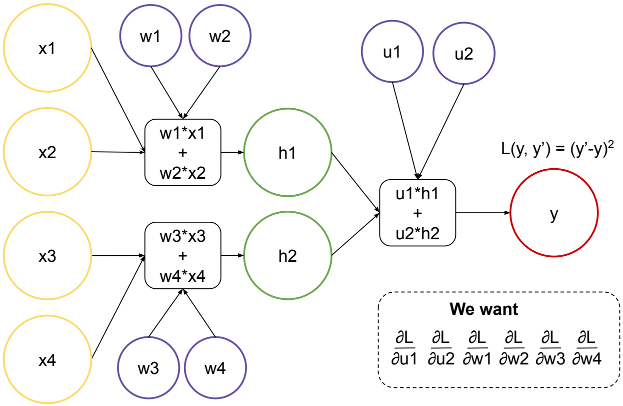
Neural Networks: Optimization

- 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 $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$

Toy Example: Analytical Partial Derivatives



We want

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

All base derivatives

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

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

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

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

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

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

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

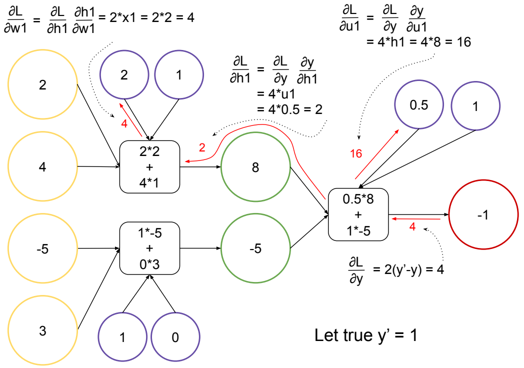
Full derivation examples

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

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

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



$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial h1}\frac{\partial h1}{\partial w1}$ = 2*x1 = 2*2 = 4
$\frac{\partial L}{\partial u1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial u1}$ = 4*h1 = 4*8 = 16
$\frac{\partial L}{\partial h1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial h1}$ = 4*u1 = 4*0.5 = 2
2
4
2
1
4
-5
3
2*2 + 4*1
1*-5 + 0*3
1
0
8
-5
4
2
0.5
1
16
0.5*8 + 1*-5
-1
4
$\frac{\partial L}{\partial y}$ = 2(y'-y) = 4
Let true y' = 1

Neuron derivatives

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

$\frac{\partial L}{\partial h1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial h1}$

$\frac{\partial L}{\partial h2} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial h2}$

$\frac{\partial L}{\partial u1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial u1}$

$\frac{\partial L}{\partial u2} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial u1}$

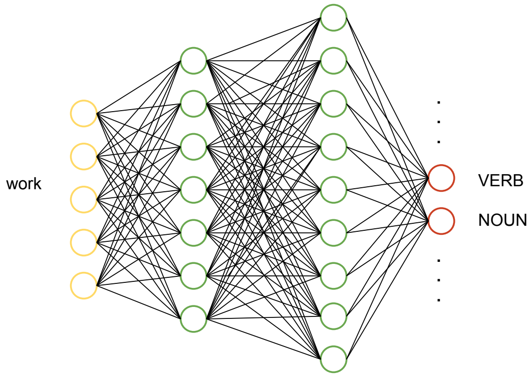
$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial h1}\frac{\partial h1}{\partial w1}$

$\frac{\partial L}{\partial w2} = \frac{\partial L}{\partial h1}\frac{\partial h1}{\partial w2}$

$\frac{\partial L}{\partial w3} = \frac{\partial L}{\partial h1}\frac{\partial h2}{\partial w3}$

$\frac{\partial L}{\partial w4} = \frac{\partial L}{\partial h1}\frac{\partial h2}{\partial w4}$

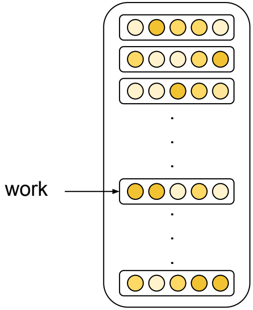
Input layer



- Consider classifying a word in isolation with a part-of-speech tag[7]
- Input is a word $\boldsymbol{x} \in \mathbb{R}^D$
- There is a fixed a finite vocabulary $\mathcal{V}$, i.e., $\boldsymbol{x} \in \mathcal{V}$

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

Input layer = Embedding layer

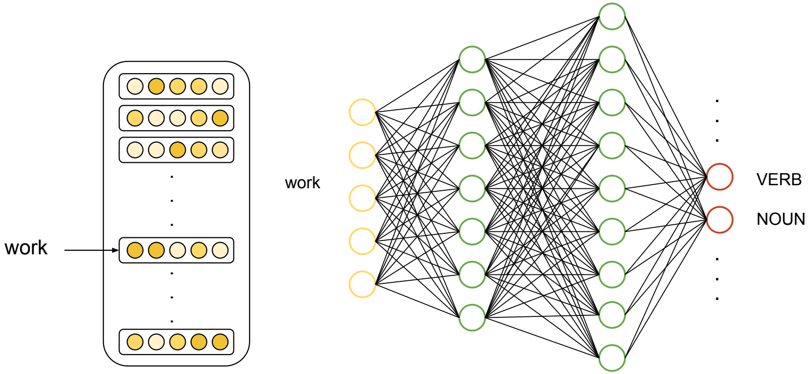


- Input is a word $\boldsymbol{x} \in \mathbb{R}^D$ for all $\boldsymbol{x} \in \mathcal{V}$
- We store these in a $|\mathcal{V}| \times D$ look up table
    - These are the model *word embeddings*
    - AKA embedding layer; word look-up table; ...

Input layer = Embedding layer

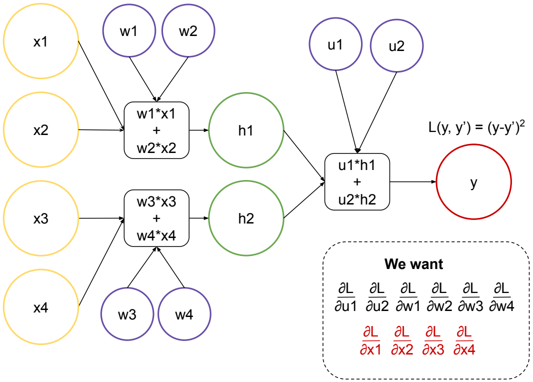
- Static embedding layer
  - Fixed word embeddings; not updated during training
  - Examples: SVD; word2vec; glove; ...
- Dynamic embedding layer
  - Randomly initialize word embeddings
  - Learn during training of the full network
  - Updated like any other layer during backpropagation
- Static + Dynamic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

Input layer



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

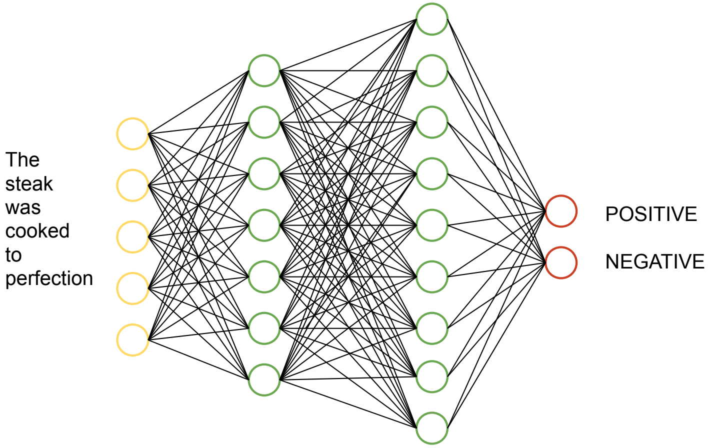
Dynamic Input layer



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

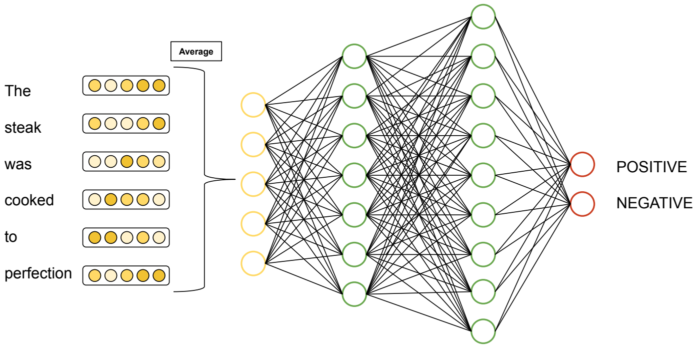
Variable Length Inputs

- 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



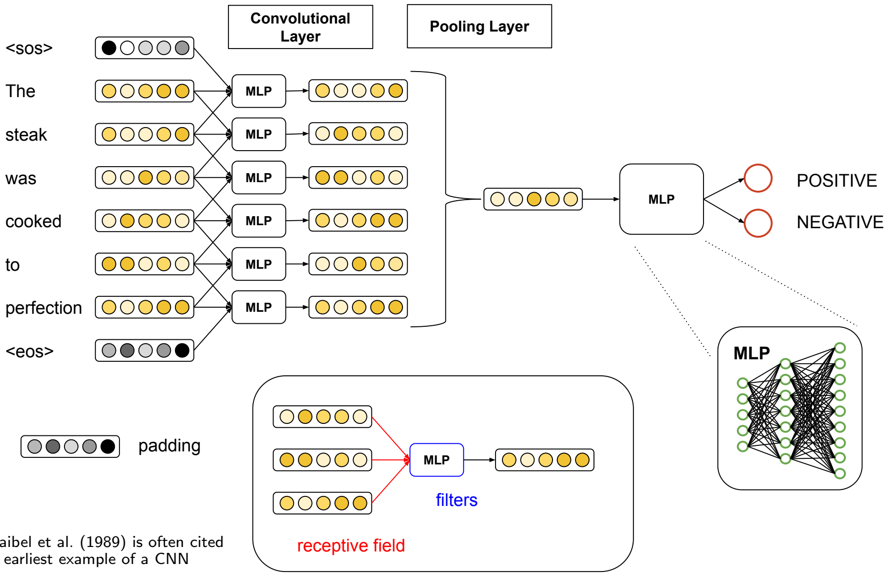
Variable Length Inputs: Options

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 (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

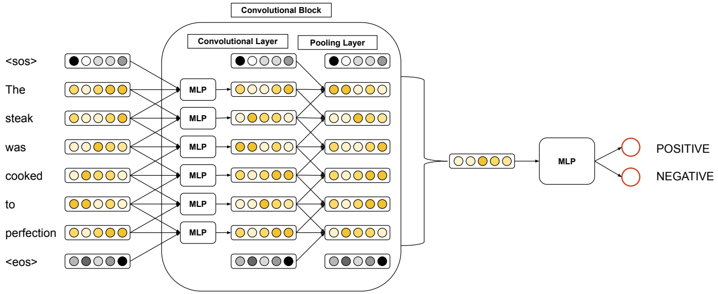


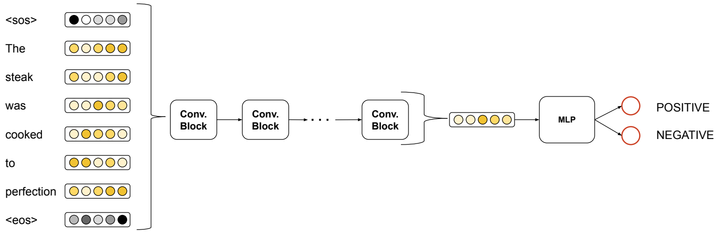
Waibel et al. (1989) is often cited as earliest example of a CNN

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
  - NN: Can also learn this, e.g., attention.

Deep Convolutional Neural Networks





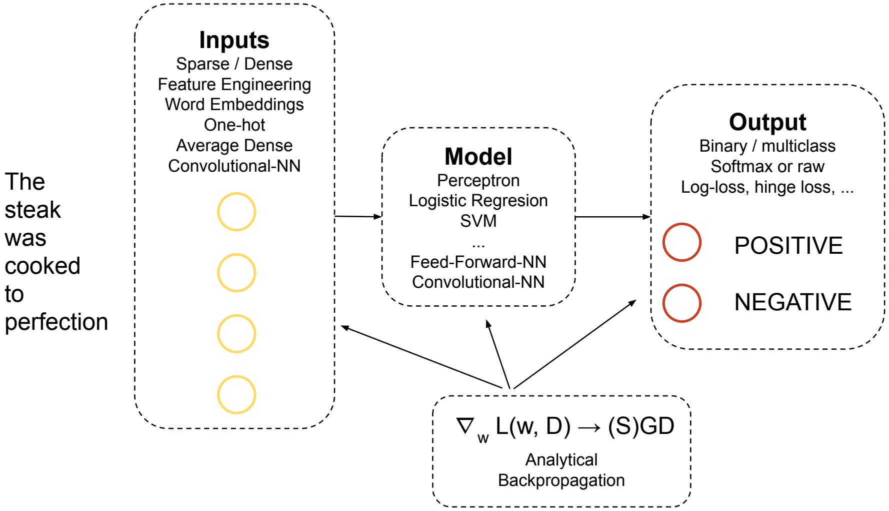
Core Neural Network Summary

- 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)



... 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
    - 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 classifiers
    - 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