Classification

Ryan McDonald [1]

Google

AthNLP, September 19, 2019

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

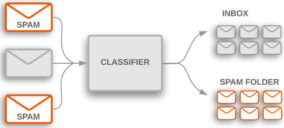
Classifiers

NOUN or VERB

How does sodium bicarbonate work ?

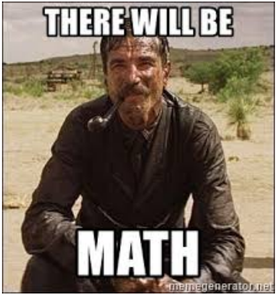
"I love this movie.
I've seen it many times
and it's still awesome." →

"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

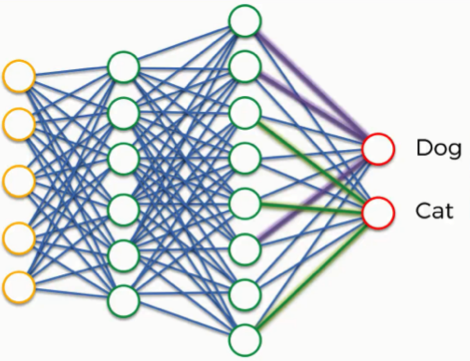
Linear Classifiers

This lecture is 2/3 about linear classifiers!

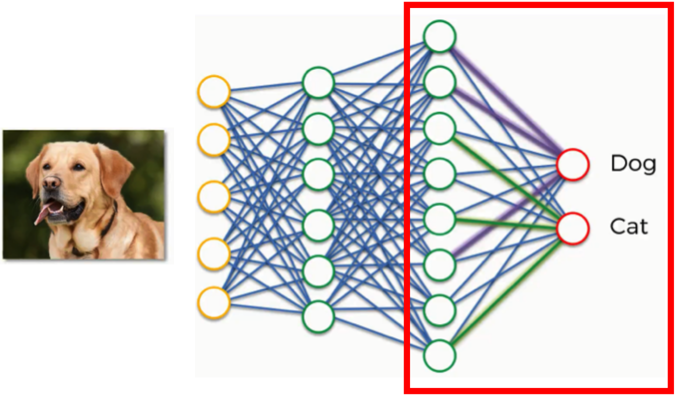
Why? It's 2019 and everybody uses neural networks.

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

Linear Classifiers and Neural Networks



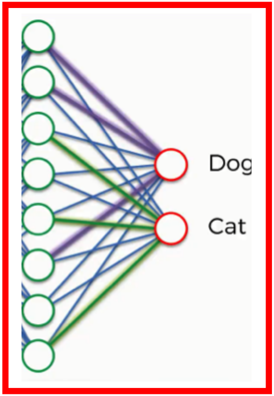
Linear Classifiers and Neural Networks



Linear Classifier

Linear Classifiers and Neural Networks

Handcrafted Features



Linear Classifier

Fake News Detection

Task: tell if a news article / quote is fake or real.

This is a **binary classification problem.**

Fake Or Real?


NATIONAL
ENQUIRER
THE WORLD'S LIVELIEST PAPER
FEATURE 15¢
Rita Hayworth Says...
I'M BACK FROM THE DEAD
For Two Years I Was a Zombie
WEEKLY WORLD
NEWS
Space creature survived UFO crash in Arkansas!
HILLARY CLINTON ADOPTS ALIEN BABY
OFFICIAL PHOTO!
Secret Service building special nursery in the White House!

Fake Or Real?


With Artificial
Intelligence we
are summoning
the demons
- Elon Musk

Fake Or Real?

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

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.

Fake Or Real?

Can a machine determine this automatically?

It can be a very hard problem, since fact-checking is hard and requires combining several knowledge sources

... also, reality surpasses fiction sometimes.

Topic Classification

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

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

It's a much easier task, we can get 80-90% accuracies with a simple ML model.

Topic Classification

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

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.

$\longrightarrow$

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

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

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

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

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. Perceptron
3. Logistic Regression
4. Support Vector Machines
5. Regularization
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., fake/not fake, a topic, a parse tree, 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 **sentence** together with a **parse tree**
  - e.g., an **image** partitioned into **segmentation regions**

Supervised Machine Learning

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

$$\mathcal{D} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n)\}_{n=1}^{N} \subseteq \mathcal{X} \times \mathcal{Y}$$

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

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

- Hopefully, $\widehat{\boldsymbol{y}} \approx \boldsymbol{y}$ 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:** $\mathcal{Y} = \mathbb{R}^K$
  - 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., fake news detection
- **Multi-class classification:** $\mathcal{Y} = \{1, 2, \ldots, K\}$
  - e.g., topic classification
- **Structured classification:** $\mathcal{Y}$ exponentially large and structured
  - e.g., machine translation, caption generation, image segmentation

Feature Representations

Feature engineering is an important step in linear classifiers:

- Bag-of-words features for text, also lemmas, parts-of-speech, ...
- SIFT features and wavelet representations in computer vision
- Other categorical, Boolean, and continuous features

Feature Representations

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

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

- $\boldsymbol{\psi}(\boldsymbol{x})$ 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

Feature Representations: Joint Feature Mappings

For multi-class/structured classification, a joint feature map $\boldsymbol{\phi} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ is sometimes more convenient

- $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ instead of $\boldsymbol{\psi}(\boldsymbol{x})$

Each feature now represents a joint property of the input $\boldsymbol{x}$ and the candidate output $\boldsymbol{y}$.

Examples

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

$$\phi_j(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains the word ``interest''} \\ & \text{and } \boldsymbol{y} = \text{``financial''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_k(\boldsymbol{x}, \boldsymbol{y}) = \text{\% of words in } \boldsymbol{x} \text{ with punctuation and } \boldsymbol{y} = \text{``scientific''}$$

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

$$\phi_j(\boldsymbol{x}, \boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} = \text{ ends in ``ed'' and } \boldsymbol{y} = \text{ Verb} \\ 0 & \text{otherwise} \end{cases}$$

Bag of Words Feature Representation

- $\boldsymbol{x}$ is a name, $\boldsymbol{y}$ is a label classifying the type of entity

$$\phi_0(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``George''} \\ & \text{and } \boldsymbol{y} = \text{``Person''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_4(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``George''} \\ & \text{and } \boldsymbol{y} = \text{``Location''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_1(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``Washington''} \\ & \text{and } \boldsymbol{y} = \text{``Person''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_5(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``Washington''} \\ & \text{and } \boldsymbol{y} = \text{``Location''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_2(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``Bridge''} \\ & \text{and } \boldsymbol{y} = \text{``Person''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_6(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``Bridge''} \\ & \text{and } \boldsymbol{y} = \text{``Location''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_3(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``General''} \\ & \text{and } \boldsymbol{y} = \text{``Person''} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_7(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains ``General''} \\ & \text{and } \boldsymbol{y} = \text{``Location''} \\ 0 & \text{otherwise} \end{cases}$$

- $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Person $\rightarrow \phi(\boldsymbol{x},\boldsymbol{y}) = [1\ 1\ 0\ 1\ 0\ 0\ 0\ 0]$
- $\boldsymbol{x}$=George Washington Bridge, $\boldsymbol{y}$=Location $\rightarrow \phi(\boldsymbol{x},\boldsymbol{y}) = [0\ 0\ 0\ 0\ 1\ 1\ 1\ 0]$
- $\boldsymbol{x}$=George Washington George, $\boldsymbol{y}$=Location $\rightarrow \phi(\boldsymbol{x},\boldsymbol{y}) = [0\ 0\ 0\ 0\ 1\ 1\ 0\ 0]$

Block Feature Vectors

- $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Person $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ = [1 1 0 1 0 0 0 0]
- $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Location $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ = [0 0 0 0 1 1 0 1]
- $\boldsymbol{x}$=George Washington Bridge, $\boldsymbol{y}$=Location $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ = [0 0 0 0 1 1 1 0]
- $\boldsymbol{x}$=George Washington George, $\boldsymbol{y}$=Location $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$ = [0 0 0 0 1 1 0 0]

- Each equal size block of the feature vector corresponds to one label
- Non-zero values allowed only in one block

Feature Representations – $\psi(x)$ vs. $\phi(x, y)$

Equivalent if $\phi(\boldsymbol{x}, \boldsymbol{y})$ conjoins input features $\psi(\boldsymbol{x})$ with one-hot label representations $\boldsymbol{e_y} := [0, \ldots, 0, 1, 0, \ldots, 0]$

$$\begin{aligned} \phi(\boldsymbol{x}, \boldsymbol{y}) &= \psi(\boldsymbol{x}) \otimes \boldsymbol{e_y} \\ &= [\mathbf{0}, \ldots, \mathbf{0}, \underbrace{\psi(\boldsymbol{x})}_{\boldsymbol{y}^{\text{th}} \text{ block}}, \mathbf{0}, \ldots, \mathbf{0}] \end{aligned}$$

- $\psi(\boldsymbol{x})$
  - $\boldsymbol{x}$=General George Washington $\rightarrow \psi(\boldsymbol{x}) = [1\ 1\ 0\ 1]$
- $\phi(\boldsymbol{x}, \boldsymbol{y})$
  - $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Person $\rightarrow \phi(\boldsymbol{x}, \boldsymbol{y}) = [1\ 1\ 0\ 1\ 0\ 0\ 0\ 0]$
  - $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Object $\rightarrow \phi(\boldsymbol{x}, \boldsymbol{y}) = [0\ 0\ 0\ 0\ 1\ 1\ 0\ 1]$

$\psi(\boldsymbol{x})$ is sometimes simpler and more convenient ... but $\phi(\boldsymbol{x}, \boldsymbol{y})$ is more expressive

Feature Engineering and NLP Pipelines

Classical NLP pipelines consist of stacking together several linear classifiers

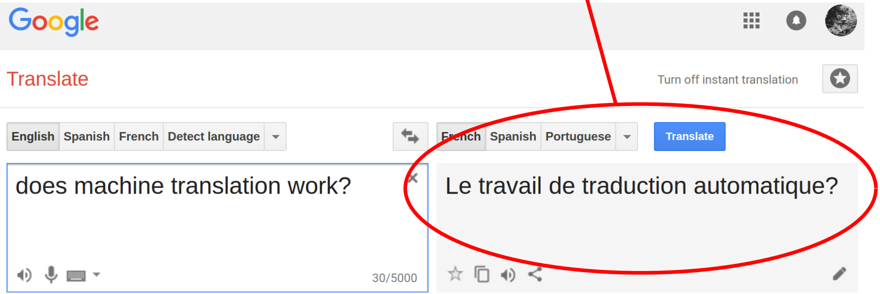
Each classifier's predictions are used to handcraft features for other classifiers

Examples of features:

- POS tags: adjective counts for sentiment analysis
- Spell checker: misspellings counts for spam detection
- Parsing: depth of tree for readability assessment.

Example: Translation Quality Estimation

Wrong translation!



Goal: estimate the quality of a translation on the fly (without a reference)!

Example: Translation Quality Estimation

Hand-crafted features:

- no of tokens in the source/target segment
- LM probability of source/target segment and their ratio
- % of source 1–3-grams observed in 4 frequency quartiles of source corpus
- average no of translations per source word
- ratio of brackets and punctuation symbols in source & target segments
- ratio of numbers, content/non-content words in source & target segments
- ratio of nouns/verbs/etc in the source & target segments
- % of dependency relations b/w constituents in source & target segments
- diff in depth of the syntactic trees of source & target segments
- diff in no of PP/NP/VP/ADJP/ADVP/CONJP in source & target
- diff in no of person/location/organization entities in source & target
- features and global score of the SMT system
- number of distinct hypotheses in the n-best list
- 1–3-gram LM probabilities using translations in the n-best to train the LM
- average size of the target phrases
- proportion of pruned search graph nodes;
- proportion of recombined graph nodes.

Our Setup

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

Linear Classifiers

- Parametrized by a weight vector $\boldsymbol{w} \in \mathbb{R}^D$ (one weight per feature)
- The score (or probability) of a particular label is based on a linear combination of features and their weights
- At test time (known $\boldsymbol{w}$), predict the class $\widehat{\boldsymbol{y}}$ which maximizes this score:

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

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

Linear Classifiers – $\psi(x)$

- Define $|\mathcal{Y}|$ weight vectors $\boldsymbol{w_y} \in \mathbb{R}^D$
  - i.e., one weight vector per output label $\boldsymbol{y}$

- **Classification**

$$\widehat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \ \boldsymbol{w_y} \cdot \boldsymbol{\psi}(\boldsymbol{x})$$

Linear Classifiers – $\psi(x)$

- Define $|\mathcal{Y}|$ weight vectors $\boldsymbol{w_y} \in \mathbb{R}^D$
  - i.e., one weight vector per output label $\boldsymbol{y}$

- **Classification**

$$\widehat{\boldsymbol{y}} = \operatorname*{arg\,max}_{\boldsymbol{y} \in \mathcal{Y}} \ \boldsymbol{w_y} \cdot \boldsymbol{\psi}(\boldsymbol{x})$$

- $\boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$
  - $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Person $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) = [1\ 1\ 0\ 1\ 0\ 0\ 0\ 0]$
  - $\boldsymbol{x}$=General George Washington, $\boldsymbol{y}$=Object $\rightarrow \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) = [0\ 0\ 0\ 0\ 1\ 1\ 0\ 1]$
  - Single $\boldsymbol{w} \in \mathbb{R}^8$

- $\boldsymbol{\psi}(\boldsymbol{x})$
  - $\boldsymbol{x}$=General George Washington $\rightarrow \boldsymbol{\psi}(\boldsymbol{x}) = [1\ 1\ 0\ 1]$
  - Two parameter vectors $\boldsymbol{w}_{\text{Person}} \in \mathbb{R}^4$, $\boldsymbol{w}_{\text{Object}} \in \mathbb{R}^4$

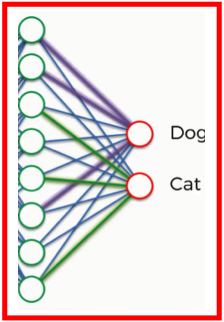
Linear Classifiers – Bias Terms

- Often linear classifiers are presented as

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

  where $b_{\boldsymbol{y}}$ is a bias or offset term
- This can be folded into $\boldsymbol{\psi}(\boldsymbol{x})$ (by defining a constant feature for each label)
- For now, we assume this for simplicity

Handcrafted Features



Linear Classifier

$$\widehat{\boldsymbol{y}} = \textbf{argmax}\left(\mathbf{W}\psi(x) + \boldsymbol{b}\right), \quad \mathbf{W} = \begin{bmatrix} \vdots \\ w_{\boldsymbol{y}}^{\top} \\ \vdots \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} \vdots \\ b_{\boldsymbol{y}} \\ \vdots \end{bmatrix}.$$

Binary Linear Classifier

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

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

Binary Linear Classifier

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

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

Binary Linear Classifier

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

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

Binary Linear Classifier

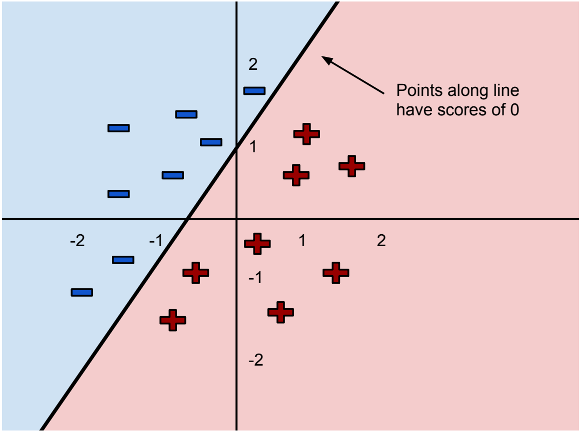
With binary labels ($\mathcal{Y} = \{\pm 1\}$) we often use a minimal parametrization:

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

That is: only half of the parameters are needed.

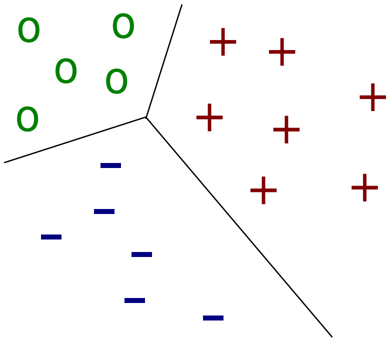
Binary Linear Classifier

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



Multiclass Linear Classifier

Defines regions of space.

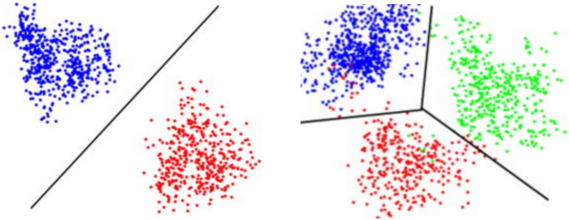


Linear Classifiers

- Prediction rule:

$$\widehat{\boldsymbol{y}} = h(\boldsymbol{x}) = \operatorname*{arg\,max}_{\boldsymbol{y} \in \mathcal{Y}} \overbrace{\boldsymbol{w} \cdot \phi(\boldsymbol{x}, \boldsymbol{y})}^{\text{linear in } \boldsymbol{w}}$$

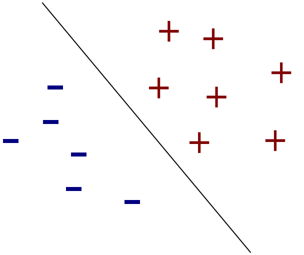
- The decision boundary is defined by the intersection of half spaces
- In the binary case ($|\mathcal{Y}| = 2$) this corresponds to a hyperplane classifier



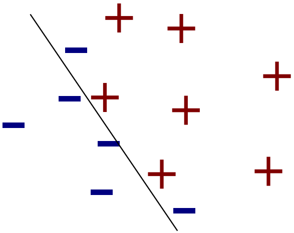
Linear Separability

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

Separable



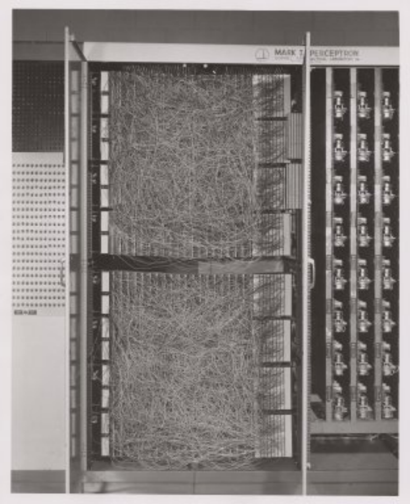
Not Separable



Outline

1. Terminology, notation and feature representations
2. **Perceptron**
3. Logistic Regression
4. Support Vector Machines
5. Regularization
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 x_i; 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 $\boldsymbol{w}^{(0)} = \mathbf{0}$
initialize $k = 0$ (number of mistakes)
repeat
  get new training example $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathcal{D}$
  predict $\widehat{\boldsymbol{y}}_i = \arg\max_{\boldsymbol{y} \in \mathcal{Y}} \boldsymbol{w}^{(k)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y})$
  if $\widehat{\boldsymbol{y}}_i \neq \boldsymbol{y}_i$ **then**
    update $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, \widehat{\boldsymbol{y}}_i)$
    increment k
  end if
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 vector $\boldsymbol{u}$ with $\|\boldsymbol{u}\| = 1$ such that

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

- radius of the data: $R = \max_{i,\ \boldsymbol{y}'_i \neq \boldsymbol{y}_i} \|\boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}'_i)\|$.

Perceptron's Mistake Bound

A couple definitions:

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

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

- radius of the data: $R = \max_{i,\ \boldsymbol{y}'_i \neq \boldsymbol{y}_i} \|\boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}'_i)\|$.

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 $\frac{R^2}{\gamma^2}$ mistakes.

One-Slide Proof

- **Lower bound on $\|w^{(k+1)}\|$:**

$$
\begin{aligned}
\boldsymbol{u} \cdot \boldsymbol{w}^{(k+1)} &= \boldsymbol{u} \cdot \boldsymbol{w}^{(k)} + \boldsymbol{u} \cdot (\phi(\boldsymbol{x}_i, \boldsymbol{y}_i) - \phi(\boldsymbol{x}_i, \widehat{\boldsymbol{y}}_i)) \\
&\geq \boldsymbol{u} \cdot \boldsymbol{w}^{(k)} + \gamma \\
&\geq k\gamma.
\end{aligned}
$$

Hence $\|\boldsymbol{w}^{(k+1)}\| = \|\boldsymbol{u}\| \cdot \|\boldsymbol{w}^{(k+1)}\| \geq \boldsymbol{u} \cdot \boldsymbol{w}^{(k+1)} \geq k\gamma$ (from CSI).

One-Slide Proof

- **Lower bound on** $\|w^{(k+1)}\|$:

$$
\begin{aligned}
\boldsymbol{u} \cdot \boldsymbol{w}^{(k+1)} &= \boldsymbol{u} \cdot \boldsymbol{w}^{(k)} + \boldsymbol{u} \cdot (\boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, \widehat{\boldsymbol{y}}_i)) \\
&\geq \boldsymbol{u} \cdot \boldsymbol{w}^{(k)} + \gamma \\
&\geq k\gamma.
\end{aligned}
$$

  Hence $\|\boldsymbol{w}^{(k+1)}\| = \|\boldsymbol{u}\| \cdot \|\boldsymbol{w}^{(k+1)}\| \geq \boldsymbol{u} \cdot \boldsymbol{w}^{(k+1)} \geq k\gamma$ (from CSI).

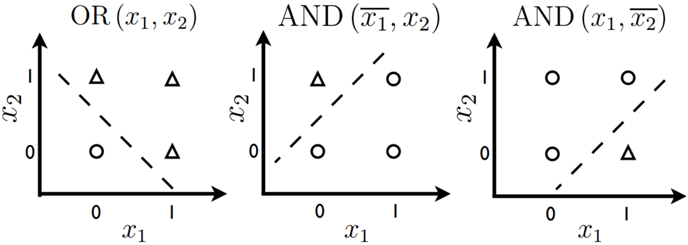
- **Upper bound on** $\|w^{(k+1)}\|$:

$$
\begin{aligned}
\|\boldsymbol{w}^{(k+1)}\|^2 &= \|\boldsymbol{w}^{(k)}\|^2 + \|\boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, \widehat{\boldsymbol{y}}_i)\|^2 \\
&\quad + 2\boldsymbol{w}^{(k)} \cdot (\boldsymbol{\phi}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \boldsymbol{\phi}(\boldsymbol{x}_i, \widehat{\boldsymbol{y}}_i)) \\
&\leq \|\boldsymbol{w}^{(k)}\|^2 + R^2 \\
&\leq kR^2.
\end{aligned}
$$

Equating both sides, we get $(k\gamma)^2 \leq kR^2 \;\Rightarrow\; k \leq R^2/\gamma^2$ (QED).

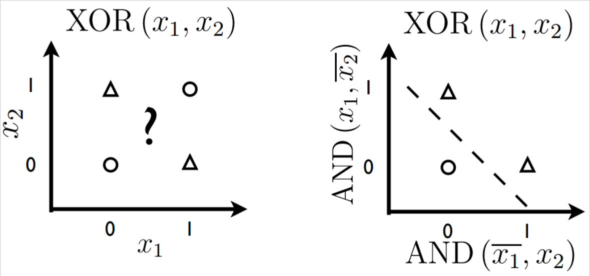
What a Simple Perceptron Can and Can't Do

- 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):



- This result is often attributed to Minsky and Papert (1969) but was known well 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

1. Terminology, notation and feature representations
2. Perceptron
3. **Logistic Regression**
4. Support Vector Machines
5. Regularization
6. Neural Networks

Logistic Regression

Define a conditional probability:

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

Exponentiating and normalizing is called the softmax transformation[2]

Critically $\sum_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) = 1$

Note: still a linear classifier

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

[2] More later during neural networks!

Logistic Regression

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

- How do we learn weights $\boldsymbol{w}$?
- Set $\boldsymbol{w}$ to minimize the negative conditional log-likelihood:

$$\begin{aligned}
\widehat{\boldsymbol{w}} &= \arg\min_{\boldsymbol{w} \in \mathbb{R}^D} -\log\left(\prod_{t=1}^{N} P_{\boldsymbol{w}}(\boldsymbol{y}_t|\boldsymbol{x}_t)\right) = \arg\min_{\boldsymbol{w} \in \mathbb{R}^D} -\sum_{t=1}^{N} \log P_{\boldsymbol{w}}(\boldsymbol{y}_t|\boldsymbol{x}_t) \\
&= \arg\min_{\boldsymbol{w} \in \mathbb{R}^D} \sum_{t=1}^{N} \left(\log \sum_{\boldsymbol{y}'_t} \exp(\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}'_t)) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) \right),
\end{aligned}$$

i.e., set $\boldsymbol{w}$ 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, ...)

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, ...)

- Logistic Regression = Maximum Entropy: maximize entropy subject to constraints on features
- Proof left as an exercise!

Recap: Convex functions

Pro: Guarantee of a global minima ✓

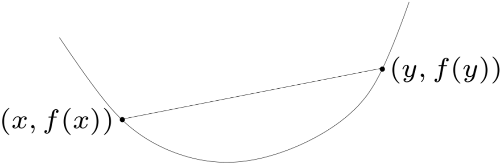


Figure: Illustration of a convex function. The line segment between any two points on the graph lies entirely above the curve.

Recap: 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.
- **Gradient descent** updates: $\boldsymbol{w}^{(k+1)} \leftarrow \boldsymbol{w}^{(k)} - \eta_k \nabla f(\boldsymbol{w}^{(k)})$

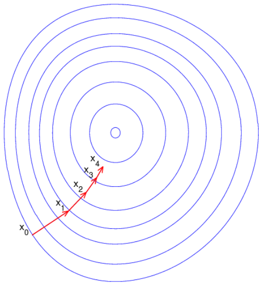


Figure: Illustration of gradient descent. The red lines correspond to steps taken in the negative gradient direction.

Gradient Descent

- Let $L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y})) = \log \sum_{\boldsymbol{y}'} \exp(\boldsymbol{w} \cdot \phi(\boldsymbol{x}, \boldsymbol{y}')) - \boldsymbol{w} \cdot \phi(\boldsymbol{x}, \boldsymbol{y})$
- This is our loss function!
  - Logistic-regressions loss function often called log-loss or cross-entropy
- We want to find $\arg\min_{\boldsymbol{w}} \sum_{t=1}^{N} L(\boldsymbol{w}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$
  - Set $\boldsymbol{w}^0 = \mathbf{0}$
  - Iterate until convergence (for suitable stepsize η_k):

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

- $\nabla_{\boldsymbol{w}} L(\boldsymbol{w})$ is gradient of L w.r.t. $\boldsymbol{w}$

- Gradient descent will always find the optimal $\boldsymbol{w}$

Stochastic Gradient Descent

If the dataset is large, we'd better do SGD instead, for more frequent updates:

- Set $\boldsymbol{w}^0 = \mathbf{0}$
- Iterate until convergence
  - Pick $(\boldsymbol{x}_t, \boldsymbol{y}_t)$ randomly
  - Update $\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta_k \nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$
- i.e. we approximate the true gradient with a noisy, unbiased, gradient, based on a single sample
- Variants exist in-between (mini-batches)
- All guaranteed to find the optimal $\boldsymbol{w}$ (for suitable step sizes)

Computing the Gradient

- For this to work, we need to be able to compute $\nabla_{\boldsymbol{w}} L(\boldsymbol{w}; (\boldsymbol{x}_t, \boldsymbol{y}_t))$, where

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

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

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

The gradient equals the "difference between the expected features under the current model and the true features."

Logistic Regression Summary

- Define conditional probability

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

- Set weights to minimize negative conditional log-likelihood:

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

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

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

The Story So Far

- Logistic regression is discriminative: maximizes conditional likelihood
  - also called log-linear model and max-entropy classifier
  - no closed form solution
  - stochastic gradient updates look like

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

- Perceptron is a discriminative, non-probabilistic classifier
  - perceptron's updates look like

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

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

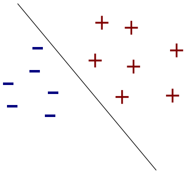
Maximizing Margin

- For a training set $\mathcal{D}$
- Margin of a weight vector $\boldsymbol{w}$ is smallest γ such that

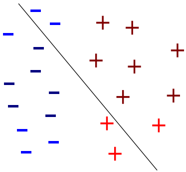
$$\boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{w} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma$$

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

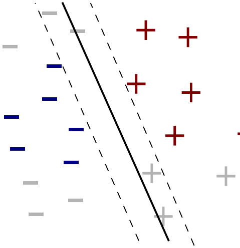
Training

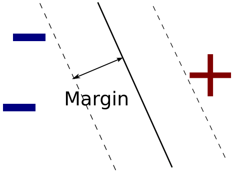


Testing



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

$$\epsilon \propto \frac{R^2}{\gamma^2 \times N}$$

- **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

1. Terminology, notation and feature representations
2. Perceptron
3. Logistic Regression
4. **Support Vector Machines**
5. Regularization
6. Neural Networks

Maximizing Margin

Let $\gamma > 0$

$$\max_{||\boldsymbol{w}|| \leq 1} \quad \gamma$$

such that:

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

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

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

- Note: algorithm still minimizes error if data is separable
- $||\boldsymbol{w}||$ is bound since scaling trivially produces larger margin

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{||\boldsymbol{w}|| \leq 1} \quad \gamma$$

such than:

$$=$$

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

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

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

Min Norm:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}||\boldsymbol{w}||^2$$

such that:

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

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

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

- Instead of fixing $||\boldsymbol{w}||$ we fix the margin $\gamma = 1$
- Make substitution $\boldsymbol{w}' = \boldsymbol{w}/\gamma$; then we have $\gamma = \frac{||\boldsymbol{w}||}{||\boldsymbol{w}'||} = \frac{1}{||\boldsymbol{w}'||}$.

Support Vector Machines

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \ \frac{1}{2}||\boldsymbol{w}||^2$$

such that:

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

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

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

Support Vector Machines

What if data is not separable?

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w},\xi} \; \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{t=1}^{N}\xi_t$$

such that:

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

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

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

Support Vector Machines

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w},\xi} \;\; \frac{\lambda}{2}||\boldsymbol{w}||^2 + \sum_{t=1}^{N} \xi_t \qquad\qquad \lambda = \frac{1}{C}$$

such that:

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

Support Vector Machines

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

such that:

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

Support Vector Machines

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

such that:

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

Support Vector Machines

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w},\xi} \;\; \frac{\lambda}{2}||\boldsymbol{w}||^2 + \sum_{t=1}^{N} \xi_t \qquad\qquad \lambda = \frac{1}{C}$$

such that:

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

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

Support Vector Machines

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

such that:

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

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

Hinge loss:

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

Support Vector Machines

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w},\xi} \; \frac{\lambda}{2}||\boldsymbol{w}||^2 + \sum_{t=1}^{N} \xi_t$$

such that:

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

Hinge loss equivalent

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

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

Summary

What we have covered

- Linear Classifiers
  - Logistic Regression
  - Perceptron
  - Support Vector Machines

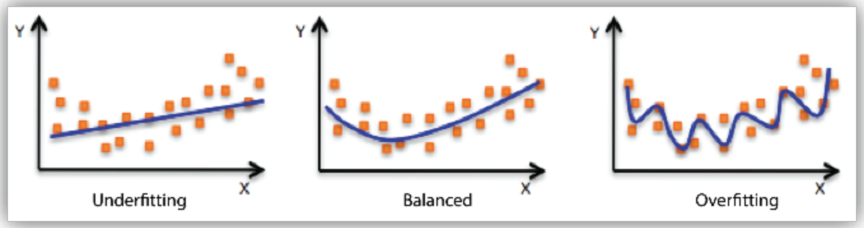
What is next

- Regularization
- Non-linear classifiers

Outline

Overfitting

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



Regularization

In practice, we regularize models to prevent overfitting

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

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

- Gaussian prior (ℓ_2), promotes smaller weights:

$$\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_2^2 = \sum_i \boldsymbol{w}_i^2.$$

- Laplacian prior (ℓ_1), promotes sparse weights!

$$\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 = \sum_i |\boldsymbol{w}_i|$$

Logistic Regression with ℓ_2 Regularization

$$\sum_{t=1}^{N} L(\boldsymbol{w}; (\boldsymbol{x}_t, \boldsymbol{y}_t)) + \lambda\Omega(\boldsymbol{w}) = -\sum_{t=1}^{N} \log\left(\exp(\boldsymbol{w} \cdot \phi(\boldsymbol{x}_t, \boldsymbol{y}_t))/Z_{\boldsymbol{x}}\right) + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

- What is the new gradient?

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

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

Support Vector Machines

Hinge-loss formulation: ℓ_2 regularization already happening!

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

$\uparrow$ SVM optimization $\uparrow$

SVMs vs. Logistic Regression

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

SVMs vs. Logistic Regression

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

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

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

SVMs vs. Logistic Regression

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

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

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

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

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

$$\boldsymbol{w} = \arg \min_{\boldsymbol{w} \in \mathbb{R}^D} \sum_{t=1}^{N} \left(\sum_{\boldsymbol{y}'_t} P(\boldsymbol{y}'_t | \boldsymbol{x}) - P(\boldsymbol{y}_t | \boldsymbol{x})\right) + \frac{\lambda}{2} \|\boldsymbol{w}\|^2$$

SVMs vs. Perceptron

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

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

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

SVMs vs. Perceptron

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

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

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

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

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

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:

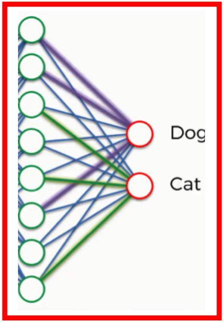
- `http://lxmls.it.pt/2019/LINEAR_LEARNERS.pdf`
- Also covers
  - Naive Bayes
  - Sub-gradient descent
    - Needed for SVMs
    - Perceptron update is sub-gradient with no margin
  - Non-Linear Classifiers $\neq$ Neural Networks
    - K-Nearest neighbors
    - Kernel methods

Outline

1. Terminology, notation and feature representations
2. Perceptron
3. Logistic Regression
4. Support Vector Machines
5. Regularization
6. **Neural Networks**

Reminder

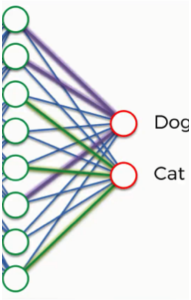
Handcrafted Features



Linear Classifier

$$\widehat{\boldsymbol{y}} = \textbf{argmax}\left(\mathbf{W}\psi(x) + \boldsymbol{b}\right), \quad \mathbf{W} = \begin{bmatrix} \vdots \\ w_{\boldsymbol{y}}^{\top} \\ \vdots \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} \vdots \\ b_{\boldsymbol{y}} \\ \vdots \end{bmatrix}.$$

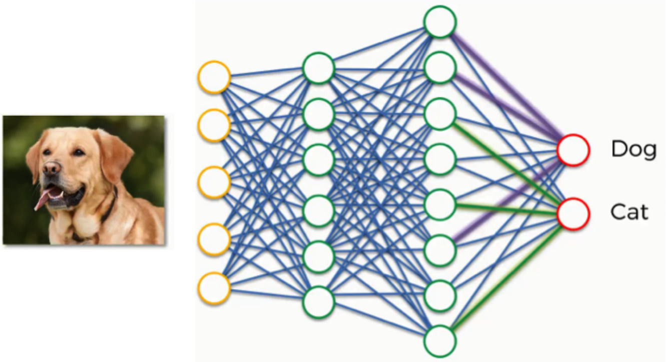
No more ψ



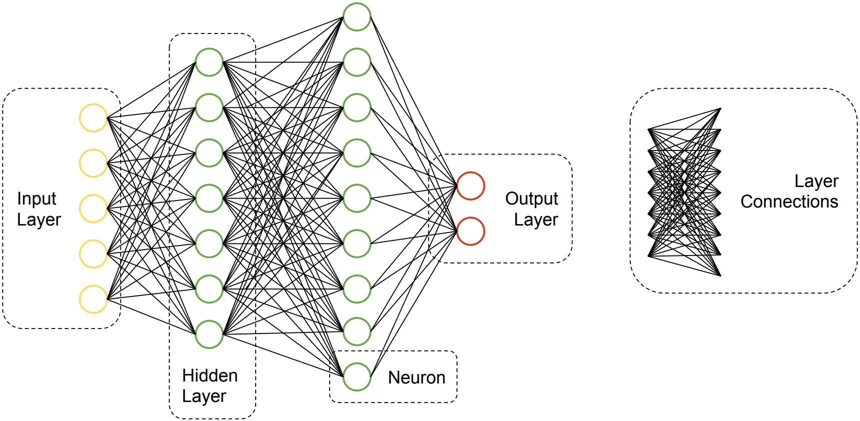
Linear Classifier

$$\widehat{y} = \textbf{argmax}\left(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}\right), \quad \boldsymbol{x} \in \mathbb{R}^D, \ \mathbf{W} = \begin{bmatrix} \vdots \\ w_{\boldsymbol{y}}^\top \\ \vdots \end{bmatrix}, \ \boldsymbol{b} = \begin{bmatrix} \vdots \\ b_{\boldsymbol{y}} \\ \vdots \end{bmatrix}$$

On to neural networks!

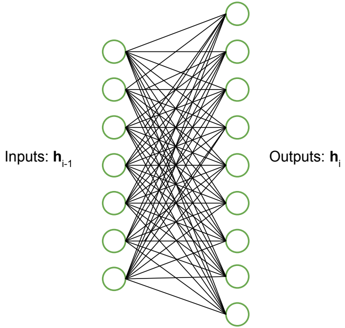


Neurons, Layers and Connections



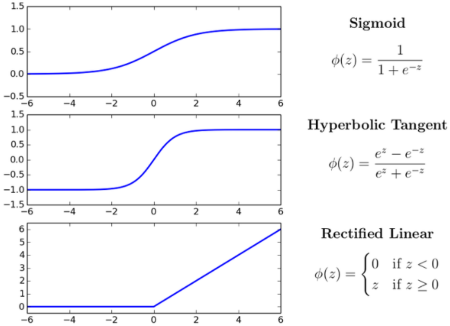
- A (dense / fully-connected) feed-forward neural network (FF-NN)
  - AKA a Multi-layer Perceptron (MLP)
- Input and output layers are special (more on this)
- However connections between layers take a similar form

Hidden Layer Connections



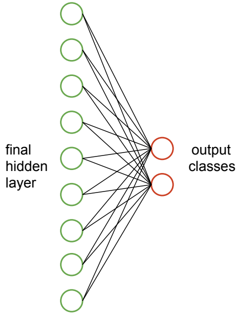
- Let $\boldsymbol{h}_i \in \mathbb{R}^{D_i}$ be the i^{th} hidden layer with D_i dimensions/neurons
- $\boldsymbol{h}_i = f_i(\mathbf{W}_i \boldsymbol{h}_{i-1} + \boldsymbol{b}_i)$
  - $\mathrm{logit}(\boldsymbol{h}_i) = \mathbf{W}_i \boldsymbol{h}_{i-1} + \boldsymbol{b}_i$
- $\mathbf{W}_i \in \mathbb{R}^{D_i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

- Non-linearity by transforming/projecting the data
- Squashes output to finite range
- Most common examples in NLP ...



From Hughes and Correll 2016

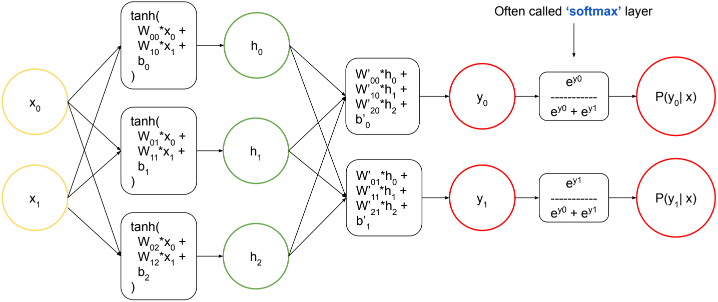
Output Layer



- This was first 2/3 of the lecture!
- $\widehat{\boldsymbol{y}} = \textbf{argmax}\, \boldsymbol{y}$; where $\boldsymbol{y} = \mathbf{W}_{\text{final}} \boldsymbol{h}_{\text{final}} + \boldsymbol{b}_{\text{final}}$
- $\text{logit}(\boldsymbol{y}_i) = \boldsymbol{y}_i$, is also used for output layer
- Various models correspond to different loss functions $L(\boldsymbol{w}; \boldsymbol{D})$
  - 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
  - Perceptron: perceptron loss (hinge at 0)

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



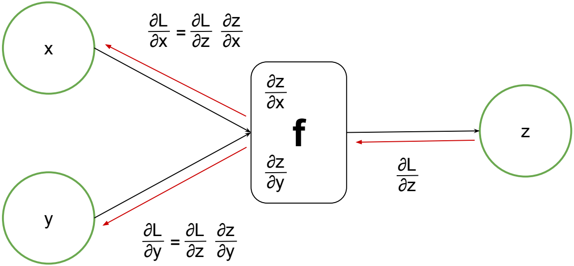
Neural Networks So Far

- 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 $\boldsymbol{w} = \{\mathbf{W}_i, \boldsymbol{b}_i \mid \forall i\}$
    - Including last output layer parameters
  - Loss function: $L(\boldsymbol{w}; (\boldsymbol{x}, \boldsymbol{y}))$
- Optimization
  - Non-linear models through input transformations
  - HOWEVER: hidden layers make model non-convex!
  - No single global optimum. Must settle for a local one.
  - If loss function and activation functions are differentiable, then can be optimized with gradient-based techniques (e.g., gradient descent)
  - Gradient computation a little trickier
    - Solution: backpropagation (Rumelhart et al. (1988))

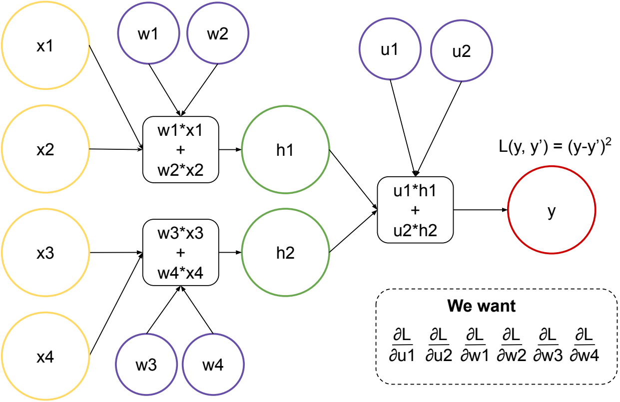
Backpropagation and the Chain Rule

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

Need to compute: $\frac{\partial L}{\partial w}$ for all variables w



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

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

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

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

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

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

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

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

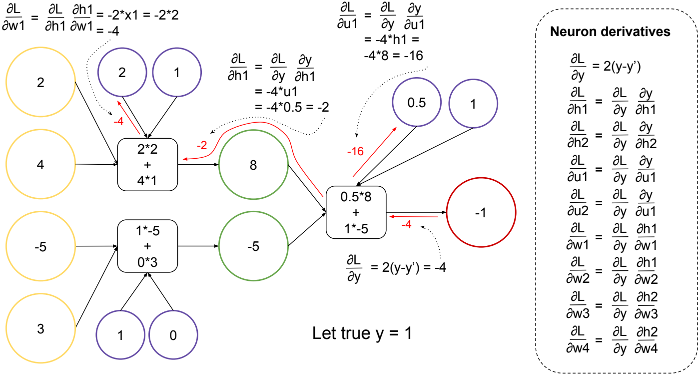
Full derivation examples

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

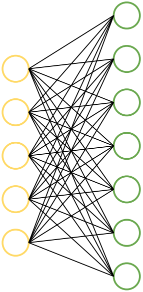
$$\frac{\partial L}{\partial w1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w1} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial h1}\frac{\partial h1}{\partial w1} = 2(y-y')*u1*x1$$

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

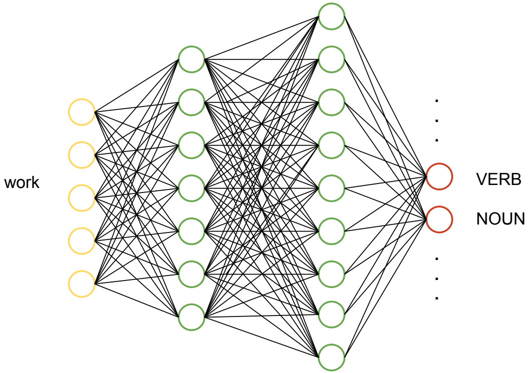


Input layer



- $\boldsymbol{x} \in \mathbb{R}^D$
- 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!

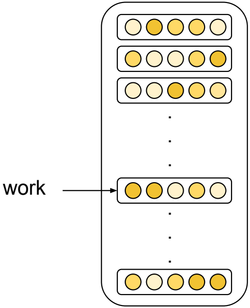
Input layer



- 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 $\boldsymbol{x}_i \in \mathcal{V}$

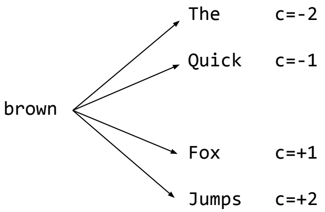
- word2vec (Mikolov et al. (2013))
  - Define two embeddings per word: $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$
  - $\boldsymbol{x}_i$ represents word as focus; $\boldsymbol{x}'_i$ as context
  - word2vec optimizes (SkipGram model):

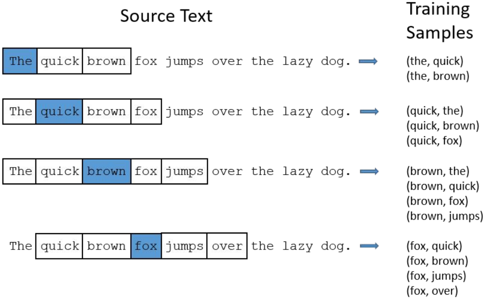
$$\sum_i^{|\mathcal{C}|} \sum_j^{|\mathcal{X}|} \sum_{-c \leq k \leq c, k \neq 0} \log p(x_{j+k}|x_j) = \sum_i^{|\mathcal{C}|} \sum_j^{|\mathcal{X}|} \sum_{-c \leq k \leq c, k \neq 0} \log \frac{e^{\boldsymbol{x}_j \cdot \boldsymbol{x}'_{j+k}}}{\sum_{\boldsymbol{x}_l \in \mathcal{V}} e^{\boldsymbol{x}_j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

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

Example Static Embedding Layer: Word2Vec

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





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

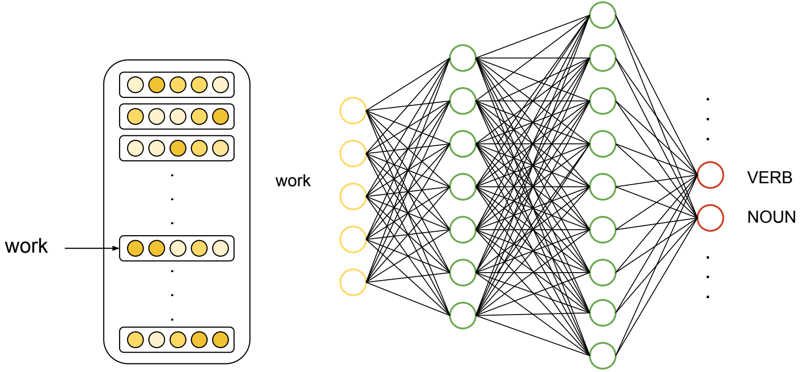
Fun word arithmetic artifact:

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

Embeddings via Language Models

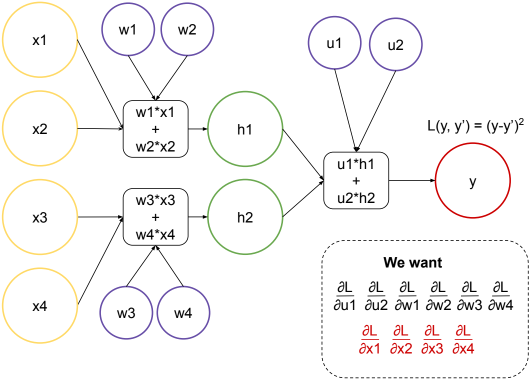
- word2vec is an example of a language model
- It models the probability of a word given a context
- Pre-trained contextual language models dominate NLP: ELMO, BERT, ROBERTA, XLNet, ...
- Huge gains in accuracy across multiple tasks
- Lecture 3 will cover RNNs, which is main building block

Input layer



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

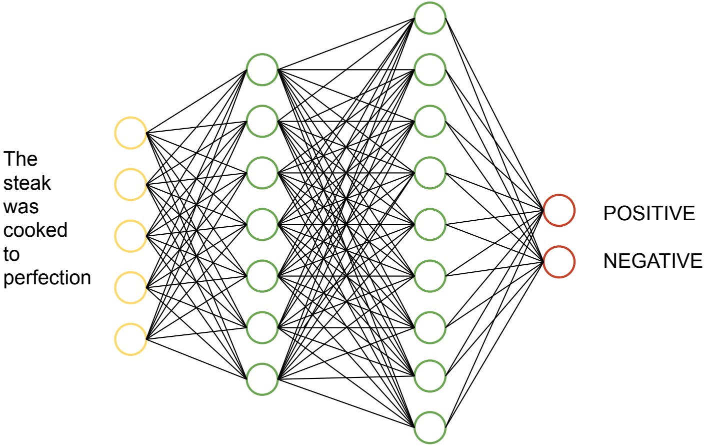
Dynamic Input layer



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

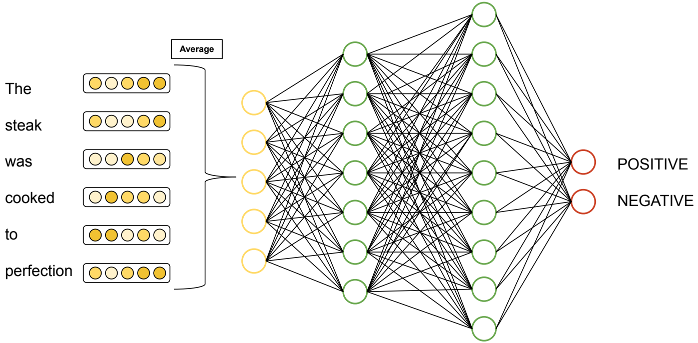
Input layer

- 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

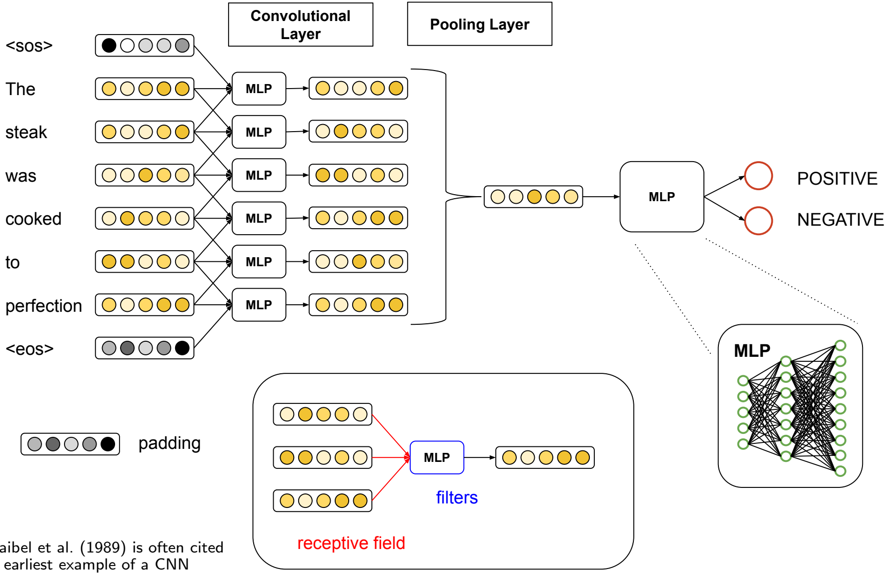


Input layer

- Truncate document at fixed length K, $\boldsymbol{x} \in \mathbb{R}^{K \times D}$
- Average embeddings (below), $\boldsymbol{x} \in \mathbb{R}^{D}$
- Even better: convolutional and recurrent neural networks (lecture 3)



Convolutional Neural Networks

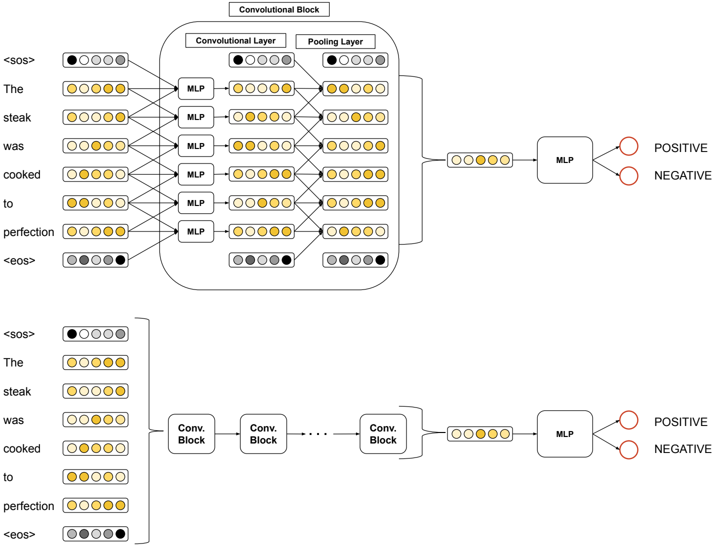


Waibel et al. (1989) is often cited as earliest example of a CNN

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?

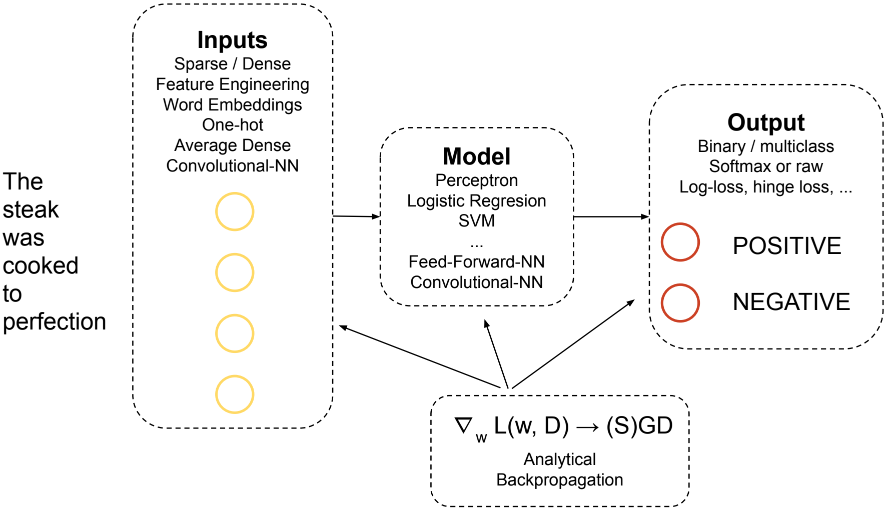
Where Does Network Structure Come From?

- 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)

Why Should I Use NNs?

- Fact:
  - Are almost always more accurate!
  - More natural to incorporate unlabeled data
    - E.g., pre-train word2vec on huge corpus and initialize
  - Multi-task learning is natural, e.g., share embedding/hidden layers
  - Tensorflow, PyTorch, Dynet, etc. lower barriers to entry
  - Entirely subsume all functionality of linear classifiers
- Fiction: No more feature engineering!
  - Feature engineering was not hard nor time consuming
  - Feature engineering was transparent (and parameters interpretable)
  - NNs replace this with less explainable hyperparameter and architecture engineering

Main Points


The steak was cooked to perfection
Inputs
Sparse / Dense
Feature Engineering
Word Embeddings
One-hot
Average Dense
Convolutional-NN
Model
Perceptron
Logistic Regresion
SVM
...
Feed-Forward-NN
Convolutional-NN
Output
Binary / multiclass
Softmax or raw
Log-loss, hinge loss, ...
POSITIVE
NEGATIVE
$\nabla_w L(w, D) \rightarrow$ (S)GD
Analytical
Backpropagation

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.