How does sodium bicarbonate work?

NOUN or VERB

Set my alarm tomorrow for 10am  -> Alarm
Quickest way to Boston          -> Navigation
Why is there summer and winter  -> Answer seeking

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

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

Ryan McDonald (Google)
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
This lecture is 2/3 about linear classifiers!


- 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 Classifier

Linear Classifier
Linear Classifiers and Neural Networks

Handcrafted Features

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

This is a binary classification problem.
Fake Or Real?

1. Fake:
   - "Rita Hayworth Says... I'M BACK FROM THE DEAD For Two Years I Was a Zombie"
   - "HILLARY CLINTON ADOPTS ALIEN BABY"

2. Real:
   - "Space creature survived UFO crash in Arkansas!"
   - "Secret Service building special nursery in the White House!"
Fake Or Real?

With Artificial Intelligence we are summoning the demons
- Elon Musk
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 Mamit 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.
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.
**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 Namit 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.

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: ⋆ ◊ ○;
  label: −1

• Example 2 – sequence: ⋆ ♥ △;
  label: −1

• Example 3 – sequence: ⋆ △ ♠;
  label: +1

• Example 4 – sequence: ◊ △ ○;
  label: +1

• New sequence: ⋆ ◊ ○; 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: ★ ◊ ◯; label: −1
- Example 2 – sequence: ★ ♥ △; label: −1
- Example 3 – sequence: ★ △ ♠; label: +1
- Example 4 – sequence: ◊ △ ◯; label: +1

- New sequence: ★ ◊ ◯; label −1
- New sequence: ★ ◊ ♥; label −1
- New sequence: ★ △ ◯; label ?

Why can we do this?
Let’s Start Simple: Machine Learning

- Example 1 – sequence: ⋆ ◊ ○; label: −1
- Example 2 – sequence: ⋆ ♥ △; label: −1
- Example 3 – sequence: ⋆ △ ♠; label: +1
- Example 4 – sequence: ◊ △ ○; label: +1

- New sequence: ⋆ ◊ ♥; label −1

\[
\begin{align*}
P(-1|⋆) &= \frac{\text{count}(⋆ \text{ and } -1)}{\text{count}(⋆)} = \frac{2}{3} = 0.67 \\
&\text{vs.} \quad P(+1|⋆) = \frac{\text{count}(⋆ \text{ and } +1)}{\text{count}(⋆)} = \frac{1}{3} = 0.33 \\
P(-1|◊) &= \frac{\text{count}(◊ \text{ and } -1)}{\text{count}(◊)} = \frac{1}{2} = 0.5 \\
&\text{vs.} \quad P(+1|◊) = \frac{\text{count}(◊ \text{ and } +1)}{\text{count}(◊)} = \frac{1}{2} = 0.5 \\
P(-1|♥) &= \frac{\text{count}(♥ \text{ and } -1)}{\text{count}(♥)} = \frac{1}{1} = 1.0 \\
&\text{vs.} \quad P(+1|♥) = \frac{\text{count}(♥ \text{ and } +1)}{\text{count}(♥)} = \frac{0}{1} = 0.0
\end{align*}
\]
Let’s Start Simple: Machine Learning

- Example 1 – sequence: ⋆ ◇ ○; label: −1
- Example 2 – sequence: ⋆ ♥ △; label: −1
- Example 3 – sequence: ⋆ △ ♠; label: +1
- Example 4 – sequence: ◇ △ ○; label: +1

- New sequence: ⋆ △ ○; label ?

<table>
<thead>
<tr>
<th>Label</th>
<th>−1</th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P(−1</td>
<td>⋆)$</td>
<td>$\frac{\text{count}(⋆ \text{ and } −1)}{\text{count}(⋆)} = \frac{2}{3} = 0.67$ vs.</td>
</tr>
<tr>
<td>$P(−1</td>
<td>△)$</td>
<td>$\frac{\text{count}(△ \text{ and } −1)}{\text{count}(△)} = \frac{1}{3} = 0.33$ vs.</td>
</tr>
<tr>
<td>$P(−1</td>
<td>○)$</td>
<td>$\frac{\text{count}(○ \text{ and } −1)}{\text{count}(○)} = \frac{1}{2} = 0.5$ vs.</td>
</tr>
</tbody>
</table>
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** $x \in X$
  - e.g., a news article, a sentence, an image, ...

• **Output** $y \in Y$
  - e.g., fake/not fake, a topic, a parse tree, an image segmentation

• **Input/Output pair:** $(x, y) \in X \times Y$
  - 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:
  \[ \mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N} \subseteq \mathcal{X} \times \mathcal{Y} \]

- **Goal**: use it to learn a **classifier** \( h : \mathcal{X} \rightarrow \mathcal{Y} \) that generalizes well to arbitrary inputs.

- At test time, given \( x \in \mathcal{X} \), we predict
  \[ \hat{y} = h(x). \]

- Hopefully, \( \hat{y} \approx y \) most of the time.
Things can go by different names depending on what $y$ is...
Deals with **continuous** output variables:

- **Regression**: $y = \mathbb{R}$
  - e.g., given a news article, how much time a user will spend reading it?

- **Multivariate regression**: $y = \mathbb{R}^K$
  - e.g., predict the X-Y coordinates in an image where the user will click
Deals with discrete output variables:

- **Binary classification**: $Y = \{\pm 1\}$
  - e.g., fake news detection

- **Multi-class classification**: $Y = \{1, 2, \ldots, K\}$
  - e.g., topic classification

- **Structured classification**: $Y$ exponentially large and structured
  - e.g., machine translation, caption generation, image segmentation
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
We need to represent information about $x$

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

- $\psi(x)$ 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 $\phi : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^D$ is sometimes more convenient

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

Each feature now represents a joint property of the input $x$ and the candidate output $y$. 
Examples

- $x$ is a document and $y$ is a topic

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

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

- $x$ is a word and $y$ is a part-of-speech tag

$$
\phi_j(x, y) = \begin{cases} 
1 & \text{if } x = \text{ ends in “ed” and } y = \text{ Verb} \\
0 & \text{otherwise}
\end{cases}
$$
- $x$ is a name, $y$ is a label classifying the type of entity

\[
\begin{align*}
\phi_0(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "George" and } y = \text{"Person"} \\
0 & \text{otherwise}
\end{cases} \\
\phi_4(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "George" and } y = \text{"Location"} \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\phi_1(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "Washington" and } y = \text{"Person"} \\
0 & \text{otherwise}
\end{cases} \\
\phi_5(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "Washington" and } y = \text{"Location"} \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\phi_2(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "Bridge" and } y = \text{"Person"} \\
0 & \text{otherwise}
\end{cases} \\
\phi_6(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "Bridge" and } y = \text{"Location"} \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

\[
\begin{align*}
\phi_3(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "General" and } y = \text{"Person"} \\
0 & \text{otherwise}
\end{cases} \\
\phi_7(x, y) &= \begin{cases} 
1 & \text{if } x \text{ contains "General" and } y = \text{"Location"} \\
0 & \text{otherwise}
\end{cases}
\end{align*}
\]

- $x=$General George Washington, $y=$Person $\rightarrow \phi(x, y) = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]$
- $x=$George Washington Bridge, $y=$Location $\rightarrow \phi(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0]$
- $x=$George Washington George, $y=$Location $\rightarrow \phi(x, y) = [0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0]$
Block Feature Vectors

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

- Each equal size block of the feature vector corresponds to one label
- Non-zero values allowed only in one block
Feature Representations – $\psi(x) \text{ vs. } \phi(x, y)$

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

$$\phi(x, y) = \psi(x) \otimes e_y$$

$$= [0, \ldots, 0, \underbrace{\psi(x)}_{y^{th} \text{ block}}, 0, \ldots, 0]$$

- $\psi(x)$
  - $x=$ General George Washington $\rightarrow \psi(x) = [1 \ 1 \ 0 \ 1]$

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

$\psi(x)$ is sometimes simpler and more convenient … but $\phi(x, y)$ is more expressive
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.
Goal: estimate the quality of a translation on the fly (without a reference)!
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.
Let’s assume a multi-class classification problem, with $|\mathcal{Y}|$ labels (classes).
Linear Classifiers

- Parametrized by a **weight vector** \( w \in \mathbb{R}^D \) (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 \( w \)), predict the class \( \hat{y} \) which maximizes this score:

  \[
  \hat{y} = h(x) = \arg \max_{y \in Y} w \cdot \phi(x, y) = \arg \max_{y \in Y} \sum_i w_i \phi(x, y);
  \]

- At training time, different strategies to learn \( w \) yield different linear classifiers: perceptron, logistic regression, SVMs, ...
Linear Classifiers – $\psi(x)$

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

- Classification

$$\hat{y} = \arg \max_{y \in \mathcal{Y}} w_y \cdot \psi(x)$$
Linear Classifiers – $\psi(x)$

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

- Classification
  $$\hat{y} = \arg\max_{y \in \mathcal{Y}} w_y \cdot \psi(x)$$

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

- $\psi(x)$
  - $x=$General George Washington $\rightarrow \psi(x) = [1 1 0 1]$
  - Two parameter vectors $w_{\text{Person}} \in \mathbb{R}^4, w_{\text{Object}} \in \mathbb{R}^4$
Often linear classifiers are presented as

$$\hat{y} = \arg \max_{y \in Y} \mathbf{w}_y \cdot \psi(x) + b_y$$

where $b_y$ is a bias or offset term.

This can be folded into $\psi(x)$ (by defining a constant feature for each label).

For now, we assume this for simplicity.
\[ \hat{y} = \arg\max (W\psi(x) + b), \quad W = \begin{bmatrix} \vdots & \vdots \\ w_y^\top & \vdots \\ \vdots \\ \vdots \end{bmatrix}, \quad b = \begin{bmatrix} \vdots \\ b_y \\ \vdots \end{bmatrix} . \]
With binary labels \( \mathcal{Y} = \{ \pm 1 \} \) we often use a minimal parametrization:

\[
\hat{y} = \arg \max_{y \in \{ \pm 1 \}} w_y \cdot \psi(x) + b_y
\]
With binary labels \((\mathcal{Y} = \{\pm 1\})\) we often use a minimal parametrization:

\[
\hat{y} = \arg \max_{y \in \{\pm 1\}} \mathbf{w}_y \cdot \psi(x) + b_y
\]

\[
= \begin{cases} 
  +1 & \text{if } \mathbf{w}_+ \cdot \psi(x) + b_+ > \mathbf{w}_- \cdot \psi(x) + b_- \\
  -1 & \text{otherwise}
\end{cases}
\]
With binary labels $(\mathbf{y} = \{\pm 1\})$ we often use a minimal parametrization:

$$
\hat{y} = \arg \max_{y \in \{\pm 1\}} w_y \cdot \psi(x) + b_y
$$

$$
= \begin{cases} 
+1 & \text{if } w_{+1} \cdot \psi(x) + b_{+1} > w_{-1} \cdot \psi(x) + b_{-1} \\
-1 & \text{otherwise}
\end{cases}
$$

$$
= \text{sign}((w_{+1} - w_{-1}) \cdot \psi(x) + (b_{+1} - b_{-1})).
$$
With binary labels ($Y = \{\pm 1\}$) we often use a minimal parametrization:

$$\hat{y} = \arg \max_{y \in \{\pm 1\}} w_y \cdot \psi(x) + b_y$$

$$= \begin{cases} 
+1 & \text{if } w_{+1} \cdot \psi(x) + b_{+1} > w_{-1} \cdot \psi(x) + b_{-1} \\
-1 & \text{otherwise} 
\end{cases}$$

$$= \text{sign}((w_{+1} - w_{-1}) \cdot \psi(x) + (b_{+1} - b_{-1})).$$

That is: only half of the parameters are needed.
Then $(\mathbf{v}, c)$ is an hyperplane that divides all points:

Points along line have scores of 0
Multiclass Linear Classifier

Defines regions of space.
Linear Classifiers

• Prediction rule:

\[
\hat{y} = h(x) = \arg \max_{y \in \mathcal{Y}} \left\{ w \cdot \phi(x, y) \right\}
\]

linear in \( w \)

• The decision boundary is defined by the intersection of half spaces

• In the binary case (\(|\mathcal{Y}| = 2\)) this corresponds to a hyperplane classifier
A set of points is **linearly separable** if there exists a $\mathbf{w}$ such that classification is perfect.
Outline

1. Terminology, notation and feature representations
2. Perceptron
3. Logistic Regression
4. Support Vector Machines
5. Regularization
6. Neural Networks
Perceptron (Rosenblatt, 1958)

- 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.

(Extracted from Wikipedia)
NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

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

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

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

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

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

Without Human Controls

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

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

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

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

1958 New York Times...

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

Learns by Doing

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

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

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.
• **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 \( \mathbf{w}^{(0)} = 0 \)
initialize \( k = 0 \) (number of mistakes)
repeat
  get new training example \( (x_i, y_i) \in \mathcal{D} \)
predict \( \hat{y}_i = \arg \max_{y \in Y} \mathbf{w}^{(k)} \cdot \phi(x_i, y) \)
if \( \hat{y}_i \neq y_i \) then
  update \( \mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \phi(x_i, y_i) - \phi(x_i, \hat{y}_i) \)
  increment \( k \)
end if
until maximum number of epochs
**output:** model weights \( \mathbf{w} \)
Perceptron’s Mistake Bound

A couple definitions:

- the training data is **linearly separable** with margin $\gamma > 0$ iff there is a weight vector $u$ with $\|u\| = 1$ such that
  \[
  u \cdot \phi(x_i, y_i) \geq u \cdot \phi(x_i, y'_i) + \gamma, \quad \forall i, \forall y'_i \neq y_i.
  \]

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

Theorem (Novikoff (1962))
The perceptron algorithm is guaranteed to find a separating hyperplane after at most $R^2 \gamma^2$ mistakes.
Perceptron’s Mistake Bound

A couple definitions:

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

$$u \cdot \phi(x_i, y_i) \geq u \cdot \phi(x_i, y'_i) + \gamma, \quad \forall i, \forall y'_i \neq y_i.$$

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

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

\[
\begin{align*}
\mathbf{u} \cdot w^{(k+1)} &= \mathbf{u} \cdot w^{(k)} + \mathbf{u} \cdot (\phi(x_i, y_i) - \phi(x_i, \hat{y}_i)) \\
&\geq \mathbf{u} \cdot w^{(k)} + \gamma \\
&\geq k\gamma.
\end{align*}
\]

Hence \( \|w^{(k+1)}\| = \|\mathbf{u}\| \cdot \|w^{(k+1)}\| \geq \mathbf{u} \cdot w^{(k+1)} \geq k\gamma \) (from CSI).
One-Slide Proof

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

  \[ u \cdot w^{(k+1)} = u \cdot w^{(k)} + u \cdot (\phi(x_i, y_i) - \phi(x_i, \hat{y}_i)) \geq u \cdot w^{(k)} + \gamma \geq k\gamma. \]

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

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

  \[ \|w^{(k+1)}\|^2 = \|w^{(k)}\|^2 + \|\phi(x_i, y_i) - \phi(x_i, \hat{y}_i)\|^2 + 2w^{(k)} \cdot (\phi(x_i, y_i) - \phi(x_i, \hat{y}_i)) \leq \|w^{(k)}\|^2 + R^2 \leq kR^2. \]

  Equating both sides, we get $(k\gamma)^2 \leq kR^2 \Rightarrow k \leq \frac{R^2}{\gamma^2}$ (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)
... 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.
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:

\[ P(y|x) = \frac{\exp(w \cdot \phi(x, y))}{Z_x}, \quad \text{where } Z_x = \sum_{y' \in \mathcal{Y}} \exp(w \cdot \phi(x, y')) \]

Exponentiating and normalizing is called the softmax transformation\(^2\)

Critically \(\sum_y P(y|x) = 1\)

Note: still a linear classifier

\[
\arg \max_y P(y|x) = \arg \max_y \frac{\exp(w \cdot \phi(x, y))}{Z_x} = \arg \max_y \frac{Z_x}{Z_x} \exp(w \cdot \phi(x, y)) = \arg \max_y w \cdot \phi(x, y)
\]

\(^2\)More later during neural networks!
Logistic Regression

\[ P_w(y|x) = \frac{\exp(w \cdot \phi(x, y))}{Z_x} \]

- How do we learn weights \( w \)?
- Set \( w \) to minimize the negative conditional log-likelihood:

\[
\hat{w} = \arg \min_{w \in \mathbb{R}^D} - \log \left( \prod_{t=1}^{N} P_w(y_t|x_t) \right) = \arg \min_{w \in \mathbb{R}^D} - \sum_{t=1}^{N} \log P_w(y_t|x_t)
\]

\[
= \arg \min_{w \in \mathbb{R}^D} \sum_{t=1}^{N} \left( \log \sum_{y'_t} \exp(w \cdot \phi(x_t, y'_t)) - w \cdot \phi(x_t, y_t) \right),
\]

i.e., set \( w \) 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, ...)

Proof left as an exercise!
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 $f : \mathbb{R}^d \rightarrow \mathbb{R}$

- Proceed in **small steps** in the **optimal direction** till a **stopping criterion** is met.
- **Gradient descent** updates: $w^{(k+1)} \leftarrow w^{(k)} - \eta_k \nabla f(w^{(k)})$

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

- Let $L(w; (x, y)) = \log \sum_{y'} \exp(w \cdot \phi(x, y')) - w \cdot \phi(x, y)$
- This is our loss function!
  - Logistic-regressions loss function often called log-loss or cross-entropy
- We want to find $\arg\min_w \sum_{t=1}^N L(w; (x_t, y_t))$
  - Set $w^0 = 0$
  - Iterate until convergence (for suitable stepsize $\eta_k$):
    $$w^{k+1} = w^k - \eta_k \nabla_w \left( \sum_{t=1}^N L(w; (x_t, y_t)) \right)$$
    $$= w^k - \eta_k \sum_{t=1}^N \nabla_w L(w; (x_t, y_t))$$
- $\nabla_w L(w)$ is gradient of $L$ w.r.t. $w$
- Gradient descent will always find the optimal $w$
If the dataset is large, we’d better do SGD instead, for more frequent updates:

- Set $w^0 = 0$
- Iterate until convergence
  - Pick $(x_t, y_t)$ randomly
  - Update $w^{k+1} = w^k - \eta_k \nabla_w L(w; (x_t, y_t))$

- 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 $w$ (for suitable step sizes)
For this to work, we need to be able to compute $\nabla_w L(w; (x_t, y_t))$, where

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

Some reminders:

1. $\nabla_w \log F(w) = \frac{1}{F(w)} \nabla_w F(w)$
2. $\nabla_w \exp F(w) = \exp(F(w)) \nabla_w F(w)$
Computing the Gradient

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

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

$$= \frac{1}{\sum_{y'} \exp(w \cdot \phi(x, y'))} \sum_{y'} \nabla_w \exp(w \cdot \phi(x, y')) - \phi(x, y)$$

$$= \frac{1}{Z_x} \sum_{y'} \exp(w \cdot \phi(x, y')) \nabla_w w \cdot \phi(x, y') - \phi(x, y)$$

$$= \sum_{y'} \frac{\exp(w \cdot \phi(x, y'))}{Z_x} \phi(x, y') - \phi(x, y)$$

$$= \sum_{y'} P_w(y'|x) \phi(x, y') - \phi(x, y).$$

The gradient equals the “difference between the expected features under the current model and the true features.”
Logistic Regression Summary

- Define conditional probability

\[ P_w(y|x) = \frac{\exp(w \cdot \phi(x, y))}{Z_x} \]

- Set weights to minimize negative conditional log-likelihood:

\[ w = \arg \min_w \sum_t - \log P_w(y_t|x_t) = \arg \min_w \sum_t L(w; (x_t, y_t)) \]

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

\[ \nabla_w L(w; (x, y)) = \sum_{y'} P_w(y'|x) \phi(x, y') - \phi(x, y) \]
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

\[ w^{k+1} = w^k + \eta \left( \phi(x, y) - \sum_{y'} P_w(y'|x) \phi(x, y') \right) \]

Perceptron is a discriminative, non-probabilistic classifier

- perceptron’s updates look like

\[ w^{k+1} = w^k + \phi(x, y) - \phi(x, \hat{y}) \]

SGD updates for logistic regression and perceptron’s updates look similar!
Maximizing Margin

- For a training set $\mathcal{D}$
- Margin of a weight vector $\mathbf{w}$ is smallest $\gamma$ such that
  \[
  \mathbf{w} \cdot \phi(x_t, y_t) - \mathbf{w} \cdot \phi(x_t, y') \geq \gamma
  \]
- for every training instance $(x_t, y_t) \in \mathcal{D}$, $y' \in \mathcal{Y}$
Denote the value of the margin by $\gamma$. 

Training

Testing

Margin
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 \( w \) that separates the data
  • However, the perceptron does not pick \( w \) 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
Let $\gamma > 0$

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

such that:

$$w \cdot \phi(x_t, y_t) - w \cdot \phi(x_t, y') \geq \gamma$$

$\forall (x_t, y_t) \in D$

and $y' \in y, y' \neq y_t$

- Note: algorithm still minimizes error if data is separable
- $\|w\|$ is bound since scaling trivially produces larger margin
Let $\gamma > 0$

**Max Margin:**

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

such that:

$$w \cdot \phi(x_t, y_t) - w \cdot \phi(x_t, y') \geq \gamma$$

$\forall (x_t, y_t) \in D$

and $y' \in Y$, $y' \neq y_t$

**Min Norm:**

$$\min_w \frac{1}{2} \|w\|^2$$

such that:

$$w \cdot \phi(x_t, y_t) - w \cdot \phi(x_t, y') \geq 1$$

$\forall (x_t, y_t) \in D$

and $y' \in Y$, $y' \neq y_t$

- Instead of fixing $\|w\|$ we fix the margin $\gamma = 1$

- Make substitution $w' = w / \gamma$; then we have $\gamma = \frac{\|w\|}{\|w'\|} = \frac{1}{\|w'\|}$. 
\[
\mathbf{w} = \text{arg min}_w \frac{1}{2}||\mathbf{w}||^2
\]
such that:
\[
\mathbf{w} \cdot \phi(x_t, y_t) - \mathbf{w} \cdot \phi(x_t, y') \geq 1
\]
\[
\forall (x_t, y_t) \in \mathcal{D} \text{ and } y' \in \mathcal{Y}, \ y' \neq y_t
\]

- **Quadratic programming problem** – a well known convex optimization problem
- Can be solved with many techniques.
What if data is not separable?

\[
\mathbf{w} = \arg\min_{\mathbf{w}, \xi} \frac{1}{2} ||\mathbf{w}||^2 + C \sum_{t=1}^{N} \xi_t
\]

such that:

\[
\mathbf{w} \cdot \phi(\mathbf{x}_t, y_t) - \mathbf{w} \cdot \phi(\mathbf{x}_t, y') \geq 1 - \xi_t \quad \text{and} \quad \xi_t \geq 0
\]

\[
\forall (\mathbf{x}_t, y_t) \in \mathcal{D} \quad \text{and} \quad y' \in \mathcal{Y}, \ y' \neq y_t
\]

\(\xi_t\): trade-off between margin per example and \(||\mathbf{w}||\)

Larger \(C = \) more examples correctly classified.
\[ w = \arg \min_{w, \xi} \frac{\lambda}{2} \|w\|^2 + \sum_{t=1}^{N} \xi_t \quad \lambda = \frac{1}{C} \]

such that:

\[ w \cdot \phi(x_t, y_t) - w \cdot \phi(x_t, y') \geq 1 - \xi_t, \quad \forall (x_t, y_t) \in \mathcal{D} \text{ and } y' \in \mathcal{Y}, \ y' \neq y_t \]
$w = \arg \min_{w,\xi} \frac{\lambda}{2} \|w\|^2 + \sum_{t=1}^{N} \xi_t \quad \lambda = \frac{1}{C}$

such that:

$w \cdot \phi(x_t, y_t) - \max_{y' \neq y_t} w \cdot \phi(x_t, y') \geq 1 - \xi_t, \quad \forall (x_t, y_t) \in D$
\[ w = \arg \min_{w, \xi} \frac{\lambda}{2} ||w||^2 + \sum_{t=1}^{N} \xi_t \quad \lambda = \frac{1}{C} \]

such that:

\[ \xi_t \geq 1 + \max_{y' \neq y_t} w \cdot \phi(x_t, y') - w \cdot \phi(x_t, y_t), \quad \forall (x_t, y_t) \in \mathcal{D} \]
Support Vector Machines

\[ \mathbf{w} = \arg \min_{\mathbf{w}, \xi} \frac{\lambda}{2} \| \mathbf{w} \|^2 + \sum_{t=1}^{N} \xi_t \quad \lambda = \frac{1}{C} \]

such that:

\[ \xi_t \geq 1 + \max_{y' \neq y_t} w \cdot \phi(x_t, y') - w \cdot \phi(x_t, y_t), \quad \forall (x_t, y_t) \in \mathcal{D} \]

If \( \mathbf{w} \) classifies \((x_t, y_t)\) with margin 1, penalty \( \xi_t = 0 \) (by def'n \( \xi_t \geq 0 \))

Otherwise penalty \( \xi_t = 1 + \max_{y' \neq y_t} w \cdot \phi(x_t, y') - w \cdot \phi(x_t, y_t) \)
Support Vector Machines

\[ w = \arg \min_{w, \xi} \quad \frac{\lambda}{2} \|w\|^2 + \sum_{t=1}^{N} \xi_t \quad \lambda = \frac{1}{C} \]

such that:

\[ \xi_t \geq 1 + \max_{y' \neq y_t} \, w \cdot \phi(x_t, y') - w \cdot \phi(x_t, y_t), \quad \forall (x_t, y_t) \in D \]

If \( w \) classifies \((x_t, y_t)\) with margin 1, penalty \( \xi_t = 0 \) (by def'n \( \xi_t \geq 0 \))

Otherwise penalty \( \xi_t = 1 + \max_{y' \neq y_t} \, w \cdot \phi(x_t, y') - w \cdot \phi(x_t, y_t) \)

Hinge loss:

\[ L(w; (x_t, y_t)) = \max \left( 0, 1 + \max_{y' \neq y_t} \, w \cdot \phi(x_t, y') - w \cdot \phi(x_t, y_t) \right) \]
Support Vector Machines

\[ w = \arg \min_{w, \xi} \quad \frac{\lambda}{2} ||w||^2 + \sum_{t=1}^{N} \xi_t \]

such that:

\[ \xi_t \geq 1 + \max_{y' \neq y_t} w \cdot \phi(x_t, y') - w \cdot \phi(x_t, y_t), \quad \forall (x_t, y_t) \in \mathcal{D} \]

Hinge loss equivalent

\[ w = \arg \min_{w} \quad \sum_{t=1}^{N} L((x_t, y_t); w) + \frac{\lambda}{2} ||w||^2 \]

\[ = \arg \min_{w} \left( \sum_{t=1}^{N} \max (0, 1 + \max_{y' \neq y_t} w \cdot \phi(x_t, y') - w \cdot \phi(x_t, y_t)) \right) + \frac{\lambda}{2} ||w||^2 \]
Summary

What we have covered

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

What is next

• Regularization
• Non-linear classifiers
Outline

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

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

where $\Omega(w)$ is the regularization function, and $\lambda$ controls how much to regularize.

- Gaussian prior ($\ell_2$), promotes smaller weights:
  $$\Omega(w) = \|w\|_2^2 = \sum_i w_i^2.$$

- Laplacian prior ($\ell_1$), promotes sparse weights!
  $$\Omega(w) = \|w\|_1 = \sum_i |w_i|.$$
Logistic Regression with $\ell_2$ Regularization

$$\sum_{t=1}^{N} L(w; (x_t, y_t)) + \lambda \Omega(w) = -\sum_{t=1}^{N} \log \left( \frac{\exp(w \cdot \phi(x_t, y_t))}{Z_x} \right) + \frac{\lambda}{2} \|w\|^2$$

- What is the new gradient?

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

- We know $\nabla_w L(w; (x_t, y_t))$

- Just need $\nabla_w \frac{\lambda}{2} \|w\|^2 = \lambda w$
Support Vector Machines

Hinge-loss formulation: $\ell_2$ regularization already happening!

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

$$= \arg \min_w \sum_{t=1}^{N} \max (0, 1 + \max_{y \neq y_t} w \cdot \phi(x_t, y) - w \cdot \phi(x_t, y_t)) + \lambda \Omega(w)$$

$$= \arg \min_w \sum_{t=1}^{N} \max (0, 1 + \max_{y \neq y_t} w \cdot \phi(x_t, y) - w \cdot \phi(x_t, y_t)) + \frac{\lambda}{2} \|w\|^2$$

↑ SVM optimization ↑
SVMs vs. Logistic Regression

\[ w = \arg \min_w \sum_{t=1}^{N} L(w; (x_t, y_t)) + \lambda \Omega(w) \]
SVMs vs. Logistic Regression

\[ w = \arg \min_w \sum_{t=1}^{N} L(w; (x_t, y_t)) + \lambda \Omega(w) \]

SVMs/hinge-loss: \[ \max (0, 1 + \max_{y \neq y_t} (w \cdot \phi(x_t, y) - w \cdot \phi(x_t, y_t))) \]

\[ w = \arg \min_w \sum_{t=1}^{N} \max (0, 1 + \max_{y \neq y_t} w \cdot \phi(x_t, y) - w \cdot \phi(x_t, y_t)) + \frac{\lambda}{2} \|w\|^2 \]
SVMs vs. Logistic Regression

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

SVMs/hinge-loss: \( \max (0, 1 + \max_{y \neq y_t} (\mathbf{w} \cdot \phi(x_t, y) - \mathbf{w} \cdot \phi(x_t, y_t))) \)

\[
\mathbf{w} = \arg \min_{\mathbf{w}} \sum_{t=1}^{N} \max (0, 1 + \max_{y \neq y_t} \mathbf{w} \cdot \phi(x_t, y) - \mathbf{w} \cdot \phi(x_t, y_t)) + \frac{\lambda}{2} \|\mathbf{w}\|^2
\]

Logistic Regression/log-loss: \( \log \sum_{y_t'} \exp(\mathbf{w} \cdot \phi(x_t, y_t')) - \mathbf{w} \cdot \phi(x_t, y_t) \)

\[
\mathbf{w} = \arg \min_{\mathbf{w} \in \mathbb{R}^D} \sum_{t=1}^{N} \left( \log \sum_{y_t'} \exp(\mathbf{w} \cdot \phi(x_t, y_t')) - \mathbf{w} \cdot \phi(x_t, y_t) \right) + \frac{\lambda}{2} \|\mathbf{w}\|^2
\]

\[
\mathbf{w} = \arg \min_{\mathbf{w} \in \mathbb{R}^D} \sum_{t=1}^{N} \left( \sum_{y_t'} P(y_t'|x) - P(y_t|x) \right) + \frac{\lambda}{2} \|\mathbf{w}\|^2
\]
SVMs vs. Perceptron

\[ w = \arg\min_w \sum_{t=1}^{N} L(w; (x_t, y_t)) + \lambda \Omega(w) \]

SVMs/hinge-loss: \( \max (0, 1 + \max_{y \neq y_t} (w \cdot \phi(x_t, y) - w \cdot \phi(x_t, y_t))) \)

\[ w = \arg\min_w \sum_{t=1}^{N} \max (0, 1 + \max_{y \neq y_t} w \cdot \phi(x_t, y) - w \cdot \phi(x_t, y_t)) + \frac{\lambda}{2} \|w\|^2 \]
SVMs vs. Perceptron

\[ w = \arg \min_w \sum_{t=1}^{N} L(w; (x_t, y_t)) + \lambda \Omega(w) \]

**SVMs/hinge-loss:** \( \max (0, 1 + \max_{y \neq y_t} (w \cdot \phi(x_t, y) - w \cdot \phi(x_t, y_t))) \)

\[ w = \arg \min_w \sum_{t=1}^{N} \max (0, 1 + \max_{y \neq y_t} w \cdot \phi(x_t, y) - w \cdot \phi(x_t, y_t)) + \frac{\lambda}{2} \|w\|^2 \]

**Perceptron/hinge-loss:** \( \max (0, 1 + \max_{y \neq y_t} (w \cdot \phi(x_t, y) - w \cdot \phi(x_t, y_t))) \)

\[ w = \arg \min_w \sum_{t=1}^{N} \max (0, 1 + \max_{y \neq y_t} w \cdot \phi(x_t, y) - w \cdot \phi(x_t, y_t)) + \frac{\lambda}{2} \|w\|^2 \]
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 ...
Could not possibly cover everything.
Please look at Andre Martins excellent lecture for LXMLS:

- Also covers
  - Naive Bayes
  - Sub-gradient descent
    - Needed for SVMs
    - Perceptron update is sub-gradient with no margin
  - Non-Linear Classifiers ≠ Neural Networks
    - K-Nearest neighbors
    - Kernel methods
1 Terminology, notation and feature representations

2 Perceptron

3 Logistic Regression

4 Support Vector Machines

5 Regularization

6 Neural Networks
A linear classifier with handcrafted features can be represented by the equation:

\[ \hat{y} = \arg\max \left( W \psi(x) + b \right), \quad W = \begin{bmatrix} \vdots \ w_y^\top \ \\
\vdots \end{bmatrix}, \quad b = \begin{bmatrix} \vdots \ b_y \end{bmatrix}. \]
\[ \hat{y} = \text{argmax} \left( Wx + b \right), \quad x \in \mathbb{R}^D, \quad W = \begin{bmatrix} \vdots \\ w_y^\top \\ \vdots \end{bmatrix}, \quad b = \begin{bmatrix} \vdots \\ b_y \end{bmatrix} \]
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
• Let \( h_i \in \mathbb{R}^{D_i} \) be the \( i^{th} \) hidden layer with \( D_i \) dimensions/neurons

\[
  h_i = f_i(W_i h_{i-1} + b_i)
\]

  \[
  \text{logit}(h_i) = W_i h_{i-1} + b_i
  \]

• \( W_i \in \mathbb{R}^{D_i \times D_{i-1}} \) and \( b_i \in D_i \) are layer parameters

• \( f_i \) is the layer’s (non-linear) activation function
Activation Functions

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

![Graph of activation functions](image)

- **Sigmoid**
  \[ \phi(z) = \frac{1}{1 + e^{-z}} \]

- **Hyperbolic Tangent**
  \[ \phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \]

- **Rectified Linear**
  \[ \phi(z) = \begin{cases} 
  0 & \text{if } z < 0 \\
  z & \text{if } z \geq 0 
\end{cases} \]

From Hughes and Correll 2016
This was first 2/3 of the lecture!

\[ \hat{y} = \text{argmax} \ y; \text{ where } y = W_{\text{final}} h_{\text{final}} + b_{\text{final}} \]

\( \text{logit}(y_i) = y_i, \) is also used for output layer

Various models correspond to different loss functions \( L(w; D) \)

- Logistic regression: log-loss/cross-entropy via softmax
  \[ \frac{e^{\text{logit}(y)}}{\sum_{y' \in y} e^{\text{logit}(y')}} \]
- SVMs: hinge-loss
- Perceptron: perceptron loss (hinge at 0)
An Wee Example

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

Often called 'softmax' layer
Neural Networks So Far

- Neural network structure (FF-NN; MLP)
  - Input layer: for now, assume given to us $x \in \mathbb{R}^D$
  - Outputs: $y \in Y$
  - Hidden layers: $h_i \in \mathbb{R}^{D_i}$; with $h_i = f_i(W_i h_{i-1} + b_i)$
    - Thus, model parameters $w = \{W_i, b_i \mid \forall i\}$
    - Including last output layer parameters
  - Loss function: $L(w; (x, y))$

- 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 $\nabla_w L(w; D) = [\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \ldots], \forall w_i \in w$
  - For linear classifiers, $w$ were feature weights
  - For NNs, $w$ is the set of all weights, e.g., $w = \{W_i, b_i \mid \forall i\}$
- 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
\[ \frac{\partial L}{\partial u_1} = 2(y-y') \frac{y}{y'} \]
\[ \frac{\partial L}{\partial u_2} = 2(y-y') \frac{u_1}{y'} \]
\[ \frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial u_1} = 2(y-y') u_1 \frac{y}{y'} \]
\[ \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial u_2} = 2(y-y') u_2 \frac{y}{y'} \]
\[ \frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial h_1} = 2(y-y') \frac{y}{y'} \]
\[ \frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial h_2} = 2(y-y') \frac{y}{y'} \]

Full derivation examples
\[ \frac{\partial L}{\partial y} = 2(y-y') \]
\[ \frac{\partial y}{\partial u_1} = u_1 \frac{y}{y'} \]
\[ \frac{\partial y}{\partial u_2} = u_2 \frac{y}{y'} \]
\[ \frac{\partial y}{\partial w_1} = \frac{y}{y'} \]
\[ \frac{\partial y}{\partial w_2} = \frac{y}{y'} \]
\[ \frac{\partial y}{\partial h_1} = \frac{y}{y'} \]
\[ \frac{\partial y}{\partial h_2} = \frac{y}{y'} \]

All base derivatives
\[ \frac{\partial L}{\partial y} = 2(y-y') \]
\[ \frac{\partial y}{\partial u_1} = u_1 \frac{y}{y'} \]
\[ \frac{\partial y}{\partial u_2} = u_2 \frac{y}{y'} \]
\[ \frac{\partial y}{\partial w_1} = \frac{y}{y'} \]
\[ \frac{\partial y}{\partial w_2} = \frac{y}{y'} \]
\[ \frac{\partial y}{\partial h_1} = \frac{y}{y'} \]
\[ \frac{\partial y}{\partial h_2} = \frac{y}{y'} \]
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

Let true $y = 1$

Neuron derivatives

- $\frac{\partial L}{\partial y} = 2(y-y')$,
- $\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial h_1}$,
- $\frac{\partial L}{\partial h_2} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial h_2}$,
- $\frac{\partial L}{\partial u_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial u_1}$,
- $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_1}$,
- $\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_2}$,
- $\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_3}$,
- $\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial w_4}$
- \( \mathbf{x} \in \mathbb{R}^D \)

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

\(^3\)This is contrived. We usually use context.
Input layer = Embedding layer

- Input is a word $x \in \mathbb{R}^D$ for all $x \in \mathcal{V}$
- 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; ...

work
Input layer = Embedding layer

- **Static embedding layer**
  - Fixed word embeddings; not updated during training
  - Examples: SVD; \texttt{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 $\mathcal{C} = \{X_1, \ldots, X_{|\mathcal{C}|}\}$
- With sentences $\mathcal{X} = x_1, \ldots, x_{|\mathcal{X}|}$
- Vocab $\mathcal{V} = \{x_i | x_i \in \mathcal{X} \text{ and } X \in \mathcal{C}\}$
- Goal: learn vector/embedding $x_i$ for all $x_i \in \mathcal{V}$

- word2vec (Mikolov et al. (2013))
  - Define two embeddings per word: $x_i$ and $x'_i$
  - $x_i$ represents word as focus; $x'_i$ as context
  - word2vec optimizes (SkipGram model):
    
    $$
    \sum_{i} \sum_{j} \sum_{-c \leq k \leq c, k \neq 0} \log p(x_{j+k} | x_j) = \sum_{i} \sum_{j} \sum_{-c \leq k \leq c, k \neq 0} \log \frac{e^{x_j \cdot x'_{j+k}}}{\sum_{x_l \in \mathcal{V}} e^{x_j \cdot x'_l}}
    $$

    Maximize the probability word embedding can predict neighbours in some context window (of size $c$)
Example Static Embedding Layer: Word2Vec

\[
\sum_{i} \sum_{j} \sum_{-c \leq k \leq c, k \neq 0} \log p(x_{j+k} | x_j) = \sum_{i} \sum_{j} \sum_{-c \leq k \leq c, k \neq 0} \log \frac{e^{x_j \cdot x'_{j+k}}}{\sum_{x_l \in V} e^{x_j \cdot x'_l}}
\]

Source Text

brown

The  c=-2
Quick  c=-1
Fox  c=+1
Jumps  c=+2

Training Samples

{the, quick, brown, fox, jumps, over, the, lazy, dog.} →
{the, quick, brown} →
{quick, the} →
{quick, brown} →
{brown, the} →
{brown, quick} →
{brown, fox} →
{brown, jumps} →
{fox, quick} →
{fox, brown} →
{fox, jumps} →
{fox, over} →

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

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

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

- Solution: negative sampling

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

- \( \mathcal{V}_s \) is randomly sampled, i.e., \( \mathcal{V}_s \subset \mathcal{V} \) and \( |\mathcal{V}_s| << |\mathcal{V}| \) (often 1)
Example Static Embedding Layer: Word2Vec

\[
\begin{align*}
\left( \sum_{i} \sum_{j} \log e^{x_j \cdot x_{j+k}} \right) & - \left( \sum_{i} \sum_{j} \sum_{-c \leq k \leq c, k \neq 0} \log \sum_{l \in V_s} e^{x_j \cdot x_l} \right) \\
\end{align*}
\]

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

Fun word arithmetic artifact:

\[
x_{\text{Greece}} - (x_{\text{Canada}} - x_{\text{Ottawa}}) = x_{\text{Athens}}
\]
• 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
• Static (e.g., word2vec) or dynamic word embeddings give us input layer.
Dynamic Input layer

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

$\frac{\partial L}{\partial u_1}$, $\frac{\partial L}{\partial u_2}$, $\frac{\partial L}{\partial w_1}$, $\frac{\partial L}{\partial w_2}$, $\frac{\partial L}{\partial w_3}$, $\frac{\partial L}{\partial w_4}$

$L(y, y') = (y-y')^2$
• But what if input is a whole document and not just a single word?
• Feed-forward neural networks assume a fixed-length input, $\mathbf{x} \in \mathbb{R}^D$
• Documents are not fixed length
• Truncate document at fixed length $K$, $x \in \mathbb{R}^{K \times D}$
• Average embeddings (below), $x \in \mathbb{R}^D$
• Even better: convolutional and recurrent neural networks (lecture 3)
Convolutional Neural Networks

The steak was cooked to perfection.

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

<sos>
The steak was cooked to perfection <eos>

Convolutional Block

Convolutional Layer

Pooling Layer

MLP

Conv. Block

Conv. Block

Conv. Block

MLP

POSITIVE

NEGATIVE

<sos>
The steak was cooked to perfection <eos>

Conv. Block

Conv. Block

Conv. Block

MLP

POSITIVE

NEGATIVE
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?

- Hyperparameters: 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
The steak was cooked to perfection.
Main Points in Words

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


