Statistical Methods for Computational Linguistics

A Basic Introduction to Machine Learning

Michael Staniek
Folien von Stefan Riezler

Computational Linguistics
Heidelberg University, Germany
staniek@cl.uni-heidelberg.de
Modeling the Frog’s Perceptual System
Modeling the Frog’s Perceptual System

- [Lettvin et al. 1959] show that the frog’s perceptual system constructs reality by four separate operations:
  - contrast detection: presence of sharp boundary?
  - convexity detection: how curved and how big is object?
  - movement detection: is object moving?
  - dimming speed: how fast does object obstruct light?

- The frog’s goal: Capture any object of the size of an insect or worm providing it moves like one.

- Can we build a model of this perceptual system and learn to capture the right objects?
Learning from Data

Assume **training data** of edible (+) and inedible (-) objects

<table>
<thead>
<tr>
<th>convex</th>
<th>speed</th>
<th>label</th>
<th>convex</th>
<th>speed</th>
<th>label</th>
</tr>
</thead>
<tbody>
<tr>
<td>small</td>
<td>small</td>
<td>-</td>
<td>small</td>
<td>large</td>
<td>+</td>
</tr>
<tr>
<td>small</td>
<td>medium</td>
<td>-</td>
<td>medium</td>
<td>large</td>
<td>+</td>
</tr>
<tr>
<td>small</td>
<td>medium</td>
<td>-</td>
<td>medium</td>
<td>large</td>
<td>+</td>
</tr>
<tr>
<td>medium</td>
<td>small</td>
<td>-</td>
<td>large</td>
<td>small</td>
<td>+</td>
</tr>
<tr>
<td>large</td>
<td>small</td>
<td>-</td>
<td>large</td>
<td>large</td>
<td>+</td>
</tr>
<tr>
<td>small</td>
<td>small</td>
<td>-</td>
<td>large</td>
<td>medium</td>
<td>+</td>
</tr>
<tr>
<td>small</td>
<td>large</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>small</td>
<td>medium</td>
<td>-</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Learning model parameters from data:
- \( p(+) = 6/14 \), \( p(-) = 8/14 \)
- \( p(\text{convex} = \text{small} | -) = 6/8 \), \( p(\text{convex} = \text{med} | -) = 1/8 \), \( p(\text{convex} = \text{large} | -) = 1/8 \)
- \( p(\text{speed} = \text{small} | -) = 4/8 \), \( p(\text{speed} = \text{med} | -) = 3/8 \), \( p(\text{speed} = \text{large} | -) = 1/8 \)
- \( p(\text{convex} = \text{small} | +) = 1/6 \), \( p(\text{convex} = \text{med} | +) = 2/6 \), \( p(\text{convex} = \text{large} | +) = 3/6 \)
- \( p(\text{speed} = \text{small} | +) = 1/6 \), \( p(\text{speed} = \text{med} | +) = 1/6 \), \( p(\text{speed} = \text{large} | +) = 4/6 \)

Predict unseen \( p(\text{label} = ?, \text{convex} = \text{med}, \text{speed} = \text{med}) \)
- \( p(-) \cdot p(\text{convex} = \text{med} | -) \cdot p(\text{speed} = \text{med} | -) = 8/14 \cdot 1/8 \cdot 3/8 = 0.027 \)
- \( p(+) \cdot p(\text{convex} = \text{med} | +) \cdot p(\text{speed} = \text{med} | +) = 6/14 \cdot 2/6 \cdot 1/6 = 0.024 \)
- **Inedible**: \( p(\text{convex} = \text{med}, \text{speed} = \text{med}, \text{label} = -) > p(\text{convex} = \text{med}, \text{speed} = \text{med}, \text{label} = +) ! \)
Machine Learning is a Frog’s World

- Machine learning problems can be seen as problems of function estimation where
  - our models are based on a combined feature representation of inputs and outputs
    - similar to the frog whose world is constructed by four-dimensional feature vector based on detection operations
  - learning of parameter weights is done by optimizing fit of model to training data
    - frog uses binary classification into edible/inedible objects as supervision signals for learning
  - The model used in the frog’s perception example is called Naive Bayes: It measures compatibility of inputs to outputs by a linear model and optimizes parameters by convex optimization
Lecture Outline

- Preliminaries
  - Data: input/output
  - Feature representations
  - Linear models

- Convex optimization for linear models
  - Naive Bayes
  - Logistic Regression
  - Perceptron
  - Large-Margin Learners (SVMs)

- Regularization

- Online learning

- Non-linear models
  - Kernel machines: Convex optimization for non-linear models
  - Neural networks: Nonconvex optimization for non-linear models
Inputs and Outputs

- **Input:** \( x \in \mathcal{X} \)
  - e.g., document or sentence with some words \( x = w_1 \ldots w_n \)
- **Output:** \( y \in \mathcal{Y} \)
  - e.g., document class, translation, parse tree
- **Input/Output pair:** \( (x, y) \in \mathcal{X} \times \mathcal{Y} \)
  - e.g., a document \( x \) and its class label \( y \),
  - a source sentence \( x \) and its translation \( y \),
  - a sentence \( x \) and its parse tree \( y \)
Feature Representations

Most NLP problems can be cast as multiclass classification where we assume a high-dimensional joint feature map on input-output pairs \((x, y)\)

\[ \phi(x, y) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m \]

Common ranges:
- categorical (e.g., counts): \(\phi_i \in \{1, \ldots, F_i\}, F_i \in \mathbb{N}^+\)
- binary (e.g., binning): \(\phi \in \{0,1\}^m\)
- continuous (e.g., word embeddings): \(\phi \in \mathbb{R}^m\)

For any vector \(v \in \mathbb{R}^m\), let \(v_j\) be the \(j^{th}\) value
Example: Text Classification

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

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

We expect this feature to have a positive weight, “interest” is a positive indicator for the label “financial”
Example: Text Classification

\[ \phi_j(x, y) = \% \text{ of words in } x \text{ containing punctuation and } y = \text{“scientific”} \]

Q&A: Punctuation symbols - positive indicator or negative indicator for scientific articles?
Example: Part-of-Speech Tagging

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

$$\phi_j(x, y) = \begin{cases} 
1 & \text{if } x = \text{"bank"} \text{ and } y = \text{Verb} \\
0 & \text{otherwise}
\end{cases}$$

Q&A: What weight would it get?
Example: Named-Entity Recognition

- $x$ is a name, $y$ is a label classifying the name

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

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

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

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

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

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

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

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

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

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

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

► $x=$George Washington Bridge, $y=$Object $\rightarrow \phi(x, y) = [0\ 0\ 0\ 0\ 1\ 1\ 1\ 0]$

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

► Each equal size block of the feature vector corresponds to one label

► Non-zero values allowed only in one block
Example: Statistical Machine Translation

- $x$ is a source sentence and $y$ is translation

$$
\phi_j(x, y) = \begin{cases} 
1 & \text{if "y a-t-il" present in } x \\
& \text{and "are there" present in } y \\
0 & \text{otherwise}
\end{cases}
$$

$$
\phi_k(x, y) = \begin{cases} 
1 & \text{if "y a-t-il" present in } x \\
& \text{and "are there any" present in } y \\
0 & \text{otherwise}
\end{cases}
$$

Q&A: Which phrase indicator should be preferred?
Example: Parsing

Statistical Methods for CL 15(161)

Note: Label $y$ includes sentence $x$
Linear Models

- **Linear model**: Defines a discriminant function that is based on a linear combination of features and weights

\[
f(x; \omega) = \arg \max_{y \in \mathcal{Y}} \omega \cdot \phi(x, y) = \arg \max_{y \in \mathcal{Y}} \sum_{j=0}^{m} \omega_j \times \phi_j(x, y)
\]

- Let \( \omega \in \mathbb{R}^m \) be a high dimensional weight vector
- Assume that \( \omega \) is known
  - **Multiclass Classification**: \( \mathcal{Y} = \{0, 1, \ldots, N\} \)
    \[
y = \arg \max_{y' \in \mathcal{Y}} \omega \cdot \phi(x, y')
    \]
  - **Binary Classification** just a special case of multiclass
Linear Models for Binary Classification

- $\omega$ defines a linear decision boundary that divides space of instances in two classes
  - 2 dimensions: line
  - 3 dimensions: plane
  - $n$ dimensions: hyperplane of $n - 1$ dimensions

Points along line have scores of 0
Multiclass Linear Model

Defines regions of space. Visualization difficult.

+ are all points \((x, y)\) where 
\[ + = \arg \max_y \omega \cdot \phi(x, y) \]
Convex Optimization for Supervised Learning

How to learn weight vector $\omega$ in order to make decisions?

- **Input:**
  - i.i.d. (independent and identically distributed) training examples $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\mathcal{T}}$
  - feature representation $\phi$

- **Output:** $\omega$ that maximizes an **objective function** on the training set

  $\omega = \arg\max L(\mathcal{T}; \omega)$

  Equivalently minimize: $\omega = \arg\min -L(\mathcal{T}; \omega)$
Objective Functions

- Ideally we can decompose $\mathcal{L}$ by training pairs $(x, y)$
  - $\mathcal{L}(\mathcal{T}; \omega) \propto \sum_{(x, y) \in \mathcal{T}} \text{loss}((x, y); \omega)$
  - $\text{loss}$ is a function that measures some value correlated with errors of parameters $\omega$ on instance $(x, y)$

- Example:
  - $y \in \{1, -1\}$, $f(x; \omega)$ is the prediction we make for $x$ using $\omega$
  - Zero-one loss function:
    $$\text{loss}((x, y); \omega) = \begin{cases} 1 & \text{if } f(x; \omega) \times y \leq 0 \\ 0 & \text{else} \end{cases}$$
Convexity

A function is convex if its graph lies on or below the line segment connecting any two points on the graph

\[ f(\alpha x + \beta y) \leq \alpha f(x) + \beta f(y) \text{ for all } \alpha, \beta \geq 0, \alpha + \beta = 1 \]

Q&A: Is the zero-one loss function convex?
Gradient

- Gradient of function $f$ is vector of partial derivatives.
  \[
  \nabla f(x) = \left( \frac{\partial}{\partial x_1} f(x), \frac{\partial}{\partial x_2} f(x), ..., \frac{\partial}{\partial x_n} f(x) \right)
  \]

- Rate of increase of $f$ at point $x$ in each of the axis-parallel directions.

Q&A: What is the gradient at $x$ for the function in the image above?
Convex Optimization

- Objectives for linear models can be defined as **convex upper bounds on zero-one loss**
Unconstrained Optimization

- Unconstrained optimization tries to find a point that minimizes our objective function.
- In order to find minimum, follow opposite direction of gradient.
- Global minimum lies at point where $\nabla f(x) = 0$.

Q&A: How can maximization be defined as minimization problem?
Constrained Optimization with Equality Constraints

- Optimization problem is finding a point among the feasible points that satisfy constraints $g_i(x) = 0$ where $f(x)$ is minimal.
- Example: For 3-dimensional domain of $f(x)$, feasible points constitute intersection of surfaces $g_1(x) = 0$ and $g_2(x) = 0$. 
Equality Constraints

Gradients $\nabla g_1(x)$, $\nabla g_2(x)$ define a normal plane to feasible set curve $C$: $\alpha_1 \nabla g_1(x) + \alpha_2 \nabla g_2(x)$, generally $\sum_{i} \alpha_i \nabla g_i(x)$

Goal: move along $C$ looking for point that minimizes $f$
Equality Constraints

\[ \nabla f(x) \text{ is a sum of vector } a \text{ (tangent to } C, \text{ pointing in direction of increase of } f \text{) and vector } b \text{ (lying in normal plane to } C) \]

- To minimize \( f \), move in opposite direction of \( a \)
- Minimum reached when there is no direction of further decrease
Lagrange Multipliers

At minimum, gradient of $f$ lies entirely in plane perpendicular to feasible set curve $C$: $\nabla f(x) = \sum_i \alpha_i \nabla g_i(x)$

Solving for $x$ solves constrained optimization problem.

Define Lagrangian $L(x) = f(x) - \sum_i \alpha_i g_i(x)$ where equality constraints have standard form $g_i = 0, \forall i$.

Setting $\nabla L(x) = 0$ and solving for $x$ gives same solution as for constrained problem, but by unconstrained optimization.
Inequality Constraints

For 3-dimensional domain of $f(x)$, inequality constraints $g_1(x) \leq 0, g_2(x) \leq 0$ describe convex solids.

Feasible set is intersection, a lentil shaped solid.

Goal: Minimize $f$ while remaining within feasible set.
Inequality Constraints

- Three cases, all reducable to equality constraints
  - Global minimum $a$ within feasible set, constraints satisfied
  - Global minimum $b$ closer to surface of binding constraint $g_1$; solve $\nabla f(x) = \alpha_1 \nabla g_1(x)$; ignore slack constraint $g_2$ by $\alpha_2 = 0$
  - Global minimum $c$ near edge where $g_1(x) = 0$ and $g_2(x) = 0$

- Kuhn-Tucker conditions: Either $g_i(x) = 0$ (binding) or $\alpha_i = 0$ (slack): $\alpha_i g_i(x) = 0, \forall i$
Naive Bayes
Naive Bayes

- Probabilistic decision model:

\[
\arg\max_y P(y|x) \propto \arg\max_y P(y)P(x|y)
\]

- Uses Bayes Rule:

\[
P(y|x) = \frac{P(y)P(x|y)}{P(x)} \text{ for fixed } x
\]

- Generative model since \(P(y)P(x|y) = P(x, y)\) is a joint probability

  - Because we model a distribution that can randomly generate outputs \textit{and} inputs, not just outputs
Naivety of Naive Bayes

- We need to decide on the structure of $P(x, y)$
- $P(x|y) = P(\phi(x)|y) = P(\phi_1(x), \ldots, \phi_m(x)|y)$

Naive Bayes Assumption
*(conditional independence)*

$$P(\phi_1(x), \ldots, \phi_m(x)|y) = \prod_i P(\phi_i(x)|y)$$

- $P(x, y) = P(y) \prod_{i=1}^m P(\phi_i(x)|y)$

Q&A: How would $P(x, y)$ be defined without independence?
Naive Bayes – Learning

- Input: \( \mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\mathcal{T}} \)
- Let \( \phi_i(x) \in \{1, \ldots, F_i\} \)
- Parameters \( \mathcal{P} = \{P(y), P(\phi_i(x)|y)\} \)
Maximum Likelihood Estimation

- What’s left? Defining an objective $\mathcal{L}(\mathcal{T})$.

- $\mathcal{P}$ plays the role of $\omega$.

- What objective to use?

- **Objective:** Maximum Likelihood Estimation (MLE)

\[
\mathcal{L}(\mathcal{T}) = \prod_{t=1}^{\mid\mathcal{T}\mid} P(x_t, y_t) = \prod_{t=1}^{\mid\mathcal{T}\mid} \left( P(y_t) \prod_{i=1}^{m} P(\phi_i(x_t) | y_t) \right)
\]
Naive Bayes – Learning

MLE has closed form solution

\[ P = \arg \max_{\mathcal{P}} \prod_{t=1}^{|\mathcal{T}|} \left( P(y_t) \prod_{i=1}^{m} P(\phi_i(x_t)|y_t) \right) \]

\[ P(y) = \frac{\sum_{t=1}^{|\mathcal{T}|} [y_t = y]}{|\mathcal{T}|} \]

\[ P(\phi_i(x)|y) = \frac{\sum_{t=1}^{|\mathcal{T}|} [\phi_i(x_t) = \phi_i(x) \text{ and } y_t = y]}{\sum_{t=1}^{|\mathcal{T}|} [y_t = y]} \]

where \([p] = \begin{cases} 1 & \text{if } p \text{ is true,} \\ 0 & \text{otherwise.} \end{cases}\]

Thus, these are just normalized counts over events in \(\mathcal{T}\)
Deriving MLE

\[
P = \arg \max_{\mathcal{P}} \prod_{t=1}^{|\mathcal{T}|} \left( P(y_t) \prod_{i=1}^{m} P(\phi_i(x_t)|y_t) \right)
\]

\[
= \arg \max_{\mathcal{P}} \sum_{t=1}^{|\mathcal{T}|} \left( \log P(y_t) + \sum_{i=1}^{m} \log P(\phi_i(x_t)|y_t) \right)
\]

\[
= \arg \max_{P(y)} \sum_{t=1}^{|\mathcal{T}|} \log P(y_t) + \arg \max_{P(\phi_i(x)|y)} \sum_{t=1}^{|\mathcal{T}|} \sum_{i=1}^{m} \log P(\phi_i(x_t)|y_t)
\]

such that \( \sum_{y} P(y) = 1, \sum_{j=1}^{F_i} P(\phi_i(x) = j|y) = 1, P(\cdot) \geq 0 \)
Deriving MLE

\[
P = \arg \max_{P(y)} \left( \sum_{t=1}^{\lvert T \rvert} \log P(y_t) \right) + \arg \max_{P(\phi_i(x)|y)} \left( \sum_{t=1}^{\lvert T \rvert} \sum_{i=1}^{m} \log P(\phi_i(x_t)|y_t) \right)
\]

Both optimizations are of the form

\[
\arg \max_{P} \sum_{v} \text{count}(v) \log P(v), \text{ s.t. } \sum_{v} P(v) = 1, P(v) \geq 0
\]

where \( v \) is event in \( T \), either \((y_t = y)\) or \((\phi_i(x_t) = \phi_i(x), y_t = y)\)

Q&A: How can this problem be classified in terms of optimization theory?
Deriving MLE

\[
\arg \max_P \sum_v \text{count}(v) \log P(v) \\
\text{s.t., } \sum_v P(v) = 1, \ P(v) \geq 0
\]

Introduce **Lagrangian** multiplier \(\lambda\), optimization becomes

\[
\arg \max_{P,\lambda} \sum_v \text{count}(v) \log P(v) - \lambda \left( \sum_v P(v) - 1 \right)
\]

- Derivative w.r.t \(P(v)\) is \(\frac{\text{count}(v)}{P(v)} - \lambda\)

- Setting this to zero \(P(v) = \frac{\text{count}(v)}{\lambda}\)

- Use \(\sum_v P(v) = 1, \ P(v) \geq 0\), then \(P(v) = \frac{\text{count}(v)}{\sum_{v'} \text{count}(v')}\)
Deriving MLE

Reinstantiate events \( \nu \) in \( \mathcal{T} \):

\[
P(y) = \frac{\sum_{t=1}^{|\mathcal{T}|} [y_t = y]}{|\mathcal{T}|}
\]

\[
P(\phi_i(x)|y) = \frac{\sum_{t=1}^{|\mathcal{T}|} [\phi_i(x_t) = \phi_i(x) \text{ and } y_t = y]}{\sum_{t=1}^{|\mathcal{T}|} [y_t = y]}
\]
Naive Bayes is a linear model

- Let \( \omega_y = \log P(y), \forall y \in \mathcal{Y} \)
- Let \( \omega_{\phi_i(x), y} = \log P(\phi_i(x)|y), \forall y \in \mathcal{Y}, \phi_i(x) \in \{1, \ldots, F_i\} \)

\[
\arg \max_y P(y|\phi(x)) \propto \arg \max_y P(\phi(x), y) = \arg \max_y P(y) \prod_{i=1}^m P(\phi_i(x)|y)
\]

\[
= \arg \max_y \log P(y) + \sum_{i=1}^m \log P(\phi_i(x)|y)
\]

\[
= \arg \max_y \omega_y + \sum_{i=1}^m \omega_{\phi_i(x), y}
\]

\[
= \arg \max_y \sum_{y'} \omega_y \psi_{y'}(y) + \sum_{i=1}^m \sum_{j=1}^{F_i} \omega_{\phi_i(x), y} \psi_{i,j}(x)
\]

where \( \psi_{i,j}(x) = [\phi_i(x) = j] \), \( \psi_{y'}(y) = [y = y'] \)
Smoothing

- doc 1: \( y_1 = \) sports, “hockey is fast”
- doc 2: \( y_2 = \) politics, “politicians talk fast”
- doc 3: \( y_3 = \) politics, “washington is sleazy”

- New doc: “washington hockey is fast”
- Q&A: What are probabilities of classes ‘sports’ or ‘politics for “washington hockey is fast”?

- Smoothing aims to assign a small amount of probability to unseen events
- E.g., Additive/Laplacian smoothing

\[
P(v) = \frac{\text{count}(v)}{\sum_{v'} \text{count}(v')} \implies P(v) = \frac{\text{count}(v) + \alpha}{\sum_{v'} (\text{count}(v') + \alpha)}
\]
Discriminative versus Generative Models

- Generative models attempt to model inputs and outputs
  - e.g., Naive Bayes = MLE of joint distribution $P(x, y)$
  - Statistical model must explain generation of input

- Occam’s Razor: “Among competing hypotheses, the one with the fewest assumptions should be selected”

- Discriminative models
  - Use $\mathcal{L}$ that directly optimizes $P(y|x)$ (or something related)
  - Logistic Regression – MLE of $P(y|x)$
  - Perceptron and SVMs – minimize classification error

- Generative and discriminative models use $P(y|x)$ for prediction; differ only on what distribution they use to set $\omega$
Logistic Regression
Logistic Regression

Define a conditional probability:

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

Note: still a linear model

\[
\arg \max_y P(y|x) = \arg \max_y \frac{e^{\omega \cdot \phi(x, y)}}{Z_x} = \arg \max_y e^{\omega \cdot \phi(x, y)} = \arg \max_y \omega \cdot \phi(x, y)
\]
Logistic Regression

$$P(y|x) = \frac{e^{\omega \cdot \phi(x,y)}}{Z_x}$$

- Q: How do we learn weights $\omega$?
- A: Set weights to maximize log-likelihood of training data:

$$\omega = \arg \max_{\omega} \mathcal{L}(\mathcal{T}; \omega)$$

$$= \arg \max_{\omega} \prod_{t=1}^{\mathcal{T}} P(y_t|x_t) = \arg \max_{\omega} \sum_{t=1}^{\mathcal{T}} \log P(y_t|x_t)$$

- In a nutshell we set the weights $\omega$ so that we assign as much probability to the correct label $y$ for each $x$ in the training set.
Logistic Regression

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

\[ \omega = \arg \max_{\omega} \sum_{t=1}^{\mathcal{T}} \log P(y_t|x_t) \] (*)

- The objective function (*) is concave
- Therefore there is a global maximum
- No closed form solution, but lots of numerical techniques
  - Gradient methods ((stochastic) gradient ascent, conjugate gradient, iterative scaling)
  - Newton methods (limited-memory quasi-newton)
Gradient Ascent
Gradient Ascent

Let $\mathcal{L}(\mathcal{T}; \omega) = \sum_{t=1}^{|\mathcal{T}|} \log \left( \frac{e^{\omega \cdot \phi(x_t, y_t)}}{Z_x} \right)$

Want to find $\arg \max_{\omega} \mathcal{L}(\mathcal{T}; \omega)$

Set $\omega^0 = 0^m$

Iterate until convergence

$$\omega^i = \omega^{i-1} + \alpha \nabla \mathcal{L}(\mathcal{T}; \omega^{i-1})$$

$\alpha > 0$ is a step size / learning rate

$\nabla \mathcal{L}(\mathcal{T}; \omega)$ is gradient of $\mathcal{L}$ w.r.t. $\omega$

A gradient is all partial derivatives over variables $\omega_i$

i.e., $\nabla \mathcal{L}(\mathcal{T}; \omega) = \left( \frac{\partial}{\partial \omega_1} \mathcal{L}(\mathcal{T}; \omega), \frac{\partial}{\partial \omega_2} \mathcal{L}(\mathcal{T}; \omega), \ldots, \frac{\partial}{\partial \omega_m} \mathcal{L}(\mathcal{T}; \omega) \right)$

Gradient ascent will always find $\omega$ to maximize $\mathcal{L}$

Q&A: How do we turn this into a minimization problem?
Gradient Descent

Let $\mathcal{L}(T; \omega) = - \sum_{t=1}^{|T|} \log \left( \frac{e^{\omega \cdot \phi(x_t, y_t)}}{Z_x} \right)$

Want to find $\arg \min \omega \mathcal{L}(T; \omega)$

- Set $\omega^0 = O^m$
- Iterate until convergence

$$\omega^i = \omega^{i-1} - \alpha \nabla \mathcal{L}(T; \omega^{i-1})$$

- $\alpha > 0$ is step size / learning rate
- $\nabla \mathcal{L}(T; \omega)$ is gradient of $\mathcal{L}$ w.r.t. $\omega$
  - A gradient is all partial derivatives over variables $\omega_i$
  - i.e., $\nabla \mathcal{L}(T; \omega) = \left( \frac{\partial}{\partial \omega_1} \mathcal{L}(T; \omega), \frac{\partial}{\partial \omega_2} \mathcal{L}(T; \omega), \ldots, \frac{\partial}{\partial \omega_m} \mathcal{L}(T; \omega) \right)$

Gradient descent will always find $\omega$ to minimize $\mathcal{L}$
Deriving Gradient

- We apply gradient descent to minimize a convex functional
- Need to find the gradient = vector of partial derivatives
- Definition of conditional negative log-likelihood:

\[
\mathcal{L}(T; \omega) = - \sum_t \log P(y_t | x_t)
\]

\[
= - \sum_t \log \frac{e^{\omega \cdot \phi(x_t, y_t)}}{\sum_{y' \in Y} e^{\omega \cdot \phi(x_t, y')}}
\]

\[
= - \sum_t \log \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}}{Z_{x_t}}
\]
Deriving Gradient

\[
\frac{\partial}{\partial \omega_i} L(T; \omega) = \frac{\partial}{\partial \omega_i} - \sum_t \log \frac{e^{\sum_j \omega_j \times \phi_j(x_t, y_t)}}{Z_{x_t}}
\]

\[
= \sum_t \left( \frac{\partial}{\partial \omega_i} - \log e^{\sum_j \omega_j \times \phi_j(x_t, y_t)} + \frac{\partial}{\partial \omega_i} \log Z_{x_t} \right)
\]

\[
= \sum_t \left( -\phi_i(x_t, y_t) + \frac{\partial}{\partial \omega_i} \log Z_{x_t} \right)
\]
Logistic Regression

Deriving Gradient

\[
\frac{\partial}{\partial \omega_i} L(T; \omega) = \sum_t \left( -\phi_i(x_t, y_t) + \frac{\partial}{\partial \omega_i} \log Z_{x_t} \right)
\]

\[
= \sum_t \left( -\phi_i(x_t, y_t) + \sum_{y' \in Y} e^{\sum_j \omega_j \times \phi_j(x_t, y')} \right)
\]

\[
= \sum_t \left( -\phi_i(x_t, y_t) + \frac{\sum_{y' \in Y} e^{\sum_j \omega_j \times \phi_j(x_t, y')} \phi_i(x_t, y')}{\sum_{y' \in Y} e^{\sum_j \omega_j \times \phi_j(x_t, y')}} \right)
\]

\[
= \sum_t \left( -\phi_i(x_t, y_t) + \sum_{y' \in Y} P(y'|x_t) \phi_i(x_t, y') \right)
\]
FINALLY!!!

- After all that,

\[
\frac{\partial}{\partial \omega_i} \mathcal{L}(T; \omega) = - \sum_t \phi_i(x_t, y_t) + \sum_t \sum_{y' \in Y} P(y'|x_t) \phi_i(x_t, y')
\]

- And the gradient is:

\[
\nabla \mathcal{L}(T; \omega) = \left( \frac{\partial}{\partial \omega_0} \mathcal{L}(T; \omega), \frac{\partial}{\partial \omega_1} \mathcal{L}(T; \omega), \ldots, \frac{\partial}{\partial \omega_m} \mathcal{L}(T; \omega) \right)
\]

- So we can now use gradient descent to find $\omega$!!
Logistic Regression Summary

▶ Define conditional probability

\[ P(y|x) = \frac{e^{\omega \cdot \phi(x, y)}}{Z_x} \]

▶ Minimize conditional negative log-likelihood of training data

\[ \omega = \arg \min_{\omega} - \sum_{t} \log P(y_t|x_t) \]

▶ Calculate gradient and apply gradient descent optimization

\[ \frac{\partial}{\partial \omega_i} \mathcal{L}(\mathcal{T}; \omega) = -\sum_{t} \phi_i(x_t, y_t) + \sum_{t} \sum_{y' \in \mathcal{Y}} P(y'|x_t) \phi_i(x_t, y') \]
Logistic Regression = Maximum Entropy

- Maximum Entropy distribution \( P = \arg \max_P H(P) \) maximizes entropy \( H(P) \) over all \( P \) subject to constraints stating that
  - empirical feature counts must equal expected counts
- Quick intuition
  - Partial derivative in logistic regression
    \[
    \frac{\partial}{\partial \omega_i} L(T; \omega) = - \sum_t \phi_i(x_t, y_t) + \sum_t \sum_{y' \in \mathcal{Y}} P(y'|x_t) \phi_i(x_t, y')
    \]
    - First term is empirical feature counts and second term is expected counts
    - At optimum of logistic regression objective we have found the optimal parameter settings for a maximum entropy model

Q&A: How can uniform distribution be shown to maximize unconstrained entropy?
Perceptron
Perceptron Learning Algorithm

Training data: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\mathcal{T}}$

1. $\omega^{(0)} = 0; \ i = 0$
2. for $n : 1..N$
3. for $t : 1..T$
4. Let $y' = \arg \max_{y'} \omega^{(i)} \cdot \phi(x_t, y')$
5. if $y' \neq y_t$
6. $\omega^{(i+1)} = \omega^{(i)} + \phi(x_t, y_t) - \phi(x_t, y')$
7. $i = i + 1$
8. return $\omega^i$
Perceptron: Separability and Margin

- Given an training instance \((x_t, y_t)\), define:
  \[ \tilde{Y}_t = \mathcal{Y} - \{y_t\} \]
  i.e., \(\tilde{Y}_t\) is the set of incorrect labels for \(x_t\)

- A training set \(\mathcal{T}\) is separable with margin \(\gamma > 0\) if there exists a vector \(u\) with \(\|u\| = 1\) such that:
  \[ u \cdot \phi(x_t, y_t) - u \cdot \phi(x_t, y') \geq \gamma \]  
  for all \(y' \in \tilde{Y}_t\) and \(\|u\| = \sqrt{\sum_j u_j^2}\)

- **Assumption**: the training set is separable with margin \(\gamma\)

Q&A: Why do we require \(\|u\| = 1\)?
Perceptron Convergence Theorem

Theorem: For any training set separable with a margin of $\gamma$, the following holds for the perceptron algorithm:

\[
\text{mistakes made during training} \leq \frac{R^2}{\gamma^2}
\]

where $R \geq \|\phi(x_t, y_t) - \phi(x_t, y')\|$ for all $(x_t, y_t) \in T$ and $y' \in \bar{Y}_t$

Thus, after a finite number of training iterations, the error on the training set will converge to zero

Let’s prove it!
Perceptron Convergence Theorem

Training data: \( \mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\mathcal{T}} \)

1. \( \omega^{(0)} = 0; \ i = 0 \)
2. for \( n : 1..N \)
3. for \( t : 1..\mathcal{T} \)
4. Let \( y' = \text{arg max}_{y'} \omega^{(i)} \cdot \phi(x_t, y') \)
5. if \( y' \neq y_t \)
6. \( \omega^{(i+1)} = \omega^{(i)} + \phi(x_t, y_t) - \phi(x_t, y') \)
7. \( i = i + 1 \)
8. return \( \omega^{i} \)

\( u \cdot \omega^{(k)} = u \cdot \omega^{(k-1)} + u \cdot (\phi(x_t, y_t) - \phi(x_t, y')) \geq u \cdot \omega^{(k-1)} + \gamma, \) by (1)

Since \( \omega^{(0)} = 0 \) and \( u \cdot \omega^{(0)} = 0, \) for all \( k: \ u \cdot \omega^{(k)} \geq k\gamma, \) by induction on \( k \)

Since \( u \cdot \omega^{(k)} \leq ||u|| \times ||\omega^{(k)}||, \) by the Cauchy-Schwarz inequality, and \( ||u|| = 1, \) then \( ||\omega^{(k)}|| \geq k\gamma \)

Q&A: What does the Cauchy-Schwarz inequality state?

Upper bound:

\[
||\omega^{(k)}||^2 = ||\omega^{(k-1)}||^2 + ||\phi(x_t, y_t) - \phi(x_t, y')||^2 + 2\omega^{(k-1)} \cdot (\phi(x_t, y_t) - \phi(x_t, y'))
\]

\[
||\omega^{(k)}||^2 \leq ||\omega^{(k-1)}||^2 + R^2, \text{ since } R \geq ||\phi(x_t, y_t) - \phi(x_t, y')||
\]

and \( \omega^{(k-1)} \cdot \phi(x_t, y_t) - \omega^{(k-1)} \cdot \phi(x_t, y') \leq 0 \)

\[
\leq kR^2 \text{ for all } k, \text{ by induction on } k
\]
Perceptron Convergence Theorem

- We have just shown that $||\omega^{(k)}|| \geq k\gamma$ and $||\omega^{(k)}||^2 \leq kR^2$

- Therefore,

$$k^2\gamma^2 \leq ||\omega^{(k)}||^2 \leq kR^2$$

- and solving for $k$

$$k \leq \frac{R^2}{\gamma^2}$$

- Therefore the number of errors is bounded!
Perceptron Objective

- What is the objective function corresponding to the perceptron update if seen as gradient descent step?

**Perceptron loss:**

\[
\text{loss}((x_t, y_t); \omega) = (\max_{y \neq y_t} \omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t))_+
\]

where \((z)_+ = \max(0, z)\).

- **Stochastic (sub)gradient:**

\[
\nabla \text{loss} = \begin{cases} 
0 & \text{if } \omega \cdot \phi(x_t, y_t) - \max_y \omega \cdot \phi(x_t, y) \geq 0 \\
\phi(x_t, y) - \phi(x_t, y_t) & \text{else, where } y = \arg \max \omega \cdot \phi(x_t, y)
\end{cases}
\]
Averaged Perceptron Algorithm

Training data: $\mathcal{T} = \{(x_t, y_t)\}_{t=1}^{\mathcal{T}}$

1. $\omega^{(0)} = 0; \ i = 0$
2. for $n : 1..N$
3. \hspace{1em} for $t : 1..T$
4. \hspace{2em} Let $y' = \arg \max_{y'} \omega^{(i)} \cdot \phi(x_t, y')$
5. \hspace{2em} if $y' \neq y_t$
6. \hspace{3em} $\omega^{(i+1)} = \omega^{(i)} + \phi(x_t, y_t) - \phi(x_t, y')$
7. \hspace{2em} else
6. \hspace{3em} $\omega^{(i+1)} = \omega^{(i)}$
7. \hspace{1em} $i = i + 1$
8. return $(\sum_i \omega^{(i)}) / (N \times T)$
Perceptron Summary

- Learns parameters of a linear model by minimizing error
- Guaranteed to find a $\omega$ in a finite amount of time
- Perceptron is an example of an Online Learning Algorithm
  - $\omega$ is updated based on a single training instance, taking a step into the negative direction of the stochastic gradient:
    \[
    \omega^{(i+1)} = \omega^{(i)} + \phi(x_t, y_t) - \phi(x_t, y')
    \]
    where $y' = \arg\max_y \omega^{(i)} \cdot \phi(x_t, y')$
- More about online learning/stochastic gradient descent later!
Support Vector Machines (SVMs)
Support Vector Machines

**Margin**

**Training**

**Testing**

Denote the value of the margin by $\gamma$
Maximizing Margin

- For a training set $\mathcal{T}$
- Margin of a weight vector $\omega$ is smallest $\gamma$ such that
  \[
  \omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq \gamma
  \]
- for every training instance $(x_t, y_t) \in \mathcal{T}$, $y' \in \bar{\mathcal{Y}}_t$
Maximizing Margin

- Intuitively maximizing margin makes sense
- By cross-validation, the generalization error on unseen test data can be shown to be proportional to the inverse of the margin
  \[ \epsilon \propto \frac{R^2}{\gamma^2 \times |T|} \]
- **Perceptron**: we have shown that:
  - If a training set is separable by some margin, the perceptron will find a \( \omega \) that separates the data
  - However, the perceptron does not pick \( \omega \) to maximize the margin!
Maximizing Margin

Let $\gamma > 0$

$$\max_{\|\omega\| = 1} \gamma$$

such that:

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

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

and $y' \in \tilde{Y}_t$

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

$$\beta(\omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y')) \geq \beta \gamma, \text{ for some } \beta \geq 1$$
Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{\|\omega\| = 1} \gamma$$

such that:

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

$$(x_t, y_t) \in T$$

and $y' \in \bar{Y}_t$$
Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{||\omega||=1} \gamma$$

such that:

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

$$\forall (x_t, y_t) \in \mathcal{T}$$

and $y' \in \bar{Y}_t$

Change variables: $u = \frac{\omega}{\gamma}$

$||\omega|| = 1$ iff $||u|| = 1/\gamma$, then $\gamma = 1/||u||$
Max Margin = Min Norm

Let $\gamma > 0$

**Max Margin:**

$$\max_{||\omega||=1} \gamma$$

such that:

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

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

and $y' \in \bar{Y}_t$

Change variables: $u = \frac{\omega}{\gamma}$

$||\omega|| = 1$ iff $||u|| = 1/\gamma$,

then $\gamma = 1/||u||$

**Min Norm (step 1):**

$$\max_u \frac{1}{||u||}$$

such that:

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

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

and $y' \in \bar{Y}_t$
Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{||\omega||=1} \gamma$$

such that:

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

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

and $y' \in \tilde{Y_t}$

Change variables: $u = \frac{\omega}{\gamma}$

$||\omega|| = 1$ iff $||u|| = 1/\gamma$, then $\gamma = 1/||u||$

Min Norm (step 1):

$$\min_u ||u||$$

such that:

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

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

and $y' \in \tilde{Y_t}$
Max Margin $=\text{Min Norm}$

Let $\gamma > 0$

**Max Margin:**

$$\max_{||\omega||=1} \gamma$$

such that:

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

$$\forall (x_t, y_t) \in \mathcal{T}$$

and $y' \in \mathcal{Y}_t$

Change variables: $u = \frac{\omega}{\gamma}$

$||\omega|| = 1$ iff $||u|| = 1/\gamma$,

then $\gamma = 1/||u||$

**Min Norm (step 2):**

$$\min_u ||u||$$

such that:

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

$$\forall (x_t, y_t) \in \mathcal{T}$$

and $y' \in \mathcal{Y}_t$
Max Margin = Min Norm

Let $\gamma > 0$

**Max Margin:**

$$\max_{\|\omega\|=1} \gamma$$

such that:

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

$\forall (x_t, y_t) \in \mathcal{T}$ and $y' \in \bar{Y}_t$

Change variables: $u = \frac{\omega}{\gamma}$

$\|\omega\| = 1$ iff $\|u\| = 1/\gamma$, then $\gamma = 1/\|u\|$

**Min Norm (step 2):**

$$\min_{u} \|u\|$$

such that:

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

$\forall (x_t, y_t) \in \mathcal{T}$ and $y' \in \bar{Y}_t$
Max Margin = Min Norm

Let \( \gamma > 0 \)

**Max Margin:**

\[
\max_{||\omega||=1} \gamma
\]

such that:

\[
\omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq \gamma
\]

\( \forall (x_t, y_t) \in T \)

and \( y' \in \bar{Y}_t \)

Change variables: \( u = \frac{\omega}{\gamma} \)

\( ||\omega|| = 1 \) iff \( ||u|| = 1/\gamma \),

then \( \gamma = 1/||u|| \)

**Min Norm (step 3):**

\[
\min_u \frac{1}{2} ||u||^2
\]

such that:

\[
u \cdot \phi(x_t, y_t) - u \cdot \phi(x_t, y') \geq 1
\]

\( \forall (x_t, y_t) \in T \)

and \( y' \in \bar{Y}_t \)
Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

$$\max_{\|\omega\|=1} \gamma$$

such that:

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

$$\forall (x_t, y_t) \in T$$
and $y' \in \bar{Y}_t$

Min Norm:

$$\min_u \frac{1}{2} \|u\|^2$$

such that:

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

$$\forall (x_t, y_t) \in T$$
and $y' \in \bar{Y}_t$

▶ Intuition: Instead of fixing $\|\omega\|$ we fix the margin $\gamma = 1$
Support Vector Machines

- **Constrained Optimization Problem**

\[ \omega = \arg \min_{\omega} \frac{1}{2} \|\omega\|^2 \]

such that:

\[ \omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq 1 \]

\[ \forall (x_t, y_t) \in \mathcal{T} \text{ and } y' \in \bar{Y}_t \]

- **Support Vectors:** Examples where

\[ \omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') = 1 \]

for training instance \((x_t, y_t) \in \mathcal{T}\) and all \(y' \in \bar{Y}_t\)

Q&A: How can the Kuhn-Tucker conditions be used to explain the concept of support vectors?
Support Vector Machines

▶ What if data is not separable?

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

such that:

$$\omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y'_t) \geq 1 - \xi_t \text{ and } \xi_t \geq 0$$

$$\forall (x_t, y_t) \in T \text{ and } y'_t \in \bar{Y}_t$$

▶ $\xi_t$: slack variable representing amount of constraint violation
▶ If data is separable, optimal solution has $\xi_i = 0, \forall i$
▶ C balances focus on margin and on error

Q&A: Which ranges of C focus on margin vs. error?
Support Vector Machines

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

such that:

\[
\omega \cdot \phi(x_t, y_t) - \omega \cdot \phi(x_t, y') \geq 1 - \xi_t
\]

where \(\xi_t \geq 0\) and \(\forall (x_t, y_t) \in T\) and \(y' \in \tilde{Y}_t\)

- Computing the dual form results in a **quadratic programming problem** – a well-known convex optimization problem
- Can we have representation of this objective that allows more direct optimization?

\[
\omega \cdot \phi(x_t, y_t) - \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') \geq 1 - \xi_t
\]

\[
\xi_t \geq 1 + \max_{y' \neq y_t} \omega \cdot \phi(x, y') - \omega \cdot \phi(x, y_t)
\]
Support Vector Machines

\[ \xi_t \geq 1 + \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t) \]

- If \( \|\omega\| \) classifies \((x_t, y_t)\) with margin 1, penalty \( \xi_t = 0 \)
- Otherwise: \( \xi_t = 1 + \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t) \)
- That means that in the end \( \xi_t \) will be:

\[ \xi_t = \max\{0, 1 + \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t)\} \]
Support Vector Machines

\[ \omega = \arg \min_{\omega, \xi} \frac{\lambda}{2} \|\omega\|^2 + \sum_{t=1}^{\mathcal{T}} \xi_t \text{ s.t. } \xi_t \geq 1 + \max_{y' \neq y_t} \omega \cdot \phi(x_t, y') - \omega \cdot \phi(x_t, y_t) \]

Hinge loss

\[ \omega = \arg \min_{\omega} L(\mathcal{T}; \omega) = \arg \min_{\omega} \sum_{t=1}^{\mathcal{T}} \text{loss}((x_t, y_t); \omega) + \frac{\lambda}{2} \|\omega\|^2 \]

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

\[ \implies \text{Hinge loss allows } \textit{unconstrained optimization} \text{ (later!)} \]
Summary

What we have covered
- Linear Models
  - Naive Bayes
  - Logistic Regression
  - Perceptron
  - Support Vector Machines

What is next
- Regularization
- Online learning
- Non-linear models
Regularization
Fit of a Model

Two sources of error:

- **Bias** error, measures how well the hypothesis class fits the space we are trying to model
- **Variance** error, measures sensitivity to training set selection

Want to balance these two things
Fitting Training Data is not Sufficient

- Two functions fitting training data, but differing in predictions on test data
- Need to restrict class of functions to one that has capacity suitable for data in question
Overfitting

- Early in lecture we made assumption data was i.i.d.
  - Rarely is this true, e.g., syntactic analyzers typically trained on 40,000 sentences from early 1990s WSJ news text

- Even more common: \( \mathcal{T} \) is very small
  - This leads to overfitting

- E.g.: ‘fake’ is never a verb in WSJ treebank (only adjective)
  - High weight on “\( \phi(x, y) = 1 \) if \( x=\text{fake} \) and \( y=\text{adjective} \)”
  - Of course: leads to high log-likelihood / low error
  - Other features might be more indicative, e.g., adjacent word identities: ‘He wants to X his death’ \( \rightarrow X=\text{verb} \)
Regularization

- In practice, we regularize models to prevent overfitting
  \[ \arg \max_\omega \mathcal{L}(T; \omega) - \lambda R(\omega) \]

- Where \( R(\omega) \) is the regularization function
- \( \lambda \) controls how much to regularize
- Most common regularizer
  - L2: \( R(\omega) \propto \|\omega\|_2 = \|\omega\| = \sqrt{\sum_i \omega_i^2} \) – smaller weights desired
Logistic Regression with L2 Regularization

- Perhaps most common learner in NLP

\[
\mathcal{L}(\mathcal{T}; \omega) - \lambda \mathcal{R}(\omega) = \sum_{t=1}^{|\mathcal{T}|} \log \left( \frac{e^{\omega \cdot \phi(x_t, y_t)}}{Z_x} \right) - \frac{\lambda}{2} \|\omega\|^2
\]

- What are the new partial derivatives?

\[
\frac{\partial}{\partial w_i} \mathcal{L}(\mathcal{T}; \omega) - \frac{\partial}{\partial w_i} \lambda \mathcal{R}(\omega)
\]

- We know \( \frac{\partial}{\partial w_i} \mathcal{L}(\mathcal{T}; \omega) \)

- Just need \( \frac{\partial}{\partial w_i} \frac{\lambda}{2} \|\omega\|^2 = \frac{\partial}{\partial w_i} \frac{\lambda}{2} \left( \sqrt{\sum_i \omega_i^2} \right)^2 = \frac{\partial}{\partial w_i} \frac{\lambda}{2} \sum_i \omega_i^2 = \lambda \omega_i \)
Support Vector Machines

▶ SVM in hinge-loss formulation: L2 regularization corresponds to margin maximization!

\[
\omega = \arg \min_{\omega} \mathcal{L}(T; \omega) + \lambda R(\omega)
\]

\[
= \arg \min_{\omega} \sum_{t=1}^{|T|} \text{loss}((x_t, y_t); \omega) + \lambda R(\omega)
\]

\[
= \arg \min_{\omega} \sum_{t=1}^{|T|} \max (0, 1 + \max_{y \neq y_t} \omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t)) + \lambda R(\omega)
\]

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

\[ \omega = \arg \min_{\omega} \mathcal{L}(\mathcal{T}; \omega) + \lambda \mathcal{R}(\omega) \]

\[ = \arg \min_{\omega} \sum_{t=1}^{\mathcal{T}} \text{loss}(x_t, y_t; \omega) + \lambda \mathcal{R}(\omega) \]

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

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

Logistic Regression/log-loss: \(- \log \left( \frac{e^{\omega \cdot \phi(x_t, y_t)}}{Z_x} \right) \)

\[ \omega = \arg \min_{\omega} \sum_{t=1}^{\mathcal{T}} -\log \left( \frac{e^{\omega \cdot \phi(x_t, y_t)}}{Z_x} \right) + \frac{\lambda}{2} \| \omega \|^2 \]
Leave-one-out Generalization Bound for Margin

By cross-validation, the generalization error on unseen test data can be shown to be proportional to the inverse of the margin

\[ \epsilon \propto \frac{R^2}{\gamma^2 \times |T|} \]

Shown for the perceptron by [Freund and Schapire 1999]

True also for SVM which optimizes margin directly

Generalizes to regularization of weight norm by equivalence of margin maximization to L2 norm minimization
Leave-one-out Generalization Bound for Support Vectors

The generalization error on unseen test data can be shown to be upper bounded by the number of support vectors found by cross-validation on a training set of size $m$

$$\epsilon \leq \frac{\#SV}{m}$$

Shown by [Vapnik 1998]

Support vectors thus can be seen as regularization in example/dual space
**Summary: Loss Functions**

- **Zero-one loss**
  \[ L(y, f(x)) = \begin{cases} 0 & \text{if } y = f(x) \\ 1 & \text{otherwise} \end{cases} \]

- **Hinge loss**
  \[ L(y, f(x)) = \max(0, 1 - yf(x)) \]

- **Perceptron loss**
  \[ L(y, f(x)) = \begin{cases} 0 & \text{if } yf(x) > 0 \\ |yf(x)| & \text{otherwise} \end{cases} \]

- **Log loss**
  \[ L(y, f(x)) = -y \log(f(x)) - (1 - y) \log(1 - f(x)) \]

- **Squared hinge loss**
  \[ L(y, f(x)) = \max(0, 1 - yf(x))^2 \]

- **Modified huber loss**
  \[ L(y, f(x)) = \begin{cases} \frac{1}{2} (y - f(x))^2 & \text{if } |y - f(x)| < \delta \\ \delta (|y - f(x)| - \frac{1}{2} \delta) & \text{otherwise} \end{cases} \]
Online Learning
Online vs. Batch Learning

Batch(\(T\));
- for 1 \(\ldots\) \(N\)
  - \(\omega \leftarrow \text{update}(T; \omega)\)
- return \(\omega\)

E.g., SVMs, logistic regression, Naive Bayes

Online(\(T\));
- for 1 \(\ldots\) \(N\)
  - for \((x_t, y_t) \in T\)
    - \(\omega \leftarrow \text{update}((x_t, y_t); \omega)\)
  - end for
- end for
- return \(\omega\)

E.g., Perceptron
\[\omega = \omega + \phi(x_t, y_t) - \phi(x_t, y)\]
Batch Gradient Descent

- Let $L(T; \omega) = \sum_{t=1}^{|T|} \text{loss}((x_t, y_t); \omega)$
  - Set $\omega^0 = O^m$
  - Iterate until convergence

$$
\omega^i = \omega^{i-1} - \alpha \nabla L(T; \omega^{i-1})
$$

$$
= \omega^{i-1} - \frac{1}{|T|} \sum_{t=1}^{|T|} \alpha \nabla \text{loss}((x_t, y_t); \omega^{i-1})
$$

- $\alpha > 0$ is step size / learning rate
Stochastic Gradient Descent

- **Stochastic Gradient Descent (SGD)**
  - Approximate batch gradient $\nabla L(T; \omega)$ with stochastic gradient $\nabla \text{loss}((x_t, y_t); \omega)$

- Let $L(T; \omega) = \sum_{t=1}^{\lvert T \rvert} \text{loss}((x_t, y_t); \omega)$
  - Set $\omega^0 = O^m$
  - Iterate until convergence
    - Sample $(x_t, y_t) \in T$ \hspace{1cm} // “stochastic”
    - $\omega^i = \omega^{i-1} - \alpha \nabla \text{loss}((x_t, y_t); \omega^{i-1})$
  - Return $\omega$
Online Logistic Regression

- Stochastic Gradient Descent (SGD)
- \( \text{loss}((x_t, y_t); \omega) = \text{log-loss} \)
- \( \nabla \text{loss}((x_t, y_t); \omega) = \nabla \left( -\log \left( \frac{e^{\omega \cdot \phi(x_t, y_t)}}{Z_{x_t}} \right) \right) \)
- From logistic regression section:

\[
\nabla \left( -\log \left( \frac{e^{\omega \cdot \phi(x_t, y_t)}}{Z_{x_t}} \right) \right) = - \left( \phi(x_t, y_t) - \sum_y P(y|x) \phi(x_t, y) \right)
\]

- Plus regularization term (if part of model)
Online SVMs

- Stochastic Gradient Descent (SGD)

\[\text{loss}(x_t, y_t; \omega) = \text{hinge-loss}\]

\[\nabla \text{loss}(x_t, y_t; \omega) = \nabla \left( \max (0, 1 + \max_{y \neq y_t} \omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t)) \right)\]

- Subgradient is:

\[\nabla \left( \max (0, 1 + \max_{y \neq y_t} \omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t)) \right) = \begin{cases} 0, & \text{if } \omega \cdot \phi(x_t, y_t) - \max_y \omega \cdot \phi(x_t, y) \geq 1 \\ \phi(x_t, y) - \phi(x_t, y_t), & \text{otherwise, where } y = \arg \max_y \omega \cdot \phi(x_t, y) \end{cases}\]

- Plus regularization term (L2 norm for SVMs):

\[\nabla \frac{\lambda}{2} ||\omega||^2 = \lambda \omega\]
Perceptron and Hinge-Loss

SVM subgradient update looks like perceptron update

\[ \omega^i = \omega^{i-1} - \alpha \begin{cases} 
\lambda \omega, & \text{if } \omega \cdot \phi(x_t, y_t) - \max_y \omega \cdot \phi(x_t, y) \geq 1 \\
\phi(x_t, y) - \phi(x_t, y_t) + \lambda \omega, & \text{otherwise, where } y = \arg \max_y \omega \cdot \phi(x_t, y) 
\end{cases} \]

**Perceptron**

\[ \omega^i = \omega^{i-1} - \alpha \begin{cases} 
0, & \text{if } \omega \cdot \phi(x_t, y_t) - \max_y \omega \cdot \phi(x_t, y) \geq 0 \\
\phi(x_t, y) - \phi(x_t, y_t), & \text{otherwise, where } y = \arg \max_y \omega \cdot \phi(x_t, y) 
\end{cases} \]

Perceptron = SGD optimization of no-margin hinge-loss (without regularization):

\[ \max (0, 1 + \max_{y \neq y_t} \omega \cdot \phi(x_t, y) - \omega \cdot \phi(x_t, y_t)) \]
Online vs. Batch Learning

- **Online algorithms**
  - Each update step relies only on the derivative for a single randomly chosen example
    - Computational cost of one step is $1/T$ compared to batch
    - Easier to implement
  - Larger variance since each gradient is different
    - Variance slows down convergence
    - Requires fine-tuning of decaying learning rate

- **Batch algorithms**
  - Higher cost of averaging gradients over $T$ for each update
    - Implementation more complex
    - Less fine-tuning, e.g., allows constant learning rates
    - Faster convergence

Q&A: What would you choose in big data scenarios - online or batch?
Variance-Reduced Online Learning

- SGD update extended by velocity vector $\nu$ weighted by momentum coefficient $0 \leq \mu < 1$ [Polyak 1964]:
  
  $$\omega^{i+1} = \omega^i - \alpha \nabla \text{loss}(x_t, y_t; \omega^i) + \mu \nu^i$$

  where

  $$\nu^i = \omega^i - \omega^{i-1}$$

- Momentum accelerates learning if gradients are aligned along same direction, and restricts changes when successive gradient are opposite of each other
- General direction of gradient reinforced, perpendicular directions filtered out
- Best of both worlds: Efficient and effective!
Online-to-Batch Conversion

- Classical online learning:
  - data are given as an infinite sequence of input examples
  - model makes prediction on next example in sequence

- Standard NLP applications:
  - finite set of training data, prediction on new batch of test data
  - online learning applied by cycling over finite data
  - online-to-batch conversion: Which model to use at test time?
    - Last model? Random model? Best model on heldout set?
Online-to-Batch Conversion by Averaging

- **Averaged Perceptron**
  - $\bar{\omega} = \left( \sum_i \omega^{(i)} \right) / (N \times T)$
  - Use weight vector averaged over online updates for prediction

- How does the perceptron mistake bound carry over to batch?
  - Let $M_k$ be number of mistakes made during online learning, then with probability of at least $1 - \delta$:
    $$\mathbb{E}[\text{loss}(x, y; \bar{\omega})] \leq M_k + \sqrt{\frac{2}{k} \ln \frac{1}{\delta}}$$
    - = generalization bound based on online performance
    - [Cesa-Bianchi et al. 2004]
    - can be applied to all online learners with convex losses
Quick Summary
Linear Learners

- Naive Bayes, Perceptron, Logistic Regression and SVMs
- Linear models and convex objectives
- Gradient descent
- Regularization
- Online vs. batch learning
Non-Linear Models
Non-Linear Models

- Some data sets require more than a linear decision boundary to be correctly modeled.
- Decision boundary is no longer a hyperplane in the feature space.
Kernel Machines = Convex Optimization for Non-Linear Models

- Projecting a linear model into a higher dimensional feature space can correspond to a non-linear model and make non-separable problems separable.

- For classifiers based on similarity functions (= kernels), computing a non-linear kernel is often more efficient than calculating the corresponding dot product in the high dimensional feature space.

- Thus, kernels allow us to efficiently learn non-linear models by convex optimization.
Monomial Features and Polynomial Kernels

- Monomial features = \(d^{th}\) order products of entries \(x_j\) of \(\mathbf{x} \in \mathbb{R}^n\) s.t. \(x_{j_1} \times x_{j_2} \times \cdots \times x_{j_d}\) for \(j_1, \ldots, j_d \in \{1 \ldots n\}\)

- Ordered monomial feature map: \(\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^4\) s.t.
  \((x_1, x_2) \mapsto (x_1^2, x_2^2, x_1x_2, x_2x_1)\)

- Computation of kernel from feature map:

\[
\phi(\mathbf{x}) \cdot \phi(\mathbf{x}') = \sum_{i=1}^{4} \phi_i(\mathbf{x})\phi_i(\mathbf{x}') \quad \text{(Def. dot product)}
\]

\[
= x_1^2x_1' + x_2^2x_2' + x_1x_2x_1'x_2' + x_2x_1x_2'x_1' \quad \text{(Def. } \phi) \\
= x_1^2x_1' + x_2^2x_2' + 2x_1x_2x_1'x_2' \\
= (x_1x_1' + x_2x_2')^2
\]

- Direct application of kernel: \(\phi(\mathbf{x}) \cdot \phi(\mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}')^2\)
Let $C_d$ be a map from $x \in \mathbb{R}^m$ to vectors $C_d(x)$ of all $d^{th}$-degree ordered products of entries of $x$. Then the corresponding kernel computing the dot product of vectors mapped by $C_d$ is:

$$K(x, x') = C_d(x) \cdot C_d(x') = (x \cdot x')^d$$

Alternative feature map satisfying this definition = unordered monomial: $\phi_2 : \mathbb{R}^2 \to \mathbb{R}^3$ s.t. $(x_1, x_2) \mapsto (x_1^2, x_2^2, \sqrt{2}x_1x_2)$

Q&A: Suppose inputs $x$ being vectors of pixel intensities. How can monomial features help to distinguish handwritten 8 from 0 in image recognition?
Non-Linear Feature Map

\[ \phi_2 : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \text{ s.t. } (x_1, x_2) \mapsto (z_1, z_2, z_3) = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \]

- Linear hyperplane parallel to \( z_3 \), e.g., mapping \((1, 1) \mapsto (1, 1, 1.4), (1, -1) \mapsto (1, 1, -1.4), \ldots, (2, 2) \mapsto (4, 4, 5.7)\)
Kernel Definition

- A kernel is a similarity function between two points that is symmetric and positive definite, which we denote by:

  \[ K(x_t, x_r) \in \mathbb{R} \]

- Let \( M \) be a \( n \times n \) matrix such that ...

  \[ M_{t,r} = K(x_t, x_r) \]

- ... for any \( n \) points. Called the Gram matrix.

- Symmetric:

  \[ K(x_t, x_r) = K(x_r, x_t) \]

- Positive definite: positivity on diagonal

  \[ K(x, x) \geq 0 \text{ for all } x \text{ with equality only for } x = 0 \]
Mercer’s Theorem

- **Mercer’s Theorem**: for any kernel $K$, there exists a $\phi$ in some $\mathbb{R}^d$, such that:

  $$K(x_t, x_r) = \phi(x_t) \cdot \phi(x_r)$$

- This means that instead of mapping input data via non-linear feature map $\phi$ and then computing dot product, we can apply kernels directly *without even knowing about* $\phi$!
Kernel Trick

- Define a kernel, and do not explicitly use dot product between vectors, only kernel calculations.
- In some high-dimensional space, this corresponds to dot product.
- In that space, the decision boundary is linear, but in the original space, we now have a non-linear decision boundary.
- Note: Since our features are over pairs \((x, y)\), we will write kernels over pairs:

\[
K((x_t, y_t), (x_r, y_r)) = \phi(x_t, y_t) \cdot \phi(x_r, y_r)
\]

- Let’s do it for the Perceptron!
Kernel Trick – Perceptron Algorithm

Training data: \( T = \{(x_t, y_t)\}_{t=1}^{\mid T \mid} \)

1. \( \omega^{(0)} = 0; \ i = 0 \)
2. for \( n : 1..N \)
3. for \( t : 1..T \)
4. Let \( y = \arg \max_y \omega^{(i)} \cdot \phi(x_t, y) \)
5. if \( y \neq y_t \)
6. \( \omega^{(i+1)} = \omega^{(i)} + \phi(x_t, y_t) - \phi(x_t, y) \)
7. \( i = i + 1 \)
8. return \( \omega^i \)

▶ Each feature function \( \phi(x_t, y_t) \) is added and \( \phi(x_t, y) \) is subtracted to \( \omega \) say \( \alpha_{y,t} \) times

▶ \( \alpha_{y,t} \) is proportional to number of times label \( y \) is predicted for example \( t \) and caused an update because of misclassification

▶ Thus,

\[
\omega = \sum_{t,y} \alpha_{y,t} [\phi(x_t, y_t) - \phi(x_t, y)]
\]

Statistical Methods for CL 118(161)
Kernel Trick – Perceptron Algorithm

We can re-write the argmax function as:

\[ y^* = \arg \max_{y^*} \omega^{(i)} \cdot \phi(x, y^*) \]

\[ = \arg \max_{y^*} \sum_{t, y} \alpha_{y,t} [\phi(x_t, y_t) - \phi(x_t, y)] \cdot \phi(x, y^*) \]

\[ = \arg \max_{y^*} \sum_{t, y} \alpha_{y,t} [\phi(x_t, y_t) \cdot \phi(x, y^*) - \phi(x_t, y) \cdot \phi(x, y^*)] \]

\[ = \arg \max_{y^*} \sum_{t, y} \alpha_{y,t} [K((x_t, y_t), (x, y^*)) - K((x_t, y), (x, y^*))] \]

We can then re-write the perceptron algorithm strictly with kernels
Kernel Trick – Perceptron Algorithm

Training: \( \mathcal{T} = \{(x_t, y_t)\}_{t=1}^{|\mathcal{T}|} \)
1. \( \forall y, t \) set \( \alpha_{y,t} = 0 \)
2. for \( n : 1..N \)
3. for \( t : 1..T \)
4. Let \( y^* = \arg \max_{y^*} \sum_{t,y} \alpha_{y,t} [K((x_t, y_t), (x_t, y^*)) - K((x_t, y), (x_t, y^*))] \)
5. if \( y^* \neq y_t \)
6. \( \alpha_{y^*,t} = \alpha_{y^*,t} + 1 \)

Testing on unseen instance \( x \):

\[ y^* = \arg \max_{y^*} \sum_{t,y} \alpha_{y,t} [K((x_t, y_t), (x, y^*)) - K((x_t, y), (x, y^*))] \]

Intuition: \( y^* \) is label that is most similar to gold standard labels and least similar to non-gold labels.
Kernels Summary

- Can turn a linear model into a non-linear model
- Kernels project feature space to higher dimensions
  - Sometimes exponentially larger
  - Sometimes an infinite space!
- Can “kernelize” algorithms to make them non-linear
- Convex optimization methods still applicable to learn parameters
- Disadvantage: Exact kernel methods depend polynomially on the number of training examples - infeasible for large datasets
Kernels for Large Training Sets

- Alternative to exact kernels: Explicit randomized feature map
  [Rahimi and Recht 2007, Lu et al. 2016]
  - Shallow neural network by random Fourier/Binning transformation:
    - Random weights from input to hidden units
    - Cosine as transfer function
    - Convex optimization of weights from hidden to output units
Neural Networks: Nonconvex Optimization for Learning Nonlinear Feature Representations

Kernel Machines
- Kernel Machines introduce nonlinearity by using specific feature maps or kernels
- Feature map or kernel is not part of optimization problem, thus convex optimization of loss function for linear model possible

Neural Networks
- Similarities and nonlinear combinations of features are learned: representation learning
- We lose the advantages of convex optimization since objective functions will be nonconvex
Neural Networks

Perceptron as Single-Unit Neural Network

New notation:
- input vector: $x \in \mathbb{R}^{d_{in}}$
- weight matrix: $W \in \mathbb{R}^{d_{in} \times d_{out}}$
- linear model: $y = xW$

Example: $d_{in} = 5$, $d_{out} = 1$, $y = \sum_{i=1}^{5} x_i w_i$

Q&A: We are implicitly assuming that $x$ is a row vector. How would a perceptron look like if we assumed that $x$ is a column vector?
Multilayer Perceptron (MLP)

- Multilayer Perceptron for 1 hidden layer:
  \[ h = f(xW^{(1)}), \]
  \[ y = g(hW^{(2)}) \]

- Input vector: \( x \in \mathbb{R}^{d_{in}} \)
- Weights between input and hidden layer: \( W^{(1)} \in \mathbb{R}^{d_{in} \times d_1} \)
- Weights between hidden layer and output: \( W^{(2)} \in \mathbb{R}^{d_1 \times d_2} \)
- Non-linear functions \( f \) and \( g \), applied elementwise
Multilayer Perceptron (MLP)

Example:
- $d_{in} = 5$, $d_1 = 5$, $d_2 = d_{out} = 1$,
- $y_k = g(\sum_{j=1}^{5} h_j w_{kj}^{(2)})$,
- $h_j = f(\sum_{i=1}^{5} x_i w_{ji}^{(1)})$. 
Layering and Non-linear Activation Functions

- **Layering structure** feeds outputs of previous layers as input into following layers.

- Each hidden node performs **feature combination** and **feature selection** by turning input feature configuration on and off.

- **Non-linear activation** (threshold, transfer) function is important.
  - Without non-linear activation function models stays linear.

- Our example of a 1-hidden layer MLP is an **universal approximator** (of any measurable function) [Hornik et al. 1989]
  - $n$-layer MLP is composition of $n$ functions $h$.
  - Multiple layers are used in practice.
Non-linear Activation Functions

Logistic function
\[
\text{sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^{-x}}
\]
output ranges from 0 to +1
Non-linear Activation Functions

Hyperbolic tangent
\[
\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}
\]
output ranges from –1 to +1
Non-linear Activation Functions

Rectified Linear

\[ \text{relu}(x) = \max(0, x) \]

output ranges from 0 to \( \infty \)
Example: XOR

XOR problem:

Suppose two input features $x_1$ and $x_2$. Classes “true” and “false” fall into opposite quadrants of the decision space and cannot be separated linearly by a hyperplane.

$-1 \text{ XOR } -1 = false$

$-1 \text{ XOR } +1 = true$

$+1 \text{ XOR } -1 = true$

$+1 \text{ XOR } +1 = false$
Example: XOR

Bias nodes $x_2$ and $h_2$ with fixed value 1, set activation thresholds by their outgoing weights.

Computation of hidden node $h_0$ for input $x_0 = 1$, $x_1 = 0$:

$$h_0 = \sigma \left( \sum_i x_i w_{0i} \right)$$

$$= \sigma \left( 1 \times 3 + 0 \times 4 + 1 \times -2 \right)$$

$$= 0.73$$
Example: XOR

<table>
<thead>
<tr>
<th>Input $x_0$</th>
<th>Input $x_1$</th>
<th>Hidden $h_0$</th>
<th>Hidden $h_1$</th>
<th>Output $y_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0.12</td>
<td>0.02</td>
<td>0.18 → 0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0.88</td>
<td>0.27</td>
<td>0.74 → 1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0.73</td>
<td>0.12</td>
<td>0.74 → 1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0.99</td>
<td>0.73</td>
<td>0.33 → 0</td>
</tr>
</tbody>
</table>

- $h_0$ acts as OR node, $h_1$ acts as AND node
- XOR is subtraction of value of AND node from OR node

Q&A: Show that nonlinearity is crucial on the example input $(1, 1)$. Value of $h_1$ needs to be pushed up by sigmoid in order to push down final value below threshold 0.5.
Optimizing MLPs by Backpropagation

- **Backpropagation:**
  - Apply stochastic gradient descent to each training example
  - Start at input layer, **feed forward computation of total input** to output layer (thus alternative name feed-forward neural networks for MLPs)
  - Compute error at output layer, **propagate error back** to previous layers (thus ...)

Statistical Methods for CL 134(161)
Backpropagation

- Weight update at output nodes
  - Output node calculation: \( s_i = \sum_j w_{i \leftarrow j} h_j, \ y_i = \sigma(s_i) \)
  - Squared error compared to target \( t \): \( E = \sum_i \frac{1}{2}(t_i - y_i)^2 \)
  - Chain rule applied to gradient: \( \frac{dE}{dw_{i \leftarrow j}} = \frac{dE}{dy_i} \frac{dy_i}{ds_i} \frac{ds_i}{dw_{i \leftarrow j}} \)
    \[
    \frac{dE}{dy_i} = \frac{d}{dy_i} \frac{1}{2}(t_i - y_i)^2 = -(t_i - y_i) \\
    \frac{dy_i}{ds_i} = \frac{d}{ds_i} \sigma(s_i) = \sigma(s_i)(1 - \sigma(s_i)) = y_i(1 - y_i) := y'_i \\
    \frac{ds_i}{dw_{i \leftarrow j}} = \frac{d}{dw_{i \leftarrow j}} \sum_j w_{i \leftarrow j} h_j = h_j \\
    \text{Alltogether} \quad \frac{dE}{dw_{i \leftarrow j}} = \frac{dE}{dy_i} \frac{dy_i}{ds_i} \frac{ds_i}{dw_{i \leftarrow j}} = -(t_i - y_i) \ y'_i \ h_j
    \]

- Weight update: \( \Delta w_{i \leftarrow j} = \mu \ \delta_i \ h_j \),
  where \( \delta_i = (t_i - y_i) \ y'_i \) is an error term and \( \mu \) is a learning rate

Q&A: Show how to recover a single-unit binary perceptron.
Backpropagation

- Weight update at hidden nodes
  - Hidden node computation: \( z_j = \sum_k w_{j \leftarrow k} x_k \), \( h_j = \sigma(z_j) \)
  - Chain rule applied to gradient of squared error:
    \[
    \frac{dE}{dw_{j \leftarrow k}} = \frac{dE}{dh_j} \frac{dh_j}{dz_j} \frac{dz_j}{dw_{j \leftarrow k}}
    \]
  - Chain rule to track how error at output of hidden node contributes to error in next layer:
    \[
    \frac{dE}{dh_j} = \sum_i \frac{dE}{dy_i} \frac{dy_i}{ds_i} \frac{ds_i}{dh_j}
    \]
    \[
    \frac{dE}{dy_i} = -(t_i - y_i) \quad y'_i = \delta_i, \quad \frac{ds_i}{dh_j} = \frac{d}{dh_j} \sum_i w_{i \leftarrow j} h_j = w_{i \leftarrow j}
    \]
    \[
    \text{Alltogether: } \frac{dE}{dh_j} = \sum_i \delta_i w_{i \leftarrow j}
    \]
  - Alltogether:
    \[
    \frac{dh_j}{dz_j} = \frac{d\sigma(z_j)}{dz_j} = \sigma(z_j)(1 - \sigma(z_j)) = h_j(1 - h_j) = h'_j
    \]
    \[
    \frac{dz_j}{dw_{j \leftarrow k}} = \frac{d}{dw_{j \leftarrow k}} \sum_k w_{j \leftarrow k} x_k = x_k
    \]
    \[
    \text{Alltogether: } \frac{dE}{dw_{j \leftarrow k}} = \frac{dE}{dh_j} \frac{dh_j}{dz_j} \frac{dz_j}{dw_{j \leftarrow k}} = \sum_i (\delta_i w_{i \leftarrow j}) \quad h'_j \quad x_k
    \]
  - Weight update: \( \Delta w_{j \leftarrow k} = \mu \delta_j x_k \) where \( \delta_j = \sum_i (\delta_i w_{i \leftarrow j}) \quad h'_j \)
Backpropagation

- Error at output node compared to target: $\delta_i = (t_i - y_i) y_i'$
- Error at hidden nodes by backpropagating error term $\delta_i$ from subsequent nodes connected by weights $w_{i \leftarrow j}$:
  $\delta_j = \sum_i (\delta_i w_{i \leftarrow j}) h'_j$
- Similar weight updates:
  - $\Delta w_{i \leftarrow j} = \mu \delta_i h_j$,
  - $\Delta w_{j \leftarrow k} = \mu \delta_j x_k$
Refinements

- Task-dependent network architecture:
  - MLP for regression: $d_{out} = 1$
  - MLP for binary classification: $d_{out} = 2$
  - MLP for $k$-fold multiclass classification: $d_{out} = k$

- Task-dependent loss functions:
  - Squared error for regression, hinge loss for multiclass classification

- Optimization issues:
  - Known techniques such as SGD/momentum/regularization applicable
  - Special considerations regarding weight initialization/learning rates/gradient flow
Feed-Forward Neural Language Model

- Goal: Word-wise learning of probability of next word given context: \( p(w_i | w_{i-4}, w_{i-3}, w_{i-2}, w_{i-1}) \)
- Key idea: Learn a feature representation for each word as continuous vector in first layer of MLP simultaneously with optimizing language model probability
Word Embeddings

- Represent each word by setting its index $i$ to 1 in a vocabulary sized vector of 0s ($= 1$-hot vector $x_i$)
- Use shared weight matrix $C$ for all words
- Words occurring in similar contexts will get similar embeddings
Learning Word Embeddings

- Train weights of embedding matrix $C$ as part of application
- OR: Train $C$ separately, lookup embedding vector by multiplying $x_iC$, concatenate embeddings into input vector $x$
- ALSO: Embeddings can be learned for arbitrary core features, e.g., by representing words by POS tags and associating a lookup table to each POS tag
Training Feed-Forward Neural Language Models

- Use standard MLP model with input $x$ being concatenation of embedding vectors for each input feature for context words.

- Output layer is probability distribution over all words in vocabulary, guaranteed by using softmax activation function over output nodes $s_i$: $p_i = \frac{e^{s_i}}{\sum_j e^{s_j}}$.

- Given context $x$ and one-hot output vector $y$, optimize negative log-likelihood: $L(W) = -\sum_k y_k \log p_k$.

- Stochastic gradient: $\frac{dL}{dW} = (p - y) h^\top$.

- Weight update: $\Delta w_{i \leftarrow j} = \mu (p_i - y_i) h_j$. 
Recurrent Neural Networks (RNN)

- Problem with MLP Language Model: Fixed context size
- RNNs can use unlimited context by recurrent definition

\[ h_t = f(x_t, h_{t-1}) \] where hidden layer of previous word is reused:

\[
\begin{align*}
    h_t &= f(x_t, h_{t-1}) \\
    &= \sigma(x_t W^{(x1)} + h_{t-1} W^{(h1)}), \\
    y_t &= \text{softmax}(h_t W^{(h2)}).
\end{align*}
\]

- \( x_t \in \mathbb{R}^{d_x} \), \( h_t \in \mathbb{R}^{d_h} \), \( y_t \in \mathbb{R}^{d_y} \),
- \( W^{(x1)} \in \mathbb{R}^{d_x \times d_h} \),
- \( W^{(h1)} \in \mathbb{R}^{d_h \times d_h} \),
- \( W^{(h2)} \in \mathbb{R}^{d_h \times d_y} \).

- Note: Columns of \( W^{(x1)} \) can also be used as word embeddings

Q&A: Unfold the RNN definition recursively over time.
RNN Language Model

Capture long term dependencies by copying contexts over time
Training RNNs

- Truncated back-propagation through time by unfolding network for a fixed number of words in context
Shortcomings and Refinements

- Neural language models require computing the value of each output node in each training step; requires expensive normalization constant $Z = \sum_j e^{s_j}$ over full vocabulary
  - **Self-normalization**: Regularize $\log Z$ in objective s.t. $\log Z \approx 0$ leads to $Z \approx 1$
  - **Noise-contrastive estimation**: Train the model to separate correct training examples from noise examples; only requires output node values for training and noise examples
- Vanishing and exploding gradients in deep networks
  - **Clip** exploding gradients $g \leftarrow \frac{\text{threshold}}{\|g\|} g$ if $\|g\| > \text{threshold}$
  - Avoid vanishing gradients by memory cells, e.g., LSTMs
Refinement: Regularization by Dropout

For each training example, drop out hidden units with probability $1 - p$

At test time, keep all units and multiply outgoing weights by $p$

→ ensures that output equals expected output under distribution used to drop out units during training
Refinement: Regularization by Dropout

- Dropout regularizes networks by training each sampled thinned network very rarely.
- Dropout prevents overfitting by approximately combining $2^n$ possible thinned networks for $n$-hidden unit architecture.
Refinement: LSTM (Long Short-Term Memory)

- LSTMs were designed to preserve gradients over time in memory cells which are accessed via gates
  - input gates regulate how much a new input changes the memory state,
  - forget gates regulate how much of the prior memory state is retained or forgotten,
  - output gates regulate how strongly a memory state is passed on to the next layer.

- Gates are set via component-wise multiplication \( \otimes \) of a (thresholded) gate vector \( a \in [0, 1]^n \) with a vector \( b \in \mathbb{R}^n \)
  - components of \( b \) corresponding to near-one values in \( a \) may pass; those corresponding to near-zero values are blocked

- Memory update via addition (won’t vanish in backprop)
Refinement: LSTM

Similar recurrent definition $h_t = f(x_t, h_{t-1})$ as RNNs, but including explicit memory component $m$:

\[
\begin{align*}
h_t &= f(x_t, h_{t-1}) \\
      &= \tanh(m_t \otimes o), \\
m_t &= m_{t-1} \otimes f + g \otimes i, \\
i &= \sigma(x_t W^{(x)}(x) + h_{t-1} W^{(h)}(h)), \\
f &= \sigma(x_t W^{(x)}(f) + h_{t-1} W^{(h)}(f)), \\
o &= \sigma(x_t W^{(x)}(o) + h_{t-1} W^{(h)}(o)), \\
g &= \tanh(x_t W^{(x)}(g) + h_{t-1} W^{(h)}(g)), \\
x_t &\in \mathbb{R}^{d_x}, m_t, h_t, i, f, o, g &\in \mathbb{R}^{d_h}, \\
W^{(x*, h*)} &\in \mathbb{R}^{d_x \times d_h}, W^{(h*)} &\in \mathbb{R}^{d_h \times d_h}.
\end{align*}
\]
Refinement: LSTM

LSTM Layer Time t-1

Preceding Layer

Next Layer

Statistical Methods for CL
RNN Encoder-Decoder for Statistical Machine Translation (SMT)

- Training data \( D = \{(x^i, y^i)\}_{i=1}^N \) where
  - \( x = (x_1, x_2, \ldots, x_{T_x}) \) is a sequence of source words,
  - \( y = (y_1, y_2, \ldots, y_{T_y}) \) is a sequence of target words.

- Conditional language model:
  - \( p(y|x) = \prod_{t=1}^{T_y} p(y_t|y_{<t}, x) \) where \( y_{<t} = y_1, \ldots, y_{t-1} \)

- Negative log-likelihood objective:
  - \( -\frac{1}{N} \sum_{i=1}^N \sum_{t=1}^{T_y} \log p(y^i_t|y^i_{<t}, x^i) \)
Simple RNN Encoder-Decoder for SMT

▶ RNN Encoder:
  ▶ Map source-language input sentence into single context vector by using last memory state of RNN/LSTM:

\[
\begin{align*}
    h_t &= f(x_t, h_{t-1}), \\
    c &= q(h_1, h_2, \ldots, h_{T_x}) = h_{T_x}.
\end{align*}
\]

▶ RNN Decoder:
  ▶ Use RNN/LSTM to decode target language words by concatenating context vector \( c \) to hidden output state representation:

\[
\begin{align*}
    s_t &= f(y_{t-1} \parallel c, s_{t-1}), \\
    p(y_t|y_{<t}, x) &= \text{softmax}(s_t W^{(h2)}).
\end{align*}
\]
Example: Translation with Simple RNN Encoder-Decoder
Refinement: Bi-directional RNN Encoder

- Forward RNN reads input from \( x_1 \) to \( x_T \) and calculates the forward hidden state sequence \( \rightarrow h_1, \ldots, \rightarrow h_T \) where \( \rightarrow h_t = f(x_t, \rightarrow h_{t-1}) \).

- Backward RNN reads input from \( x_T \) to \( x_1 \) and calculates the backward hidden state sequence \( \leftarrow h_1, \ldots, \leftarrow h_T \) where \( \leftarrow h_t = f(x_t, \leftarrow h_{t+1}) \).

- Concatenate hidden states of forward and backward RNNs:
  \[
  h_t = \leftarrow h_t \| \rightarrow h_t,
  \]
Refinement: Attention-Based RNN Decoder

Attention Mechanism:
- Instead of encoding whole source sentence into \( c \), use weighted average of source context vectors \( c_i = \sum_{j=1}^{T_x} a_{ij} h_j \),
- **Attention weights** \( a_{ij} = \frac{e^{e_{ij}}}{\sum_{j'=1}^{T} e^{e_{ij'}}} \) are computed by softmax over the relevance of a source-word context vector \( h_j \) for translating the next target word represented by target word context \( s_{i-1} \) just before emitting word \( y_i \)
- This matrix encodes a **soft alignment model** for translation
- Can be learned by MLP \( e_{ij} = v \tanh(s_{i-1}W + h_jU) \)
Attention Mechanism: Example

Economic growth has slowed down in recent years.

Das Wirtschaftswachstum hat sich in den letzten Jahren verlangsamt.

Economic growth has slowed down in recent years.

La croissance économique s’est ralentie ces dernières années.

- Soft alignments learned by attention mechanism
Attention-Based RNN Encoder-Decoder for SMT

- Encoder: Concatenate left-to-right and right-to-left RNNs
- Decoder: Predict next output word, given previous output words and contexts, and alignment-weighted input contexts
- Not shown: Generate output words from hidden output states
Summary

- Basic principles of machine learning:
  - To do learning, we set up an **objective function** that tells the fit of the model to the data
    - For linear models, the objective will be convex
  - Apply **optimization** techniques to train model parameters (weights, probabilities, etc.)
    - For linear models, even if non-linearity is introduced by kernels, we can apply convex optimization techniques
  - **Algorithms** can be set up as batch or online learners, with and without regularization
Summary

▶ Extension of models
  ▶ Kernel Machines
    ▶ Kernel Machines introduce nonlinearity by using specific feature maps or kernels
    ▶ Feature map or kernel is not part of optimization problem, thus convex optimization of loss function for linear model possible
  ▶ Neural Networks
    ▶ Similarities and nonlinear combinations of features are learned: representation learning
    ▶ We lose the advantages of convex optimization since objective functions will be nonconvex
    ▶ However, basic building blocks (e.g. perceptron) and optimization techniques (e.g. stochastic gradient descent, regularization) stay the same
Further Reading

- **Introductory Example:**
  

- **Naive Bayes:**
  

- **Logistic Regression:**
  


  Stefan Riezler, Detlef Prescher, Jonas Kuhn, and Mark Johnson. 2000.

- **Perceptron:**

- **SVM:**
  - Olivier Chapelle. 2007.


**Kernels and Regularization:**


References


- **Convex and Non-Convex Optimization:**

- **Online/Stochastic Optimization:**
  - Boris T. Polyak. 1964.


**Neural Networks:**


Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010.


Thanks