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

convex	speed	label	convex	speed	label
small	small	-	small	large	+
small	medium	-	medium	large	+
small	medium	-	medium	large	+
medium	small	-	large	small	+
large	small	-	large	large	+
small	small	-	large	medium	+
small	large	-			
small	medium	-			

- Learning model parameters from data:
  - p(+) = 6/14, p(-) = 8/14
  - p(convex = small\|-) = 6/8, p(convex = med\|-) = 1/8, p(convex = large\|-) = 1/8
    p(speed = small\|-) = 4/8, p(speed = med\|-) = 3/8, p(speed = large\|-) = 1/8
    p(convex = small\|+) = 1/6, p(convex = med\|+) = 2/6, p(convex = large\|+) = 3/6
    p(speed = small\|+) = 1/6, p(speed = med\|+) = 1/6, p(speed = large\|+) = 4/6
- Predict unseen p(label = ?, convex = med, speed = med)
  - p(-) · p(convex = med\|-) · p(speed = med\|-) = 8/14 · 1/8 · 3/8 = 0.027
  - p(+) · p(convex = med\|+) · p(speed = med\|+) = 6/14 · 2/6 · 1/6 = 0.024
  - Inedible: p(convex = med, speed = med, label = -) > p(convex = med, speed = med, 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: $\boldsymbol{x} \in \mathcal{X}$
  - e.g., document or sentence with some words $\boldsymbol{x} = w_1 \ldots w_n$
- Output: $\boldsymbol{y} \in \mathcal{Y}$
  - e.g., document class, translation, parse tree
- Input/Output pair: $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{X} \times \mathcal{Y}$
  - e.g., a document $\boldsymbol{x}$ and its class label $\boldsymbol{y}$,
  - a source sentence $\boldsymbol{x}$ and its translation $\boldsymbol{y}$,
  - a sentence $\boldsymbol{x}$ and its parse tree $\boldsymbol{y}$

Feature Representations

- Most NLP problems can be cast as multiclass classification where we assume a high-dimensional joint feature map on input-output pairs $(\boldsymbol{x}, \boldsymbol{y})$
  - $\phi(\boldsymbol{x}, \boldsymbol{y}) : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^m$
- 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 $\mathbf{v} \in \mathbb{R}^m$, let $\mathbf{v}_j$ be the j^{th} value

Example: Text Classification

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

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

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

Example: Text Classification

$\boldsymbol{\phi}_j(\boldsymbol{x}, \boldsymbol{y}) =$ % of words in $\boldsymbol{x}$ containing punctuation and $\boldsymbol{y} =$ "scientific"

Q&A: Punctuation symbols - positive indicator or negative indicator for scientific articles?

Example: Part-of-Speech Tagging

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

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

Q&A: What weight would it get?

Example: Named-Entity Recognition

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

$$\phi_0(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "George"} \\ & \text{and } \boldsymbol{y} = \text{"Person"} \\ 0 & \text{otherwise} \end{cases} \qquad \phi_4(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "George"} \\ & \text{and } \boldsymbol{y} = \text{"Object"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_1(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "Washington"} \\ & \text{and } \boldsymbol{y} = \text{"Person"} \\ 0 & \text{otherwise} \end{cases} \qquad \phi_5(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "Washington"} \\ & \text{and } \boldsymbol{y} = \text{"Object"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_2(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "Bridge"} \\ & \text{and } \boldsymbol{y} = \text{"Person"} \\ 0 & \text{otherwise} \end{cases} \qquad \phi_6(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "Bridge"} \\ & \text{and } \boldsymbol{y} = \text{"Object"} \\ 0 & \text{otherwise} \end{cases}$$

$$\phi_3(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "General"} \\ & \text{and } \boldsymbol{y} = \text{"Person"} \\ 0 & \text{otherwise} \end{cases} \qquad \phi_7(\boldsymbol{x},\boldsymbol{y}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \text{ contains "General"} \\ & \text{and } \boldsymbol{y} = \text{"Object"} \\ 0 & \text{otherwise} \end{cases}$$

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

Block Feature Vectors

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

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

Example: Statistical Machine Translation

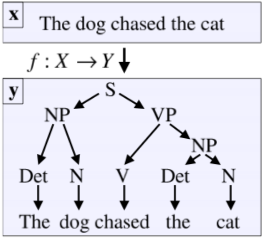
- $\boldsymbol{x}$ is a source sentence and $\boldsymbol{y}$ is translation

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

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

Q&A: Which phrase indicator should be preferred?

Example: Parsing



$$\Psi(\mathbf{x},\mathbf{y}) = \begin{array}{cl} \left(\begin{array}{c} 1 \\ 0 \\ 2 \\ 1 \\ \vdots \\ 0 \\ 2 \\ 1 \\ 1 \\ 1 \end{array}\right) & \begin{array}{c} S \rightarrow NP\,VP \\ S \rightarrow NP \\ NP \rightarrow Det\ N \\ VP \rightarrow V\ NP \\ \\ Det \rightarrow dog \\ Det \rightarrow the \\ N \rightarrow dog \\ V \rightarrow chased \\ N \rightarrow cat \end{array} \end{array}$$

Note: Label $\boldsymbol{y}$ includes sentence $\boldsymbol{x}$

Linear Models

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

$$
\begin{aligned}
f(\boldsymbol{x}; \boldsymbol{\omega}) &= \underset{\boldsymbol{y} \in \mathcal{Y}}{\arg\max}\ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \\
&= \underset{\boldsymbol{y} \in \mathcal{Y}}{\arg\max}\ \sum_{j=0}^{m} \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}, \boldsymbol{y})
\end{aligned}
$$

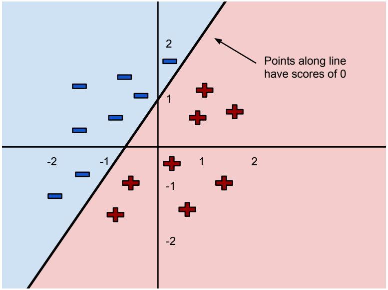
- Let $\boldsymbol{\omega} \in \mathbb{R}^m$ be a high dimensional weight vector
- Assume that $\boldsymbol{\omega}$ is known
  - **Multiclass Classification**: $\mathcal{Y} = \{0, 1, \ldots, N\}$

$$
\boldsymbol{y} = \underset{\boldsymbol{y}' \in \mathcal{Y}}{\arg\max}\ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}')
$$

  - **Binary Classification** just a special case of multiclass

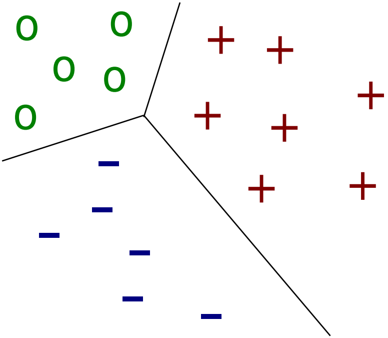
Linear Models for Binary Classification

- ω 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



Multiclass Linear Model

Defines regions of space. Visualization difficult.



- $+$ are all points $(\boldsymbol{x}, \boldsymbol{y})$ where $+ = \arg\max_{\boldsymbol{y}} \; \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y})$

Convex Optimization for Supervised Learning

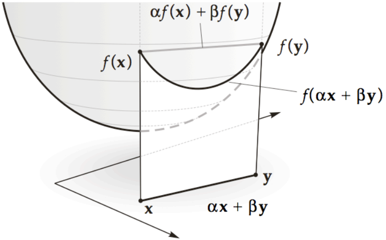
How to learn weight vector $\boldsymbol{\omega}$ in order to make decisions?

- Input:
  - i.i.d. (independent and identically distributed) training examples $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$
  - feature representation ϕ
- Output: $\boldsymbol{\omega}$ that maximizes an objective function on the training set
  - $\boldsymbol{\omega} = \arg\max \mathcal{L}(\mathcal{T}; \boldsymbol{\omega})$
  - Equivalently minimize: $\boldsymbol{\omega} = \arg\min -\mathcal{L}(\mathcal{T}; \boldsymbol{\omega})$

Objective Functions

- Ideally we can decompose $\mathcal{L}$ by training pairs $(\boldsymbol{x}, \boldsymbol{y})$
  - $\mathcal{L}(\mathcal{T}; \boldsymbol{\omega}) \propto \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{T}} loss((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\omega})$
  - *loss* is a function that measures some value correlated with errors of parameters $\boldsymbol{\omega}$ on instance $(\boldsymbol{x}, \boldsymbol{y})$
- Example:
  - $\boldsymbol{y} \in \{1, -1\}$, $f(\boldsymbol{x}; \boldsymbol{\omega})$ is the prediction we make for $\boldsymbol{x}$ using $\boldsymbol{\omega}$
  - zero-one loss function:
    $$loss((\boldsymbol{x}, \boldsymbol{y}); \boldsymbol{\omega}) = \begin{cases} 1 & \text{if } f(\boldsymbol{x}; \boldsymbol{\omega}) \times \boldsymbol{y} \leq 0 \\ 0 & \text{else} \end{cases}$$

Convexity

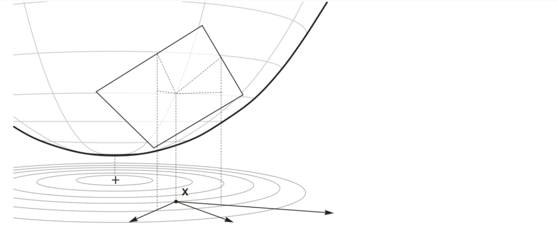


- A function is convex if its graph lies on or below the line segment connecting any two points on the graph

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

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

Gradient

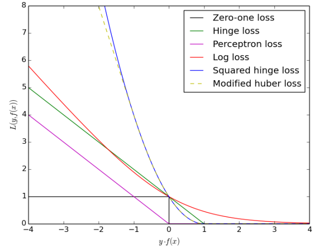


- Gradient of function f is vector of partial derivatives.
  $\nabla f(x) = \left(\frac{\partial}{\partial x_1} f(x), \frac{\partial}{\partial x_2} f(x), \ldots, \frac{\partial}{\partial x_n} f(x) \right)$
- Rate of increase of f at point x in each of the axis-parallel directions.

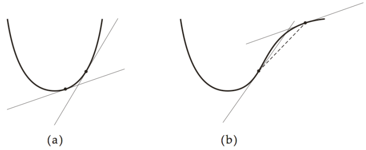
Q&A: What is the gradient at x for the function in the image above?

Convex Optimization

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



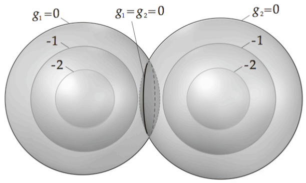
Unconstrained Optimization



- 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 $\nabla f(x) = 0$

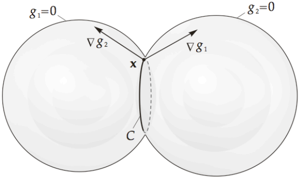
Q&A: How can maximization be defined as minimization problem?

Constrained Optimization with Equality Constraints



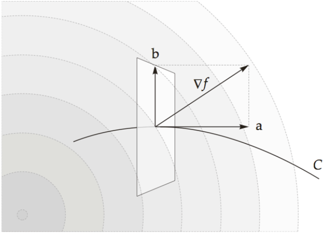
- Optimization problem is finding a point among the feasible points that satisfy constraints $g_i(x) = 0$ where $f(x)$ is minimal
- Example: For 3-dimensional domain of $f(x)$, feasible points constitute intersection of surfaces $g_1(x) = 0$ and $g_2(x) = 0$

Equality Constraints



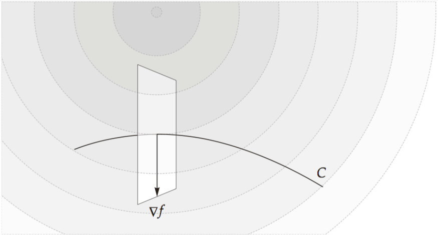
- Gradients $\nabla g_1(x)$, $\nabla g_2(x)$ define a normal plane to feasible set curve C: $\alpha_1 \nabla g_1(x) + \alpha_2 \nabla g_2(x)$, generally $\sum_i \alpha_i \nabla g_i(x)$
- Goal: move along C looking for point that minimizes f

Equality Constraints



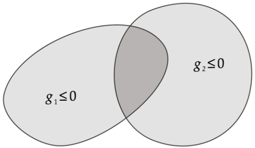
- $\nabla f(x)$ is a sum of vector a (= tangent to C, pointing in direction of increase of f) and vector b (= lying in normal plane to C)
- To minimize f, move in opposite direction of a
- Minimium reached when there is no direction of further decrease

Lagrange Multipliers



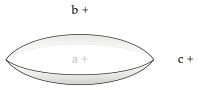
- At minimum, gradient of f lies entirely in plane perpendicular to feasible set curve C: $\nabla f(x) = \sum_i \alpha_i \nabla g_i(x)$
- Solving for x solves *constrained optimization* problem.
- Define Lagrangian $L(x) = f(x) - \sum_i \alpha_i g_i(x)$ where equality constraints have standard form $g_i = 0, \forall i$.
- Setting $\nabla L(x) = 0$ and solving for x gives same solution as for constrained problem, but by *unconstrained optimization*

Inquality 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.

Inquality 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:

$$\arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) \propto \arg\max_{\boldsymbol{y}} P(\boldsymbol{y})P(\boldsymbol{x}|\boldsymbol{y})$$

  - Uses Bayes Rule:

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{P(\boldsymbol{y})P(\boldsymbol{x}|\boldsymbol{y})}{P(\boldsymbol{x})} \text{ for fixed } \boldsymbol{x}$$

- Generative model since $P(\boldsymbol{y})P(\boldsymbol{x}|\boldsymbol{y}) = P(\boldsymbol{x}, \boldsymbol{y})$ is a joint probability
  - Because we model a distribution that can randomly generate outputs *and* inputs, not just outputs

Naivety of Naive Bayes

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

Naive Bayes Assumption

(conditional independence)

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

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

Q&A: How would $P(\boldsymbol{x}, \boldsymbol{y})$ be defined without independence?

Naive Bayes – Learning

- Input: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$
- Let $\phi_i(\boldsymbol{x}) \in \{1, \ldots, F_i\}$
- Parameters $\mathcal{P} = \{P(\boldsymbol{y}), P(\phi_i(\boldsymbol{x})|\boldsymbol{y})\}$

Maximum Likelihood Estimation

- What's left? Defining an objective $\mathcal{L}(\mathcal{T})$
- $\mathcal{P}$ plays the role of $\boldsymbol{\omega}$
- What objective to use?
- Objective: Maximum Likelihood Estimation (MLE)

$$\mathcal{L}(\mathcal{T}) = \prod_{t=1}^{|\mathcal{T}|} P(\boldsymbol{x}_t, \boldsymbol{y}_t) = \prod_{t=1}^{|\mathcal{T}|} \left(P(\boldsymbol{y}_t) \prod_{i=1}^{m} P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right)$$

Naive Bayes – Learning

MLE has closed form solution

$$\mathcal{P} = \underset{\mathcal{P}}{\arg\max} \prod_{t=1}^{|\mathcal{T}|} \left(P(\boldsymbol{y}_t) \prod_{i=1}^{m} P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right)$$

$$P(\boldsymbol{y}) = \frac{\sum_{t=1}^{|\mathcal{T}|} [\![\boldsymbol{y}_t = \boldsymbol{y}]\!]}{|\mathcal{T}|}$$

$$P(\phi_i(\boldsymbol{x})|\boldsymbol{y}) = \frac{\sum_{t=1}^{|\mathcal{T}|} [\![\phi_i(\boldsymbol{x}_t) = \phi_i(\boldsymbol{x}) \text{ and } \boldsymbol{y}_t = \boldsymbol{y}]\!]}{\sum_{t=1}^{|\mathcal{T}|} [\![\boldsymbol{y}_t = \boldsymbol{y}]\!]}$$

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

Thus, these are just normalized counts over events in $\mathcal{T}$

Deriving MLE

$$
\begin{aligned}
\mathcal{P} &= \arg\max_{\mathcal{P}} \prod_{t=1}^{|\mathcal{T}|} \left(P(\boldsymbol{y}_t) \prod_{i=1}^{m} P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right) \\
&= \arg\max_{\mathcal{P}} \sum_{t=1}^{|\mathcal{T}|} \left(\log P(\boldsymbol{y}_t) + \sum_{i=1}^{m} \log P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t) \right) \\
&= \arg\max_{P(\boldsymbol{y})} \sum_{t=1}^{|\mathcal{T}|} \log P(\boldsymbol{y}_t) + \arg\max_{P(\phi_i(\boldsymbol{x})|\boldsymbol{y})} \sum_{t=1}^{|\mathcal{T}|} \sum_{i=1}^{m} \log P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t)
\end{aligned}
$$

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

Deriving MLE

$$\mathcal{P} = \arg\max_{P(\boldsymbol{y})} \sum_{t=1}^{|\mathcal{T}|} \log P(\boldsymbol{y}_t) + \arg\max_{P(\phi_i(\boldsymbol{x})|\boldsymbol{y})} \sum_{t=1}^{|\mathcal{T}|} \sum_{i=1}^{m} \log P(\phi_i(\boldsymbol{x}_t)|\boldsymbol{y}_t)$$

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 $\mathcal{T}$, either $(\boldsymbol{y}_t = \boldsymbol{y})$ or $(\phi_i(\boldsymbol{x}_t) = \phi_i(\boldsymbol{x}), \boldsymbol{y}_t = \boldsymbol{y})$

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 λ, 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 v in $\mathcal{T}$:

$$P(\boldsymbol{y}) = \frac{\sum_{t=1}^{|\mathcal{T}|} [\![\boldsymbol{y}_t = \boldsymbol{y}]\!]}{|\mathcal{T}|}$$

$$P(\phi_i(\boldsymbol{x})|\boldsymbol{y}) = \frac{\sum_{t=1}^{|\mathcal{T}|} [\![\phi_i(\boldsymbol{x}_t) = \phi_i(\boldsymbol{x}) \text{ and } \boldsymbol{y}_t = \boldsymbol{y}]\!]}{\sum_{t=1}^{|\mathcal{T}|} [\![\boldsymbol{y}_t = \boldsymbol{y}]\!]}$$

Naive Bayes is a linear model

- Let $\boldsymbol{\omega}_{\boldsymbol{y}} = \log P(\boldsymbol{y}),\ \forall \boldsymbol{y} \in \mathcal{Y}$
- Let $\boldsymbol{\omega}_{\phi_i(\boldsymbol{x}),\boldsymbol{y}} = \log P(\phi_i(\boldsymbol{x})|\boldsymbol{y}),\ \forall \boldsymbol{y} \in \mathcal{Y}, \phi_i(\boldsymbol{x}) \in \{1, \ldots, F_i\}$

$$
\begin{aligned}
\arg\max_{\boldsymbol{y}}\ P(\boldsymbol{y}|\phi(\boldsymbol{x})) \quad &\propto \quad \arg\max_{\boldsymbol{y}}\ P(\phi(\boldsymbol{x}), \boldsymbol{y}) = \arg\max_{\boldsymbol{y}}\ P(\boldsymbol{y}) \prod_{i=1}^{m} P(\phi_i(\boldsymbol{x})|\boldsymbol{y}) \\
&= \quad \arg\max_{\boldsymbol{y}}\ \log P(\boldsymbol{y}) + \sum_{i=1}^{m} \log P(\phi_i(\boldsymbol{x})|\boldsymbol{y}) \\
&= \quad \arg\max_{\boldsymbol{y}}\ \boldsymbol{\omega}_{\boldsymbol{y}} + \sum_{i=1}^{m} \boldsymbol{\omega}_{\phi_i(\boldsymbol{x}),\boldsymbol{y}} \\
&= \quad \arg\max_{\boldsymbol{y}}\ \sum_{\boldsymbol{y}'} \boldsymbol{\omega}_{\boldsymbol{y}} \psi_{\boldsymbol{y}'}(\boldsymbol{y}) + \sum_{i=1}^{m} \sum_{j=1}^{F_i} \boldsymbol{\omega}_{\phi_i(\boldsymbol{x}),\boldsymbol{y}} \psi_{i,j}(\boldsymbol{x})
\end{aligned}
$$

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

Smoothing

- doc 1: $\boldsymbol{y}_1 =$ sports, "hockey is fast"
- doc 2: $\boldsymbol{y}_2 =$ politics, "politicians talk fast"
- doc 3: $\boldsymbol{y}_3 =$ 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 $P(\boldsymbol{x}, \boldsymbol{y})$
  - 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 $P(\boldsymbol{y}|\boldsymbol{x})$ (or something related)
  - Logistic Regression – MLE of $P(\boldsymbol{y}|\boldsymbol{x})$
  - Perceptron and SVMs – minimize classification error
- Generative and discriminative models use $P(\boldsymbol{y}|\boldsymbol{x})$ for prediction; differ only on what distribution they use to set $\boldsymbol{\omega}$

Logistic Regression

Logistic Regression

Define a conditional probability:

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

Note: still a linear model

$$\begin{aligned}
\arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) &= \arg\max_{\boldsymbol{y}} \frac{e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y})}}{Z_{\boldsymbol{x}}} \\
&= \arg\max_{\boldsymbol{y}} e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y})} \\
&= \arg\max_{\boldsymbol{y}} \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x},\boldsymbol{y})
\end{aligned}$$

Logistic Regression

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

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

$$\begin{aligned}\boldsymbol{\omega} &= \arg\max_{\boldsymbol{\omega}} \mathcal{L}(\mathcal{T};\boldsymbol{\omega}) \\ &= \arg\max_{\boldsymbol{\omega}} \prod_{t=1}^{|\mathcal{T}|} P(\boldsymbol{y}_t|\boldsymbol{x}_t) = \arg\max_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \log P(\boldsymbol{y}_t|\boldsymbol{x}_t)\end{aligned}$$

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

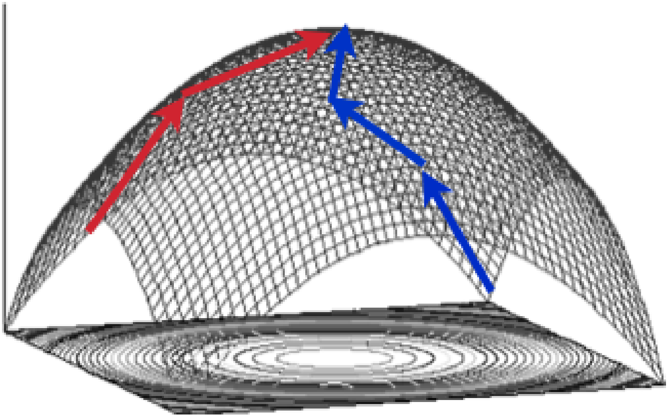
Logistic Regression

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

$$\boldsymbol{\omega} = \arg\max_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \log P(\boldsymbol{y}_t|\boldsymbol{x}_t) \ (*)$$

- 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 $\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = \sum_{t=1}^{|\mathcal{T}|} \log\left(e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t)}/Z_{\boldsymbol{x}}\right)$
- Want to find $\arg\max_{\boldsymbol{\omega}} \mathcal{L}(\mathcal{T};\boldsymbol{\omega})$
  - Set $\boldsymbol{\omega}^0 = O^m$
  - Iterate until convergence

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

- $\alpha > 0$ is a step size / learning rate
- $\nabla \mathcal{L}(\mathcal{T};\boldsymbol{\omega})$ is gradient of $\mathcal{L}$ w.r.t. $\boldsymbol{\omega}$
  - A gradient is all partial derivatives over variables w_i
  - i.e., $\nabla \mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = (\frac{\partial}{\partial \boldsymbol{\omega}_1}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}), \frac{\partial}{\partial \boldsymbol{\omega}_2}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}), \ldots, \frac{\partial}{\partial \boldsymbol{\omega}_m}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}))$
- Gradient ascent will always find $\boldsymbol{\omega}$ to maximize $\mathcal{L}$

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

Gradient Descent

- Let $\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = -\sum_{t=1}^{|\mathcal{T}|} \log\left(e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t)}/Z_{\boldsymbol{x}}\right)$
- Want to find $\arg\min_{\boldsymbol{\omega}} \mathcal{L}(\mathcal{T};\boldsymbol{\omega})$
  - Set $\boldsymbol{\omega}^0 = O^m$
  - Iterate until convergence

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

- $\alpha > 0$ is step size / learning rate
- $\nabla\mathcal{L}(\mathcal{T};\boldsymbol{\omega})$ is gradient of $\mathcal{L}$ w.r.t. $\boldsymbol{\omega}$
  - A gradient is all partial derivatives over variables w_i
  - i.e., $\nabla\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = \left(\frac{\partial}{\partial\boldsymbol{\omega}_1}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}), \frac{\partial}{\partial\boldsymbol{\omega}_2}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}), \ldots, \frac{\partial}{\partial\boldsymbol{\omega}_m}\mathcal{L}(\mathcal{T};\boldsymbol{\omega})\right)$
- Gradient descent will always find $\boldsymbol{\omega}$ 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:

$$
\begin{aligned}
\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) &= -\sum_t \log P(\boldsymbol{y}_t|\boldsymbol{x}_t) \\
&= -\sum_t \log \frac{e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t)}}{\sum_{\boldsymbol{y}'\in\mathcal{Y}} e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}')}} \\
&= -\sum_t \log \frac{e^{\sum_j \boldsymbol{\omega}_j\times\boldsymbol{\phi}_j(\boldsymbol{x}_t,\boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}}
\end{aligned}
$$

Deriving Gradient

$$
\begin{aligned}
\frac{\partial}{\partial \omega_i} \mathcal{L}(\mathcal{T};\boldsymbol{\omega}) &= \frac{\partial}{\partial \omega_i} - \sum_t \log \frac{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t,\boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}} \\
&= \sum_t \frac{\partial}{\partial \omega_i} - \log \frac{e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t,\boldsymbol{y}_t)}}{Z_{\boldsymbol{x}_t}} \\
&= \sum_t \left(\frac{\partial}{\partial \omega_i} - \log e^{\sum_j \boldsymbol{\omega}_j \times \boldsymbol{\phi}_j(\boldsymbol{x}_t,\boldsymbol{y}_t)} + \frac{\partial}{\partial \omega_i} \log Z_{\boldsymbol{x}_t} \right) \\
&= \sum_t \left(-\boldsymbol{\phi}_i(\boldsymbol{x}_t,\boldsymbol{y}_t) + \frac{\partial}{\partial \omega_i} \log Z_{\boldsymbol{x}_t} \right)
\end{aligned}
$$

Deriving Gradient

$$
\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\omega}_i}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) &= \sum_t \left(-\phi_i(\boldsymbol{x}_t, \boldsymbol{y}_t) + \frac{\partial}{\partial \boldsymbol{\omega}_i} \log Z_{\boldsymbol{x}_t}\right) \\
&= \sum_t \left(-\phi_i(\boldsymbol{x}_t, \boldsymbol{y}_t) + \frac{\partial}{\partial \boldsymbol{\omega}_i} \log \sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \boldsymbol{\omega}_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}')}\right) \\
&= \sum_t \left(-\phi_i(\boldsymbol{x}_t, \boldsymbol{y}_t) + \frac{\sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \boldsymbol{\omega}_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}')} \phi_i(\boldsymbol{x}_t, \boldsymbol{y}')}{\sum_{\boldsymbol{y}' \in \mathcal{Y}} e^{\sum_j \boldsymbol{\omega}_j \times \phi_j(\boldsymbol{x}_t, \boldsymbol{y}')}}\right) \\
&= \sum_t \left(-\phi_i(\boldsymbol{x}_t, \boldsymbol{y}_t) + \sum_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}'|\boldsymbol{x}_t)\phi_i(\boldsymbol{x}_t, \boldsymbol{y}')\right)
\end{aligned}
$$

FINALLY!!!

- After all that,

$$\frac{\partial}{\partial \omega_i}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) \;=\; -\sum_t \phi_i(\boldsymbol{x}_t,\boldsymbol{y}_t) + \sum_t \sum_{\boldsymbol{y}'\in\mathcal{Y}} P(\boldsymbol{y}'|\boldsymbol{x}_t)\phi_i(\boldsymbol{x}_t,\boldsymbol{y}')$$

- And the gradient is:

$$\nabla\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = \left(\frac{\partial}{\partial \boldsymbol{\omega}_0}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}), \frac{\partial}{\partial \boldsymbol{\omega}_1}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}), \ldots, \frac{\partial}{\partial \boldsymbol{\omega}_m}\mathcal{L}(\mathcal{T};\boldsymbol{\omega})\right)$$

- So we can now use gradient descent to find ω!!

Logistic Regression Summary

- Define conditional probability

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

- Minimize conditional negative log-likelihood of training data

$$\boldsymbol{\omega} = \arg\min_{\boldsymbol{\omega}} -\sum_t \log P(\boldsymbol{y}_t|\boldsymbol{x}_t)$$

- Calculate gradient and apply gradient descent optimization

$$\frac{\partial}{\partial \omega_i}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = -\sum_t \phi_i(\boldsymbol{x}_t,\boldsymbol{y}_t) + \sum_t \sum_{\boldsymbol{y}'\in\mathcal{Y}} P(\boldsymbol{y}'|\boldsymbol{x}_t)\phi_i(\boldsymbol{x}_t,\boldsymbol{y}')$$

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

$$\frac{\partial}{\partial \boldsymbol{\omega}_i}\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = -\sum_t \phi_i(\boldsymbol{x}_t, \boldsymbol{y}_t) + \sum_t \sum_{\boldsymbol{y}' \in \mathcal{Y}} P(\boldsymbol{y}'|\boldsymbol{x}_t)\phi_i(\boldsymbol{x}_t, \boldsymbol{y}')$$

  - 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: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

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

Perceptron: Separability and Margin

- Given an training instance $(\boldsymbol{x}_t, \boldsymbol{y}_t)$, define:
  - $\bar{\mathcal{Y}}_t = \mathcal{Y} - \{\boldsymbol{y}_t\}$
  - i.e., $\bar{\mathcal{Y}}_t$ is the set of incorrect labels for $\boldsymbol{x}_t$
- A training set $\mathcal{T}$ is separable with margin $\gamma > 0$ if there exists a vector $\mathbf{u}$ with $\|\mathbf{u}\| = 1$ such that:

$$\mathbf{u} \cdot \phi(\boldsymbol{x}_t, \boldsymbol{y}_t) - \mathbf{u} \cdot \phi(\boldsymbol{x}_t, \boldsymbol{y}') \geq \gamma \tag{1}$$

  for all $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$ and $||\mathbf{u}|| = \sqrt{\sum_j \mathbf{u}_j^2}$
- **Assumption**: the training set is separable with margin γ

Q&A: Why do we require $\|\mathbf{u}\| = 1$?

Perceptron Convergence Theorem

- **Theorem**: For any training set separable with a margin of γ, the following holds for the perceptron algorithm:

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

  where $R \geq ||\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')||$ for all $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$ and $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$
- 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: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

$\boldsymbol{\omega}^{(0)} = 0;\ i = 0$
for $n : 1..N$
  for $t : 1..T$
    Let $\boldsymbol{y}' = \arg\max_{\boldsymbol{y}'} \boldsymbol{\omega}^{(i)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$
    if $\boldsymbol{y}' \neq \boldsymbol{y}_t$
      $\boldsymbol{\omega}^{(i+1)} = \boldsymbol{\omega}^{(i)} + \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$
      $i = i + 1$
return $\boldsymbol{\omega}^{i}$

- Lower bound:

  $\boldsymbol{\omega}^{(k-1)}$ are weights before k^{th} error

  Suppose k^{th} error made at $(\boldsymbol{x}_t, \boldsymbol{y}_t)$

  $\boldsymbol{y}' = \arg\max_{\boldsymbol{y}'} \boldsymbol{\omega}^{(k-1)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$

  $\boldsymbol{y}' \neq \boldsymbol{y}_t$

  $\boldsymbol{\omega}^{(k)} =$
  $\boldsymbol{\omega}^{(k-1)} + \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$

  $\mathbf{u} \cdot \boldsymbol{\omega}^{(k)} = \mathbf{u} \cdot \boldsymbol{\omega}^{(k-1)} + \mathbf{u} \cdot (\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')) \geq \mathbf{u} \cdot \boldsymbol{\omega}^{(k-1)} + \gamma$, by (1)
  Since $\boldsymbol{\omega}^{(0)} = 0$ and $\mathbf{u} \cdot \boldsymbol{\omega}^{(0)} = 0$, for all k: $\mathbf{u} \cdot \boldsymbol{\omega}^{(k)} \geq k\gamma$, by induction on k
  Since $\mathbf{u} \cdot \boldsymbol{\omega}^{(k)} \leq ||\mathbf{u}|| \times ||\boldsymbol{\omega}^{(k)}||$, by the Cauchy-Schwarz inequality, and $||\mathbf{u}|| = 1$, then $||\boldsymbol{\omega}^{(k)}|| \geq k\gamma$

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

- Upper bound:

$$
\begin{aligned}
||\boldsymbol{\omega}^{(k)}||^2 &= ||\boldsymbol{\omega}^{(k-1)}||^2 + ||\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')||^2 + 2\boldsymbol{\omega}^{(k-1)} \cdot (\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')) \\
||\boldsymbol{\omega}^{(k)}||^2 &\leq ||\boldsymbol{\omega}^{(k-1)}||^2 + R^2, \text{ since } R \geq ||\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')|| \\
&\qquad \text{and } \boldsymbol{\omega}^{(k-1)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega}^{(k-1)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \leq 0 \\
&\leq kR^2 \text{ for all } k, \text{ by induction on } k
\end{aligned}
$$

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:

$$loss((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega}) = (\max_{\boldsymbol{y} \neq \boldsymbol{y}_t} \boldsymbol{\omega} \cdot \phi(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{\omega} \cdot \phi(\boldsymbol{x}_t, \boldsymbol{y}_t))_+$$

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

- Stochastic (sub)gradient:

$$\nabla loss = \begin{cases} 0 & \text{if } \boldsymbol{\omega} \cdot \phi(\boldsymbol{x}_t, \boldsymbol{y}_t) - \max_{\boldsymbol{y}} \boldsymbol{\omega} \cdot \phi(\boldsymbol{x}_t, \boldsymbol{y}) \geq 0 \\ \phi(\boldsymbol{x}_t, \boldsymbol{y}) - \phi(\boldsymbol{x}_t, \boldsymbol{y}_t) & \text{else, where } \boldsymbol{y} = \arg\max_{\boldsymbol{y}} \boldsymbol{\omega} \cdot \phi(\boldsymbol{x}_t, \boldsymbol{y}) \end{cases}$$

Averaged Perceptron Algorithm

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

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

Perceptron Summary

- Learns parameters of a linear model by minimizing error
- Guaranteed to find a $\boldsymbol{\omega}$ in a finite amount of time
- Perceptron is an example of an Online Learning Algorithm
  - $\boldsymbol{\omega}$ is updated based on a single training instance, taking a step into the negative direction of the stochastic gradient:

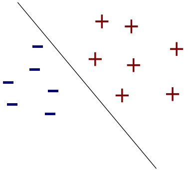
$$\boldsymbol{\omega}^{(i+1)} = \boldsymbol{\omega}^{(i)} + \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$$

    where $\boldsymbol{y}' = \arg\max_{\boldsymbol{y}'} \boldsymbol{\omega}^{(i)} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')$
- More about online learning/stochastic gradient descent later!

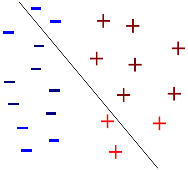
Support Vector Machines (SVMs)

Margin

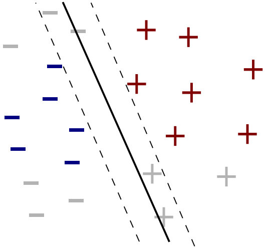
Training

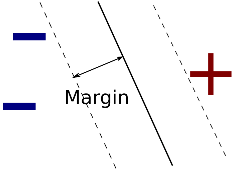


Testing



Denote the value of the margin by γ





Maximizing Margin

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

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

- for every training instance $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$, $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$

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

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

- **Perceptron**: we have shown that:
  - If a training set is separable by some margin, the perceptron will find a $\boldsymbol{\omega}$ that separates the data
  - However, the perceptron does not pick $\boldsymbol{\omega}$ to maximize the margin!

Maximizing Margin

Let $\gamma > 0$

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

such that:

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

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

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

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

$$\beta(\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}')) \geq \beta\gamma, \text{ for some } \beta \geq 1$$

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

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

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

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

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

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

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

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

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

$||\boldsymbol{\omega}|| = 1$ iff $||\mathbf{u}|| = 1/\gamma$,
then $\gamma = 1/||\mathbf{u}||$

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

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

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

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

Change variables: $\mathbf{u} = \dfrac{\boldsymbol{\omega}}{\gamma}$

$||\boldsymbol{\omega}|| = 1$ iff $||\mathbf{u}|| = 1/\gamma$,
then $\gamma = 1/||\mathbf{u}||$

Min Norm (step 1):

$$\max_{\mathbf{u}} \quad \frac{1}{||\mathbf{u}||}$$

such that:

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

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

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

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

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

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

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

Change variables: $\mathbf{u} = \dfrac{\boldsymbol{\omega}}{\gamma}$

$||\boldsymbol{\omega}|| = 1$ iff $||\mathbf{u}|| = 1/\gamma$,
then $\gamma = 1/||\mathbf{u}||$

Min Norm (step 1):

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

such that:

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

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

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

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

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

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

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

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

$||\boldsymbol{\omega}|| = 1$ iff $||\mathbf{u}|| = 1/\gamma$,
then $\gamma = 1/||\mathbf{u}||$

Min Norm (step 2):

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

such that:

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

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

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

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

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

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

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

Change variables: $\mathbf{u} = \dfrac{\boldsymbol{\omega}}{\gamma}$

$||\boldsymbol{\omega}|| = 1$ iff $||\mathbf{u}|| = 1/\gamma$,
then $\gamma = 1/||\mathbf{u}||$

Min Norm (step 2):

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

such that:

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

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

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

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

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

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

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

Change variables: $\mathbf{u} = \dfrac{\boldsymbol{\omega}}{\gamma}$

$||\boldsymbol{\omega}|| = 1$ iff $||\mathbf{u}|| = 1/\gamma$,
then $\gamma = 1/||\mathbf{u}||$

Min Norm (step 3):

$$\min_{\mathbf{u}} \quad \frac{1}{2}||\mathbf{u}||^2$$

such that:

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

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

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

Max Margin = Min Norm

Let $\gamma > 0$

Max Margin:

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

such that:

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

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

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

Min Norm:

$$\min_{\mathbf{u}} \quad \frac{1}{2}||\mathbf{u}||^2$$

such that:

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

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

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

- Intuition: Instead of fixing $||\boldsymbol{\omega}||$ we fix the margin $\gamma = 1$

Support Vector Machines

- **Constrained Optimization Problem**

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

such that:

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

$$\forall(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T} \text{ and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

- **Support Vectors:** Examples where

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

for training instance $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$ and all $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$

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?**

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

such that:

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

$$\forall (\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T} \text{ and } \boldsymbol{y}' \in \bar{\mathcal{Y}}_t$$

- ξ_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

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

such that:

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

where $\xi_t \geq 0$ and $\forall(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$ and $\boldsymbol{y}' \in \bar{\mathcal{Y}}_t$

- 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?

$$\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') \geq 1 - \xi_t$$

$$\xi_t \geq 1 + \max \ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)$$

Support Vector Machines

$$\xi_t \geq 1 + \underbrace{\max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)}_{\textit{negated margin for example}}$$

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

$$\xi_t = \max\{0, 1 + \max_{\boldsymbol{y}' \neq \boldsymbol{y}_t} \ \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}') - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)\}$$

Support Vector Machines

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

Hinge loss

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

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

- Hinge loss allows **unconstrained optimization** (later!)

Summary

What we have covered

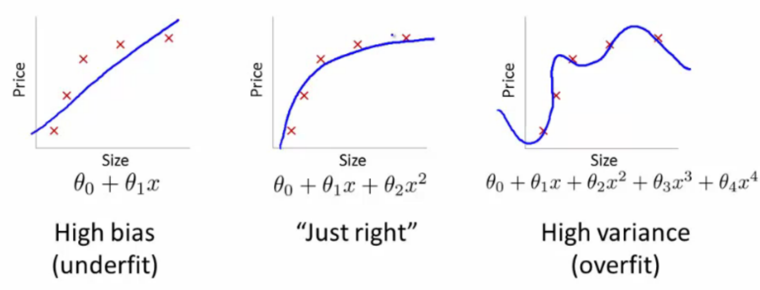
- Linear Models
  - Naive Bayes
  - Logistic Regression
  - Perceptron
  - Support Vector Machines

What is next

- Regularization
- Online learning
- Non-linear models

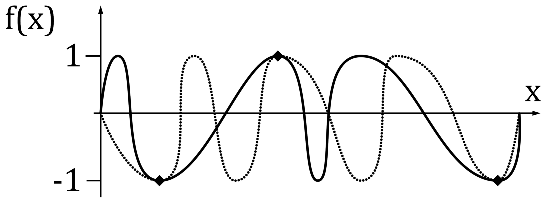
Regularization

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 "$\phi(\boldsymbol{x}, \boldsymbol{y}) = 1$ if $\boldsymbol{x}$=fake and $\boldsymbol{y}$=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=verb

Regularization

- In practice, we regularize models to prevent overfitting

$$\arg\max_{\boldsymbol{\omega}} \ \mathcal{L}(\mathcal{T};\boldsymbol{\omega}) - \lambda\mathcal{R}(\boldsymbol{\omega})$$

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

Logistic Regression with L2 Regularization

- Perhaps most common learner in NLP

$$\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) - \lambda\mathcal{R}(\boldsymbol{\omega}) = \sum_{t=1}^{|\mathcal{T}|} \log\left(e^{\boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t)}/Z_{\boldsymbol{x}}\right) - \frac{\lambda}{2}\|\boldsymbol{\omega}\|^2$$

- What are the new partial derivatives?

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

- We know $\frac{\partial}{\partial w_i}\mathcal{L}(\mathcal{T};\boldsymbol{\omega})$
- Just need $\frac{\partial}{\partial w_i}\frac{\lambda}{2}\|\boldsymbol{\omega}\|^2 = \frac{\partial}{\partial w_i}\frac{\lambda}{2}\left(\sqrt{\sum_i \boldsymbol{\omega}_i^2}\right)^2 = \frac{\partial}{\partial w_i}\frac{\lambda}{2}\sum_i \boldsymbol{\omega}_i^2 = \lambda\boldsymbol{\omega}_i$

Support Vector Machines

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

$$
\begin{aligned}
\boldsymbol{\omega} &= \arg\min_{\boldsymbol{\omega}} \mathcal{L}(\mathcal{T};\boldsymbol{\omega}) + \lambda\mathcal{R}(\boldsymbol{\omega}) \\
&= \arg\min_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \mathit{loss}((\boldsymbol{x}_t,\boldsymbol{y}_t);\boldsymbol{\omega}) + \lambda\mathcal{R}(\boldsymbol{\omega}) \\
&= \arg\min_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \max\,(0, 1 + \max_{\boldsymbol{y}\neq\boldsymbol{y}_t}\ \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}) - \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t)) + \lambda\mathcal{R}(\boldsymbol{\omega}) \\
&= \arg\min_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \max\,(0, 1 + \max_{\boldsymbol{y}\neq\boldsymbol{y}_t}\ \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}) - \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t)) + \frac{\lambda}{2}\|\boldsymbol{\omega}\|^2
\end{aligned}
$$

SVMs vs. Logistic Regression

$$\begin{aligned}\boldsymbol{\omega} &= \arg\min_{\boldsymbol{\omega}} \; \mathcal{L}(\mathcal{T};\boldsymbol{\omega}) + \lambda\mathcal{R}(\boldsymbol{\omega}) \\ &= \arg\min_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} \mathit{loss}((\boldsymbol{x}_t,\boldsymbol{y}_t);\boldsymbol{\omega}) + \lambda\mathcal{R}(\boldsymbol{\omega})\end{aligned}$$

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

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

Logistic Regression/log-loss: $-\log\left(e^{\boldsymbol{\omega}\cdot\phi(\boldsymbol{x}_t,\boldsymbol{y}_t)}/Z_{\boldsymbol{x}}\right)$

$$\boldsymbol{\omega} = \arg\min_{\boldsymbol{\omega}} \sum_{t=1}^{|\mathcal{T}|} -\log\left(e^{\boldsymbol{\omega}\cdot\phi(\boldsymbol{x}_t,\boldsymbol{y}_t)}/Z_{\boldsymbol{x}}\right) + \frac{\lambda}{2}\|\boldsymbol{\omega}\|^2$$

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

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

- 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

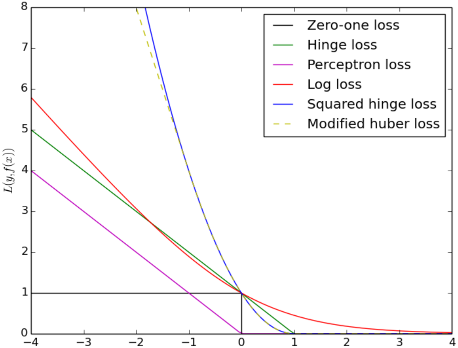
Leave-one-out Generalization Bound for Support Vectors

- 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



Online Learning

Online vs. Batch Learning

Batch($\mathcal{T}$);

- for 1 ... N
  - $\boldsymbol{\omega} \leftarrow \text{update}(\mathcal{T}; \boldsymbol{\omega})$
- return $\boldsymbol{\omega}$

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

Online($\mathcal{T}$);

- for 1 ... N
  - for $(\boldsymbol{x}_t, \boldsymbol{y}_t) \in \mathcal{T}$
    - $\boldsymbol{\omega} \leftarrow \text{update}((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega})$
  - end for
- end for
- return $\boldsymbol{\omega}$

E.g., Perceptron

$$\boldsymbol{\omega} = \boldsymbol{\omega} + \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y})$$

Batch Gradient Descent

- Let $\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = \sum_{t=1}^{|\mathcal{T}|} loss((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega})$
  - Set $\boldsymbol{\omega}^0 = O^m$
  - Iterate until convergence

$$\begin{aligned}\boldsymbol{\omega}^i &= \boldsymbol{\omega}^{i-1} - \alpha \nabla \mathcal{L}(\mathcal{T}; \boldsymbol{\omega}^{i-1}) \\ &= \boldsymbol{\omega}^{i-1} - \sum_{t=1}^{|\mathcal{T}|} \alpha \nabla loss((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega}^{i-1})\end{aligned}$$

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

Stochastic Gradient Descent

- Stochastic Gradient Descent (SGD)
  - Approximate batch gradient $\nabla\mathcal{L}(\mathcal{T};\boldsymbol{\omega})$ with stochastic gradient $\nabla \mathit{loss}((\boldsymbol{x}_t,\boldsymbol{y}_t);\boldsymbol{\omega})$
- Let $\mathcal{L}(\mathcal{T};\boldsymbol{\omega}) = \sum_{t=1}^{|\mathcal{T}|} \mathit{loss}((\boldsymbol{x}_t,\boldsymbol{y}_t);\boldsymbol{\omega})$
  - Set $\boldsymbol{\omega}^0 = O^m$
  - iterate until convergence
    - sample $(\boldsymbol{x}_t,\boldsymbol{y}_t) \in \mathcal{T}$ // "stochastic"
    - $\boldsymbol{\omega}^i = \boldsymbol{\omega}^{i-1} - \alpha\nabla \mathit{loss}((\boldsymbol{x}_t,\boldsymbol{y}_t);\boldsymbol{\omega}^{i-1})$
  - return $\boldsymbol{\omega}$

Online Logistic Regression

- Stochastic Gradient Descent (SGD)
- $loss((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega}) = \text{log-loss}$
- $\nabla loss((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega}) = \nabla \left(-\log \left(e^{\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)} / Z_{\boldsymbol{x}_t}\right)\right)$
- From logistic regression section:

$$\nabla \left(-\log \left(e^{\boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)} / Z_{\boldsymbol{x}_t}\right)\right) = -\left(\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \sum_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y})\right)$$

- Plus regularization term (if part of model)

Online SVMs

- Stochastic Gradient Descent (SGD)
- $\mathit{loss}((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega}) = \text{hinge-loss}$

$$\nabla \mathit{loss}((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega}) = \nabla \left(\max \left(0, 1 + \max_{\boldsymbol{y} \neq \boldsymbol{y}_t} \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)\right)\right)$$

- Subgradient is:

$$\nabla \left(\max \left(0, 1 + \max_{\boldsymbol{y} \neq \boldsymbol{y}_t} \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)\right)\right)$$

$$= \begin{cases} 0, & \text{if } \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \max_{\boldsymbol{y}} \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) \geq 1 \\ \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t), & \text{otherwise, where } \boldsymbol{y} = \arg\max_{\boldsymbol{y}} \boldsymbol{\omega} \cdot \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) \end{cases}$$

- Plus regularization term (L2 norm for SVMs):

$$\nabla \frac{\lambda}{2} ||\boldsymbol{\omega}||^2 = \lambda \boldsymbol{\omega}$$

Perceptron and Hinge-Loss

SVM subgradient update looks like perceptron update

$$\boldsymbol{\omega}^i = \boldsymbol{\omega}^{i-1} - \alpha \begin{cases} \lambda\boldsymbol{\omega}, & \text{if } \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t) - \max_{\boldsymbol{y}} \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}) \geq 1 \\ \boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t) + \lambda\boldsymbol{\omega}, & \text{otherwise, where } \boldsymbol{y} = \arg\max_{\boldsymbol{y}} \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}) \end{cases}$$

Perceptron

$$\boldsymbol{\omega}^i = \boldsymbol{\omega}^{i-1} - \alpha \begin{cases} 0, & \text{if } \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t) - \max_{\boldsymbol{y}} \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}) \geq 0 \\ \boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}) - \boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t), & \text{otherwise, where } \boldsymbol{y} = \arg\max_{\boldsymbol{y}} \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}) \end{cases}$$

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

$$\max\left(0, 1 + \max_{\boldsymbol{y}\neq\boldsymbol{y}_t} \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}) - \boldsymbol{\omega}\cdot\boldsymbol{\phi}(\boldsymbol{x}_t,\boldsymbol{y}_t)\right)$$

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 $1/\mathcal{T}$ 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 $\mathcal{T}$ 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 $\boldsymbol{v}$ weighted by momentum coefficient $0 \leq \mu < 1$ [Polyak 1964]:
  - $$\boldsymbol{\omega}^{i+1} = \boldsymbol{\omega}^{i} - \alpha \nabla loss((\boldsymbol{x}_t, \boldsymbol{y}_t); \boldsymbol{\omega}^{i}) + \mu \boldsymbol{v}^{i}$$

    where
    $$\boldsymbol{v}^{i} = \boldsymbol{\omega}^{i} - \boldsymbol{\omega}^{i-1}$$
  - 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
  - $\bar{\boldsymbol{\omega}} = \left(\sum_i \boldsymbol{\omega}^{(i)}\right) / \left(N \times T\right)$
  - 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 $1 - \delta$:

$$\mathbb{E}[loss((\boldsymbol{x}, \boldsymbol{y}); \bar{\boldsymbol{\omega}})] \leq M_k + \sqrt{\frac{2}{k} \ln \frac{1}{\delta}}$$

  - $=$ generalization bound based on online performance [Cesa-Bianchi et al. 2004]
  - can be applied to all online learners with convex losses

Quick Summary

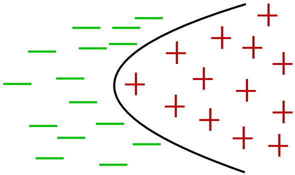
Linear Learners

- 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 $\boldsymbol{x} \in \mathbb{R}^n$ s.t. $x_{j_1} * x_{j_2} * \cdots * x_{j_d}$ for $j_1, \ldots, j_d \in \{1 \ldots n\}$
- Ordered monomial feature map: $\boldsymbol{\phi} : \mathbb{R}^2 \to \mathbb{R}^4$ s.t. $(x_1, x_2) \mapsto (x_1^2, x_2^2, x_1 x_2, x_2 x_1)$
- Computation of kernel from feature map:

$$
\begin{aligned}
\boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{x}') &= \sum_{i=1}^{4} \phi_i(\boldsymbol{x}) \phi_i(\boldsymbol{x}') \text{ (Def. dot product)} \\
&= x_1^2 {x'}_1^2 + x_2^2 {x'}_2^2 + x_1 x_2 {x'}_1 {x'}_2 + x_2 x_1 {x'}_2 {x'}_1 \text{ (Def. } \boldsymbol{\phi}\text{)} \\
&= x_1^2 {x'}_1^2 + x_2^2 {x'}_2^2 + 2 x_1 x_2 {x'}_1 {x'}_2 \\
&= \left(x_1 {x'}_1 + x_2 {x'}_2 \right)^2
\end{aligned}
$$

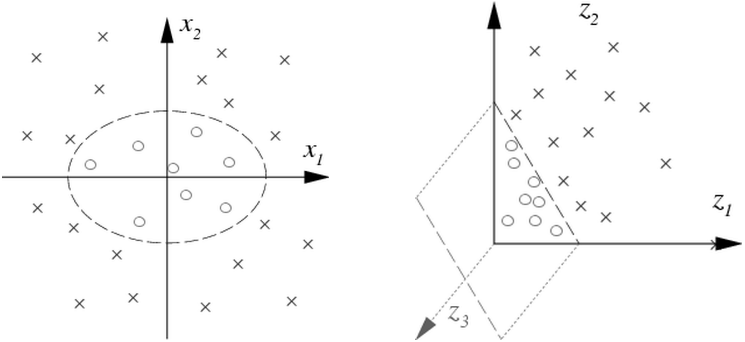
- Direct application of kernel: $\boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{\phi}(\boldsymbol{x}') = (\boldsymbol{x} \cdot \boldsymbol{x}')^2$

Direct Application of Kernel

- Let C_d be a map from $\boldsymbol{x} \in \mathbb{R}^m$ to vectors $C_d(\boldsymbol{x})$ of all d^{th}-degree ordered products of entries of $\boldsymbol{x}$.
  Then the corresponding kernel computing the dot product of vectors mapped by C_d is:
  $K(\boldsymbol{x}, \boldsymbol{x}') = C_d(\boldsymbol{x}) \cdot C_d(\boldsymbol{x}') = (\boldsymbol{x} \cdot \boldsymbol{x}')^d$
- Alternative feature map satisfying this definition = unordered monomial: $\phi_2 : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ s.t. $(x_1, x_2) \mapsto (x_1^2, x_2^2, \sqrt{2}x_1x_2)$

Q&A: Suppose inputs $\boldsymbol{x}$ being vectors of pixel intensities. How can monomial features help to distinguish handwritten 8 from 0 in image recognition?

Non-Linear Feature Map



- $\phi_2 : \mathbb{R}^2 \to \mathbb{R}^3$ 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 $z3$, 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:

$$K(\boldsymbol{x}_t, \boldsymbol{x}_r) \in \mathbb{R}$$

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

$$M_{t,r} = K(\boldsymbol{x}_t, \boldsymbol{x}_r)$$

- ... for any n points. Called the Gram matrix.
- Symmetric:

$$K(\boldsymbol{x}_t, \boldsymbol{x}_r) = K(\boldsymbol{x}_r, \boldsymbol{x}_t)$$

- Positive definite: positivity on diagonal

$$K(\boldsymbol{x}, \boldsymbol{x}) \geq 0 \text{ forall } \boldsymbol{x} \text{ with equality only for } \boldsymbol{x} = 0$$

Mercer's Theorem

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

$$K(\boldsymbol{x}_t, \boldsymbol{x}_r) = \boldsymbol{\phi}(\boldsymbol{x}_t) \cdot \boldsymbol{\phi}(\boldsymbol{x}_r)$$

- This means that instead of mapping input data via non-lineear feature map ϕ and then computing dot product, we can apply kernels directly *without even knowing about* ϕ!

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 $(\boldsymbol{x}, \boldsymbol{y})$, we will write kernels over pairs

$$K((\boldsymbol{x}_t, \boldsymbol{y}_t), (\boldsymbol{x}_r, \boldsymbol{y}_r)) = \phi(\boldsymbol{x}_t, \boldsymbol{y}_t) \cdot \phi(\boldsymbol{x}_r, \boldsymbol{y}_r)$$

- Let's do it for the Perceptron!

Kernel Trick – Perceptron Algorithm

Training data: $\mathcal{T} = \{(\boldsymbol{x}_t, \boldsymbol{y}_t)\}_{t=1}^{|\mathcal{T}|}$

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

- Each feature function $\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t)$ is added and $\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y})$ is subtracted to $\boldsymbol{\omega}$ say $\alpha_{\boldsymbol{y},t}$ times
  - $\alpha_{\boldsymbol{y},t}$ is proportional to number of times label $\boldsymbol{y}$ is predicted for example t and caused an update because of misclassification
- Thus,

$$\boldsymbol{\omega} = \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t}[\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y})]$$

Kernel Trick – Perceptron Algorithm

- We can re-write the argmax function as:

$$
\begin{aligned}
\boldsymbol{y}* &= \arg\max_{\boldsymbol{y}^*} \boldsymbol{\omega}^{(i)} \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}^*) \\
&= \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t} [\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y})] \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}^*) \\
&= \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t} [\boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}_t) \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}^*) - \boldsymbol{\phi}(\boldsymbol{x}_t, \boldsymbol{y}) \cdot \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}^*)] \\
&= \arg\max_{\boldsymbol{y}^*} \sum_{t,\boldsymbol{y}} \alpha_{\boldsymbol{y},t} [K((\boldsymbol{x}_t, \boldsymbol{y}_t), (\boldsymbol{x}, \boldsymbol{y}^*)) - K((\boldsymbol{x}_t, \boldsymbol{y}), (\boldsymbol{x}, \boldsymbol{y}^*))]
\end{aligned}
$$

- We can then re-write the perceptron algorithm strictly with kernels

Kernel Trick – Perceptron Algorithm

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

- Testing on unseen instance $\boldsymbol{x}$:

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

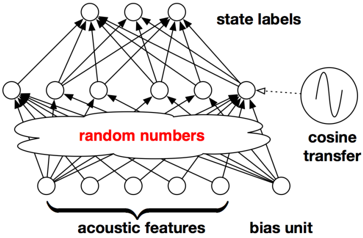
Intuition: y^* is label that is most similar to gold standard labels and least similar to non-gold labels.

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

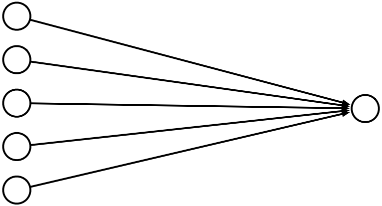
- Alternative to exact kernels: Explicit randomized feature map
  [Rahimi and Recht 2007, Lu et al. 2016]
  - Shallow neural network by random Fourier/Binning transformation:
    - Random weights from input to hidden units
    - Cosine as transfer function
    - Convex optimization of weights from hidden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

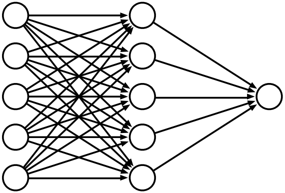
Perceptron as Single-Unit Neural Network



- New notation:
  - input vector: $\mathbf{x} \in \mathbb{R}^{d_{in}}$
  - weight matrix: $\mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}$
  - linear model: $\mathbf{y} = \mathbf{x}\mathbf{W}$
- Example: $d_{in} = 5,\ d_{out} = 1,\ y = \sum_{i=1}^{5} x_i w_i$

Q&A: We are implicitly assuming that $\mathbf{x}$ is a row vector. How would a perceptron look like if we assumed that $\mathbf{x}$ is a column vector?

Multilayer Perceptron (MLP)

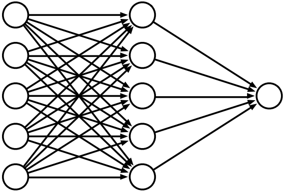


- Multilayer Perceptron for 1 hidden layer:

$$\mathbf{h} = f(\mathbf{x}\mathbf{W}^{(1)}),$$
$$\mathbf{y} = g(\mathbf{h}\mathbf{W}^{(2)})$$

  - input vector: $\mathbf{x} \in \mathbb{R}^{d_{in}}$
  - weights between input and hidden layer: $\mathbf{W}^{(1)} \in \mathbb{R}^{d_{in} \times d_1}$
  - weights between hidden layer and output: $\mathbf{W}^{(2)} \in \mathbb{R}^{d_1 \times d_2}$
  - non-linear functions f and g, applied elementwise

Multilayer Perceptron (MLP)



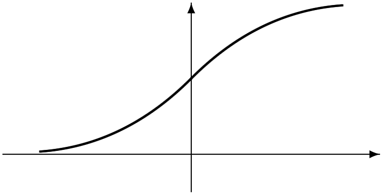
- Example:
  - $d_{in} = 5, d_1 = 5,\ d_2 = d_{out} = 1,$
  - $y_k = g(\sum_{j=1}^{5} h_j w_{kj}^{(2)}),$
  - $h_j = f(\sum_{i=1}^{5} x_i w_{ji}^{(1)}).$

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 $\mathbf{h}$
  - 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 $+1$

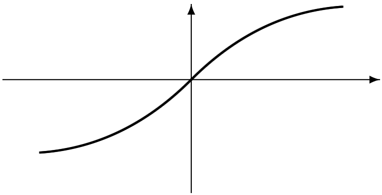


Non-linear Activation Functions

Hyperbolic tangent

$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

output ranges from –1 to $+1$

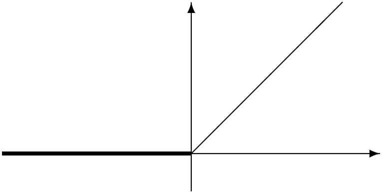


Non-linear Activation Functions

Rectified Linear

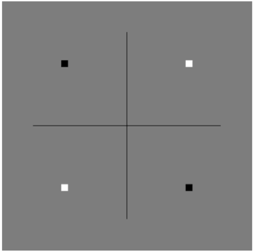
$\text{relu}(x) = \max(0,x)$

output ranges from 0 to ∞



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.



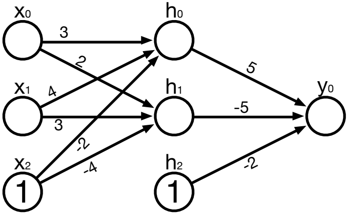
-1 XOR -1 = *false*

-1 XOR +1 = *true*

+1 XOR -1 = *true*

+1 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 $x_0 = 1,\ x_1 = 0$:

$$\begin{aligned} h_0 &= \sigma(\sum_i x_i w_{0i}) \\ &= \sigma\left(1 \times 3 + 0 \times 4 + 1 \times -2\right) \\ &= 0.73 \end{aligned}$$

Example: XOR

Input x_0	Input x_1	Hidden h_0	Hidden h_1	Output y_0
0	0	0.12	0.02	$0.18 \rightarrow 0$
0	1	0.88	0.27	$0.74 \rightarrow 1$
1	0	0.73	0.12	$0.74 \rightarrow 1$
1	1	0.99	0.73	$0.33 \rightarrow 0$

- 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 ...)

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: $\frac{dE}{dw_{i \leftarrow j}} = \frac{dE}{dy_i} \frac{dy_i}{ds_i} \frac{ds_i}{dw_{i \leftarrow j}}$
    - $\frac{dE}{dy_i} = \frac{d}{dy_i} \frac{1}{2}(t_i - y_i)^2 = -(t_i - y_i)$
    - $\frac{dy_i}{ds_i} = \frac{d\ \sigma(s_i)}{ds_i} = \sigma(s_i)(1 - \sigma(s_i)) = y_i(1 - y_i) := y_i'$
    - $\frac{ds}{dw_{i \leftarrow j}} = \frac{d}{dw_{i \leftarrow j}} \sum_j w_{i \leftarrow j} h_j = h_j$
    - Alltogether $\frac{dE}{dw_{i \leftarrow j}} = \frac{dE}{dy_i} \frac{dy_i}{ds_i} \frac{ds}{dw_{i \leftarrow j}} = -(t_i - y_i) \quad y_i' \quad h_j$
- 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 μ 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: $\frac{dE}{dh_j} = \sum_i \frac{dE}{dy_i}\frac{dy_i}{ds_i}\frac{ds_i}{dh_j}$
    - $\frac{dE}{dy_i}\frac{dy_i}{ds_i} = -(t_i - y_i)\; y_i' = \delta_i$, $\frac{ds_i}{dh_j} = \frac{d}{dh_j}\sum_i w_{i\leftarrow j} h_j = w_{i\leftarrow j}$
    - Alltogether: $\frac{dE}{dh_j} = \sum_i \delta_i w_{i\leftarrow j}$
  - $\frac{dh_j}{dz_j} = \frac{d\ \sigma(z_j)}{dz_j} = \sigma(z_j)(1-\sigma(z_j)) = h_j(1-h_j) = h_j'$
  - $\frac{dz_j}{dw_{j\leftarrow k}} = \frac{d}{dw_{j\leftarrow k}}\sum_k w_{j\leftarrow k} x_k = x_k$
  - Alltogether: $\frac{dE}{dw_{j\leftarrow k}} = \frac{dE}{dh_j}\frac{dh_j}{dz_j}\frac{dz_j}{dw_{j\leftarrow k}} = \sum_i (\delta_i w_{i\leftarrow j})\;\; h_j'\;\; x_k$
- Weight update: $\Delta w_{j\leftarrow k} = \mu\ \delta_j\ x_k$ where $\delta_j = \sum_i (\delta_i w_{i\leftarrow j})\;\; h_j'$

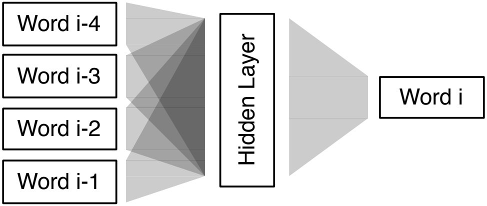
Backpropagation

- Error at output node compared to target: $\delta_i = (t_i - y_i)\ y'_i$
- Error at hidden nodes by backpropagating error term δ_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

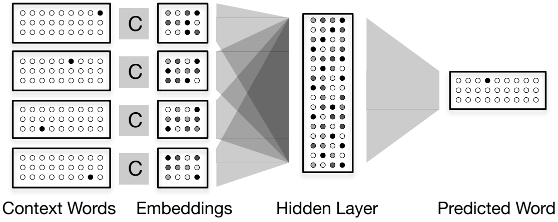
- Task-dependent network architecture:
  - MLP for regression: $d_{out} = 1$
  - MLP for binary classification: $d_{out} = 2$
  - MLP for k-fold multiclass classification: $d_{out} = k$
- Task-dependent loss functions:
  - Squared error for regression, hinge loss for multiclass classification
- Optimization 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 $\mathbf{x}_i$)
- 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 $\mathbf{C}$ as part of application
- OR: Train C separately, lookup embedding vector by multiplying $\mathbf{x}_i\mathbf{C}$, concatenate embeddings into input vector $\mathbf{x}$
- 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 $\mathbf{x}$ 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 $\mathbf{x}$ and one-hot output vector $\mathbf{y}$, optimize negative log-likelihood: $L(\mathbf{W}) = -\sum_k y_k \log p_k$
- Stochastic gradient: $\frac{dL}{d\mathbf{W}} = (\mathbf{p} - \mathbf{y})\mathbf{h}^\top$
- Weight update: $\Delta w_{i \leftarrow j} = \mu \left(p_i - y_i\right) h_j$

Recurrent Neural Networks (RNN)

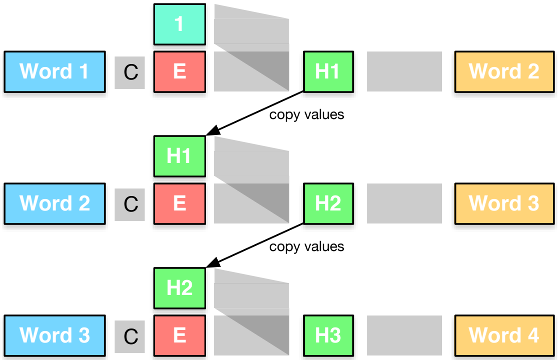
- Problem with MLP Language Model: Fixed context size
- RNNs can use unlimited context by recurrent definition $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$ where hidden layer of previous word is reused:

$$\begin{aligned}\mathbf{h}_t &= f(\mathbf{x}_t, \mathbf{h}_{t-1}) \\ &= \sigma(\mathbf{x}_t \mathbf{W}^{(\mathbf{x1})} + \mathbf{h}_{t-1}\mathbf{W}^{(\mathbf{h1})}), \\ \mathbf{y}_t &= \text{softmax}(\mathbf{h}_t \mathbf{W}^{(\mathbf{h2})}).\end{aligned}$$

  - $\mathbf{x}_t \in \mathbb{R}^{d_x}$, $\mathbf{h}_t \in \mathbb{R}^{d_h}$, $\mathbf{y}_t \in \mathbb{R}^{d_y}$,
  - $\mathbf{W}^{(\mathbf{x1})} \in \mathbb{R}^{d_x \times d_h}$,
  - $\mathbf{W}^{(\mathbf{h1})} \in \mathbb{R}^{d_h \times d_h}$,
  - $\mathbf{W}^{(\mathbf{h2})} \in \mathbb{R}^{d_h \times d_y}$.
- Note: Columns of $\mathbf{W}^{(\mathbf{x1})}$ can also be used as word embeddings

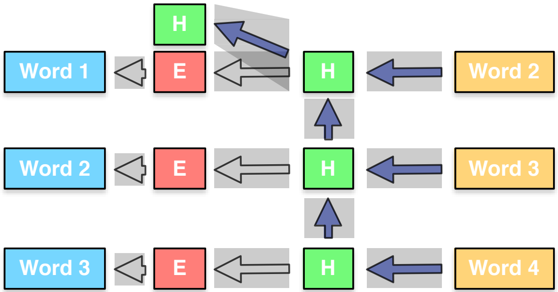
Q&A: Unfold the RNN definition recursively over time.

RNN Language Model



- Capture long term dependencies by copying contexts over time

Training RNNs

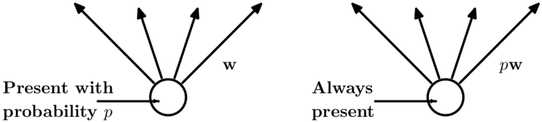


- Truncated back-propagation through time by unfolding network for a fixed number of words in context

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. $\log\ Z \simeq 0$ leads to $Z \simeq 1$
  - **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

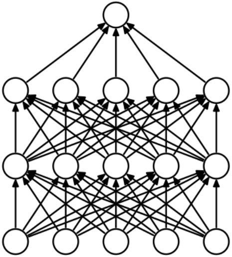
Refinement: Regularization by Dropout



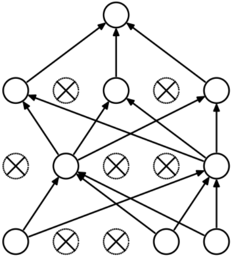
(a) At training time (b) At test time

- For each training example, drop out hidden units with probability $1 - p$
- At test time, keep all units and multiply outgoing weights by p
- $\rightarrow$ ensures that output equals expected output under distribution used to drop out units during training

Refinement: Regularization by Dropout



(a) Standard Neural Net



(b) After applying 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)

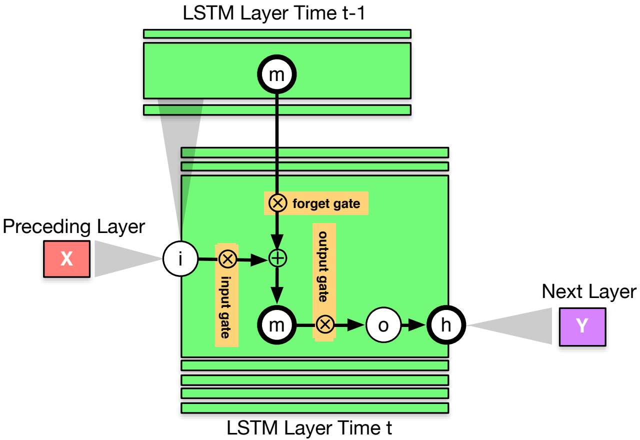
- LSTMs were designed to preserve gradients over time in **memory cells** which are accessed via **gates**
  - **input gates** regulate how much a new input changes the memory state,
  - **forget gates** regulate how much of the prior memory state is retained or forgotten,
  - **output gates** regulate 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 $\mathbf{a} \in [0,1]^n$ with a vector $\mathbf{b} \in \mathbb{R}^n$
  - components of $\mathbf{b}$ corresponding to near-one values in $\mathbf{a}$ may pass; those corresponding to near-zero values are blocked
- Memory update via **addition** (won't vanish in backprop)

Refinement: LSTM

- Similar recurrent definition $\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1})$ as RNNs, but including explicit memory component $\mathbf{m}$:

$$
\begin{aligned}
\mathbf{h}_t &= f(\mathbf{x}_t, \mathbf{h}_{t-1}) \\
&= \tanh(\mathbf{m}_t \otimes \mathbf{o}), \\
\mathbf{m}_t &= \mathbf{m}_{t-1} \otimes \mathbf{f} + \mathbf{g} \otimes \mathbf{i}, \\
\mathbf{i} &= \sigma(\mathbf{x}_t \mathbf{W}^{(\mathbf{xi})} + \mathbf{h}_{t-1} \mathbf{W}^{(\mathbf{hi})}), \\
\mathbf{f} &= \sigma(\mathbf{x}_t \mathbf{W}^{(\mathbf{xf})} + \mathbf{h}_{t-1} \mathbf{W}^{(\mathbf{hf})}), \\
\mathbf{o} &= \sigma(\mathbf{x}_t \mathbf{W}^{(\mathbf{xo})} + \mathbf{h}_{t-1} \mathbf{W}^{(\mathbf{ho})}), \\
\mathbf{g} &= \tanh(\mathbf{x}_t \mathbf{W}^{(\mathbf{xg})} + \mathbf{h}_{t-1} \mathbf{W}^{(\mathbf{hg})}), \\
& \mathbf{x}_t \in \mathbb{R}^{d_x}, \mathbf{m}_t, \mathbf{h}_t, \mathbf{i}, \mathbf{f}, \mathbf{o}, \mathbf{g} \in \mathbb{R}^{d_h}, \\
& \mathbf{W}^{(\mathbf{x}*)} \in \mathbb{R}^{d_x \times d_h}, \mathbf{W}^{(\mathbf{h}*)} \in \mathbb{R}^{d_h \times d_h}.
\end{aligned}
$$

Refinement: LSTM



RNN Encoder-Decoder for Statistical Machine Translation (SMT)

- Training data $D = \{(\mathbf{x}^i, \mathbf{y}^i)\}_{i=1}^N$ where
  - $\mathbf{x} = (x_1, x_2, \ldots, x_{T_x})$ is a sequence of source words,
  - $\mathbf{y} = (y_1, y_2, \ldots, y_{T_y})$ is a sequence of target words.
- **Conditional language model**:
  - $p(\mathbf{y}|\mathbf{x}) = \prod_{t=1}^{T_y} p(y_t|y_{<t}, \mathbf{x})$ where $y_{<t} = y_1, \ldots, y_{t-1}$
- Negative log-likelihood objective:
  - $-\frac{1}{N} \sum_{i=1}^N \sum_{t=1}^{T_y} \log p(y_t^i|y_{<t}^i, \mathbf{x}^i)$

Simple RNN Encoder-Decoder for SMT

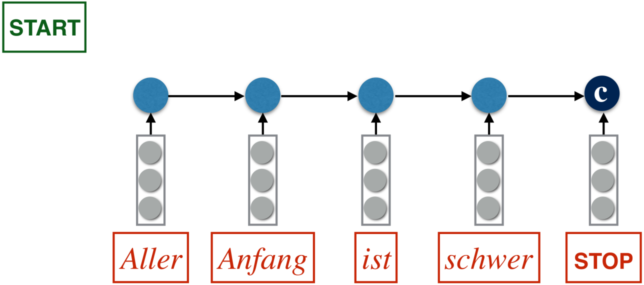
- RNN Encoder:
  - Map source-language input sentence into single context vector by using last memory state of RNN/LSTM:

$$\mathbf{h}_t = f(\mathbf{x}_t, \mathbf{h}_{t-1}),$$
$$\mathbf{c} = q(\mathbf{h_1}, \mathbf{h_2}, \ldots, \mathbf{h_{T_x}}) = \mathbf{h}_{T_x}.$$

- RNN Decoder:
  - Use RNN/LSTM to decode target language words by concatenating context vector **c** to hidden output state representation:

$$\mathbf{s}_t = f(\mathbf{y}_{t-1} \| \mathbf{c}, \mathbf{s}_{t-1}),$$
$$p(y_t | y_{<t}, \mathbf{x}) = \mathrm{softmax}(\mathbf{s}_t \mathbf{W}^{(\mathbf{h2})}).$$

Example: Translation with Simple RNN Encoder-Decoder



Refinement: Bi-directional RNN Encoder

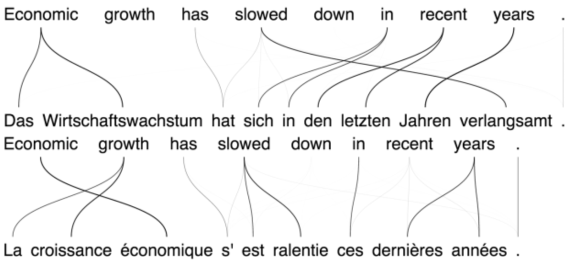
- Forward RNN reads input from x_1 to x_{T_x} and calculates the forward hidden state sequence $\overrightarrow{\mathbf{h}}_1, \ldots, \overrightarrow{\mathbf{h}}_{T_x}$ where $\overrightarrow{\mathbf{h}}_t = f(\mathbf{x}_t, \overrightarrow{\mathbf{h}}_{t-1})$,
- Backward RNN reads input from x_{T_x} to x_1 and calculates the backward hidden state sequence $\overleftarrow{\mathbf{h}}_1, \ldots, \overleftarrow{\mathbf{h}}_{T_x}$ where $\overleftarrow{\mathbf{h}}_t = f(\mathbf{x}_t, \overleftarrow{\mathbf{h}}_{t+1})$,
- Concatenate hidden states of forward and backward RNNs:

$$\mathbf{h}_t = \overleftarrow{\mathbf{h}}_t \parallel \overrightarrow{\mathbf{h}}_t,$$

Refinement: Attention-Based RNN Decoder

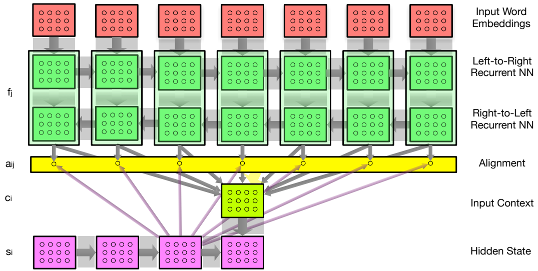
- Attention Mechanism:
  - Instead of encoding whole source sentence into $\mathbf{c}$, use weighted average of source context vectors $\mathbf{c}_i = \sum_{j=1}^{T_x} a_{ij}\mathbf{h}_j$,
  - **Attention weights** $a_{ij} = \frac{e^{e_{ij}}}{\sum_{j'=1}^{T} e^{e_{ij'}}}$ are computed by softmax over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

Attention Mechanism: Example



- 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 by 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:**

- J. Y. Lettvin, H. R. Maturana, W. S. McCulloch, and W. H. Pitts. 1959.
  What the frog's eye tells the frog's brain. *Proc. Inst. Radio Engr.*, 47:1940–1951.

- **Naive Bayes:**

- Pedro Domingos and Michael Pazzani. 1997.
  On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, (29):103–130.

- **Logistic Regression:**

- Bradley Efron. 1975.
  The efficiency of logistic regression compared to normal discriminant analysis. *Journal of the American Statistical Association*, 70(352):892–898.

- Adam L. Berger, Vincent J. Della Pietra, and Stephen A. Della Pietra. 1996.
  A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

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

Lexicalized Stochastic Modeling of Constraint-Based Grammars using Log-Linear Measures and EM Training. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics (ACL'00)*, Hong Kong.

- **Perceptron:**
- Albert B.J. Novikoff. 1962.
  On convergence proofs on perceptrons. *Symposium on the Mathematical Theory of Automata*, 12:615–622.
- Yoav Freund and Robert E. Schapire. 1999.
  Large margin classification using the perceptron algorithm. *Journal of Machine Learning Research*, 37:277–296.
- Michael Collins. 2002.
  Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms. In *Proceedings of the conference on Empirical Methods in Natural Language Processing (EMNLP'02)*, Philadelphia, PA.
- **SVM:**
- Vladimir N. Vapnik. 1998.
  Statistical Learning Theory. Wiley.
- Olivier Chapelle. 2007.

Training a support vector machine in the primal. *Neural Computation*, 19(5):1155–1178.

- Ben Taskar, Carlos Guestrin, and Daphne Koller. 2003.
  Max-margin markov networks. In *Advances in Neural Information Processing Systems 17 (NIPS'03)*, Vancouver, Canada.

- Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004.
  Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*, Banff, Canada.

- **Kernels and Regularization:**

- Bernhard Schölkopf and Alexander J. Smola. 2002.
  Learning with Kernels. Support Vector Machines, Regularization, Optimization, and Beyond. The MIT Press.

- Ali Rahimi and Ben Recht. 2007.
  Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*, Vancouver, B.C., Canada.

- Zhiyun Lu, Dong Guo, Alireza Bagheri Garakani, Kuan Liu, Avner May, Aurelien Bellet, Linxi Fan, Michael Collins, Brian Kingsbury, Michael Picheny, and Fei Sha. 2016.
  A comparison between deep neural nets and kernel acoustic models for speech recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*.

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

- Yurii Nesterov. 2004.
  Introductory lectures on convex optimization: A basic course. Springer.

- Stephen Boyd and Lieven Vandenberghe. 2004.
  Convex Optimization. Cambridge University Press.

- Dimitri P. Bertsekas and John N. Tsitsiklis. 1996.
  Neuro-Dynamic Programming. Athena Scientific.

- **Online/Stochastic Optimization:**

- Herbert Robbins and Sutton Monro. 1951.
  A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407.

- Boris T. Polyak. 1964.

Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1 – 17.

- Nicolò Cesa-Bianchi, Alex Conconi, and Claudio Gentile. 2004.
  On the generalization ablility of on-line learning algorithms. *IEEE Transactons on Information Theory*, 50(9):2050–2057.

- Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. 2013.
  On the importance of initialization and momentum in deep learning. In *Proceedings of International Conference on Machine Learning (ICML)*, pages 1139–1147.

- Leon Bottou, Frank E. Curtis, and Jorge Nocedal. 2016.
  Optimization methods for large-scale machine learning. *CoRR*, abs/arXiv:1606.04838v1.

- **Neural Networks:**

- David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. 1986.
  Learning representations by back-propagating errors. *Nature*, 323:533–536.

- Kurt Hornik, Maxwell Stinchcombe, and Halber White. 1989.
  Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366.

- Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. 2003.

A neural probabilistic language model. *Joural of Machine Learning Research*, 3:1137–1155.

- Ronan Collobert, Jason Weston, Leon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011.
  Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2461–2505.

- Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. 2010.
  Recurrent neural network based language model. In *Proceedings of Interspeech*, Makuhari, Chiba, Japan.

- Tomas Mikolov, Wen tau Yih, and Geoffrey Zweig. 2013.
  Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, Atlanta, Georgia.

- Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015.
  Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR)*, San Diego, CA.

Thanks