Learning with Structured Inputs and Outputs

Christoph H. Lampert
IST Austria (Institute of Science and Technology Austria), Vienna

ENS/INRIA Summer School, Paris, July 2013

IST AUSTRIA
Institute of Science and Technology

Slides: `http://www.ist.ac.at/~chl/`

Schedule

Monday	Introduction to Graphical Models
9:00-9:45	Conditional Random Fields
9:45-10:30	Structured Support Vector Machines

Slides available on my home page:

`http://www.ist.ac.at/~chl`

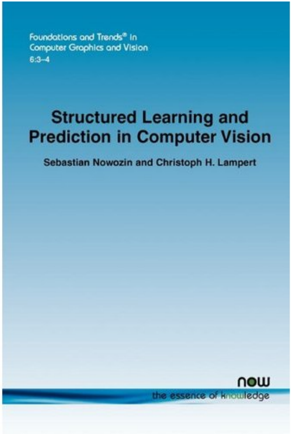
Extended version lecture in book form (180 pages)

Foundations and Trends in
Computer Graphics and Vision

now publisher

`http://www.nowpublishers.com/`

Available as PDF on
`http://pub.ist.ac.at/~chl/`



Standard Regression/Classification:

$$f : \mathcal{X} \to \mathbb{R}.$$

Structured Output Learning:

$$f : \mathcal{X} \to \mathcal{Y}.$$

Standard Regression/Classification:

$$f : \mathcal{X} \to \mathbb{R}.$$

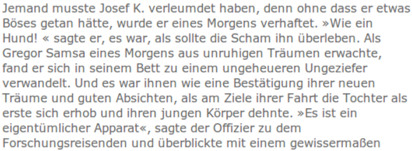
- inputs $\mathcal{X}$ can be any kind of objects
- output y is a real number

Structured Output Learning:

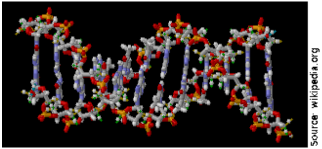
$$f : \mathcal{X} \to \mathcal{Y}.$$

- inputs $\mathcal{X}$ can be any kind of objects
- outputs $y \in \mathcal{Y}$ are complex (structured) objects

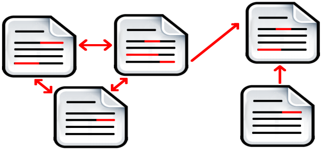
Ad hoc definition: data that consists of several parts, and not only the parts themselves contain information, but also the way in which the parts belong together.


Jemand musste Josef K. verleumdet haben, denn ohne dass er etwas Böses getan hätte, wurde er eines Morgens verhaftet. »Wie ein Hund! « sagte er, es war, als sollte die Scham ihn überleben. Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu einem ungeheueren Ungeziefer verwandelt. Und es war ihnen wie eine Bestätigung ihrer neuen Träume und guten Absichten, als am Ziele ihrer Fahrt die Tochter als erste sich erhob und ihren jungen Körper dehnte. »Es ist ein eigentümlicher Apparat«, sagte der Offizier zu dem Forschungsreisenden und überblickte mit einem gewissermaßen

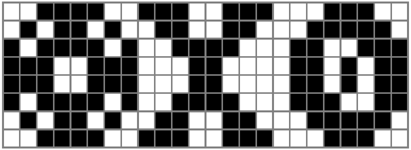
Text



Molecules / Chemical Structures



Documents/HyperText



Images

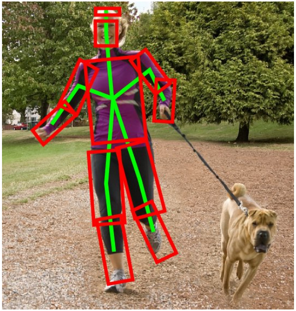
What is structured output prediction?

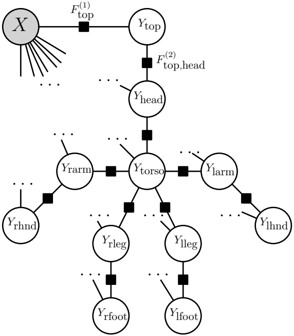
Ad hoc definition: predicting *structured* outputs from input data
(in contrast to predicting just a single number, like in classification or regression)

- Natural Language Processing:
  - Automatic Translation (output: sentences)
  - Sentence Parsing (output: parse trees)
- Bioinformatics:
  - Secondary Structure Prediction (output: bipartite graphs)
  - Enzyme Function Prediction (output: path in a tree)
- Speech Processing:
  - Automatic Transcription (output: sentences)
  - Text-to-Speech (output: audio signal)
- Robotics:
  - Planning (output: sequence of actions)

This tutorial: Applications and Examples from Computer Vision

Reminder: Graphical Model for Pose Estimation



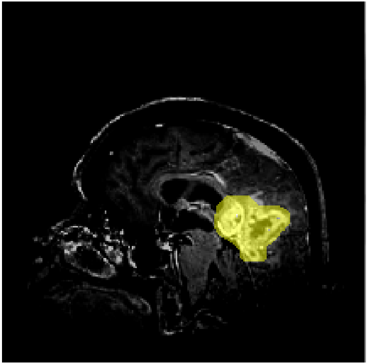


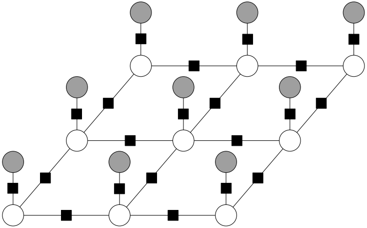
- Joint probability distribution of all body parts

$$p(y|x) = \frac{1}{Z(x)} \exp(-\sum_{F \in \mathcal{F}} E_F(y_F; x)).$$

Exponent ("energy") decomposes into small but interacting *factors*.

Reminder: Graphical Model for Image Segmentation





- Probability distribution over all foreground/background segmentations

$$p(y|x) = \frac{1}{Z(x)} \exp(-\sum_{F \in \mathcal{F}} E_F(y_F; x)).$$

Exponent ("energy") decomposes into small but interacting *factors*.

Monday: Probabilistic Inference

Compute *marginal probabilities*

$$p(y_F|x)$$

for any factor F, in particular, $p(y_i|x)$ for all $i \in V$.

Monday: MAP Prediction

Predict $f : \mathcal{X} \to \mathcal{Y}$ by solving

$$y^* = \operatorname*{argmax}_{y\in\mathcal{Y}} p(y|x) = \operatorname*{argmin}_{y\in\mathcal{Y}} E(y,x)$$

Today: Parameter Learning

Learn **learn potentials/energy terms** from training data.

Part 1: Conditional Random Fields

Supervised Learning Problem

- Given training examples $(x^1, y^1), \dots, (x^N, y^N) \in \mathcal{X} \times \mathcal{Y}$
- How to make predictions $g : \mathcal{X} \to \mathcal{Y}$?

Approach 1) Discrimitive Probabilistic Learning

1) Use training data to obtain an estimate $p(y|x)$.
2) Use $f(x) = \operatorname{argmin}_{\bar{y} \in \mathcal{Y}} \sum_y p(y|x) \Delta(y, \bar{y})$ to make predictions.

Approach 2) Loss-minimizing Parameter Estimation

1) Use training data to learn an energy function $E(x, y)$
2) Use $f(x) := \operatorname{argmin}_{y \in \mathcal{Y}} E(x, y)$ to make predictions.

Conditional Random Field Learning

Goal: learn a posterior distribution

$$p(y|x) = \frac{1}{Z(x)} \exp(\ -\sum_{F\in\mathcal{F}} E_F(y_F;x)\rangle \ ).$$

with $\mathcal{F} = \{$ all factors $\}$: all unary, pairwise, potentially higher order, ...

- parameterize each $E_F(y_F;x) = \langle w_F, \phi_F(x, y_F)\rangle$.
- fixed feature functions $(\ \phi_1(x_1, y), \dots, \phi_{|\mathcal{F}|}(x_F, y)\ ) \equiv: \phi(x,y)$
- weight vectors $(\ w_1, \dots, w_{|\mathcal{F}|}\ ) \equiv: w$

Result: log-linear model with parameter vector w

$$p(y|x;w) = \frac{1}{Z(x;w)} \exp(-\langle w, \phi(y,x)\rangle).$$

$$\text{with} \qquad Z(x;w) = \sum_{\bar{y}\in\mathcal{Y}} \exp(-\langle w, \phi(\bar{y},x)\rangle)$$

New goal: find best parameter vector $w \in \mathbb{R}^D$.

Maximum Likelihood Parameter Estimation

Idea 1: Maximize likelihood of outputs $y^1, \dots, y^N$ for inputs $x^1, \dots, x^N$

$$
\begin{aligned}
w^* &= \operatorname*{argmax}_{w \in \mathbb{R}^D} \; p(y^1, \dots, y^N | x^1, \dots, x^N, w) \\
&\overset{i.i.d.}{=} \operatorname*{argmax}_{w \in \mathbb{R}^D} \; \prod_{n=1}^{N} p(y^n | x^n, w) \\
&\overset{-\log(\cdot)}{=} \operatorname*{argmin}_{w \in \mathbb{R}^D} \; \underbrace{- \sum_{n=1}^{N} \log p(y^n | x^n, w)}_{\text{negative conditional log-likelihood (of } \mathcal{D}\text{)}}
\end{aligned}
$$

MAP Estimation of w

Idea 2: Treat w as random variable; maximize posterior $p(w|\mathcal{D})$

Idea 2: Treat w as random variable; maximize posterior $p(w|\mathcal{D})$

$$p(w|\mathcal{D}) \overset{\text{Bayes}}{=} \frac{p(x^1, y^1, \dots, x^N, y^N|w)p(w)}{p(\mathcal{D})} \overset{i.i.d.}{=} p(w) \prod_{n=1}^{N} \frac{p(y^n|x^n, w)}{p(y^n|x^n)}$$

$p(w)$: *prior belief* on w (cannot be estimated from data).

$$\begin{aligned} w^* &= \operatorname*{argmax}_{w \in \mathbb{R}^D} \; p(w|\mathcal{D}) = \operatorname*{argmin}_{w \in \mathbb{R}^D} \big[-\log p(w|\mathcal{D}) \big] \\ &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[-\log p(w) - \sum_{n=1}^{N} \log p(y^n|x^n, w) + \underbrace{\log p(y^n|x^n)}_{\text{indep. of } w} \Big] \\ &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[-\log p(w) - \sum_{n=1}^{N} \log p(y^n|x^n, w) \Big] \end{aligned}$$

$$w^* = \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[-\log p(w) - \sum_{n=1}^{N} \log p(y^n | x^n, w) \Big]$$

Choices for $p(w)$:

- $p(w) :\equiv$ const. (uniform; in $\mathbb{R}^D$ not really a distribution)

$$w^* = \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[\underbrace{-\sum_{n=1}^{N} \log p(y^n | x^n, w)}_{\text{negative conditional log-likelihood}} + \text{const.} \Big]$$

- $p(w) := \textit{const.} \cdot e^{-\frac{\lambda}{2}\|w\|^2}$ (Gaussian)

$$w^* = \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[\underbrace{\frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N} \log p(y^n | x^n, w)}_{\textbf{regularized}\text{ negative conditional log-likelihood}} + \text{const.} \Big]$$

Probabilistic Models for Structured Prediction - Summary

Negative (Regularized) Conditional Log-Likelihood (of $\mathcal{D}$)

$$\mathcal{L}(w) = \frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N} \left[\langle w, \phi(x^n, y^n)\rangle + \log \sum_{y \in \mathcal{Y}} e^{-\langle w, \phi(x^n, y)\rangle}\right]$$

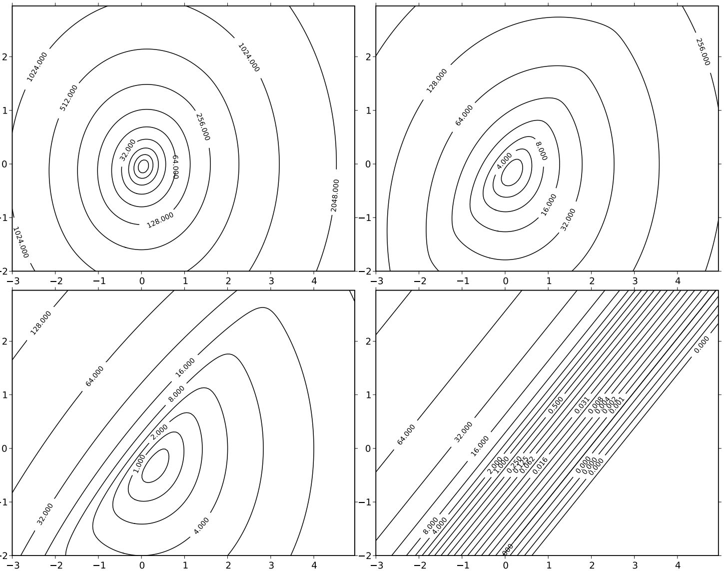
($\lambda \to 0$ makes it *unregularized*)

Probabilistic parameter estimation or *training* means solving

$$w^* = \operatorname*{argmin}_{w \in \mathbb{R}^D} \mathcal{L}(w).$$

Same optimization problem as for multi-class **logistic regression**.

Negative Conditional Log-Likelihood (Toy Example)



Steepest Descent Minimization – minimize $\mathcal{L}(w)$

input tolerance $\epsilon > 0$
1: $w_{cur} \leftarrow 0$
2: **repeat**
3: $\quad v \leftarrow \nabla_w \mathcal{L}(w_{cur})$
4: $\quad \eta \leftarrow \operatorname{argmin}_{\eta \in \mathbb{R}} \mathcal{L}(w_{cur} - \eta v)$
5: $\quad w_{cur} \leftarrow w_{cur} - \eta v$
6: **until** $\|v\| < \epsilon$
output w_{cur}

Alternatives:

- L-BFGS (second-order descent without explicit Hessian)
- Conjugate Gradient

We always need (at least) the gradient of $\mathcal{L}$.

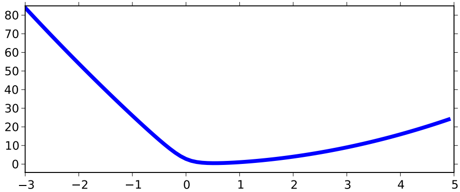
$$\mathcal{L}(w) = \frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N} \Big[\langle w, \phi(x^n, y^n)\rangle + \log \sum_{y\in\mathcal{Y}} e^{-\langle w, \phi(x^n, y)\rangle}\Big]$$

$$\begin{aligned}
\nabla_w \mathcal{L}(w) &= \lambda w + \sum_{n=1}^{N} \Big[\phi(x^n, y^n) - \frac{\sum_{y\in\mathcal{Y}} e^{-\langle w, \phi(x^n, y)\rangle} \phi(x^n, y)}{\sum_{\bar{y}\in\mathcal{Y}} e^{-\langle w, \phi(x^n, \bar{y})\rangle}}\Big] \\
&= \lambda w + \sum_{n=1}^{N} \Big[\phi(x^n, y^n) - \sum_{y\in\mathcal{Y}} p(y|x^n, w)\phi(x^n, y)\Big] \\
&= \lambda w + \sum_{n=1}^{N} \Big[\phi(x^n, y^n) - \mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n, y)\Big]
\end{aligned}$$

$$\Delta\mathcal{L}(w) = \lambda \mathit{Id}_{D\times D} + \sum_{n=1}^{N} \mathbb{E}_{y\sim p(y|x^n,w)}\Big\{\ \phi(x^n, y)\phi(x^n, y)^\top\Big\}$$

$$\mathcal{L}(w) = \frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N}\left[\langle w, \phi(x^n, y^n)\rangle + \log \sum_{y\in\mathcal{Y}} e^{-\langle w,\phi(x^n,y)\rangle}\right]$$

- continuous (not discrete), C^∞-differentiable on all $\mathbb{R}^D$.



slice through objective value ($w_x \in [-3, 5]$, $w_y = 0$)

$$\nabla_w \mathcal{L}(w) = \lambda w + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n,w)} \phi(x^n, y) \right]$$

- For $\lambda \to 0$:

$$\mathbb{E}_{y \sim p(y|x^n,w)} \phi(x^n, y) = \phi(x^n, y^n) \qquad \Rightarrow \quad \nabla_w \mathcal{L}(w) = 0,$$

criticial point of $\mathcal{L}$ (local minimum/maximum/saddle point).

Interpretation:

- We want the model distribution to match the empirical one:

$$\mathbb{E}_{y \sim p(y|x,w)} \phi(x, y) \stackrel{!}{=} \phi(x, y^{\text{obs}})$$

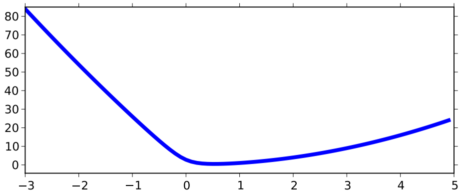
- E.g. Image Segmentation

  ϕ_{unary}: correct amount of foreground vs. background

  ϕ_{pairwise}: correct amount of fg/bg transitions $\to$ smoothness

$$\Delta\mathcal{L}(w) = \lambda Id_{D\times D} + \sum_{n=1}^{N} \mathbb{E}_{y\sim p(y|x^n,w)}\Big\{ \phi(x^n,y)\phi(x^n,y)^\top \Big\}$$

- positive definite Hessian matrix $\to \mathcal{L}(w)$ is *convex*
  $\to \nabla_w \mathcal{L}(w) = 0$ implies *global minimum*.



slice through objective value ($w_x \in [-3, 5]$, $w_y = 0$)

- $p(y|x, w)$ log-linear in $w \in \mathbb{R}^D$.
- Training: minimize negative conditional log-likelihood, $\mathcal{L}(w)$
- $\mathcal{L}(w)$ is differentiable and *convex*,
  $\rightarrow$ gradient descent will find global optimum with $\nabla_w \mathcal{L}(w) = 0$
- Same structure as multi-class *logistic regression*.

Milestone I: Probabilistic Training (Conditional Random Fields)

- $p(y|x, w)$ log-linear in $w \in \mathbb{R}^D$.
- Training: minimize negative conditional log-likelihood, $\mathcal{L}(w)$
- $\mathcal{L}(w)$ is differentiable and *convex*,
  $\rightarrow$ gradient descent will find global optimum with $\nabla_w \mathcal{L}(w) = 0$
- Same structure as multi-class *logistic regression*.

For logistic regression: this is where the textbook ends. We're done.

For conditional random fields: we're not in safe waters, yet!

Solving the Training Optimization Problem Numerically

Task: Compute $v = \nabla_w \mathcal{L}(w_{cur})$, evaluate $\mathcal{L}(w_{cur} + \eta v)$:

$$\mathcal{L}(w) = \frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N} \left[\langle w, \phi(x^n, y^n)\rangle + \log \sum_{y\in\mathcal{Y}} e^{-\langle w, \phi(x^n, y)\rangle}\right]$$

$$\nabla_w \mathcal{L}(w) = \frac{\lambda}{2} w + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \sum_{y\in\mathcal{Y}} p(y|x^n, w)\phi(x^n, y)\right]$$

Problem: $\mathcal{Y}$ typically is very (exponentially) large:

- binary image segmentation: $|\mathcal{Y}| = 2^{640\times 480} \approx 10^{92475}$
- ranking N images: $|\mathcal{Y}| = N!$, e.g. $N = 1000$: $|\mathcal{Y}| \approx 10^{2568}$.

We must use the **structure** in $\mathcal{Y}$, or we're lost.

Solving the Training Optimization Problem Numerically

$$\nabla_w \mathcal{L}(w) = \lambda w + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n, w)} \phi(x^n, y) \right]$$

Computing the Gradient (naive): $O(K^M N D)$

$$\mathcal{L}(w) = \frac{\lambda}{2} \|w\|^2 + \sum_{n=1}^{N} \left[\langle w, \phi(x^n, y^n) \rangle + \log Z(x^n, w) \right]$$

Line Search (naive): $O(K^M N D)$ per evaluation of $\mathcal{L}$

- N: number of samples
- D: dimension of feature space
- M: number of output nodes
- K: number of possible labels of each output nodes

Solving the Training Optimization Problem Numerically

$$\nabla_w \mathcal{L}(w) = \lambda w + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n, w)} \phi(x^n, y) \right]$$

Computing the Gradient (naive): $O(K^M ND)$

$$\mathcal{L}(w) = \frac{\lambda}{2} \|w\|^2 + \sum_{n=1}^{N} \left[\langle w, \phi(x^n, y^n) \rangle + \log Z(x^n, w) \right]$$

Line Search (naive): $O(K^M ND)$ per evaluation of $\mathcal{L}$

- N: number of samples
- D: dimension of feature space
- M: number of output nodes $\approx$ 100s to 1,000,000s
- K: number of possible labels of each output nodes $\approx$ 2 to 1000s

In a graphical model with factors $\mathcal{F}$, the features decompose:

$$\phi(x,y) = \Big(\phi_F(x,y_F)\Big)_{F\in\mathcal{F}}$$

$$\begin{aligned}\mathbb{E}_{y\sim p(y|x,w)}\phi(x,y) &= \Big(\mathbb{E}_{y\sim p(y|x,w)}\phi_F(x,y_F)\Big)_{F\in\mathcal{F}}\\ &= \big(\mathbb{E}_{y_F\sim p(y_F|x,w)}\phi_F(x,y_F)\big)_{F\in\mathcal{F}}\end{aligned}$$

$$\mathbb{E}_{y_F\sim p(y_F|x,w)}\phi_F(x,y_F) = \underbrace{\sum_{y_F\in\mathcal{Y}_F}}_{K^{|F|}\text{ terms}} \underbrace{p(y_F|x,w)}_{\text{factor marginals}}\ \phi_F(x,y_F)$$

Factor marginals $\mu_F = p(y_F|x,w)$

- are much smaller than complete joint distribution $p(y|x,w)$,
- can be computed/approximated, e.g., with *(loopy) belief propagation*.

$$\nabla_w \mathcal{L}(w) = \lambda w + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n, w)} \phi(x^n, y) \right]$$

Computing the Gradient: $\cancel{O(K^M nd)}$, $O(MK^{|F_{max}|}ND)$:

$$\mathcal{L}(w) = \frac{\lambda}{2} \|w\|^2 + \sum_{n=1}^{N} \left[\langle w, \phi(x^n, y^n) \rangle + \log \sum_{y \in \mathcal{Y}} e^{-\langle w, \phi(x^n, y) \rangle} \right]$$

Line Search: $\cancel{O(K^M nd)}$, $O(MK^{|F_{max}|}ND)$ per evaluation of $\mathcal{L}$

- N: number of samples
- D: dimension of feature space
- M: number of output nodes
- K: number of possible labels of each output nodes

Solving the Training Optimization Problem Numerically

$$\nabla_w \mathcal{L}(w) = \lambda w + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n, w)} \phi(x^n, y) \right]$$

Computing the Gradient: ~~$O(K^M nd)$~~, $O(MK^{|F_{max}|} N D)$:

$$\mathcal{L}(w) = \frac{\lambda}{2} \|w\|^2 + \sum_{n=1}^{N} \left[\langle w, \phi(x^n, y^n) \rangle + \log \sum_{y \in \mathcal{Y}} e^{-\langle w, \phi(x^n, y) \rangle} \right]$$

Line Search: ~~$O(K^M nd)$~~, $O(MK^{|F_{max}|} N D)$ per evaluation of $\mathcal{L}$

- N: number of samples $\approx$ 10s to 1,000,000s
- D: dimension of feature space
- M: number of output nodes
- K: number of possible labels of each output nodes

Solving the Training Optimization Problem Numerically

What, if the training set $\mathcal{D}$ is too large (e.g. millions of examples)?

Stochastic Gradient Descent (SGD)

- Minimize $\mathcal{L}(w)$, but without ever computing $\mathcal{L}(w)$ or $\nabla\mathcal{L}(w)$ exactly
- In each gradient descent step:
  - Pick random subset $\mathcal{D}' \subset \mathcal{D}$, $\leftarrow$ **often just 1–3 elements!**
  - Follow approximate gradient

$$\tilde{\nabla}\mathcal{L}(w) = \lambda w + \frac{|\mathcal{D}|}{|\mathcal{D}'|}\sum_{(x^n,y^n)\in\mathcal{D}'} \left[\phi(x^n,y^n) - \mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n,y)\right]$$

more: see L. Bottou, O. Bousquet: "*The Tradeoffs of Large Scale Learning*", NIPS 2008.
also: `http://leon.bottou.org/research/largescale`

Solving the Training Optimization Problem Numerically

What, if the training set $\mathcal{D}$ is too large (e.g. millions of examples)?

Stochastic Gradient Descent (SGD)

- Minimize $\mathcal{L}(w)$, but without ever computing $\mathcal{L}(w)$ or $\nabla\mathcal{L}(w)$ exactly
- In each gradient descent step:
  - Pick random subset $\mathcal{D}' \subset \mathcal{D}$, $\leftarrow$ **often just 1–3 elements!**
  - Follow approximate gradient

$$\tilde{\nabla}\mathcal{L}(w) = \lambda w + \frac{|\mathcal{D}|}{|\mathcal{D}'|}\sum_{(x^n,y^n)\in\mathcal{D}'} \left[\phi(x^n,y^n) - \mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n,y)\right]$$

- Avoid *line search* by using fixed stepsize rule η (new parameter)

more: see L. Bottou, O. Bousquet: "*The Tradeoffs of Large Scale Learning*", NIPS 2008.
also: `http://leon.bottou.org/research/largescale`

Solving the Training Optimization Problem Numerically

What, if the training set $\mathcal{D}$ is too large (e.g. millions of examples)?

Stochastic Gradient Descent (SGD)

- Minimize $\mathcal{L}(w)$, but without ever computing $\mathcal{L}(w)$ or $\nabla\mathcal{L}(w)$ exactly
- In each gradient descent step:
  - Pick random subset $\mathcal{D}' \subset \mathcal{D}$, $\leftarrow$ **often just 1–3 elements!**
  - Follow approximate gradient

$$\tilde{\nabla}\mathcal{L}(w) = \lambda w + \frac{|\mathcal{D}|}{|\mathcal{D}'|}\sum_{(x^n,y^n)\in\mathcal{D}'} \Big[\phi(x^n,y^n) - \mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n,y)\Big]$$

- Avoid *line search* by using fixed stepsize rule η (new parameter)
- SGD converges to $\operatorname{argmin}_w \mathcal{L}(w)$! (if η chosen right)

more: see L. Bottou, O. Bousquet: "*The Tradeoffs of Large Scale Learning*", NIPS 2008.
also: `http://leon.bottou.org/research/largescale`

Solving the Training Optimization Problem Numerically

What, if the training set $\mathcal{D}$ is too large (e.g. millions of examples)?

Stochastic Gradient Descent (SGD)

- Minimize $\mathcal{L}(w)$, but without ever computing $\mathcal{L}(w)$ or $\nabla\mathcal{L}(w)$ exactly
- In each gradient descent step:
  - Pick random subset $\mathcal{D}' \subset \mathcal{D}$, $\leftarrow$ **often just 1–3 elements!**
  - Follow approximate gradient

$$\tilde{\nabla}\mathcal{L}(w) = \lambda w + \frac{|\mathcal{D}|}{|\mathcal{D}'|}\sum_{(x^n,y^n)\in\mathcal{D}'} \Big[\phi(x^n,y^n) - \mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n,y)\Big]$$

- Avoid *line search* by using fixed stepsize rule η (new parameter)
- SGD converges to $\operatorname{argmin}_w \mathcal{L}(w)$! (if η chosen right)
- SGD needs more iterations, but each one is much faster

---

more: see L. Bottou, O. Bousquet: "*The Tradeoffs of Large Scale Learning*", NIPS 2008.
also: `http://leon.bottou.org/research/largescale`

$$\nabla_w \mathcal{L}(w) = \lambda w + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n, w)} \phi(x^n, y) \right]$$

Computing the Gradient: ~~$O(K^M nd)$~~, $O(MK^2ND)$ (if BP is possible):

$$\mathcal{L}(w) = \frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N} \left[\langle w, \phi(x^n, y^n) \rangle + \log \sum_{y \in \mathcal{Y}} e^{-\langle w, \phi(x^n, y) \rangle} \right]$$

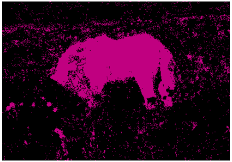
Line Search: ~~$O(K^M nd)$~~, $O(MK^2ND)$ per evaluation of $\mathcal{L}$

- N: number of samples
- D: dimension of feature space: $\approx \phi_{i,j}$ 1–10s, ϕ_i: 100s to 10000s
- M: number of output nodes
- K: number of possible labels of each output nodes

Typical feature functions in **image segmentation:**

- $\phi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: local image features, e.g. bag-of-words
  $\rightarrow \quad \langle w_i, \phi_i(y_i, x) \rangle$: local classifier (like logistic-regression)
- $\phi_{i,j}(y_i, y_j) = [\![y_i = y_j]\!] \in \mathbb{R}^1$: test for same label
  $\rightarrow \quad \langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle$: penalizer for label changes (if $w_{ij} > 0$)
- combined: $\operatorname{argmax}_y p(y|x)$ is smoothed version of local cues

original



local confidence

local + smoothness

Solving the Training Optimization Problem Numerically

Typical feature functions in **pose estimation:**

- $\phi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: local image representation, e.g. HoG
  $\rightarrow \quad \langle w_i, \phi_i(y_i, x) \rangle$: local confidence map

- $\phi_{i,j}(y_i, y_j) = \textit{good\_fit}(y_i, y_j) \in \mathbb{R}^1$: test for geometric fit
  $\rightarrow \quad \langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle$: penalizer for unrealistic poses

- together: $\operatorname{argmax}_y p(y|x)$ is sanitized version of local cues

original

local confidence

local + geometry

[V. Ferrari, M. Marin-Jimenez, A. Zisserman: "Progressive Search Space Reduction for Human Pose Estimation", CVPR 2008.]

Solving the Training Optimization Problem Numerically

Idea: split learning of unary potentials into two parts:

- local classifiers,
- their importance.

Two-Stage Training

- pre-train $f_i^y(x) \hat{=} \log p(y_i|x)$
- use $\tilde{\phi}_i(y_i, x) := f_i^y(x) \in \mathbb{R}^K$ (low-dimensional)
- keep $\phi_{ij}(y_i, y_j)$ are before
- perform CRF learning with $\tilde{\phi}_i$ and ϕ_{ij}

Solving the Training Optimization Problem Numerically

Idea: split learning of unary potentials into two parts:

- local classifiers,
- their importance.

Two-Stage Training

- pre-train $f_i^y(x) \hat{=} \log p(y_i|x)$
- use $\tilde{\phi}_i(y_i, x) := f_i^y(x) \in \mathbb{R}^K$ (low-dimensional)
- keep $\phi_{ij}(y_i, y_j)$ are before
- perform CRF learning with $\tilde{\phi}_i$ and ϕ_{ij}

Advantage:

- lower dimensional feature space during inference $\rightarrow$ faster
- $f_i^y(x)$ can be any classifiers, e.g. non-linear SVMs, deep network,...

Disadvantage:

- if local classifiers are bad, CRF training cannot fix that.

CRF training methods is based on gradient-descent optimization.
The faster we can do it, the better (more realistic) models we can use:

$$\tilde{\nabla}_w \mathcal{L}(w) = \lambda w - \sum_{n=1}^{N} \Big[\phi(x^n, y^n) - \sum_{y\in\mathcal{Y}} p(y|x^n, w)\, \phi(x^n, y)\Big] \quad \in \mathbb{R}^D$$

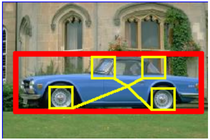
A lot of research on accelerating CRF training:

problem	"solution"	method(s)
$\lvert\mathcal{Y}\rvert$ too large	exploit structure	(loopy) belief propagation
	smart sampling	contrastive divergence
	use approximate $\mathcal{L}$	e.g. pseudo-likelihood
N too large	mini-batches	stochastic gradient descent
D too large	trained ϕ_{unary}	two-stage training

CRFs with Latent Variables

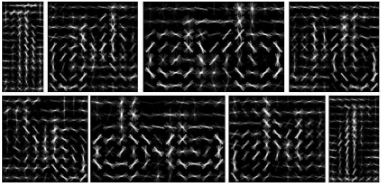
So far, training was *fully supervised*, all variables were observed.
In real life, some variables can be *unobserved* even during training.

missing labels in training data



latent variables, e.g. part location

latent variables, e.g. part occlusion



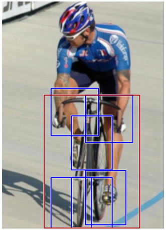
latent variables, e.g. viewpoint

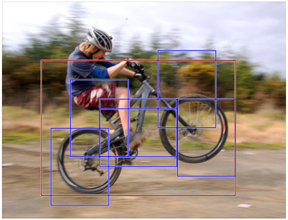
Three types of variables in graphical model:

- $x \in \mathcal{X}$ always observed (input),
- $y \in \mathcal{Y}$ observed only in training (output),
- $z \in \mathcal{Z}$ never observed (latent).

Example:

- x : image
- y : part positions
- $z \in \{0, 1\}$: flag *front-view* or *side-view*





images: [Felzenszwalb et al., "Object Detection with Discriminatively Trained Part Based Models", T-PAMI, 2010]

Marginalization over Latent Variables

Construct conditional likelihood as usual:

$$p(y, z|x, w) = \frac{1}{Z(x, w)} \exp(-\langle w, \phi(x, y, z)\rangle)$$

Derive $p(y|x, w)$ by marginalizing over z:

$$p(y|x, w) = \sum_{z \in \mathcal{Z}} p(y, z|x, w) \;=\; \frac{1}{Z(x, w)} \sum_{z \in \mathcal{Z}} \exp(-\langle w, \phi(x, y, z)\rangle)$$

Negative regularized conditional log-likelihood:

$$\begin{aligned}
\mathcal{L}(w) &= \frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N} \log p(y^n|x^n, w) \\
&= \frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N} \log \sum_{z\in\mathcal{Z}} p(y^n, z|x^n, w) \\
&= \frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N} \log \sum_{z\in\mathcal{Z}} \exp(-\langle w, \phi(x^n, y^n, z)\rangle) \\
&\qquad - \sum_{n=1}^{N} \log \sum_{\substack{z\in\mathcal{Z}\\ y\in\mathcal{Y}}} \exp(-\langle w, \phi(x^n, y, z)\rangle)
\end{aligned}$$

- $\mathcal{L}$ is *not convex* in w $\rightarrow$ local minima possible

How to train CRFs with latent variables is active research.

Summary – CRF Learning

Given:

- training set $\{(x^1, y^1), \dots, (x^N, y^N)\} \subset \mathcal{X} \times \mathcal{Y}$
- feature functions $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$
  that decomposes over factors, $\phi_F : \mathcal{X} \times \mathcal{Y}_F \to \mathbb{R}^d$ for $F \in \mathcal{F}$

Overall model is *log-linear* (in parameter w)

$$p(y|x; w) \propto e^{-\langle w, \phi(x,y) \rangle}$$

CRF training requires minimizing *negative conditional log-likelihood*:

$$w^* = \operatorname*{argmin}_w \; \frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N} \left[\langle w, \phi(x^n, y^n) \rangle - \log \sum_{y \in \mathcal{Y}} e^{-\langle w, \phi(x^n, y) \rangle} \right]$$

- *convex* optimization problem $\to$ (stochastic) gradient descent works
- training needs repeated runs of *probabilistic inference*
- latent variables are possible, but make training non-convex

Part 2: Structured Support Vector Machines

Supervised Learning Problem

- Training examples $(x^1, y^1), \dots, (x^N, y^N) \in \mathcal{X} \times \mathcal{Y}$
- Loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.
- How to make predictions $g : \mathcal{X} \to \mathcal{Y}$?

Approach 2) Loss-minimizing Parameter Estimation

1) Use training data to learn an energy function $E(x, y)$
2) Use $f(x) := \operatorname{argmin}_{y \in \mathcal{Y}} E(x, y)$ to make predictions.

Slight variation (for historic reasons):

1) Learn a compatibility function $g(x, y)$ (think: "$g = -E$")
2) Use $f(x) := \operatorname{argmax}_{y \in \mathcal{Y}} g(x, y)$ to make predictions.

Loss-Minimizing Parameter Learning

- $\mathcal{D} = \{(x^1, y^1), \dots, (x^N, y^N)\}$ i.i.d. training set
- $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ be a feature function.
- $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function.

- Find a weight vector w^* that minimizes the *expected loss*

$$\mathbb{E}_{(x,y)} \Delta(y, f(x))$$

for $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle$.

Loss-Minimizing Parameter Learning

- $\mathcal{D} = \{(x^1, y^1), \dots, (x^N, y^N)\}$ i.i.d. training set
- $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ be a feature function.
- $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function.

- Find a weight vector w^* that minimizes the *expected loss*

$$\mathbb{E}_{(x,y)} \Delta(y, f(x))$$

for $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle$.

Advantage:

- We directly optimize for the quantity of interest: expected loss.
- No expensive-to-compute partition function Z will show up.

Disadvantage:

- We need to know the loss function already at training time.
- We can't use probabilistic reasoning to find w^*.

Task: for $f(x) = \operatorname{argmax}_{y\in\mathcal{Y}} \langle w, \phi(x,y)\rangle$

$$\min_{w\in\mathbb{R}^D} \quad \mathbb{E}_{(x,y)}\Delta(y, f(x))$$

Two major problems:

- data distribution is unknown $\to$ we can't compute $\mathbb{E}$
- $f : \mathcal{X} \to \mathcal{Y}$ has output in a discrete space
  $\to$ f is piecewise constant w.r.t. w
  $\to$ $\Delta(y, f(x))$ is discontinuous, piecewise constant w.r.t w

  we can't apply gradient-based optimization

Task: for $f(x) = \text{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x,y) \rangle$

$$\min_{w \in \mathbb{R}^D} \quad \mathbb{E}_{(x,y)} \Delta(y, f(x))$$

Problem 1:

- data distribution is unknown

Solution:

- Replace $\mathbb{E}_{(x,y) \sim d(x,y)} \big(\cdot \big)$ with *empirical estimate* $\frac{1}{N} \sum_{(x^n, y^n)} \big(\cdot \big)$
- To avoid overfitting: add a *regularizer*, e.g. $\frac{\lambda}{2} \|w\|^2$.

New task:

$$\min_{w \in \mathbb{R}^D} \quad \frac{\lambda}{2} \|w\|^2 + \frac{1}{N} \sum_{n=1}^{N} \Delta(\ y^n, f(x^n)\ ).$$

Reminder: Regularized Risk Minimization

Task: for $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x,y) \rangle$

$$\min_{w \in \mathbb{R}^D} \quad \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \Delta(\ y^n, f(x^n)\ ).$$

Problem:

- $\Delta(\ y^n, f(x^n)\ ) = \Delta(\ y, \operatorname{argmax}_y \langle w, \phi(x,y) \rangle\ )$ discontinuous w.r.t. w.

Solution:

- Replace $\Delta(y, y')$ with *well behaved* $\ell(x, y, w)$
- Typically: ℓ *upper bound* to Δ, *continuous* and *convex* w.r.t. w.

New task:

$$\min_{w \in \mathbb{R}^D} \quad \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ell(x^n, y^n, w))$$

Reminder: Regularized Risk Minimization

$$\min_{w \in \mathbb{R}^D} \quad \frac{\lambda}{2}\|w\|^2 \quad + \quad \frac{1}{N}\sum_{n=1}^{N} \ell(x^n, y^n, w))$$

Regularization + *Loss on training data*

Reminder: Regularized Risk Minimization

$$\min_{w \in \mathbb{R}^D} \qquad \frac{\lambda}{2}\|w\|^2 \quad + \quad \frac{1}{N}\sum_{n=1}^{N} \ell(x^n, y^n, w))$$

Regularization + *Loss on training data*

Hinge loss: maximum margin training

$$\ell(x^n, y^n, w) := \max_{y \in \mathcal{Y}} \left[\ \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\ \right]$$

Reminder: Regularized Risk Minimization

$$\min_{w \in \mathbb{R}^D} \qquad \frac{\lambda}{2}\|w\|^2 \quad + \quad \frac{1}{N}\sum_{n=1}^{N} \ell(x^n, y^n, w))$$

Regularization + *Loss on training data*

Hinge loss: maximum margin training

$$\ell(x^n, y^n, w) := \max_{y \in \mathcal{Y}} \left[\ \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle \ \right]$$

- ℓ is maximum over linear functions $\rightarrow$ *continuous*, *convex*.
- ℓ is an upper bound to Δ: "small $\ell \Rightarrow$ small Δ"

Reminder: Regularized Risk Minimization

$$\min_{w \in \mathbb{R}^D} \qquad \frac{\lambda}{2}\|w\|^2 \quad + \quad \frac{1}{N}\sum_{n=1}^{N} \ell(x^n, y^n, w))$$

Regularization + *Loss on training data*

Hinge loss: maximum margin training

$$\ell(x^n, y^n, w) := \max_{y \in \mathcal{Y}} \left[\; \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle \; \right]$$

Alternative:

Logistic loss: probabilistic training

$$\ell(x^n, y^n, w) := \log \sum_{y \in \mathcal{Y}} \exp\big(\langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\big)$$

Differentiable, convex, not an upper bound to $\Delta(y, y')$.

Structured Output Support Vector Machine

$$\min_{w} \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \max_{y\in\mathcal{Y}} \left[\Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\right]$$

Conditional Random Field

$$\min_{w} \frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N} \underbrace{\log\sum_{y\in\mathcal{Y}} \exp\left(\langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\right)}_{= -\langle w,\phi(x^n,y^n)\rangle + \exp(\langle w,\phi(x^n,y)\rangle) \quad = \text{cond.log.likelihood}}$$

CRFs and SSVMs have more in common than usually assumed.

- $\log\sum_y \exp(\cdot)$ can be interpreted as a *soft-max*
- but: CRF doesn't take loss function into account at training time

Example: Multiclass SVM

- $\mathcal{Y} = \{1, 2, \dots, K\}, \quad \Delta(y, y') = \begin{cases} 1 & \text{for } y \neq y' \\ 0 & \text{otherwise} \end{cases}.$
- $\phi(x, y) = \Big([\![y = 1]\!]\phi(x),\ [\![y = 2]\!]\phi(x),\ \dots,\ [\![y = K]\!]\phi(x) \Big)$

Solve:

$$\min_{w}\ \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \max_{y \in \mathcal{Y}}\ \Big[\Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle \Big]$$

Classification: $\quad f(x) = \operatorname{argmax}_{y \in \mathcal{Y}}\ \langle w, \phi(x, y)\rangle.$

Crammer-Singer Multiclass SVM

[K. Crammer, Y. Singer: "On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines", JMLR, 2001]

Example: Multiclass SVM

- $\mathcal{Y} = \{1, 2, \ldots, K\}, \quad \Delta(y, y') = \begin{cases} 1 & \text{for } y \neq y' \\ 0 & \text{otherwise} \end{cases}.$
- $\phi(x, y) = \Big([\![y = 1]\!]\phi(x),\ [\![y = 2]\!]\phi(x),\ \ldots,\ [\![y = K]\!]\phi(x) \Big)$

Solve:

$$\min_{w} \ \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \max_{y \in \mathcal{Y}} \ \underbrace{\Big[\Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\Big]}_{= \begin{cases} 0 & \text{for } y = y^n \\ 1 + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle & \text{for } y \neq y^n \end{cases}}$$

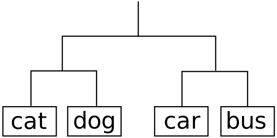
Classification: $\quad f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \ \langle w, \phi(x, y)\rangle.$

Crammer-Singer Multiclass SVM

[K. Crammer, Y. Singer: "On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines", JMLR, 2001]

Hierarchical Multiclass Loss:

$$\Delta(y, y') := \frac{1}{2}(\text{distance in tree})$$
$$\Delta(\texttt{cat}, \texttt{cat}) = 0, \quad \Delta(\texttt{cat}, \texttt{dog}) = 1,$$
$$\Delta(\texttt{cat}, \texttt{bus}) = 2, \quad etc.$$



$$\min_w \; \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \max_{y\in\mathcal{Y}} \; \Big[\Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\Big]$$

[L. Cai, T. Hofmann: "Hierarchical Document Categorization with Support Vector Machines", ACM CIKM, 2004]
[A. Binder, K.-R. Müller, M. Kawanabe: "On taxonomies for multi-class image categorization", IJCV, 2011]

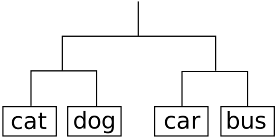
Example: Hierarchical Multiclass SVM

Hierarchical Multiclass Loss:

$$\Delta(y, y') := \frac{1}{2}(\text{distance in tree})$$

$$\Delta(\texttt{cat}, \texttt{cat}) = 0, \quad \Delta(\texttt{cat}, \texttt{dog}) = 1,$$

$$\Delta(\texttt{cat}, \texttt{bus}) = 2, \quad etc.$$



$$\min_w \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \max_{y\in\mathcal{Y}} \underbrace{\Big[\Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\Big]}$$

e.g. if $y^n = \texttt{cat}$, $\begin{cases} \langle w, \phi(x^n, \texttt{cat})\rangle - \langle w, \phi(x^n, \texttt{dog})\rangle \overset{!}{\geq} 1 \\ \langle w, \phi(x^n, \texttt{cat})\rangle - \langle w, \phi(x^n, \texttt{car})\rangle \overset{!}{\geq} 2 \\ \langle w, \phi(x^n, \texttt{cat})\rangle - \langle w, \phi(x^n, \texttt{bus})\rangle \overset{!}{\geq} 2. \end{cases}$

- labels that cause more loss are pushed further away
  $\rightarrow$ lower chance of high-loss mistake at test time

[L. Cai, T. Hofmann: "Hierarchical Document Categorization with Support Vector Machines", ACM CIKM, 2004]
[A. Binder, K.-R. Müller, M. Kawanabe: "On taxonomies for multi-class image categorization", IJCV, 2011]

We can solve SSVM training like CRF training:

$$\min_{w} \; \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N}\Big[\max_{y\in\mathcal{Y}} \; \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\Big]$$

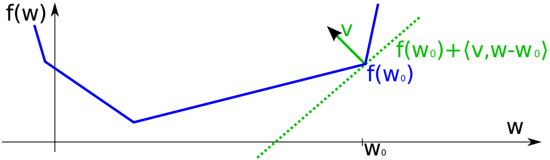
- continuous ☺
- unconstrained ☺
- convex ☺
- non-differentiable ☹
  $\rightarrow$ we can't use gradient descent directly.
  $\rightarrow$ we'll have to use **subgradients**

Solving S-SVM Training Numerically – Subgradient Method

Definition

Let $f : \mathbb{R}^D \to \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** of f at w_0, if

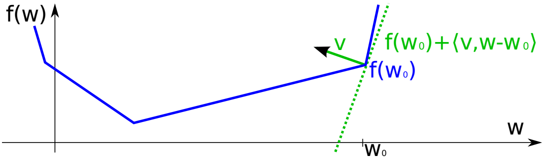
$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



Definition

Let $f : \mathbb{R}^D \to \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** of f at w_0, if

$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$

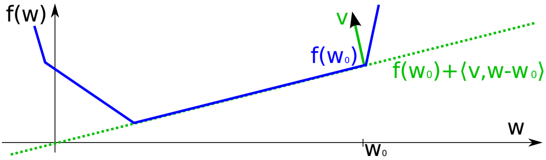


Solving S-SVM Training Numerically – Subgradient Method

Definition

Let $f : \mathbb{R}^D \to \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** of f at w_0, if

$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$

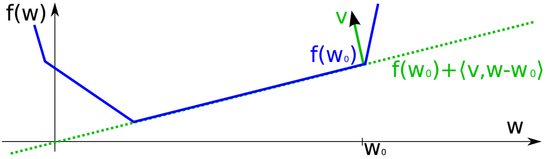


Solving S-SVM Training Numerically – Subgradient Method

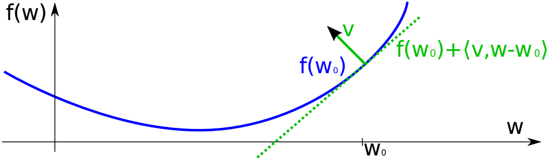
Definition

Let $f : \mathbb{R}^D \to \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** of f at w_0, if

$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



For differentiable f, the gradient $v = \nabla f(w_0)$ is the only subgradient.



Subgradient method works basically like gradient descent:

Subgradient Method Minimization – minimize $F(w)$

- **require:** tolerance $\epsilon > 0$, stepsizes η_t
- $w_{cur} \leftarrow 0$
- **repeat**
  - $v \in \nabla_w^{\text{sub}} F(w_{cur})$
  - $w_{cur} \leftarrow w_{cur} - \eta_t v$
- **until** F changed less than ϵ
- **return** w_{cur}

Converges to global minimum, but rather inefficient if F non-differentiable.

[Shor, "Minimization methods for non-differentiable functions", Springer, 1985.]

Computing a subgradient:

$$\min_{w} \; \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

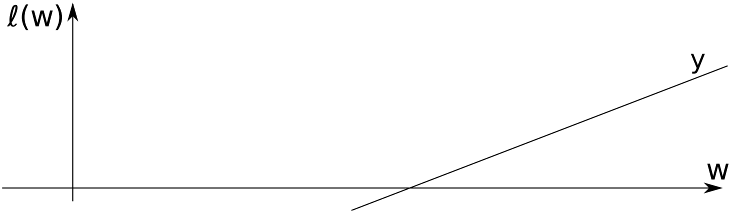
$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$

Computing a subgradient:

$$\min_{w} \; \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



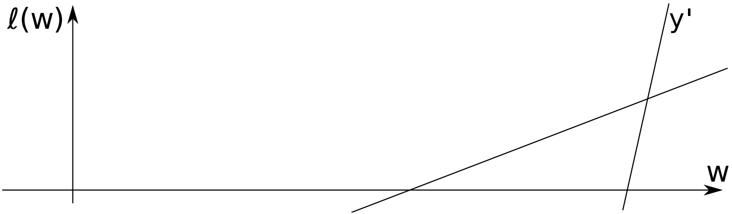
For each $y \in \mathcal{Y}$, $\ell^n_y(w)$ is a linear function of w.

Computing a subgradient:

$$\min_w \; \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



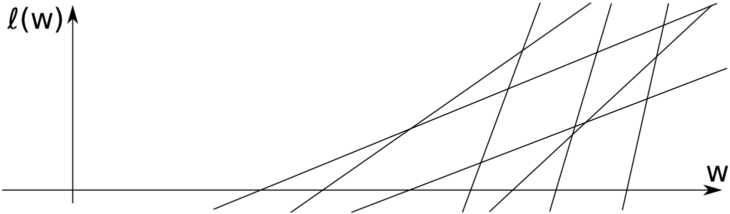
For each $y \in \mathcal{Y}$, $\ell^n_y(w)$ is a linear function of w.

Computing a subgradient:

$$\min_{w} \; \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



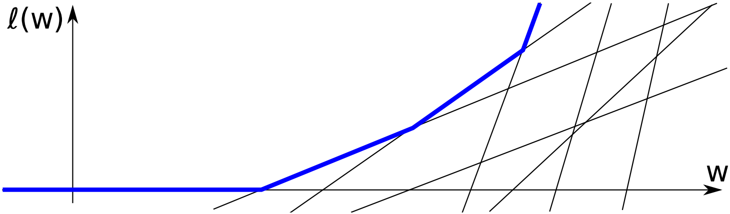
For each $y \in \mathcal{Y}$, $\ell^n_y(w)$ is a linear function of w.

Computing a subgradient:

$$\min_{w} \ \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



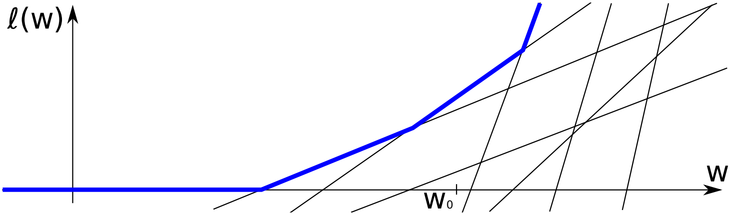
$\max$ over finite $\mathcal{Y}$: piece-wise linear

Computing a subgradient:

$$\min_w \; \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



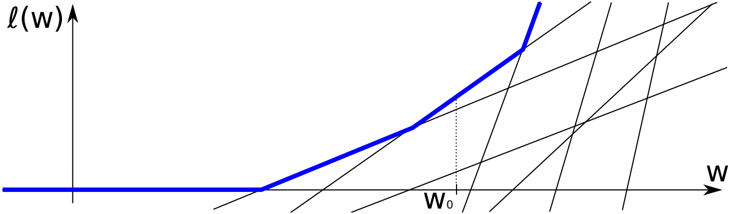
Subgradient of ℓ^n at w_0:

Computing a subgradient:

$$\min_w \; \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



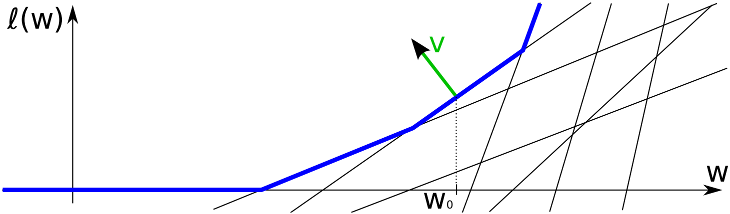
Subgradient of ℓ^n at w_0: find maximal (active) y.

Computing a subgradient:

$$\min_w \ \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



Subgradient of ℓ^n at w_0: find maximal (active) y, use $v = \nabla \ell^n_y(w_0)$.

Subgradient Method S-SVM Training

input training pairs $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
input feature map $\phi(x, y)$, loss function $\Delta(y, y')$, regularizer λ,
input number of iterations T, stepsizes η_t for $t = 1, \dots, T$

$w \leftarrow \vec{0}$
for t=1,...,T **do**
  for i=1,...,n **do**
    $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \quad \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$
    $v^n \leftarrow \phi(x^n, \hat{y}) - \phi(x^n, y^n)$
  end for
  $w \leftarrow w - \eta_t(\lambda w - \frac{1}{N}\sum_n v^n)$
end for

output prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x, y)\rangle$.

Obs: each update of w needs N argmax-prediction (one per example).

Solving S-SVM Training Numerically – Subgradient Method

Same trick as for CRFs: **stochastic updates**:

Stochastic Subgradient Method S-SVM Training

input training pairs $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
input feature map $\phi(x, y)$, loss function $\Delta(y, y')$, regularizer λ,
input number of iterations T, stepsizes η_t for $t = 1, \dots, T$

$w \leftarrow \vec{0}$
for t=1,. . . ,T **do**
    $(x^n, y^n) \;\leftarrow$ randomly chosen training example pair
    $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$
    $w \leftarrow w - \eta_t(\lambda w - \frac{1}{N}[\phi(x^n, \hat{y}) - \phi(x^n, y^n)])$
end for

output prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x, y)\rangle$.

Observation: each update of w needs only 1 argmax-prediction
(but we'll need many iterations until convergence)

Example: Image Segmenatation

- $\mathcal{X}$ images, $\quad \mathcal{Y} = \{$ binary segmentation masks $\}$.
- Training example(s): $(x^n, y^n) = \Big($  ,  $\Big)$
- $\Delta(y, \bar{y}) = \sum_p [\![y_p \neq \bar{y}_p]\!]$ (Hamming loss)

Images: [Carreira, Li, Sminchisescu, "Object Recognition by Sequential Figure-Ground Ranking", IJCV 2010]

Example: Image Segmenatation

- $\mathcal{X}$ images, $\mathcal{Y} = \{$ binary segmentation masks $\}$.
- Training example(s): $(x^n, y^n) = \Bigg($  ,  $\Bigg)$
- $\Delta(y, \bar{y}) = \sum_p [\![y_p \neq \bar{y}_p]\!]$ (Hamming loss)

$t = 1$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $+$, white $+$, green $-$, blue $-$, gray $-$

Images: [Carreira, Li, Sminchisescu, "Object Recognition by Sequential Figure-Ground Ranking", IJCV 2010]

Example: Image Segmenatation

- $\mathcal{X}$ images, $\mathcal{Y} = \{$ binary segmentation masks $\}$.
- Training example(s): $(x^n, y^n) = \Bigg($  ,  $\Bigg)$
- $\Delta(y, \bar{y}) = \sum_p [\![y_p \neq \bar{y}_p]\!]$ (Hamming loss)

$t = 1$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $+$, white $+$, green $-$, blue $-$, gray $-$

$t = 2$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $+$, white $+$, green $=$, blue $=$, gray $-$

Images: [Carreira, Li, Sminchisescu, "Object Recognition by Sequential Figure-Ground Ranking", IJCV 2010]

Example: Image Segmenatation

- $\mathcal{X}$ images, $\mathcal{Y} = \{$ binary segmentation masks $\}$.
- Training example(s): $(x^n, y^n) = \Big($ ,  $\Big)$
- $\Delta(y, \bar{y}) = \sum_p [\![y_p \neq \bar{y}_p]\!]$ (Hamming loss)

$t = 1$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $+$, white $+$, green $-$, blue $-$, gray $-$

$t = 2$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $+$, white $+$, green $=$, blue $=$, gray $-$

$t = 3$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $=$, white $=$, green $-$, blue $-$, gray $-$

Images: [Carreira, Li, Sminchisescu, "Object Recognition by Sequential Figure-Ground Ranking", IJCV 2010]

Example: Image Segmenatation

- $\mathcal{X}$ images, $\mathcal{Y} = \{$ binary segmentation masks $\}$.
- Training example(s): $(x^n, y^n) = \Bigg($  ,  $\Bigg)$
- $\Delta(y, \bar{y}) = \sum_p [\![y_p \neq \bar{y}_p]\!]$ (Hamming loss)

$t = 1$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $+$, white $+$, green $-$, blue $-$, gray $-$

$t = 2$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $+$, white $+$, green $=$, blue $=$, gray $-$

$t = 3$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $=$, white $=$, green $-$, blue $-$, gray $-$

$t = 4$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $=$, white $=$, green $-$, blue $=$, gray $=$

Images: [Carreira, Li, Sminchisescu, "Object Recognition by Sequential Figure-Ground Ranking", IJCV 2010]

Example: Image Segmenatation

- $\mathcal{X}$ images, $\mathcal{Y} = \{$ binary segmentation masks $\}$.
- Training example(s): $(x^n, y^n) = \Big($  ,  $\Big)$
- $\Delta(y, \bar{y}) = \sum_p [\![y_p \neq \bar{y}_p]\!]$ (Hamming loss)

$t = 1$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $+$, white $+$, green $-$, blue $-$, gray $-$

$t = 2$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $+$, white $+$, green $=$, blue $=$, gray $-$

$t = 3$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $=$, white $=$, green $-$, blue $-$, gray $-$

$t = 4$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $=$, white $=$, green $-$, blue $=$, gray $=$

$t = 5$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $=$, white $=$, green $=$, blue $=$, gray $=$

Images: [Carreira, Li, Sminchisescu, "Object Recognition by Sequential Figure-Ground Ranking", IJCV 2010]

Example: Image Segmenatation

- $\mathcal{X}$ images, $\quad \mathcal{Y} = \{$ binary segmentation masks $\}$.
- Training example(s): $(x^n, y^n) = \Bigg($  ,  $\Bigg)$
- $\Delta(y, \bar{y}) = \sum_p [\![y_p \neq \bar{y}_p]\!]$ (Hamming loss)

$t = 1$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $+$, white $+$, green $-$, blue $-$, gray $-$

$t = 2$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $+$, white $+$, green $=$, blue $=$, gray $-$

$t = 3$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $=$, white $=$, green $-$, blue $-$, gray $-$

$t = 4$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $=$, white $=$, green $-$, blue $=$, gray $=$

$t = 5$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: black $=$, white $=$, green $=$, blue $=$, gray $=$

$t = 6, \ldots$: no more changes.

Images: [Carreira, Li, Sminchisescu, "Object Recognition by Sequential Figure-Ground Ranking", IJCV 2010]

Solving S-SVM Training Numerically

Structured Support Vector Machine:

$$\min_{w} \quad \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \max_{y\in\mathcal{Y}} \Big[\Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\Big]$$

Subgradient method converges slowly. Can we do better?

Structured Support Vector Machine:

$$\min_{w} \quad \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \max_{y\in\mathcal{Y}} \Big[\Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\Big]$$

Subgradient method converges slowly. Can we do better?

Remember from SVM:
We can use **inequalities** and **slack variables** to encode the loss.

Structured SVM (equivalent formulation):

Idea: *slack variables*

$$\min_{w,\xi} \quad \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \xi^n$$

subject to, for $n = 1, \ldots, N$,

$$\max_{y \in \mathcal{Y}} \Big[\Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle - \langle w, \phi(x^n, y^n) \rangle \Big] \leq \xi^n$$

Note: $\xi^n \geq 0$ automatic, because left hand side is non-negative.

Differentiable objective, convex, N *non-linear* contraints,

Structured SVM (also equivalent formulation):

Idea: expand $\max$-constraint into individual cases

$$\min_{w,\xi} \quad \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N}\xi^n$$

subject to, for $n = 1, \ldots, N$,

$$\Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle \leq \xi^n, \quad \text{for all } y \in \mathcal{Y}$$

Differentiable objective, convex, $N\,|\mathcal{Y}|$ linear constraints

Solve an S-SVM like a linear SVM:

$$\min_{w\in\mathbb{R}^D,\xi\in\mathbb{R}^n} \quad \frac{\lambda}{2}\|w\|^2 \; + \; \frac{1}{N}\sum_{n=1}^{N}\xi^n$$

subject to, for $i = 1, \dots n$,

$$\langle w, \phi(x^n, y^n)\rangle - \langle w, \phi(x^n, y)\rangle \geq \Delta(y^n, y) \; - \; \xi^n, \quad \text{for all } y \in \mathcal{Y}.$$

Introduce feature vectors $\delta\phi(x^n, y^n, y) := \phi(x^n, y^n) - \phi(x^n, y)$.

Solving S-SVM Training Numerically

Solve

$$\min_{w \in \mathbb{R}^D, \xi \in \mathbb{R}^n_+} \frac{\lambda}{2}\|w\|^2 \; + \; \frac{1}{N}\sum_{n=1}^{N} \xi^n$$

subject to, for $i = 1, \ldots n$, for all $y \in \mathcal{Y}$,

$$\langle w, \delta\phi(x^n, y^n, y)\rangle \geq \Delta(y^n, y) \; - \; \xi^n.$$

Same structure as an ordinary SVM!

- quadratic objective ☺
- linear constraints ☺

Solving S-SVM Training Numerically

Solve

$$\min_{w \in \mathbb{R}^D, \xi \in \mathbb{R}^n_+} \frac{\lambda}{2}\|w\|^2 \; + \; \frac{1}{N}\sum_{n=1}^{N} \xi^n$$

subject to, for $i = 1, \ldots n$, for all $y \in \mathcal{Y}$,

$$\langle w, \delta\phi(x^n, y^n, y)\rangle \geq \Delta(y^n, y) \; - \; \xi^n.$$

Same structure as an ordinary SVM!

- quadratic objective ☺
- linear constraints ☺

Question: Can we use an ordinary SVM/QP solver?

Solving S-SVM Training Numerically

Solve

$$\min_{w\in\mathbb{R}^D,\xi\in\mathbb{R}^n_+} \frac{\lambda}{2}\|w\|^2 \; + \; \frac{1}{N}\sum_{n=1}^{N}\xi^n$$

subject to, for $i = 1, \ldots n$, for all $y \in \mathcal{Y}$,

$$\langle w, \delta\phi(x^n, y^n, y)\rangle \geq \Delta(y^n, y) \; - \; \xi^n.$$

Same structure as an ordinary SVM!

- quadratic objective ☺
- linear constraints ☺

Question: Can we use an ordinary SVM/QP solver?

Answer: Almost! We could, if there weren't $N|\mathcal{Y}|$ constraints.

- E.g. 100 binary 16×16 images: 10^{79} constraints

Solving S-SVM Training Numerically – Working Set

Solution: working set training

- It's enough if we enforce the **active constraints**.
  The others will be fulfilled automatically.
- We don't know which ones are active for the optimal solution.
- But it's likely to be only a small number $\leftarrow$ can of course be formalized.

Keep a set of potentially active constraints and update it iteratively:

Solving S-SVM Training Numerically – Working Set

Solution: working set training

- It's enough if we enforce the **active constraints**.
  The others will be fulfilled automatically.
- We don't know which ones are active for the optimal solution.
- But it's likely to be only a small number ← can of course be formalized.

Keep a set of potentially active constraints and update it iteratively:

Solving S-SVM Training Numerically – Working Set

- Start with working set $S = \emptyset$ (no contraints)
- Repeat until convergence:
  - Solve S-SVM training problem with constraints from S
  - Check, if solution violates any of the *full* constraint set
    - if no: we found the optimal solution, *terminate*.
    - if yes: add most violated constraints to S, *iterate*.

Solving S-SVM Training Numerically – Working Set

Solution: working set training

- It's enough if we enforce the **active constraints**.
  The others will be fulfilled automatically.
- We don't know which ones are active for the optimal solution.
- But it's likely to be only a small number $\leftarrow$ can of course be formalized.

Keep a set of potentially active constraints and update it iteratively:

Solving S-SVM Training Numerically – Working Set

- Start with working set $S = \emptyset$ (no contraints)
- Repeat until convergence:
  - Solve S-SVM training problem with constraints from S
  - Check, if solution violates any of the *full* constraint set
    - if no: we found the optimal solution, *terminate*.
    - if yes: add most violated constraints to S, *iterate*.

Good *practical performance* and *theoretic guarantees*:

- polynomial time convergence ϵ-close to the global optimum

Working Set S-SVM Training

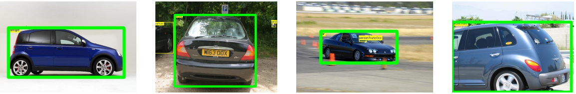
input training pairs $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
input feature map $\phi(x, y)$, loss function $\Delta(y, y')$, regularizer λ

$w \leftarrow 0$, $S \leftarrow \emptyset$
repeat
  $(w, \xi) \leftarrow$ *solution to QP only with constraints from* S
  for i=1,. . . ,n **do**
    $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \quad \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle$
    if $\hat{y} \neq y^n$ **then**
      $S \leftarrow S \cup \{(x^n, \hat{y})\}$
    end if
  end for
until S doesn't change anymore.

output prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle$.

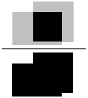
Obs: each update of w needs N argmax-predictions (one per example), but we solve globally for next w, not by local steps.

Example: Object Localization

- $\mathcal{X}$ images, $\mathcal{Y} = \{$ object bounding box $\} \subset \mathbb{R}^4$.
- Training examples: 
- Goal: $f : \mathcal{X} \to \mathcal{Y}$

$\mapsto$

- Loss function: area overlap $\Delta(y, y') = 1 - \frac{\text{area}(y \cap y')}{\text{area}(y \cup y')}$



[Blaschko, Lampert: "Learning to Localize Objects with Structured Output Regression", ECCV 2008]

Structured SVM:

- $\phi(x, y) :=$ "bag-of-words histogram of region y in image x"

$$\min_{w \in \mathbb{R}^D, \xi \in \mathbb{R}^n} \quad \frac{\lambda}{2}\|w\|^2 \;+\; \frac{1}{N}\sum_{n=1}^{N} \xi^n$$

subject to, for $i = 1, \ldots n$,

$$\langle w, \phi(x^n, y^n)\rangle - \langle w, \phi(x^n, y)\rangle \geq \Delta(y^n, y) \;-\; \xi^n, \quad \text{for all } y \in \mathcal{Y}.$$

Interpretation:

- For every image, the *correct* bounding box, y^n, should have a higher score than any *wrong* bounding box.
- Less overlap between the boxes $\rightarrow$ bigger difference in score

Working set training – Step 1:

- $w \leftarrow 0$.

For every example:

- $\hat{y} \leftarrow \text{argmax}_{y \in \mathcal{Y}} \quad \Delta(y^n, y) + \underbrace{\langle w, \phi(x^n, y) \rangle}_{=0}$

  maximal Δ-loss $\quad \equiv \quad$ minimal overlap with $y^n \quad \equiv \quad \hat{y} \cap y^n = \emptyset$

- add constraint

$$\langle w, \phi(x^n, y^n) \rangle - \langle w, \phi(x^n, \hat{y}) \rangle \geq 1 \; - \; \xi^n$$

Note: similar to binary SVM training for object detection:

- positive examples: ground truth bounding boxes
- negative examples: random boxes from 'image background'

Working set training – Later Steps:

For every example:

- $\hat{y} \leftarrow \text{argmax}_{y \in \mathcal{Y}} \quad \underbrace{\Delta(y^n, y)}_{\text{bias towards 'wrong' regions}} \quad + \quad \underbrace{\langle w, \phi(x^n, y) \rangle}_{\text{object detection score}}$
- if $\hat{y} = y^n$: do nothing,
  else: add constraint

$$\langle w, \phi(x^n, y^n) \rangle - \langle w, \phi(x^n, \hat{y}) \rangle \geq \Delta(y^n, \hat{y}) \; - \; \xi^n$$

  enforces $\hat{y}$ to have lower score after re-training.

Note: similar to hard negative mining for object detection:

- perform detection on training image
- if detected region is far from ground truth, add as negative example

Difference: S-SVM handles regions that overlap with ground truth.

Kernelized S-SVM

We can also **kernelize** S-SVM optimization:

$$\max_{\alpha \in \mathbb{R}_+^{N|\mathcal{Y}|}} \sum_{\substack{n=1,\dots,N \\ y \in \mathcal{Y}}} \alpha_{ny} \Delta(y^n, y) - \frac{1}{2} \sum_{\substack{y, \bar{y} \in \mathcal{Y} \\ n, \bar{n} = 1,\dots,N}} \alpha_{ny} \alpha_{\bar{n}\bar{y}} K_{n\bar{n}y\bar{y}}$$

subject to, for $n = 1, \dots, N$,

$$\sum_{y \in \mathcal{Y}} \alpha_{ny} \leq \frac{2}{\lambda N}.$$

$N|\mathcal{Y}|$ many variables: train with **working set** of α_{iy}.

Kernelized prediction function:

$$f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \sum_{n, y'} \alpha_{ny'} k(\, (x^n, y'), (x, y) \,)$$

Not very popular in Computer Vision (quickly becomes inefficient)

Latent variables also possible in S-SVMs

- $x \in \mathcal{X}$ always observed,
- $y \in \mathcal{Y}$ observed only in training,
- $z \in \mathcal{Z}$ never observed (latent).

Decision function: $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \max_{z \in \mathcal{Z}} \langle w, \phi(x, y, z) \rangle$

SSVMs with Latent Variables

Latent variables also possible in S-SVMs

- $x \in \mathcal{X}$ always observed,
- $y \in \mathcal{Y}$ observed only in training,
- $z \in \mathcal{Z}$ never observed (latent).

Decision function: $f(x) = \operatorname{argmax}_{y\in\mathcal{Y}} \max_{z\in\mathcal{Z}} \langle w, \phi(x,y,z)\rangle$

Maximum Margin Training with Maximization over Latent Variables

Solve: $$\min_{w,\xi} \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \max_{y\in\mathcal{Y}} \ \ell_w^n(y)$$

with

$$\ell_w^n(y) = \Delta(y^n, y) + \max_{z\in\mathcal{Z}} \langle w, \phi(x^n, y, z)\rangle - \max_{z\in\mathcal{Z}} \langle w, \phi(x^n, y^n, z)\rangle$$

Problem: not convex $\rightarrow$ can have local minima

[Yu, Joachims, "Learning Structural SVMs with Latent Variables", 2009]
similar: [Felzenszwalb *et al.*, "A Discriminatively Trained, Multiscale, Deformable Part Model", 2008], but $\mathcal{Y} = \{\pm 1\}$

Summary – S-SVM Learning

Given:

- training set $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$
- loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.
- parameterize $f(x) := \operatorname{argmax}_y \langle w, \phi(x, y) \rangle$

Task: find w that minimizes expected loss on future data, $\mathbb{E}_{(x,y)} \Delta(y, f(x))$

Summary – S-SVM Learning

Given:

- training set $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$
- loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.
- parameterize $f(x) := \operatorname{argmax}_y \langle w, \phi(x, y) \rangle$

Task: find w that minimizes expected loss on future data, $\mathbb{E}_{(x,y)} \Delta(y, f(x))$

S-SVM solution derived from *regularized risk minimization*:

- enforce correct output to be better than all others by a margin:

$$\langle w, \phi(x^n, y^n) \rangle \geq \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle \quad \text{for all } y \in \mathcal{Y}.$$

- convex optimization problem, but non-differentiable
- many equivalent formulations $\to$ different training algorithms
- training needs many argmax predictions, but no probabilistic inference

Latent variable possible, but optimization becomes non-convex.

Structured Learning is full of Open Research Questions

- How to train faster?
  - CRFs need many runs of probablistic inference,
  - SSVMs need many runs of argmax-predictions.
- How to reduce the necessary amount of training data?
  - semi-supervised learning? transfer learning?
- How can we better understand different loss function?
  - how important is it to optimize the "right" loss?
- Can we understand structured learning with approximate inference?
  - often computing $\nabla\mathcal{L}(w)$ or $\mathrm{argmax}_y\langle w, \phi(x,y)\rangle$ *exactly* is infeasible.
  - can we guarantee good results even with approximate inference?
- More and new applications!

More info: **www.ist.ac.at**

IST Austria Graduate School

- enter with MSc **or BSc**
- 1(2) + 3 yr PhD program
  - **Computer Vision/Machine Learning** (me, Vladimir Kolmogorov)
  - Computer Graphics (C. Wojtan)
  - Comp. Topology (H. Edelsbrunner)
  - Game Theory (K. Chatterjee)
  - Software Verification (T. Henzinger)
  - Cryptography (K. Pietrzak)
  - Comp. Neuroscience (G. Tkacik)
  - Random Matrix Theory (L. Erdös)
  - Statistics (C. Uhler), and more...
- fully funded positions

Postdoc Positions in my Group

- see `http://www.ist.ac.at/~chl`

Internships: send me an email!

Additional Material

One-Slack Formulation of S-SVM:
(equivalent to ordinary S-SVM formulation by $\xi = \frac{1}{N}\sum_n \xi^n$)

$$\min_{w\in\mathbb{R}^D, \xi\in\mathbb{R}_+} \quad \frac{\lambda}{2}\|w\|^2 + \xi$$

subject to, for all $(\hat{y}^1, \dots, \hat{y}^N) \in \mathcal{Y} \times \cdots \times \mathcal{Y}$,

$$\sum_{n=1}^{N} \big[\Delta(y^n, \hat{y}^N) + \langle w, \phi(x^n, \hat{y}^n)\rangle - \langle w, \phi(x^n, y^n)\rangle\big] \leq N\xi,$$

One-Slack Formulation of S-SVM:
(equivalent to ordinary S-SVM formulation by $\xi = \frac{1}{N}\sum_n \xi^n$)

$$\min_{w\in\mathbb{R}^D,\xi\in\mathbb{R}_+} \quad \frac{\lambda}{2}\|w\|^2 + \xi$$

subject to, for all $(\hat{y}^1,\dots,\hat{y}^N) \in \mathcal{Y}\times\cdots\times\mathcal{Y}$,

$$\sum_{n=1}^{N}\big[\Delta(y^n,\hat{y}^N) + \langle w,\phi(x^n,\hat{y}^n)\rangle - \langle w,\phi(x^n,y^n)\rangle\big] \leq N\xi,$$

$|\mathcal{Y}|^N$ linear constraints, convex, differentiable objective.

We blew up the constraint set even further:

- 100 binary 16×16 images: 10^{177} constraints (instead of 10^{79}).

Solving S-SVM Training Numerically – One-Slack

Working Set One-Slack S-SVM Training

input training pairs $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
input feature map $\phi(x, y)$, loss function $\Delta(y, y')$, regularizer λ

$S \leftarrow \emptyset$
repeat
  $(w, \xi) \leftarrow$ *solution to QP only with constraints from* S
  for i=1,…,n **do**
    $\hat{y}^n \leftarrow \mathrm{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle$
  end for
  $S \leftarrow S \cup \{\big((x^1, \dots, x^n), (\hat{y}^1, \dots, \hat{y}^n)\big)\}$
until S doesn't change anymore.

output prediction function $f(x) = \mathrm{argmax}_{y \in \mathcal{Y}} \langle w, \phi(x, y) \rangle$.

Often faster convergence:
We add one *strong* constraint per iteration instead of n weak ones.