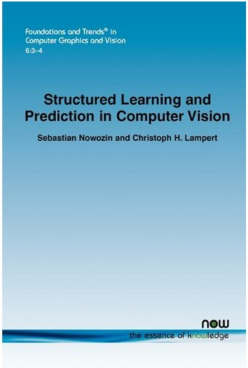
Likelihood based Parameter Learning

Sebastian Nowozin
Microsoft Research

ICCV 2015 Tutorial on Graphical Models

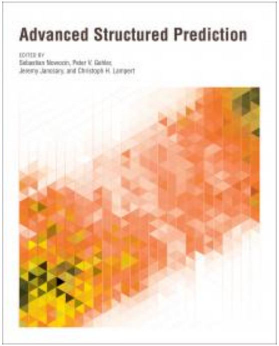
(Slides by Christoph Lampert, http://pub.ist.ac.at/~ chl/)

References



- Nowozin and Lampert, "Structured Learning and Prediction in Computer Vision", 2011
- PDF freely available on author homepages

References



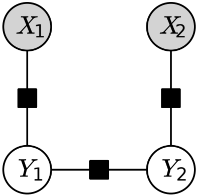
- Nowozin, Gehler, Jancsary, Lampert (Eds.), "Advanced Structured Prediction", MIT Press, 2014.
- Many chapters freely available on the web

Model of a conditional probability distribution:

$$p(y|x) = \frac{1}{Z(x)} e^{-E(x,y)}$$

with $$E(x,y) = \sum_{F \in \mathcal{F}} E_F(x, y_F)$$

e.g. $$E(x,y) = E_1(x_1, y_1) + E_{12}(y_1, y_2) + E_2(x_2, y_2)$$



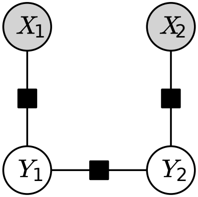
Factor graph

Model of a conditional probability distribution:

$$p(y|x) = \frac{1}{Z(x)} e^{-E(x,y)}$$

with $$E(x,y) = \sum_{F \in \mathcal{F}} E_F(x, y_F)$$

e.g. $$E(x,y) = E_1(x_1, y_1) + E_{12}(y_1, y_2) + E_2(x_2, y_2)$$



Factor graph

Probabilistic Inference:

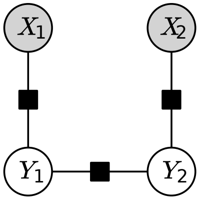
- compute probabilities of some variables/factors, e.g. $p(y_1|x)$

Model of a conditional probability distribution:

$$p(y|x) = \frac{1}{Z(x)} e^{-E(x,y)}$$

with $$E(x,y) = \sum_{F \in \mathcal{F}} E_F(x, y_F)$$

e.g. $$E(x,y) = E_1(x_1, y_1) + E_{12}(y_1, y_2) + E_2(x_2, y_2)$$



Factor graph

Probabilistic Inference:

- compute probabilities of some variables/factors, e.g. $p(y_1|x)$

MAP Prediction / Energy Minimization:

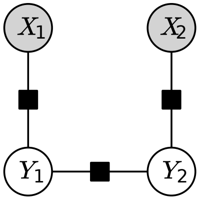
- compute $\operatorname{argmax}_y p(y|x)$, or equivalently $\operatorname{argmin}_y E(x,y)$

Model of a conditional probability distribution:

$$p(y|x) = \frac{1}{Z(x)} e^{-E(x,y)}$$

with $$E(x,y) = \sum_{F \in \mathcal{F}} E_F(x, y_F)$$

e.g. $$E(x,y) = E_1(x_1, y_1) + E_{12}(y_1, y_2) + E_2(x_2, y_2)$$



Factor graph

Probabilistic Inference:

- compute probabilities of some variables/factors, e.g. $p(y_1|x)$

MAP Prediction / Energy Minimization:

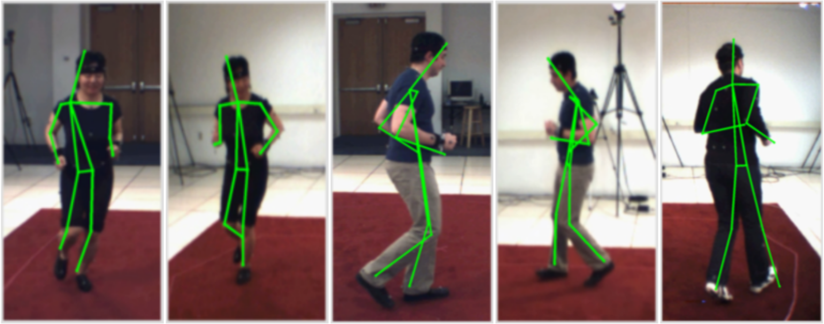
- compute $\operatorname{argmax}_y p(y|x)$, or equivalently $\operatorname{argmin}_y E(x,y)$

Structured Loss Functions:

- $\Delta(y, \bar{y})$: *"how bad is predicting $\bar{y}$ if y is correct?"*

Supervised Learning Problem

- Given training examples $(x^1, y^1), \ldots, (x^N, y^N) \in \mathcal{X} \times \mathcal{Y}$
  $x \in \mathcal{X}$: input, e.g. image
  $y \in \mathcal{Y}$: structured output, e.g. human pose, sentence



Images: HumanEva dataset

- How to make predictions for new inputs, i.e. learn a function $f : \mathcal{X} \to \mathcal{Y}$?

Supervised Learning Problem

- Given training examples $(x^1, y^1), \dots, (x^N, y^N) \in \mathcal{X} \times \mathcal{Y}$
  $x \in \mathcal{X}$: input, e.g. image
  $y \in \mathcal{Y}$: structured output, e.g. human pose, sentence
- How to make predictions for new inputs, i.e. learn a function $f : \mathcal{X} \to \mathcal{Y}$?

Approach 1) Discriminative 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 Fields

$$\max_w \; p(y|x;w)$$

Goal: learn a posterior distribution

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

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

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

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

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 $x^1, \dots, x^N$

$$w^* = \underset{w \in \mathbb{R}^D}{\operatorname{argmax}} \; p(y^1, \dots, y^N | x^1, \dots, x^N; w) \overset{i.i.d.}{=} \underset{w \in \mathbb{R}^D}{\operatorname{argmax}} \; \prod_{n=1}^{N} p(y^n | x^n; w)$$

$$\overset{-\log(\cdot)}{=} \underset{w \in \mathbb{R}^D}{\operatorname{argmin}} \qquad \underbrace{-\sum_{n=1}^{N} \log p(y^n | x^n; w)}_{\text{negative conditional log-likelihood (of } \mathcal{D}\text{)}}$$

Maximum Likelihood Parameter Estimation

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

$$w^* = \operatorname*{argmax}_{w \in \mathbb{R}^D} p(y^1, \dots, y^N | x^1, \dots, x^N; w) \overset{i.i.d.}{=} \operatorname*{argmax}_{w \in \mathbb{R}^D} \prod_{n=1}^{N} p(y^n | x^n; w)$$

$$\overset{-\log(\cdot)}{=} \operatorname*{argmin}_{w \in \mathbb{R}^D} \underbrace{-\sum_{n=1}^{N} \log p(y^n | x^n; w)}_{\text{negative conditional log-likelihood (of } \mathcal{D})}$$

$$= \operatorname*{argmin}_{w \in \mathbb{R}^D} -\sum_{n=1}^{N} \left[\log e^{-\langle w, \phi(x^n, y) \rangle} - \log Z(x; w) \right]$$

$$= \operatorname*{argmin}_{w \in \mathbb{R}^D} \sum_{n=1}^{N} \left[\langle w, \phi(x^n, y^n) \rangle + \underbrace{\log \sum_{y \in \mathcal{Y}} e^{-\langle w, \phi(x^n, y) \rangle}}_{\text{log-partition function}} \right]$$

MAP Estimation of w

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

MAP Estimation of w

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

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

$p(w)$: *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)

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

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

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

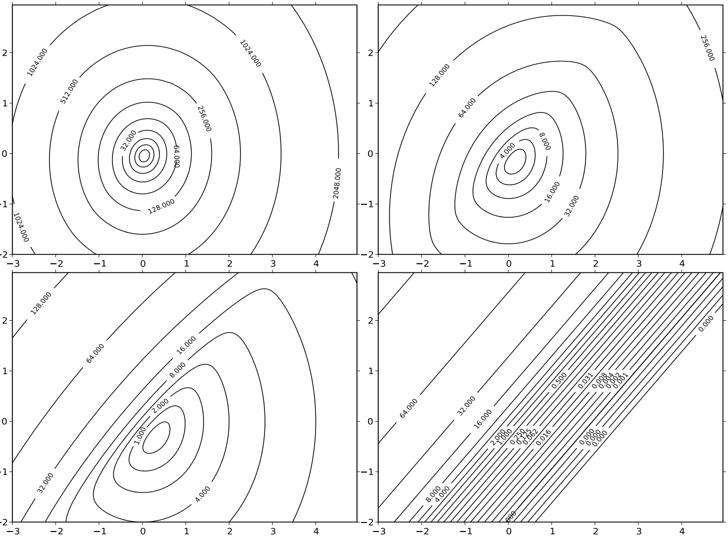
($\lambda \to 0$ makes it *unregularized*)

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 $\epsilon > 0$

```
w_cur ← 0
repeat
    v ← ∇_w L(w_cur)
    η ← argmin_{η∈ℝ} L(w_cur − ηv)
    w_cur ← w_cur − ηv
until ‖v‖ < ε
```

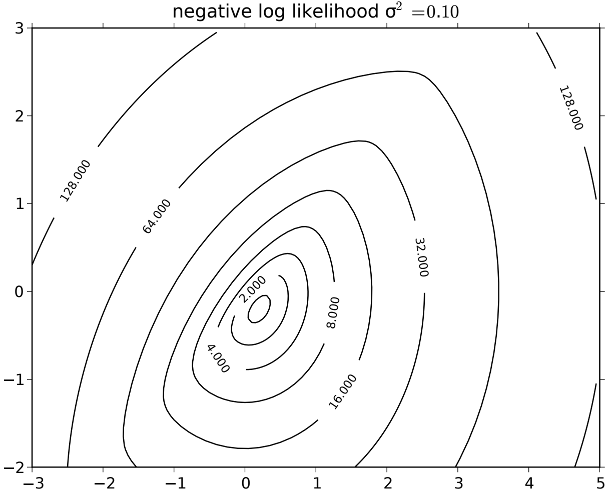
output w_{cur}

Alternatives:

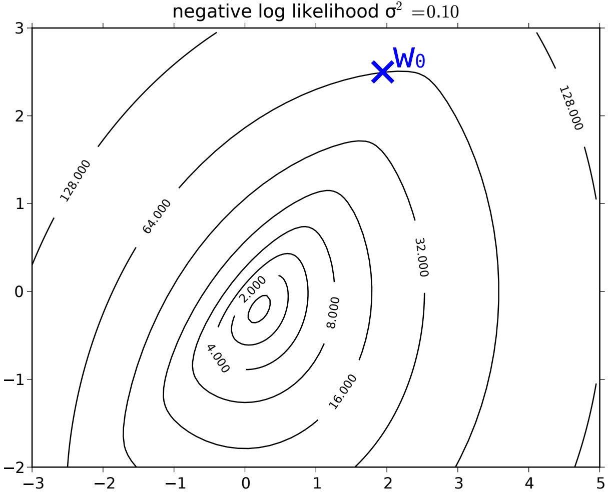
- L-BFGS (second-order descent without explicit Hessian)
- Conjugate Gradient

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

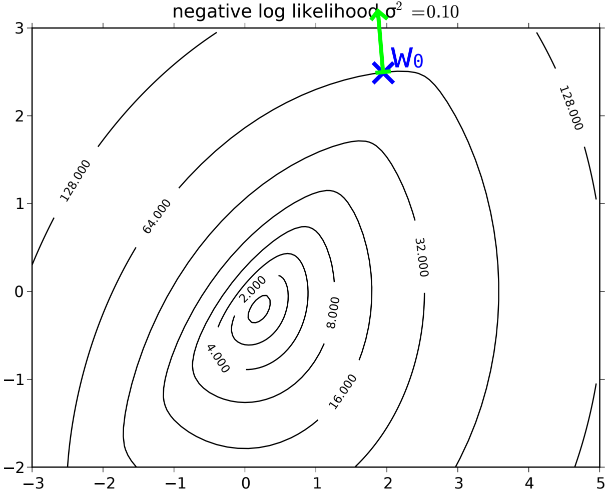
Steepest Descent Minimization – minimize $\mathcal{L}(w)$



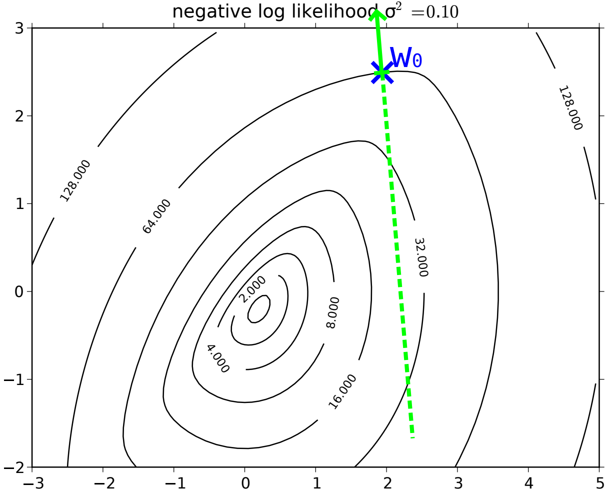
Steepest Descent Minimization – minimize $\mathcal{L}(w)$



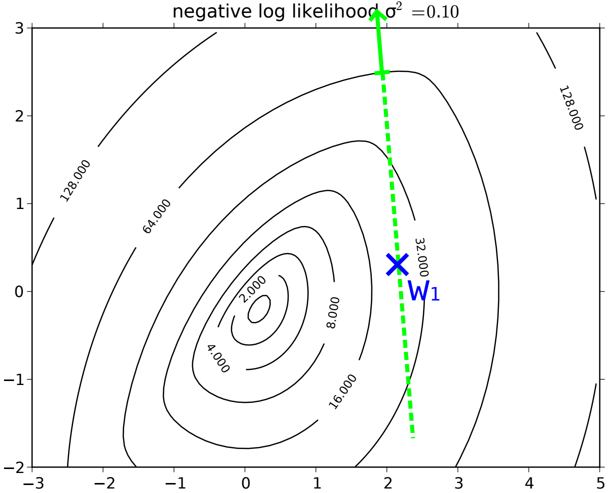
Steepest Descent Minimization – minimize $\mathcal{L}(w)$



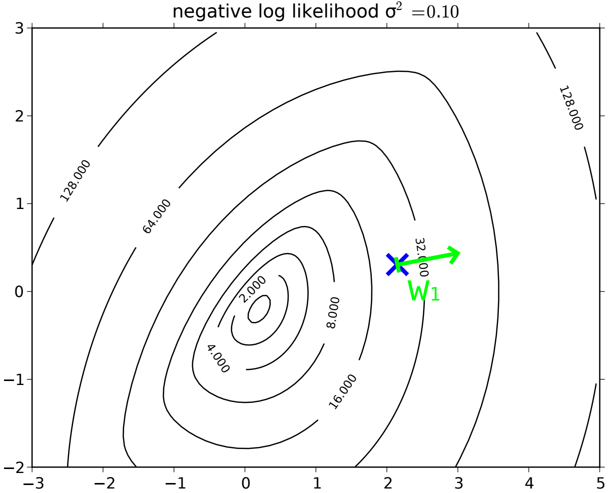
Steepest Descent Minimization – minimize $\mathcal{L}(w)$



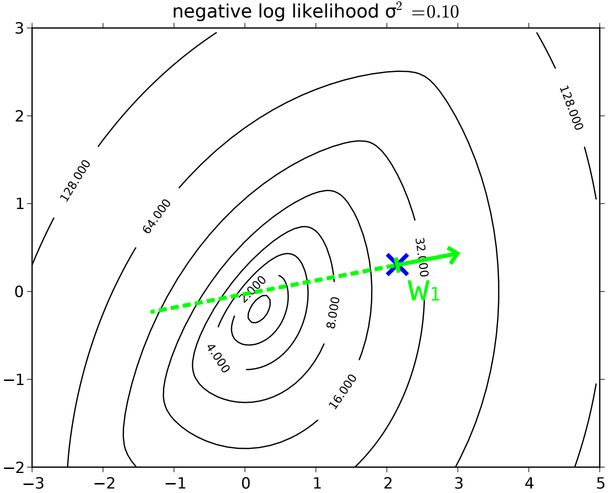
Steepest Descent Minimization – minimize $\mathcal{L}(w)$



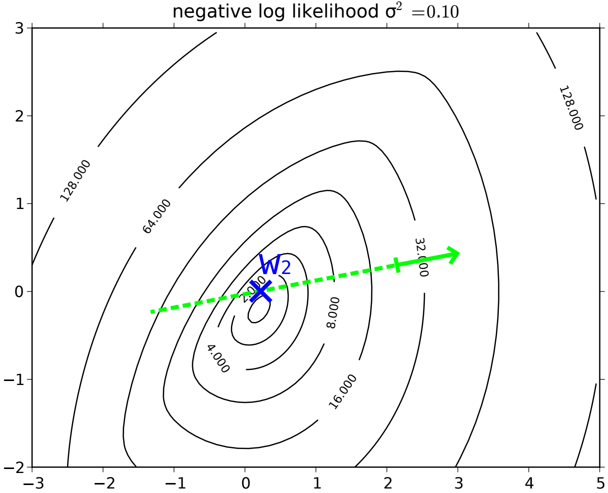
Steepest Descent Minimization – minimize $\mathcal{L}(w)$



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



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



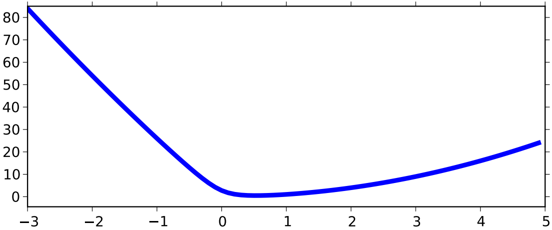
$$\mathcal{L}(w) = \frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N} \left[\langle w, \phi(x^n, y^n)\rangle + \log \sum_{y\in\mathcal{Y}} e^{-\langle w, \phi(x^n, y)\rangle}\right]$$

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

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

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

- 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:

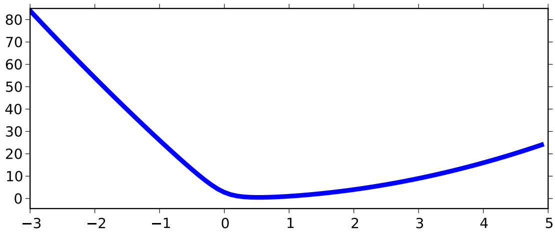
- We want the model distribution to match the empirical one:

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

- E.g. image segmentation
  - ϕ_{unary}: correct amount of foreground vs. background
  - ϕ_{pairwise}: correct amount of fg/bg transitions $\to$ smoothness

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

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



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

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*.

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 variables
- K: number of possible labels of each output variables

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 variables ≈ 10s to 1,000,000s
- K: number of possible labels of each output variables ≈ 2 to 1000s

Solving the Training Optimization Problem Numerically

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 $p(y|x;w)$,
- compute/approximate them by probabilistic inference.

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

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

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

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

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

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

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

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

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

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

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

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

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

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: ~~$O(K^M N D)$~~, $O(MK^2 N D)$ (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: ~~$O(K^M N D)$~~, $O(MK^2 N D)$ per evaluation of $\mathcal{L}$

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

Solving the Training Optimization Problem Numerically

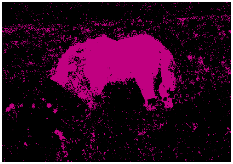
Typical feature functions in **image segmentation:**

- $\phi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: 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: $\mathrm{argmax}_y\, p(y|x)$ 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: $\text{argmax}_y\, p(y|x)$ 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.

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: $O(K^M ND)$, (if BP is possible: $O(MK^2ND)$)

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

Line Search: $O(K^M ND)$ (if BP is possible: $O(MK^2ND)$)

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

Training with Approximate Likelihood

Problem: what if probabilistic inference is still too expensive?

Idea: optimize a simpler quantity instead of $\mathcal{L}$

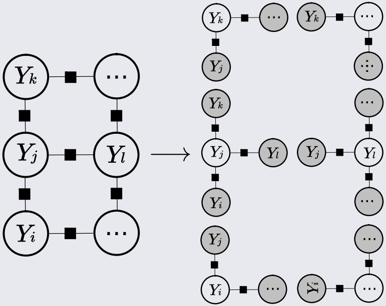
Problem: what if probabilistic inference is still too expensive?

Idea: optimize a simpler quantity instead of $\mathcal{L}$

Pseudolikelihood [Besag, 1987]

$$p(y|x) \approx \prod_{i \in V} p(y_i | y_{V \setminus \{i\}}, x)$$

$$= \prod_{i \in V} p(y_i | y_{N(i)}, x)$$



Training with Approximate Likelihood – Pseudolikelihood (PL)

$$p(y|x) \approx p_{\mathsf{PL}}(y|x) = \prod_{i \in V} p(y_i | y_{N(i)}, x; w)$$

For training data $\{(x^1, y^1), \ldots, (x^N, y^N)\}$:

$$\begin{aligned}
\mathcal{L}_{PL}(w) &= \log \prod_{n=1}^{N} p_{\mathsf{PL}}(y^n | x^n; w) \\
&= \sum_{n=1}^{N} \sum_{i \in V} \log p(y_i^n | y_{N(i)}^n, x) \\
&= \sum_{n=1}^{N} \sum_{i \in V} \Big[\langle w, \phi(y^n, x^n) \rangle - \log \sum_{\boldsymbol{k} \in \mathcal{Y}_i} e^{\langle w, \phi(y_1^n, \ldots, y_{i-1}^n, \boldsymbol{k}, y_{i+1}^n, \ldots, y_{|V|}^n, x^n) \rangle} \Big]
\end{aligned}$$

Training with Approximate Likelihood – Pseudolikelihood (PL)

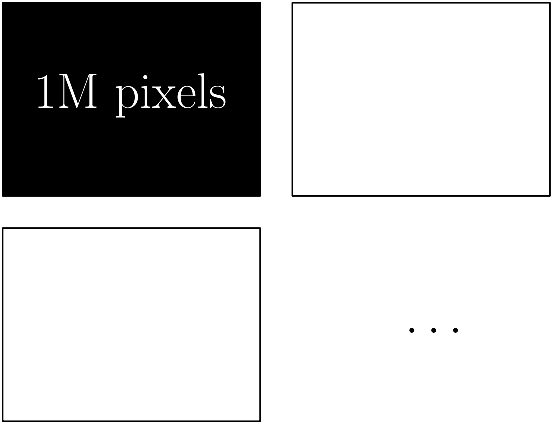
$$p(y|x) \approx p_{\mathsf{PL}}(y|x) = \prod_{i \in V} p(y_i | y_{N(i)}, x; w)$$

For training data $\{(x^1, y^1), \ldots, (x^N, y^N)\}$:

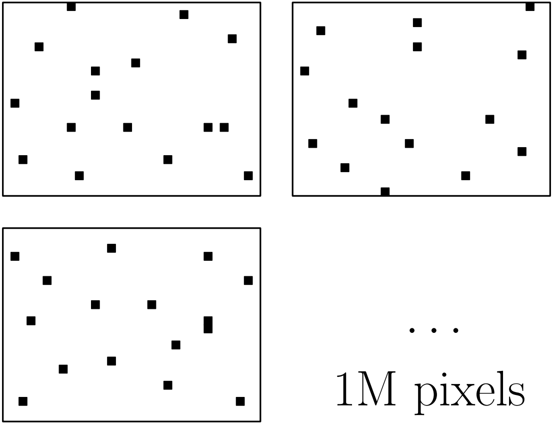
$$\begin{aligned}
\mathcal{L}_{PL}(w) &= \log \prod_{n=1}^{N} p_{\mathsf{PL}}(y^n | x^n; w) \\
&= \sum_{n=1}^{N} \sum_{i \in V} \log p(y_i^n | y_{N(i)}^n, x) \\
&= \sum_{n=1}^{N} \sum_{i \in V} \Big[\langle w, \phi(y^n, x^n) \rangle - \log \sum_{\boldsymbol{k} \in \mathcal{Y}_i} e^{\langle w, \phi(y_1^n, \ldots, y_{i-1}^n, \boldsymbol{k}, y_{i+1}^n, \ldots, y_{|V|}^n, x^n) \rangle} \Big]
\end{aligned}$$

Partition functions sum only over one variable at a time $\rightarrow$ tractable

Efficient Training by Pseudolikelihood Subsampling



- (Nowozin et al., ICCV 2011), pseudolikelihood subsampling
- Decouples training complexity from instance count N



- (Nowozin et al., ICCV 2011), pseudolikelihood subsampling
- Decouples training complexity from instance count N

Training with Approximate Likelihood

Problem: what if probabilistic inference is still too expensive?

Idea: optimize a simpler quantity instead of $\mathcal{L}$

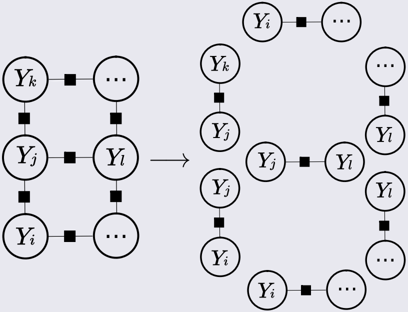
Problem: what if probabilistic inference is still too expensive?

Idea: optimize a simpler quantity instead of $\mathcal{L}$

Piecewise Training [Sutton, McCallum, 2005]

$$p(y|x) \approx \prod_{F \in \mathcal{F}} p_F(y_F|x) \quad \text{for}$$

$$p_F(y_F|x) = \frac{1}{Z_F(x;w)} e^{-\langle w_F, \phi_F(y_F,x) \rangle}$$



Training with Approximate Likelihood – Piecewise Training (PW)

$$p(y|x) \approx \prod_{F \in \mathcal{F}} p_F(y_F|x; w_F) \quad \text{for} \quad p_F(y_F|x) \propto e^{-\langle w_F, \phi_F(y_F, x)\rangle}$$

For training data $\{(x^1, y^1), \ldots, (x^N, y^N)\}$:

$$\begin{aligned}
\mathcal{L}_{PW}(w) &= \log \prod_{n=1}^{N} p_{\mathsf{PW}}(y_F^n | x^n; w) = \sum_{n=1}^{N} \sum_{F \in \mathcal{F}} \log p_F(y_F^n | x) \\
&= \sum_{n=1}^{N} \sum_{F \in \mathcal{F}} \Big[\langle w_F, \phi_F(y_F^n, x^n)\rangle - \log \sum_{\bar{y}_F \in \mathcal{Y}_F} e^{\langle w_F, \phi_F(\bar{y}_F, x^n)\rangle} \Big]
\end{aligned}$$

Training with Approximate Likelihood – Piecewise Training (PW)

$$p(y|x) \approx \prod_{F\in\mathcal{F}} p_F(y_F|x;w_F) \quad \text{for} \quad p_F(y_F|x) \propto e^{-\langle w_F, \phi_F(y_F,x)\rangle}$$

For training data $\{(x^1,y^1),\dots,(x^N,y^N)\}$:

$$\mathcal{L}_{PW}(w) = \log\prod_{n=1}^{N} p_{\mathsf{PW}}(y_F^n|x^n;w) = \sum_{n=1}^{N}\sum_{F\in\mathcal{F}} \log p_F(y_F^n|x)$$

$$= \sum_{n=1}^{N}\sum_{F\in\mathcal{F}}\Big[\langle w_F, \phi_F(y_F^n,x^n)\rangle - \log \sum_{\bar{y}_F\in\mathcal{Y}_F} e^{\langle w_F,\phi_F(\bar{y}_F,x^n)\rangle}\Big]$$

Partition functions sum over $|F|$ variables at a time $\to$ usually tractable

Optimization decomposes into a sum over the w_F $\to$ easy to parallelize

Training with Approximate Likelihood – Piecewise Training (PW)

$$p(y|x) \approx \prod_{F\in\mathcal{F}} p_F(y_F|x;w_F) \quad \text{for} \quad p_F(y_F|x) \propto e^{-\langle w_F, \phi_F(y_F,x)\rangle}$$

For training data $\{(x^1,y^1),\dots,(x^N,y^N)\}$:

$$\mathcal{L}_{PW}(w) = \log \prod_{n=1}^{N} p_{\mathsf{PW}}(y_F^n|x^n;w) = \sum_{n=1}^{N}\sum_{F\in\mathcal{F}} \log p_F(y_F^n|x)$$

$$= \sum_{F\in\mathcal{F}}\sum_{n=1}^{N}\Big[\langle w_F, \phi_F(y_F^n,x^n)\rangle - \log \sum_{\bar{y}_F\in\mathcal{Y}_F} e^{\langle w_F, \phi_F(\bar{y}_F,x^n)\rangle}\Big]$$

Partition functions sum over $|F|$ variables at a time $\rightarrow$ usually tractable

Optimization decomposes into a sum over the w_F $\rightarrow$ easy to parallelize

Solving the Training Optimization Problem Numerically

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

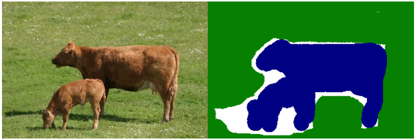
$$\tilde{\nabla}_w \mathcal{L}(w) = \lambda w - \boldsymbol{\sum_{n=1}^{N}} \Big[\phi(x^n, y^n) - \boldsymbol{\sum_{y \in \mathcal{Y}} p(y|x^n; w)}\, \phi(x^n, y)\Big] \quad \in \mathbb{R}^{\boldsymbol{D}}$$

A lot of research on accelerating CRF training:

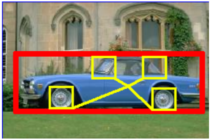
problem	"solution"	method(s)
$\lvert\mathcal{Y}\rvert$ too large	exploit structure	(loopy) belief propagation
	fast sampling	contrastive divergence
	use approximate $\mathcal{L}$	pseudo-likelihood, piecewise
N too large	mini-batches	stochastic gradient descent
D too large	pretrained ϕ_{unary}	two-stage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

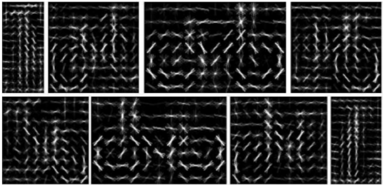


missing labels in training data



latent variables, e.g. part location

latent variables, e.g. part occlusion



latent variables, e.g. viewpoint

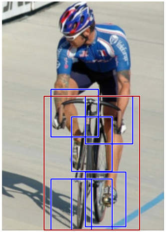
CRFs with Latent Variables

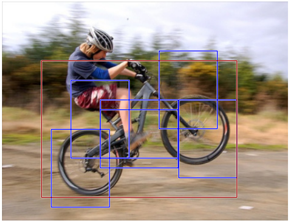
Three types of variables in graphical model:

- $x \in \mathcal{X}$ 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:

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

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

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

Negative regularized conditional log-likelihood:

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

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

How to best train CRFs with latent variables is active research.

Summary – CRF Learning

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

Energy is linear in parameter vector $w = (w_F)_{F \in \mathcal{F}}$

$$E(y, x; w) = \langle w, \phi(x, y) \rangle = \sum_{F \in \mathcal{F}} \langle w_F, \phi_F(y_F, x) \rangle$$

Overall model is log-linear: $\quad p(y|x; w) \propto e^{-\langle w, \phi(x,y) \rangle}$

Summary – CRF Learning

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

Energy is linear in parameter vector $w = (w_F)_{F \in \mathcal{F}}$

$$E(y, x; w) = \langle w, \phi(x, y) \rangle = \sum_{F \in \mathcal{F}} \langle w_F, \phi_F(y_F, x) \rangle$$

Overall model is log-linear: $p(y|x; w) \propto e^{-\langle w, \phi(x,y) \rangle}$

CRF training requires minimizing negative conditional log-likelihood:

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

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

Structured Support Vector Machines

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

Supervised Learning Problem

- Training examples $(x^1, y^1), \ldots, (x^N, y^N) \in \mathcal{X} \times \mathcal{Y}$
- 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^*.

Reminder: Regularized Risk Minimization

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

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

Two major problems:

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

  we can't apply gradient-based optimization

Reminder: Regularized Risk Minimization

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

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

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

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

Regularization $+$ *Loss on training data*

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

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

- 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

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

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

Structured Output Support Vector Machine

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

Conditional Random Field

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

CRFs and SSVMs have more in common than usually assumed.

- $\log \sum_y \exp(\cdot)$ can be interpreted as a soft-max (differentiable)
- SSVM training takes loss function into account
- CRF is trained without specific loss, but loss enters at prediction time

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

Solve:

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

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 Support Vector Machine

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

Solve:

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

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]

Solving S-SVM Training Numerically

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

Solving S-SVM Training Numerically

We can solve SSVM training like CRF training:

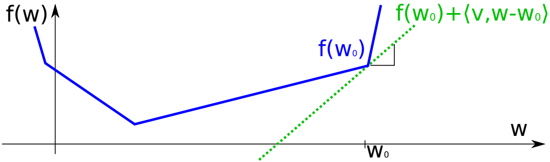
$$\min_{w} \; \frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N}\Big[\max_{y\in\mathcal{Y}} \; \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\Big]$$

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

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

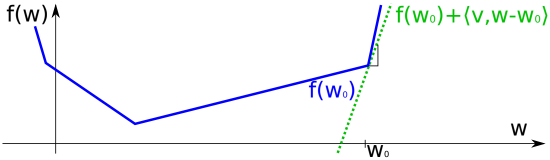
$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



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

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

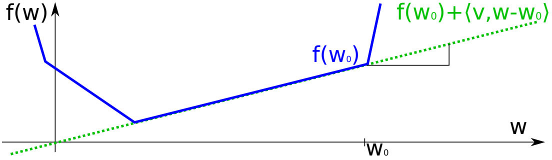


Solving S-SVM Training Numerically – Subgradient Method

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

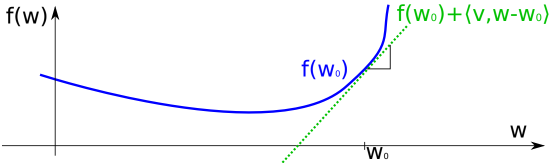
$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



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 Minimization – minimize $F(w)$ [Shor, 1985]

- **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}

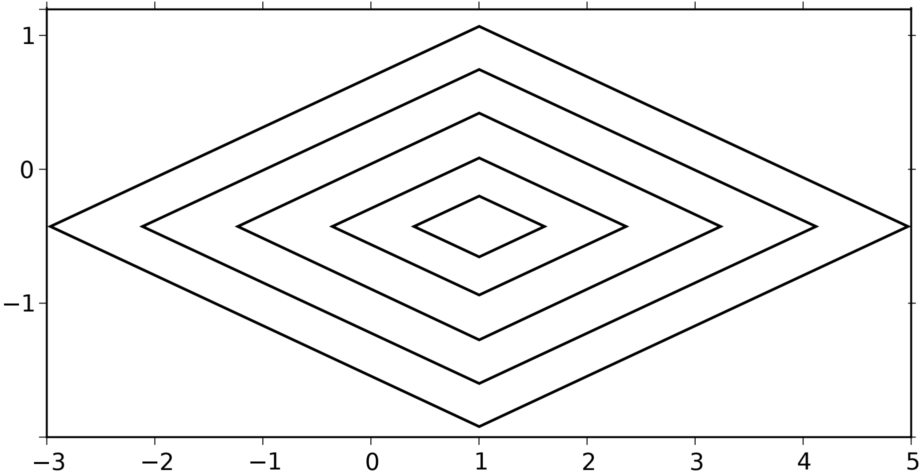
Subgradient method looks very similar to gradient descent:

- iterative update in opposite direction of (sub)gradients
- converges to global minimum for convex F,

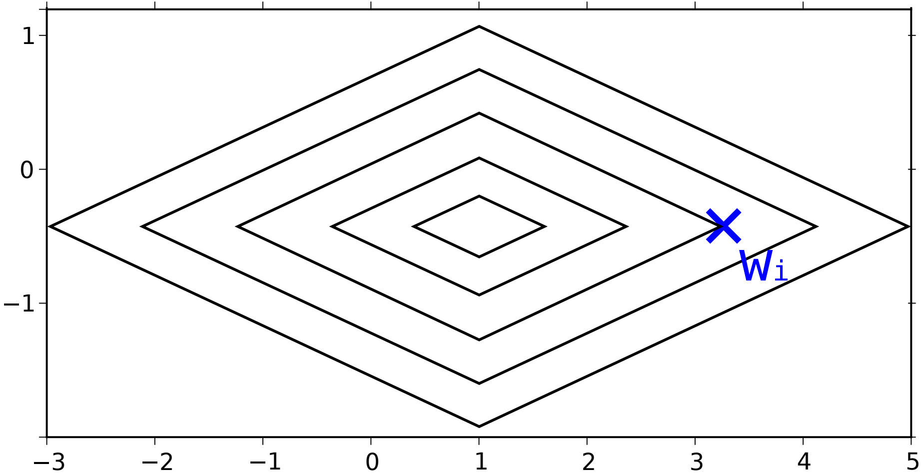
Caveats for non-differentiable F:

- only possible for convex functions (unlike gradient descent)
- not a descent method: **the objective can go up in some steps**

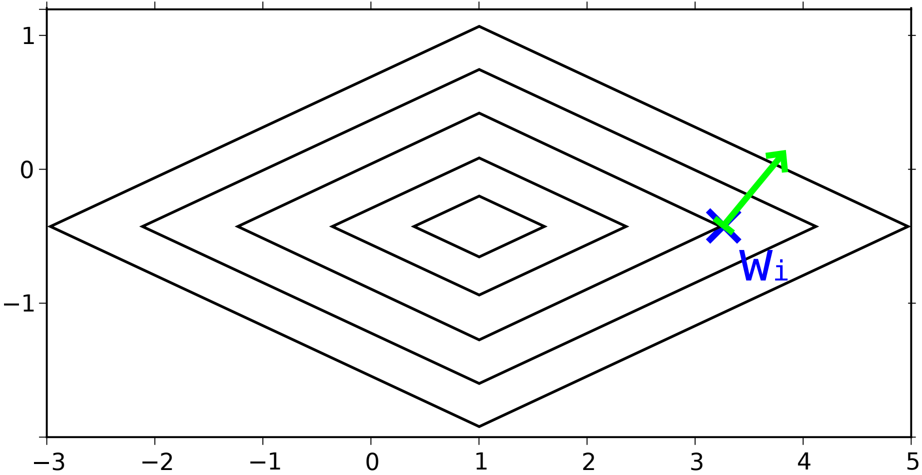
Subgradient method

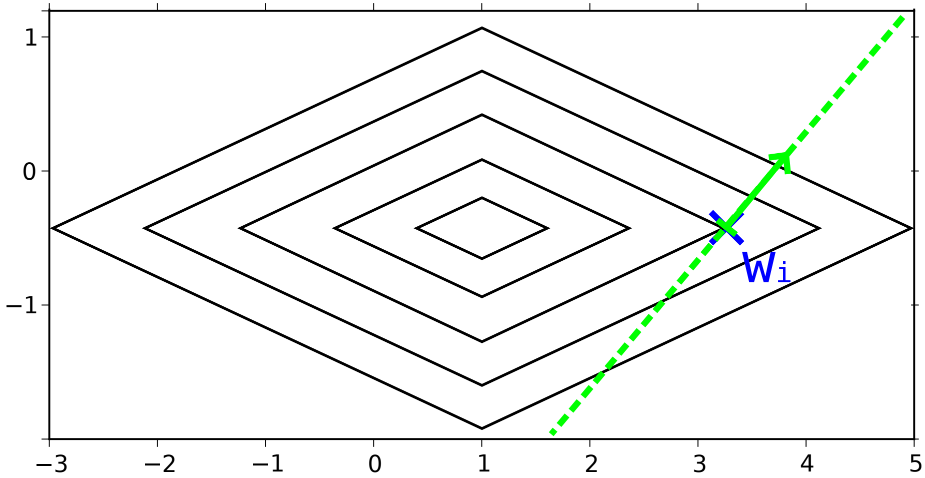


Subgradient method

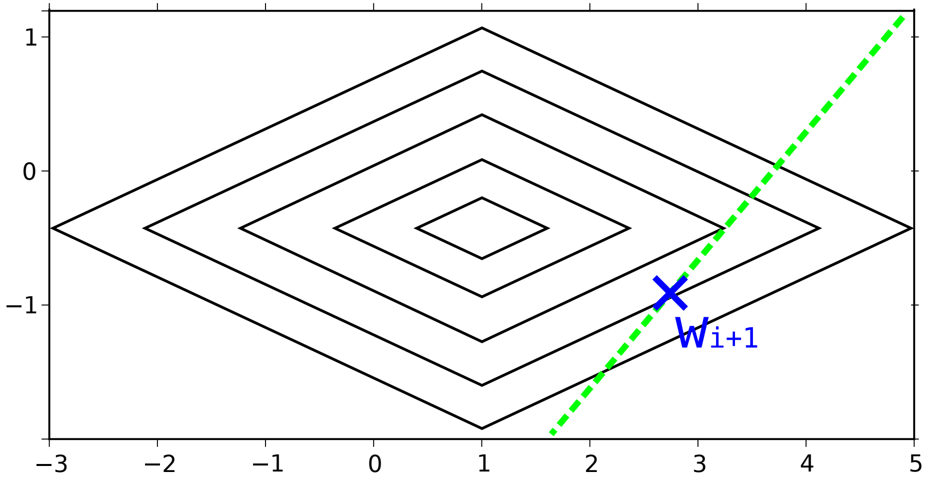


Subgradient method



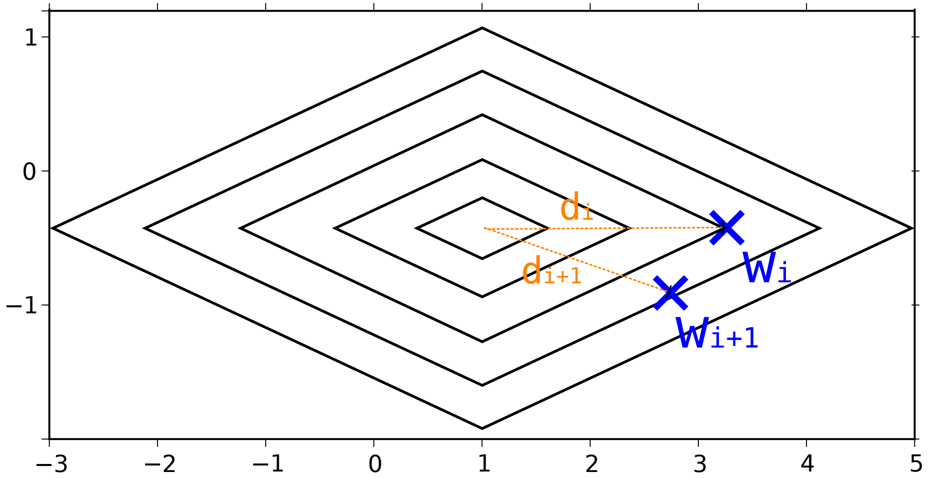


All points along subgradient have larger objective than starting point!



All points along subgradient have larger objective than starting point!

Subgradient method



Why does it work anyway? Distance to optimum decreases in every step!

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

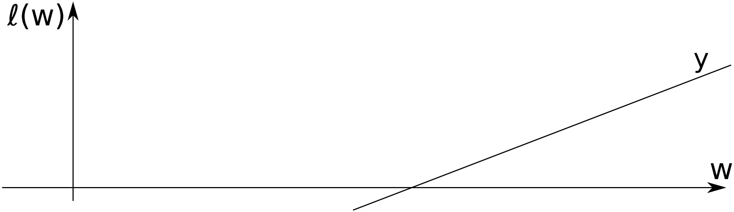
$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$

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

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



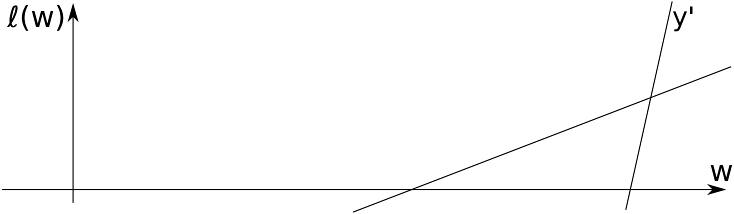
For each $y \in \mathcal{Y}$, $\ell^n_y(w)$ is a linear function of w.

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

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



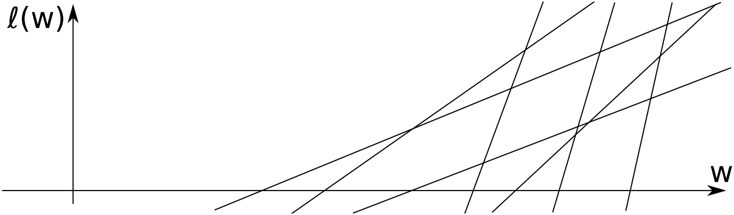
For each $y \in \mathcal{Y}$, $\ell^n_y(w)$ is a linear function of w.

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

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



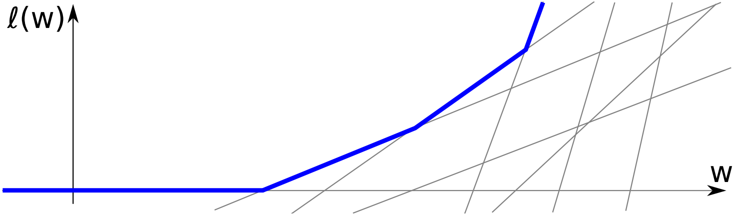
For each $y \in \mathcal{Y}$, $\ell^n_y(w)$ is a linear function of w.

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

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



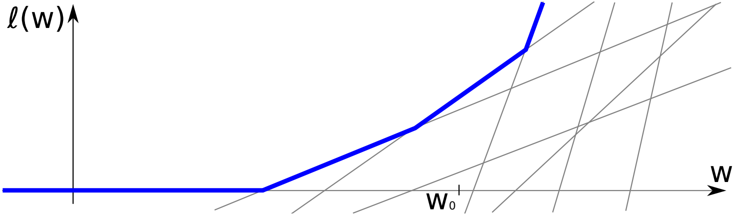
$\max$ over finite $\mathcal{Y}$: piece-wise linear

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

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



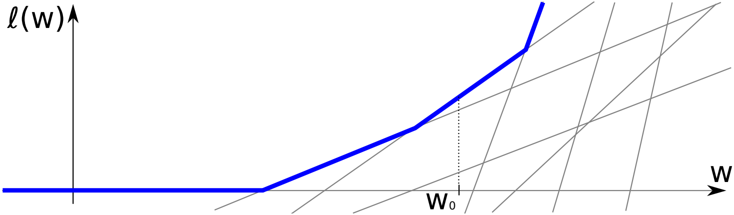
Subgradient of ℓ^n at w_0:

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

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



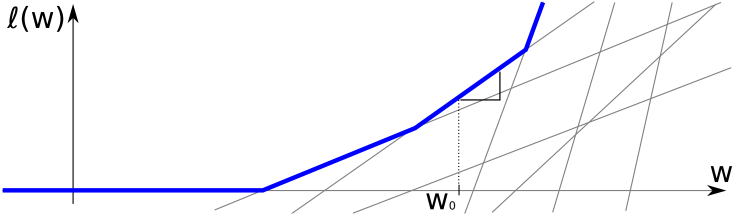
Subgradient of ℓ^n at w_0: find maximal (active) y.

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 $v = \nabla \ell^n_y(w_0)$.

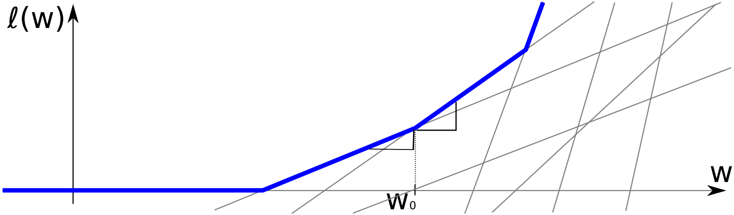
Solving S-SVM Training Numerically – Subgradient Method

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

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



Not necessarily unique, but $v = \nabla \ell^n_y(w_0)$ works for any maximal y

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).
Obs: computing the argmax is (loss augmented) **energy minimization**

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)

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)

$t = 1$: $w = 0$,

$$\hat{y} = \operatorname*{argmax}_y \Big[\langle w, \phi(x^n, y) \rangle + \Delta(y^n, y) \Big]$$

$$\stackrel{w=0}{=} \operatorname*{argmax}_y \Delta(y^n, y) \quad = \text{"the opposite of } y^n\text{"}$$

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 $-$

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 $-$

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 $-$

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 $=$

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)

$t = 1$: $\hat{y} =$  $\phi(y^n) - \phi(\hat{y})$: 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.

Solving S-SVM Training Numerically – Subgradient Method

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)

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?

We can use **inequalities** and **slack variables** to reformulate the optimization.

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 $n = 1, \dots, N$,

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

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 $\max$ term into individual constraints

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

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

Solving S-SVM Training Numerically

Solve an S-SVM like a linear Support Vector Machine:

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

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

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

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

```
w ← 0, S ← ∅
repeat
    (w, ξ) ← solution to QP only with constraints from S
    for i=1,...,n do
        ŷ ← argmax_{y∈𝒴} Δ(y^n, y) + ⟨w, φ(x^n, y)⟩
        if ŷ ≠ y^n then
            S ← S ∪ {(x^n, ŷ)}
        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.

Frank-Wolfe Algorithm

Most important algorithm in use today:

- Frank-Wolfe algorithm for S-SVM training (Lacoste-Julien et al., 2013)
- Iteration complexity of primal stochastic subgradient method
- Explicit duality gap stopping criterion
- Simpler to implement than cutting plane approaches

SSVMs with Latent Variables [Yu, Joachims, 2009], see also [Felzenszwalb *et al.*, 2008]

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 [Yu, Joachims, 2009], see also [Felzenszwalb *et al.*, 2008]

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

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 $f(x) := \mathrm{argmax}_y \langle w, \phi(x, y) \rangle$

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 $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$
- 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, $\mathbb{E}_{(x,y)} \Delta(y, f(x))$

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?
- Can we understand structured learning with approximate inference?
  - often computing $\nabla\mathcal{L}(w)$ or $\operatorname{argmax}_y \langle w, \phi(x,y)\rangle$ exactly is infeasible.
  - can we guarantee good results even with approximate inference?
- Learning data representations
  - e.g. by combinations with deep learning
- More and new applications!