# Introduction to Probabilistic Graphical Models

Christoph Lampert

IST Austria (Institute of Science and Technology Austria)

IST AUSTRIA
Institute of Science and Technology

# Schedule

| | |
|---|---|
| | **Refresher of Probabilities** |
| | **Introduction to Probabilistic Graphical Models** |
| | **Probabilistic Inference** |
| | **Learning Conditional Random Fields** |
| | **MAP Prediction / Energy Minimization** |
| | **Learning Structured Support Vector Machines** |

Links to slide download: `http://pub.ist.ac.at/~chl/courses/PGM_W16/`

Password for ZIP files (if any): `pgm2016`

Email for questions, suggestions or typos that you found: `chl@ist.ac.at`

# 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 $f : \mathcal{X} \to \mathcal{Y}$ ?

## 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 $f : \mathcal{X} \to \mathcal{Y}$ ?

**Approach 1)** Probabilistic Learning

1) Use training data to learn a probability distribution $p(y|x)$
2) Use $f(x) := \operatorname{argmin}_{y \in \mathcal{Y}} \mathbb{E}_{\bar{y} \sim p(y|x)} \Delta(\bar{y}, y)$ to make predictions.

## 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 $f : \mathcal{X} \to \mathcal{Y}$ ?

### **Approach 1)** Probabilistic Learning

1) Use training data to learn a probability distribution $p(y|x)$
2) Use $f(x) := \operatorname{argmin}_{y \in \mathcal{Y}} \mathbb{E}_{\bar{y} \sim p(y|x)} \Delta(\bar{y}, y)$ to make predictions.

For example, if $\Delta(\bar{y}, y) = [\![\bar{y} \neq y]\!]$ or intractable otherwise:

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

for $p(y|x) \propto e^{-E(x,y)}$ and $E(x, y) = \langle \theta, \phi(x, y) \rangle$.

## 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 $f : \mathcal{X} \to \mathcal{Y}$ ?

**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.

## 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 $f : \mathcal{X} \to \mathcal{Y}$ ?

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

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

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

## 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 $\mathbb{E}_{(x,y) \sim d(x,y)} (\,\cdot\,)$ with empirical estimate $\frac{1}{N} \sum_{(x^n, y^n)} (\,\cdot\,)$
- 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: $\ell$ upper bound to $\Delta$, 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

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

$$\textit{Regularization} \; + \; \textit{Loss on training data}$$

# 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*

## 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

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

*Regularization* + *Loss on training data*

## Hinge loss: maximum margin training

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

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

# Reminder: Regularized Risk Minimization

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

$$\textit{Regularization} \; + \; \textit{Loss on training data}$$

## 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

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

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

## Hinge loss

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

## Log-loss

$$\min_{w} \frac{\lambda}{2}\|w\|^2 + \sum_{n=1}^{N} \log \sum_{y\in\mathcal{Y}} \exp\big(\langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\big)$$

## 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

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

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

# Example: Multiclass Support Vector Machine

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

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

**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

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

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]

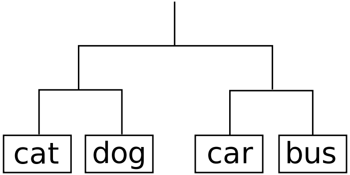
# Example: Hierarchical Multiclass SVM

**Hierarchical Multiclass Loss:**

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

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

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




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

[L. Cai, T. Hofmann: "Hierarchical Document Categorization with Support Vector Machines", ACM CIKM, 2004]

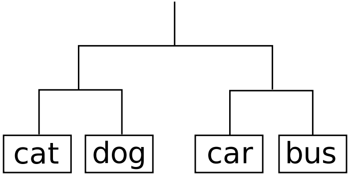
# Example: Hierarchical Multiclass SVM

**Hierarchical Multiclass Loss:**

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

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

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




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

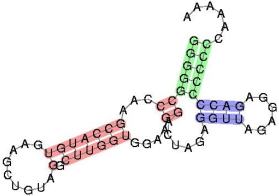
$$\textit{e.g.}\text{ if } y^n = \texttt{cat}, \begin{cases} \langle w, \phi(x^n, \texttt{cat})\rangle - \langle w, \phi(x^n, \texttt{dog})\rangle \overset{!}{\geq} 1 \\ \langle w, \phi(x^n, \texttt{cat})\rangle - \langle w, \phi(x^n, \texttt{car})\rangle \overset{!}{\geq} 2 \\ \langle w, \phi(x^n, \texttt{cat})\rangle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

- 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]

# Example: RNA Secondary Structure Prediction De Bona *et al.*, 2007]

AAAAACCCCCCCCAGAGGAGAUUG
GAGAUCAAAGGUGGUUCGGAUGUC
GAAGUGUACCGAACCCGGGGG $\rightarrow$

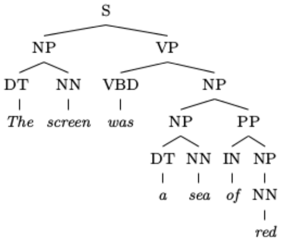


- $\mathcal{X} = \Sigma^*$ for $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$ (nucleotide sequence)
- $\mathcal{Y} = \{(i,j) : i,j \in \mathbb{N}, i < j\}$ $\quad (i,j)$ mean "$x_i$ binds with $x_j$"
- $\phi(x,y)$ stacked domain-specific features, e.g. binding energy of $x_i \leftrightarrow x_j$, prefered patterns (motifs), loop properties, ...
- $\Delta(\bar{y}, y)$: number of wrong/missing bindings (Hamming loss)

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

# Example: Sentence Parsing [Taskar *et al.*, 2004]

`The screen was a sea of red.` $\rightarrow$




- $\mathcal{X} = \{\text{English sentences}\}$
- $\mathcal{Y} = \{\text{parse tree}\}$
- $\phi(x, y)$ domain-specific features:
  - word properties, *e.g.* "· starts with capital letter", "· ends in `ing`"
  - grammatical rules: $NP \rightarrow DT + NN$
- $\Delta(\bar{y}, y)$: number of wrong assignments

# Solving S-SVM Training Numerically

We can solve SSVM training like CRF training:

# Solving S-SVM Training Numerically

We can solve SSVM training like CRF training:

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

# 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.

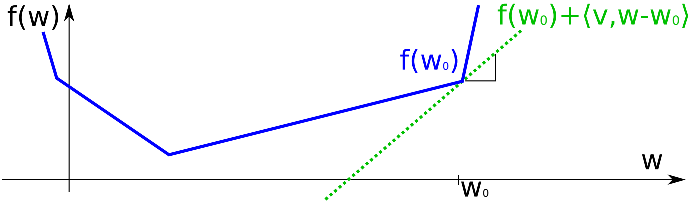
  $\rightarrow$ we'll have to use **subgradients**

# 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

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

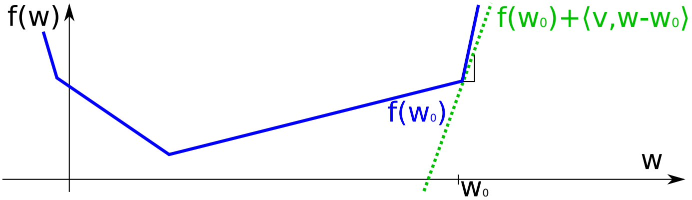


# Solving S-SVM Training Numerically – Subgradient Method

## 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

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

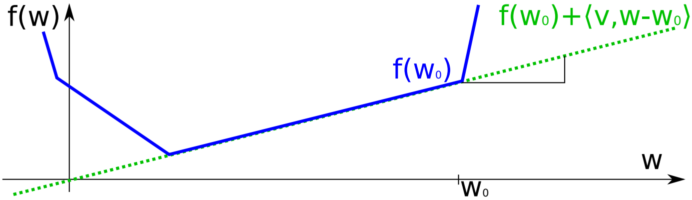


# Solving S-SVM Training Numerically – Subgradient Method

## 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

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

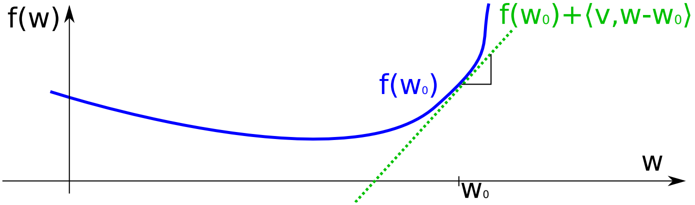


# Solving S-SVM Training Numerically – Subgradient Method

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

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

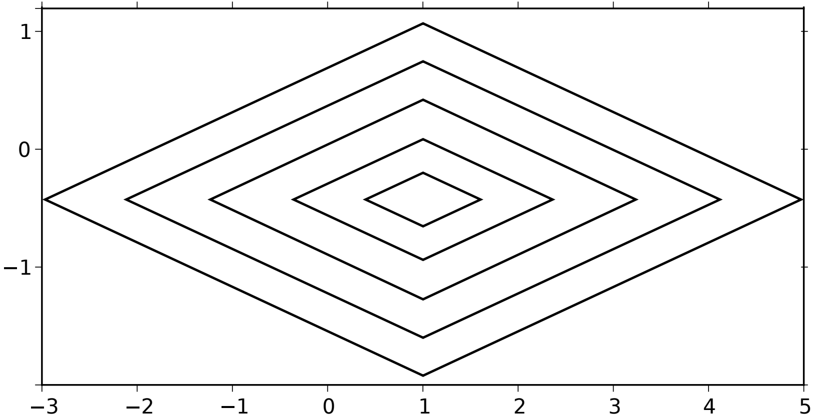
Subgradient method looks very similar to gradient descent:

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

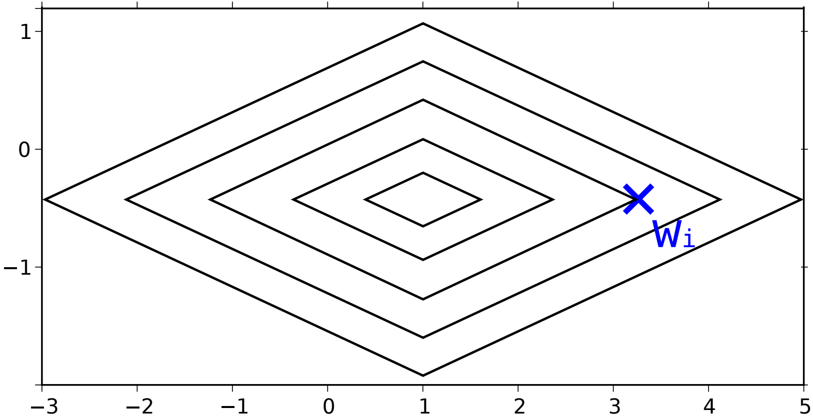
**Caveats for non-differentiable $F$:**

- only possible for convex functions (unlike gradient descent)
- not a descent method: **the objective can sometimes go up**, but overall it will decrease

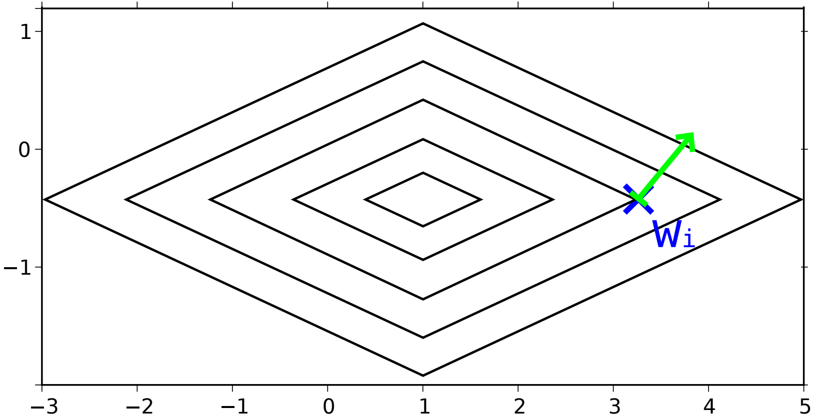
# Subgradient method



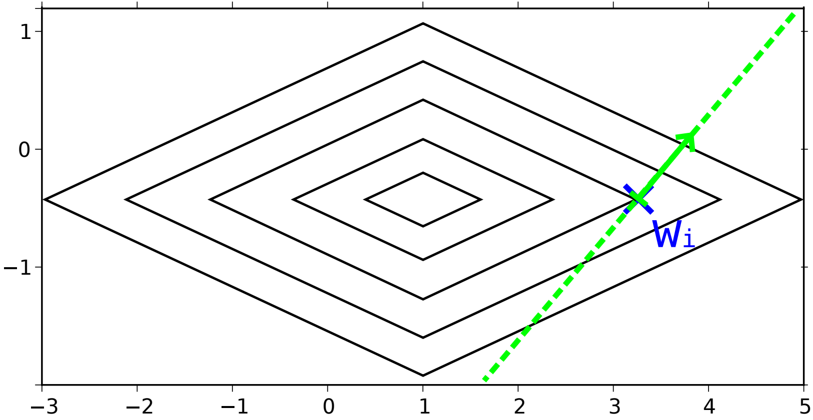
# Subgradient method



# Subgradient method

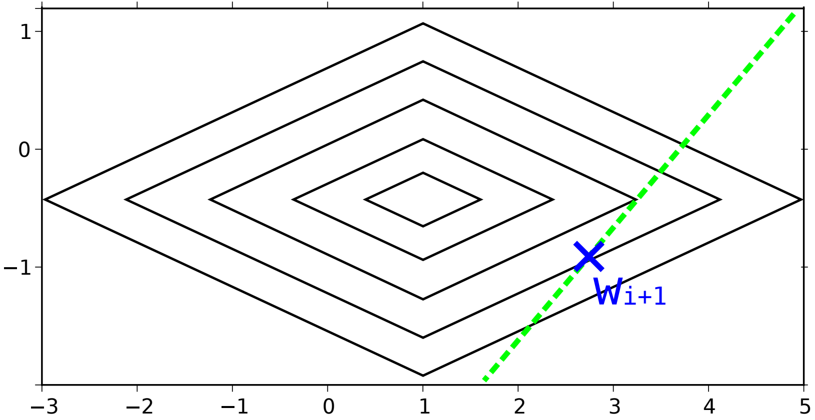


# Subgradient method



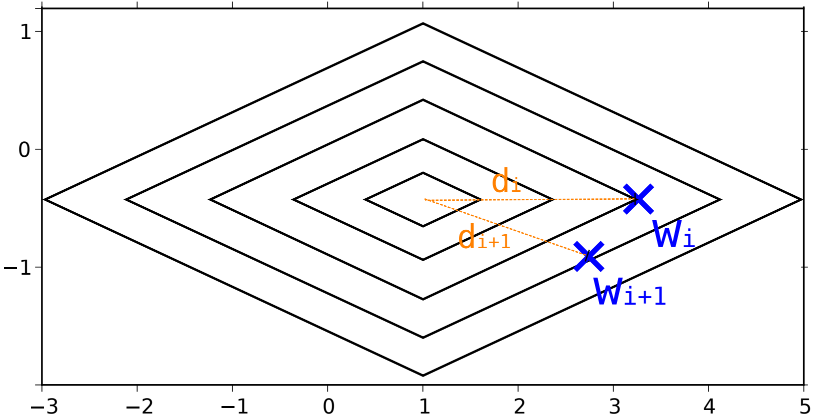

All points along subgradient have larger objective than starting point!

# Subgradient method




All points along subgradient have larger objective than starting point!

# Subgradient method




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

# 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

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

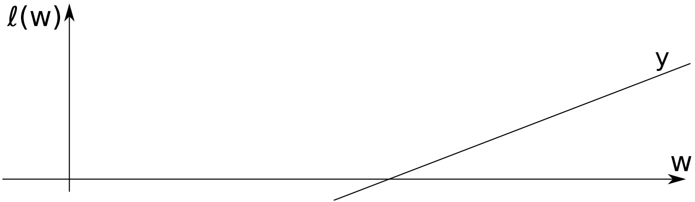
# Solving S-SVM Training Numerically – Subgradient Method

**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

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




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

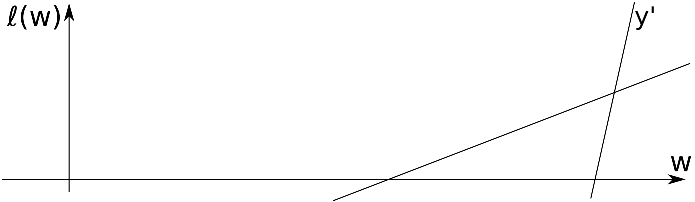
# Solving S-SVM Training Numerically – Subgradient Method

**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

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




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

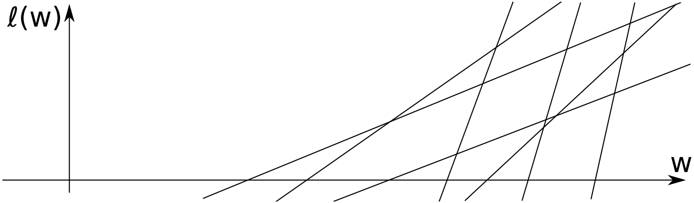
# Solving S-SVM Training Numerically – Subgradient Method

**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

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




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

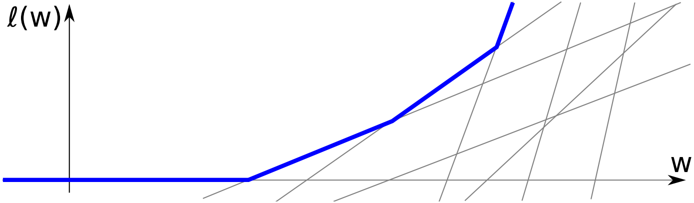
# Solving S-SVM Training Numerically – Subgradient Method

**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

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




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

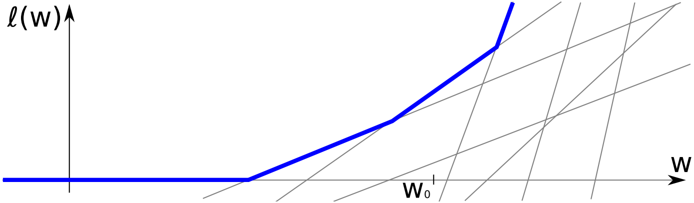
# Solving S-SVM Training Numerically – Subgradient Method

**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

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




Subgradient of $\ell^n$ at $w_0$:

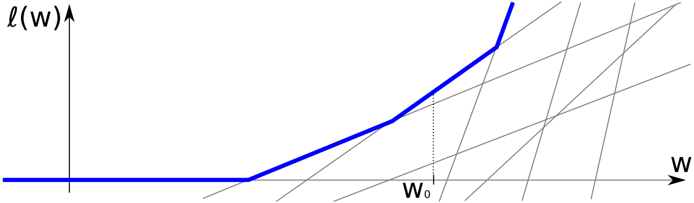
# Solving S-SVM Training Numerically – Subgradient Method

**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

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




Subgradient of $\ell^n$ at $w_0$: find maximal (active) $y$.

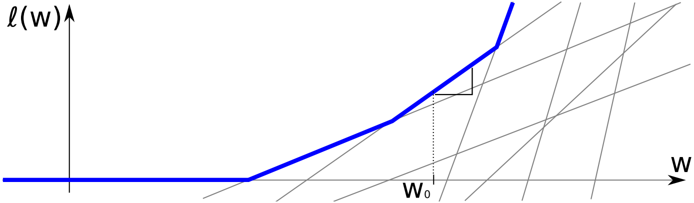
# Solving S-SVM Training Numerically – Subgradient Method

**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

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




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

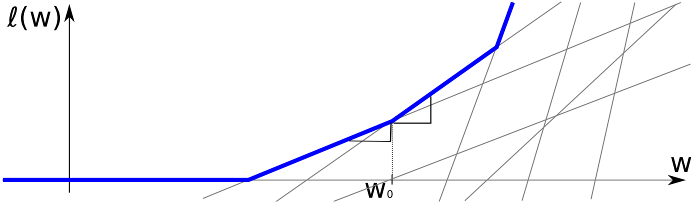
# Solving S-SVM Training Numerically – Subgradient Method

**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 $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
**input** feature map $\phi(x, y)$, loss function $\Delta(y, y')$, regularizer $\lambda$,
**input** number of iterations $T$, stepsizes $\eta_t$ for $t = 1, \ldots, T$

$w \leftarrow \vec{0}$
**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): $(x^n, y^n) =$ ( ,  )
- $\Delta(y, \bar{y}) = \sum_p [\![ y_p \neq \bar{y}_p ]\!]$ (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)

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

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

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

# 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 $-$

# 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 $-$

# 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): $(x^n, y^n) = \Big($  ,  $\Big)$
- $\Delta(y, \bar{y}) = \sum_p [\![ y_p \neq \bar{y}_p ]\!]$ (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 $=$

## 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.

# 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 $\lambda$,
**input** number of iterations $T$, stepsizes $\eta_t$ for $t = 1, \dots, T$

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

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

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

# 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?

# Solving S-SVM Training Numerically

**Structured Support Vector Machine:**

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

Subgradient method converges slowly. Can we do better?

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

# Solving S-SVM Training Numerically

**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,

# Solving S-SVM Training Numerically

**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 $y \in \mathcal{Y}$,

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

"Quadratic program":

- quadratic objective ☺
- linear constraints ☺
- (same structure as an ordinary support vector machine)

## 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 $y \in \mathcal{Y}$ ,

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

"Quadratic program":

- quadratic objective ☺
- linear constraints ☺
- (same structure as an ordinary support vector machine)

**Question:** Can we use an ordinary QP or SVM 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 $y \in \mathcal{Y}$,

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

"Quadratic program":

- quadratic objective ☺
- linear constraints ☺
- (same structure as an ordinary support vector machine)

**Question:** Can we use an ordinary QP or SVM solver?

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

- *E.g.* 100 binary $16 \times 16$ 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 $\epsilon$-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 $\lambda$

$w \leftarrow 0$, $S \leftarrow \emptyset$
**repeat**
    $(w, \xi) \leftarrow$ *solution to QP only with constraints from S*
    **for** i=1,...,n **do**
        $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \; \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle$
        **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.

# Dual S-SVM

We can also dualize the S-SVM optimization:

$$\max_{\alpha \in \mathbb{R}^{N|\mathcal{Y}|}} \quad -\frac{1}{2} \sum_{\substack{y,\bar{y} \in \mathcal{Y} \\ n,\bar{n}=1,\ldots,N}} \alpha_{ny} \alpha_{\bar{n}\bar{y}} \langle \phi(x^n, y), \phi(x^{\bar{n}}, \bar{y}) \rangle + \sum_{\substack{n=1,\ldots,N \\ y \in \mathcal{Y}}} \alpha_{ny} \Delta(y^n, y)$$

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

$$\alpha_{ny} \geq 0, \qquad \text{and} \qquad \sum_{y \in \mathcal{Y}} \alpha_{ny} \leq \frac{2}{\lambda N}.$$

Quadratic (convex) objective, linear constraints, $N|\mathcal{Y}|$ unknowns

# Dual S-SVM

We can also dualize the S-SVM optimization:

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

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

$$\alpha_{ny} \geq 0, \qquad \text{and} \qquad \sum_{y \in \mathcal{Y}} \alpha_{ny} \leq \frac{2}{\lambda N}.$$

Quadratic (convex) objective, linear constraints, $N|\mathcal{Y}|$ unknowns

Recover weight vector from dual coefficients: $w = \sum_{n,\alpha} \alpha_{ny} \phi(x^n, y)$

Some current state-of-the-art methods work solve the dual: [Lacoste-Julien et al. ICML 2013], [Shah et al. CVPR 2015]

# Summary – S-SVM Learning

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

## Summary – S-SVM Learning

- 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

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

# 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 – Structured Prediction and Learning

## Structured Prediction and 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 $\text{argmax}_y\langle w, \phi(x,y)\rangle$ 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!