Part 5: Structured Support Vector Machines

Sebastian Nowozin and Christoph H. Lampert

Colorado Springs, 25th June 2011

Microsoft Research
IST AUSTRIA

Problem (Loss-Minimizing Parameter Learning)

Let $d(x, y)$ *be the (unknown) true data distribution.*
Let $\mathcal{D} = \{(x^1, y^1), \ldots, (x^N, y^N)\}$ *be i.i.d. samples from* $d(x, y)$*.*
Let $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ *be a feature function.*
Let $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ *be a loss function.*

- *Find a weight vector* w^* *that leads to minimal expected loss*

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

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

Problem (Loss-Minimizing Parameter Learning)

Let $d(x, y)$ *be the (unknown) true data distribution.*
Let $\mathcal{D} = \{(x^1, y^1), \ldots, (x^N, y^N)\}$ *be i.i.d. samples from* $d(x, y)$.
Let $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ *be a feature function.*
Let $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ *be a loss function.*

- *Find a weight vector* w^* *that leads to minimal expected loss*

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

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

Pro:

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

Con:

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

Reminder: learning by regularized risk minimization

For compatibility function $g(x, y; w) := \langle w, \phi(x, y) \rangle$ find w^* that minimizes

$$\mathbb{E}_{(x,y) \sim d(x,y)} \; \Delta(\; y, \operatorname{argmax}_y g(x, y; w) \;).$$

Two major problems:

- $d(x, y)$ is unknown
- $\operatorname{argmax}_y g(x, y; w)$ maps into a discrete space
  $\rightarrow \Delta(\; y, \operatorname{argmax}_y g(x, y; w))$ is discontinuous, piecewise constant

Task:

$$\min_w \quad \mathbb{E}_{(x,y)\sim d(x,y)} \, \Delta(\, y, \operatorname{argmax}_y g(x,y;w) \,).$$

Problem 1:

- $d(x,y)$ 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. $\lambda\|w\|^2$.

New task:

$$\min_w \quad \lambda\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \Delta(\, y^n, \operatorname{argmax}_y g(x^n,y;w) \,).$$

Task:

$$\min_w \quad \lambda \|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \Delta(\ y^n, \operatorname{argmax}_y g(x^n, y; w)\ ).$$

Problem:

- $\Delta(\ y, \operatorname{argmax}_y g(x, y; w)\ )$ 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:

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

Regularized Risk Minimization

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

Regularization + *Loss on training data*

Regularized Risk Minimization

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

Regularization + *Loss on training data*

Hinge loss: maximum margin training

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

Regularized Risk Minimization

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

Regularization + *Loss on training data*

Hinge loss: maximum margin training

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

- ℓ is maximum over linear functions $\rightarrow$ *continuous*, *convex*.
- ℓ bounds Δ from above.

  Proof: Let $\bar{y} = \operatorname{argmax}_y g(x^n, y, w)$

$$\begin{aligned}\Delta(y^n, \bar{y}) &\leq \Delta(y^n, \bar{y}) + g(x^n, \bar{y}, w) - g(x^n, y^n, w) \\ &\leq \max_{y\in\mathcal{Y}} \left[\Delta(y^n, y) + g(x^n, y, w) - g(x^n, y^n, w)\right]\end{aligned}$$

Regularized Risk Minimization

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

Regularization + *Loss on training data*

Hinge loss: maximum margin training

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

Alternative:

Logistic loss: probabilistic training

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

Structured Output Support Vector Machine

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

Conditional Random Field

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

CRFs and SSVMs have more in common than usually assumed.

- both do regularized risk minimization
- $\log\sum_y \exp(\cdot)$ can be interpreted as a *soft-max*

Solving the Training Optimization Problem Numerically

Structured Output Support Vector Machine:

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

Unconstrained optimization, convex, non-differentiable objective.

Structured Output SVM (equivalent formulation):

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

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

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

N non-linear contraints, convex, differentiable objective.

Structured Output SVM (also equivalent formulation):

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

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

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

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

Example: Multiclass SVM

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

Solve: $$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \xi^n$$

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

$$\langle w, \phi(x^n, y^n)\rangle - \langle w, \phi(x^n, y)\rangle \geq 1 - \xi^n \quad \text{for all } y \in \mathcal{Y} \setminus \{y^n\}.$$

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

Crammer-Singer Multiclass SVM

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

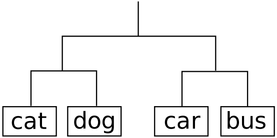
Example: Hierarchical SVM

Hierarchical Multiclass Loss:

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

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

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



Solve:
$$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \xi^n$$

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

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

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

Solving the Training Optimization Problem Numerically

We can solve SSVM training like CRF training:

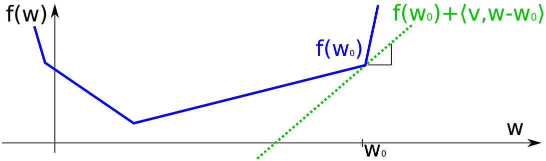
$$\min_w \; \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N}\Big[\max_{y\in\mathcal{Y}} \; \Delta(y^n,y) + \langle w,\phi(x^n,y)\rangle - \langle w,\phi(x^n,y^n)\rangle\Big]$$

- 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

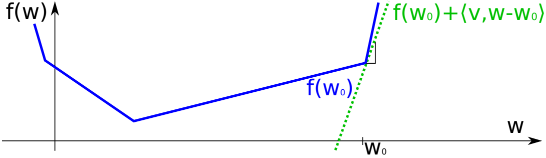
$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



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

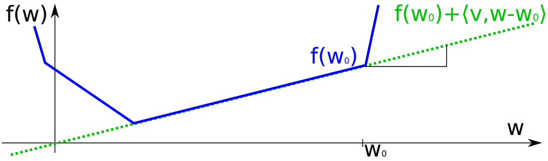
$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



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

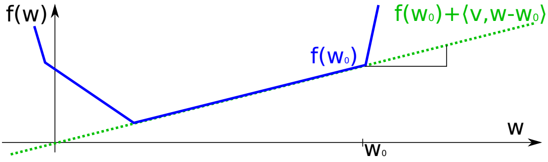
$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



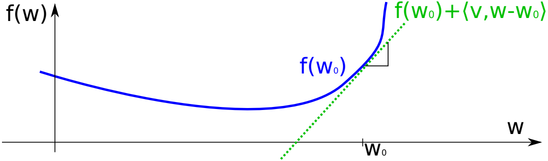
Definition

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 descent works basically like gradient descent:

Subgradient Descent Minimization – minimize $F(w)$

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

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

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

Computing a subgradient:

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

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

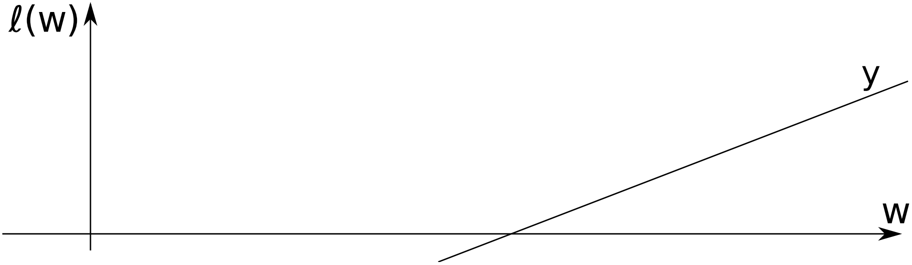
$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$

Computing a subgradient:

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

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

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



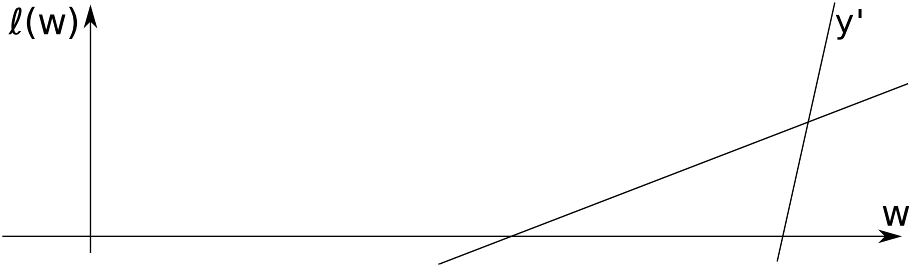
For each $y \in \mathcal{Y}$, $\ell_y(w)$ is a linear function.

Computing a subgradient:

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

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

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



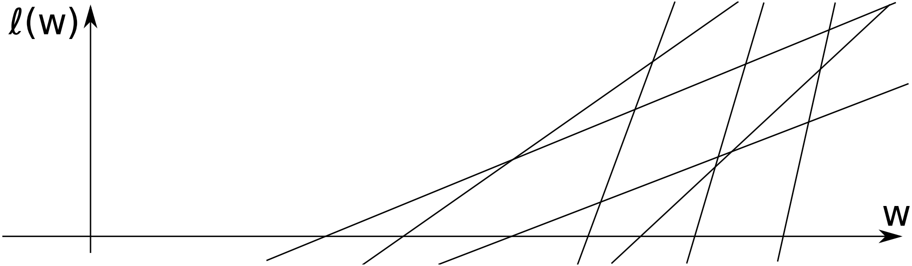
For each $y \in \mathcal{Y}$, $\ell_y(w)$ is a linear function.

Computing a subgradient:

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

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

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



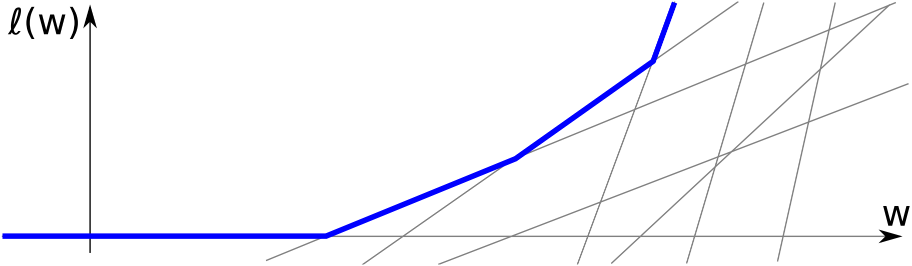
For each $y \in \mathcal{Y}$, $\ell_y(w)$ is a linear function.

Computing a subgradient:

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

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

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



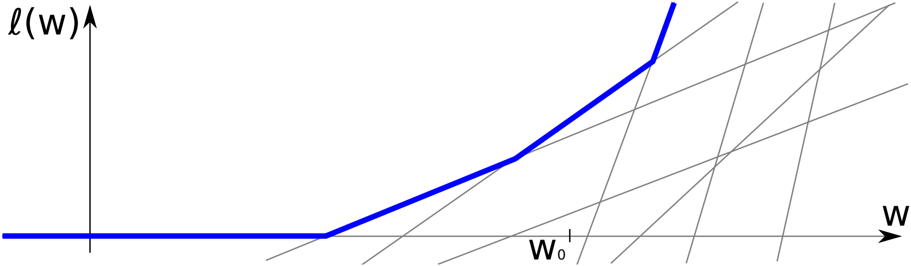
$\ell(w) = \max_y \ell_y(w)$: maximum over all $y \in \mathcal{Y}$.

Computing a subgradient:

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

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

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



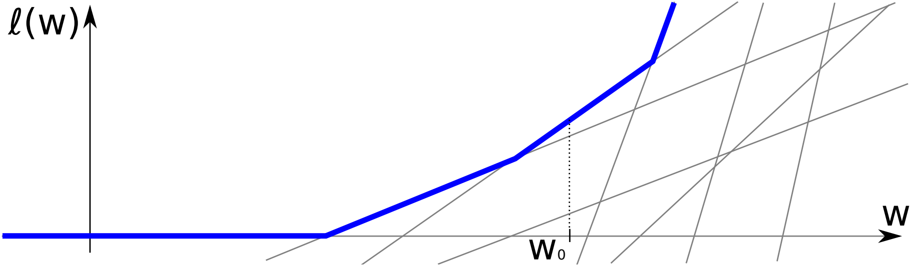
Subgradient of ℓ^n at w_0:

Computing a subgradient:

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

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

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



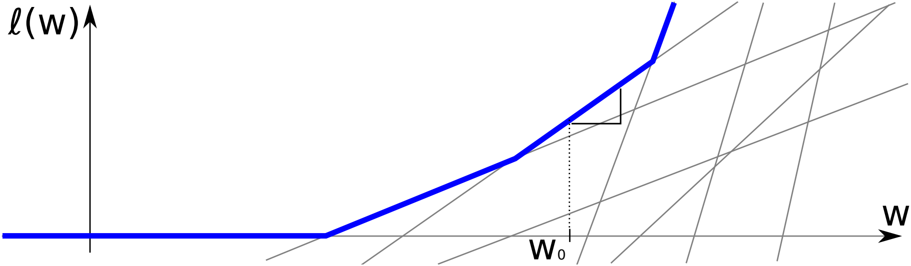
Subgradient of ℓ^n at w_0: find maximal (active) y.

Computing a subgradient:

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

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

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



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

Subgradient Descent 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 C,
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**
    $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle - \langle w, \phi(x^n, y^n) \rangle$
    $v^n \leftarrow \phi(x^n, \hat{y}) - \phi(x^n, y^n)$
  end for
  $w \leftarrow w - \eta_t (w - \frac{C}{N} \sum_n v^n)$
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 1 argmax-prediction per example.

We can use the same tricks as for CRFs, e.g. **stochastic updates**:

Stochastic Subgradient Descent 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 C,
input number of iterations T, stepsizes η_t for $t = 1, \dots, T$

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

Solving the Training Optimization Problem Numerically

We can solve an S-SVM like a linear SVM:

One of the equivalent formulations was:

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

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

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

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

Solve

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

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

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

This has the same structure as an ordinary SVM!

- quadratic objective ☺
- linear constraints ☺

Solve

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

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

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

This has the same structure as an ordinary SVM!

- quadratic objective ☺
- linear constraints ☺

Question: Can't we use a ordinary SVM/QP solver?

Solve

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

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

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

This has the same structure as an ordinary SVM!

- quadratic objective ☺
- linear constraints ☺

Question: Can't we use a 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

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:

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:

Working Set Training

- 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*.

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:

Working Set Training

- 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

[Tsochantaridis et al. "Large Margin Methods for Structured and Interdependent Output Variables", JMLR, 2005.]

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 C

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

Observation: each update of w needs 1 argmax-prediction per example.
(but we solve globally for next w, not by local steps)

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

$$\min_{w\in\mathbb{R}^D,\xi\in\mathbb{R}_+} \quad \frac{1}{2}\|w\|^2 + C\xi$$

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

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

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

$$\min_{w\in\mathbb{R}^D,\xi\in\mathbb{R}_+} \quad \frac{1}{2}\|w\|^2 + C\xi$$

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

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

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

We blew up the constraint set even further:

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

Working Set One-Slack 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 C

1: $S \leftarrow \emptyset$
2: **repeat**
3: $\quad (w, \xi) \leftarrow$ *solution to QP only with constraints from* S
4: $\quad$ **for** i=1,. . . ,n **do**
5: $\quad\quad \hat{y}^n \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle$
6: $\quad$ **end for**
7: $\quad S \leftarrow S \cup \{\big((x^1, \ldots, x^n), (\hat{y}^1, \ldots, \hat{y}^n)\big)\}$
8: **until** S doesn't change anymore.

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

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

We can solve an S-SVM like a non-linear SVM: compute Lagrangian dual

- $\min$ becomes $\max$,
- original (primal) variables w, ξ disappear,
- new (dual) variables α_{iy}: one per constraint of the original problem.

Dual S-SVM problem

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

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

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

N linear contraints, convex, differentiable objective, $N|\mathcal{Y}|$ **variables**.

We can **kernelize**:

- Define joint kernel function $k : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$

$$k(\,(x,y)\,,\,(\bar{x},\bar{y})\,) = \langle \phi(x,y), \phi(\bar{x},\bar{y})\rangle.$$

- k measure similarity between two *(input,output)*-pairs.

- We can express the optimization in terms of k:

$$\begin{aligned}
\langle \delta\phi(x^n,y^n,y)\,,\delta\phi(x^{\bar{n}},y^{\bar{n}},\bar{y})\rangle \\
&= \big\langle\, \phi(x^n,y^n) - \phi(x^n,y)\ ,\ \phi(x^{\bar{n}},y^{\bar{n}}) - \phi(x^{\bar{n}},\bar{y})\,\big\rangle \\
&= \langle\, \phi(x^n,y^n), \phi(x^{\bar{n}},y^{\bar{n}})\,\rangle - \langle\, \phi(x^n,y^n), \phi(x^{\bar{n}},\bar{y})\,\rangle \\
&\quad - \langle\, \phi(x^n,y), \phi(x^{\bar{n}},y^{\bar{n}})\rangle + \langle\, \phi(x^n,y), \phi(x^{\bar{n}},\bar{y})\rangle \\
&= k(\,(x^n,y^n),(x^{\bar{n}},y^{\bar{n}})\,) - k(\,(x^n,y^n),\phi(x^{\bar{n}},\bar{y})\,) \\
&\quad - k(\,(x^n,y),(x^{\bar{n}},y^{\bar{n}})\,) + k(\,(x^n,y),\phi(x^{\bar{n}},\bar{y})\,) \\
&=: K_{i\bar{\imath}y\bar{y}}
\end{aligned}$$

Kernelized S-SVM problem:

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

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

$$\sum_{y \in \mathcal{Y}} \alpha_{iy} \leq \frac{C}{N}.$$

- too many variables: train with **working set** of α_{iy}.

Kernelized prediction function:

$$f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \sum_{iy'} \alpha_{iy'} k(\,(x_i, y_i), (x, y)\,)$$

What do "joint kernel functions" look like?

$$k(\,(x,y)\,,\,(\bar{x},\bar{y})\,) = \langle \phi(x,y), \phi(\bar{x},\bar{y}) \rangle.$$

As in **graphical model:** easier if ϕ decomposes w.r.t. factors:

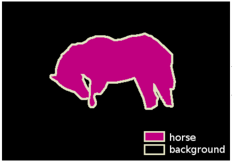
- $\phi(x,y) = \big(\phi_F(x,y_F)\big)_{F\in\mathcal{F}}$

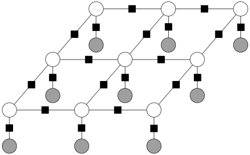
Then the kernel k decomposes into sum over *factors*:

$$\begin{aligned} k(\,(x,y)\,,\,(\bar{x},\bar{y})\,) &= \Big\langle\, \big(\phi_F(x,y_F)\big)_{F\in\mathcal{F}}, \big(\phi_F(x',y'_F)\big)_{F\in\mathcal{F}} \,\Big\rangle \\ &= \sum_{F\in\mathcal{F}} \langle\, \phi_F(x,y_F), \phi_F(x',y'_F)\,\rangle \\ &= \sum_{F\in\mathcal{F}} k_F(\,(x,y_F),(x',y'_F)\,) \end{aligned}$$

We can define kernels for each factor (e.g. nonlinear).

Example: figure-ground segmentation with grid structure

$(x, y) \hat{=} ($  , 

$)$ 

Typical kernels: arbirary in x, linear (or at least simple) w.r.t. y:

- Unary factors:

$$k_p((x_p, y_p), (x'_p, y'_p)) = k(x_p, x'_p)[\![y_p = y'_p]\!]$$

with $k(x_p, x'_p)$ local image kernel, e.g. χ^2 or *histogram intersection*

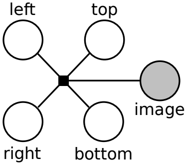
- Pairwise factors:

$$k_{pq}((y_p, y_q), (y'_p, y'_p)) = [\![y_q = y'_q]\!]\,[\![y_q = y'_q]\!]$$

More powerful than all-linear, and argmax-prediction still possible.

Example: object localization

$(x, y) \hat{=} ($  ,  $)$



Only one factor that includes all x and y:

$$k(\,(x,y)\,,\,(x',y')\,) = k_{image}(x|_y, x'|_{y'})$$

with k_{image} image kernel and $x|_y$ is image region within box y.

argmax-prediction as difficult as object localization with k_{image}-SVM.

Summary – S-SVM Learning

Given:

- training set $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$
- loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.

Task: learn parameter w for $f(x) := \mathrm{argmax}_y \langle w, \phi(x, y) \rangle$ that minimizes expected loss on future data.

Summary – S-SVM Learning

Given:

- training set $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$
- loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.

Task: learn parameter w for $f(x) := \operatorname{argmax}_y \langle w, \phi(x, y) \rangle$ that minimizes expected loss on future data.

S-SVM solution derived by *maximum margin* framework:

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

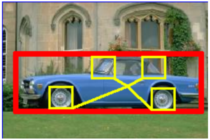
$$\langle w, \phi(x^n, y^n) \rangle \geq \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle \quad \text{for all } y \in \mathcal{Y}.$$

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

Extra I: Beyond Fully Supervised Learning

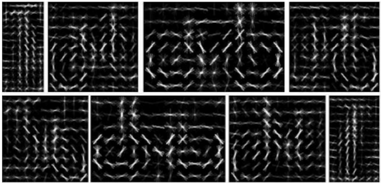
So far, training was *fully supervised*, all variables were observed.
In real life, some variables are *unobserved* even during training.

missing labels in training data



latent variables, e.g. part location

latent variables, e.g. part occlusion



latent variables, e.g. viewpoint

Three types of variables:

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

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

Three types of variables:

- $x \in \mathcal{X}$ 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:
$$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \xi^n$$

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

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

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

[C. Yu, T. Joachims, "Learning Structural SVMs with Latent Variables", ICML, 2009]
similar idea: [Felzenszwalb, McAllester, Ramaman. A Discriminatively Trained, Multiscale, Deformable Part Model, CVPR'08]

Structured Learning is full of Open Research Questions

- How to train faster?
  - CRFs need many runs of probablistic inference,
  - SSVMs need many runs of argmax-predictions.
- How to reduce the necessary amount of training data?
  - semi-supervised learning? transfer learning?
- How can we better understand different loss function?
  - when to use *probabilistic training*, when *maximum margin*?
  - CRFs are "consistent", SSVMs are not. Is this relevant?
- Can we understand structured learning with approximate inference?
  - often computing $\nabla\mathcal{L}(w)$ or $\mathrm{argmax}_y\langle w, \phi(x,y)\rangle$ *exactly* is infeasible.
  - can we guarantee good results even with approximate inference?
- More and new applications!

Lunch-Break

Continuing at 13:30

Slides available at
`http://www.nowozin.net/sebastian/cvpr2011tutorial/`