Combinatorial Optimization in Computer Vision (IN2245)

Frank R. Schmidt
Csaba Domokos

Winter Semester 2015/2016

23. Loss minimizing parameter learning 2 / 31

Parameter learning 3 / 31

Parameter learning

Learning graphical models (from training data) is a way to find among a large class of possible models a single one that is *best* in some sense for the task at hand.

We assume a fixed underlying graphical model with **parameterized conditional probability distribution**

$$p(y \mid x, w) = \frac{1}{Z(x,w)} \exp(-E(x,y,w)) = \frac{1}{Z(x,w)} \exp(-\langle w, \varphi(x,y)\rangle) ,$$

where $Z(x,w) = \sum_{y \in \mathcal{Y}} \exp(-\langle w, \varphi(x,y)\rangle)$. The only unknown quantity is the *parameter vector* w, on which the energy $E(x,y,w)$ depends linearly.

Learning tasks

Let $d(y \mid x)$ be the (*unknown*) conditional distribution of labels for a problem to be solved. For a parameterized conditional distribution $p(y \mid x, w)$ with parameters $w \in \mathbb{R}^D$, **probabilistic parameter learning** is the task of finding a point estimate of the parameter w^* that minimizes the **expected dissimilarity** of $p(y \mid x, w^*)$ and $d(y \mid x)$:

$$\mathsf{KL}_{\mathsf{tot}}(p \| d) = \sum_{x \in \mathcal{X}} d(x) \sum_{y \in \mathcal{Y}} d(y \mid x) \log \frac{d(y \mid x)}{p(y \mid x, w)} .$$

Let $d(x,y)$ be the unknown distribution of data in labels, and let $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ be a loss function. **Loss minimizing parameter learning** is the task of finding a parameter value w^* such that the *expected prediction loss*

$$\mathbb{E}_{(x,y) \sim d(x,y)}[\Delta(y, f_p(x))]$$

is as small as possible, where $f_p(x) = \operatorname{argmax}_{y \in \mathcal{Y}} p(y \mid x, w^*)$.

Regularized Maximum Conditional Likelihood Training

Let $p(y \mid x, w) = \frac{1}{Z(x,w)} \exp(-\langle w, \varphi(x, y)\rangle)$ be a *probability distribution parameterized by* $w \in \mathbb{R}^D$, and let $\mathcal{D} = \{(x^n, y^n)\}_{n=1,\ldots,N}$ be a set of *i.i.d. training examples*. For any $\lambda > 0$, **regularized maximum conditional likelihood** training chooses the parameter as

$$w = \operatorname*{argmin}_{w \in \mathbb{R}^D} L(w) = \operatorname*{argmin}_{w \in \mathbb{R}^D} \lambda \|w\|^2 + \sum_{n=1}^{N} \langle w, \varphi(x^n, y^n)\rangle + \sum_{n=1}^{N} \log Z(x^n, w) \; .$$

For $\lambda = 0$ the simplified rule is given by

$$w = \operatorname*{argmin}_{w \in \mathbb{R}^D} \sum_{n=1}^{N} \langle w, \varphi(x^n, y^n)\rangle + \sum_{n=1}^{N} \log Z(x^n, w) \; .$$

Numerical solution

$$\nabla_w L(w) = 2\lambda w + \sum_{n=1}^{N} \big(\varphi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n, w)}[\varphi(x^n, y)]\big) \; .$$

In a naive way, the complexity of the gradient computation is $\mathcal{O}(K^M N D)$.

$$\lambda \|w\|^2 + \sum_{n=1}^{N} \langle w, \varphi(x^n, y^n)\rangle + \sum_{n=1}^{N} \log Z(x^n, w) \; .$$

In a naive way, the complexity of a line search is $\mathcal{O}(K^M N D)$ (for each evaluation of L), where

- N is the number of samples,
- D is the dimension of weight vector,
- $M = |\mathcal{V}|$ is number of output nodes, and
- $K = \max_{i \in \mathcal{V}} |\mathcal{Y}_i|$ is (maximal) number of possible labels of each output nodes.

Stochastic gradient descent

If the training set $\mathcal{D}$ is too large, one can create a random subset $\mathcal{D}' \subset \mathcal{D}$ and estimate the gradient $\nabla_w L(w)$ on $\mathcal{D}'$. In an extreme case, one may randomly select only **one** sample and calculate the gradient

$$\tilde{\nabla}_w^{(x^n,y^n)} L(w) = 2\lambda w + \varphi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n,w)}[\varphi(x^n, y)] \, .$$

This approach is called **stochastic gradient descent** (SGD). Note that line search is not possible, therefore, we need for an extra parameter, referred to as step-size η_t for each iteration.

Stochastic gradient descent

Input: Step-sizes $\eta_1, \dots, \eta_T$ for all the T iterations.
Output: The learned weight vector $w \in \mathbb{R}^D$.

```
w ← 0
for t = 1, ..., T do
    (x^n, y^n) ← randomly chosen training example pair
    d ← −∇̃_w^{(x^n,y^n)} L(w)
    w ← w + η_t d
end for
return w
```

If the step-size is chosen correctly (e.g., $\eta_t = \frac{1}{t}$), then SGD converges to $\operatorname{argmin}_{w \in \mathbb{R}^D} L(w)$. However, it needs more iterations, but each one is much faster.

Using of the output structure

Assume the set of factors $\mathcal{F}$ in a graphical model representation, such that $\varphi(x,y)$ decomposes as $\varphi(x,y) = [\varphi_F(x_F, y_F)]_{F\in\mathcal{F}}$. Thus

$$\begin{aligned}\mathbb{E}_{y\sim p(y|x,w)}[\varphi(x,y)] =&[\mathbb{E}_{y\sim p(y|x,w)}[\varphi_F(x_F,y_F)]]_{F\in\mathcal{F}}\\ =&[\mathbb{E}_{y_F\sim p(y_F|x_F,w)}[\varphi_F(x_F,y_F)]]_{F\in\mathcal{F}}\ ,\end{aligned}$$

where

$$\mathbb{E}_{y_F\sim p(y_F|x_F,w)}[\varphi_F(x_F,y_F)] = \sum_{y_F\in\mathcal{Y}_F} p(y_F \mid x_F, w)\varphi_F(x_F,y_F)\,.$$

Factor marginals $\mu_F = p(y_F \mid x_F, w)$ are generally (much) easier to calculate than the complete joint distribution $p(y \mid x, w)$.

They can be either computed exactly (e.g., by applying Belief propagation yielding $\mathcal{O}(K^2MND)$) or approximated. In general, the approximation yields $\mathcal{O}(K^{|F_{\mathsf{max}}|}MND)$, where $|F_{\mathsf{max}}|$ is the maximal factor size.

Gradient approximation via sampling

$$\nabla_w L(w) = 2\lambda w + \sum_{n=1}^{N}\left(\varphi(x^n,y^n) - \mathbb{E}_{y\sim p(y|x^n,w)}[\varphi(x^n,y)]\right)\ .$$

We have seen that the computationally demanding part in the gradient computation has the form of the *expectation* of $\varphi(x,y)$ with respect to the distribution $p(y \mid x, w)$.

If we have a method to obtain *i.i.d* samples $\{y^{(1)},\dots,y^{(S)}\}$ from this distribution, we can form an estimator

$$\mathbb{E}_{y\sim p(y|x^n,w)}[\varphi(x^n,y)] \approx \frac{1}{S}\sum_{i=1}^{S}\varphi(x^n,y^{(i)})\ .$$

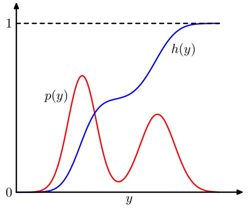
Inserting this into $\nabla_w L$, the law of large numbers guarantees convergence of the approximation to the exact gradient. Consequently, any procedure to sample from $p(y \mid x^n, w)$ for $n = 1,\dots,N$ provides us with a tool for estimating $\nabla_w L$.

Basic sampling

Let Z be a uniformly distributed random variable on the interval $[0,1]$ and $h(y)$ be a continuous and strictly monotonic cumulative distributive function. Then

$$Y = h^{-1}(Z)$$

is a random variable with cumulative distributive function (cdf.) $h(y)$, where $h^{-1}(y)$ is the inverse of $h(y)$.



The cdf. of the uniformly distributed Z is given by

$$F(z) = \begin{cases} 0, & \text{if } z \leqslant 0 \\ z, & \text{if } 0 < z \leqslant 1 \\ 1, & \text{if } 1 < z \end{cases}$$

Therefore, the cdf. of Y is given by

$$P(Y < y) = P(h^{-1}(Z) < y) = P(Z < h(y)) = F(h(y)) = h(y) \ .$$

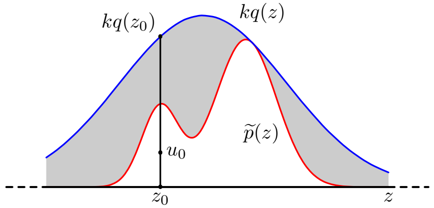
Rejection sampling

Suppose we wish to sample from a distribution $p(z)$ that can be a relatively complex distributions, and that sampling directly from $p(z)$ is *difficult*.

Furthermore suppose that we are easily able to evaluate $p(z)$ for any given value of z, up to some normalizing constant Z, so that

$$p(z) = \frac{1}{Z_p}\tilde{p}(z) \ ,$$

where $\tilde{p}(z)$ can readily be evaluated, but Z_p is unknown.

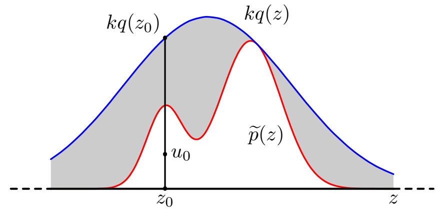


We need some simpler distribution $q(z)$, called a **proposal distribution**, from which we can readily draw samples. Let k a constant such that $kq(z) \geqslant \tilde{p}(z)$ for all values of z.

Rejection sampling

1. Generate a sample z_0 from the distribution $q(z)$.
2. Generate a sample $u_0 \sim \mathcal{U}(0, kq(z_0))$.

This pair of random samples has uniform distribution under the curve of the function $kq(z)$.



If $u_0 > \tilde{p}(z_0)$ then the sample is *rejected*, otherwise u_0 is retained. Note that the remaining pairs then have uniform distribution under the curve of $\tilde{p}(z)$.

The values of z are generated from $q(z)$, and these samples are accepted with probability $\tilde{p}(z)/kq(z)$, therefore

$$p(\text{accept}) = \int \frac{\tilde{p}(z)}{kq(z)} q(z) \mathrm{d}z = \frac{1}{k} \int \tilde{p}(z) \mathrm{d}z \ .$$

Metropolis-Hasting algorithm*

Input: $\tilde{p}(y \mid x, w) \propto p(y \mid x, w)$, unnormalized target distribution and $q(y' \mid y)$, proposal distribution
Output: $y^{(t)}$, sequence of samples with approximately $y^{(t)} \sim p(y \mid x, w)$

$y^0 \leftarrow$ arbitrary in $\mathcal{Y}$
for $t = 1, \dots, T$ **do**
  $y^{(t)} \sim q(y' \mid y^{(t-1)})$ ▷ Generate candidate
  $\sigma \leftarrow \min\left(1, \frac{\tilde{p}(y' \mid x, w) q(y^{(t-1)} \mid y')}{\tilde{p}(y^{(t-1)} \mid x, w) q(y' \mid y^{(t-1)})}\right)$ ▷ Compute accept. prob.
  $y^{(t)} \leftarrow \begin{cases} y' & \text{with probability } \sigma \text{ (accept)} \\ y^{(t1)} & \text{otherwise (reject)} \end{cases}$ ▷ Update
  output $y^{(t)}$
end for

Loss-minimizing parameter learning

Let $\mathcal{D} = \{(x^1, y^1), \ldots, (x^N, y^N)\}$ be *i.i.d.* samples from the (unknown) *true data distribution* $d(x, y)$ and $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ be a *loss function*. The task is to find a weight vector w that leads to **minimal expected loss**

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

for a *prediction function* $f(x) = \operatorname{argmax}_{y\in\mathcal{Y}}\, g(x, y; w)$, where $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ is an auxiliary function that is parameterized by $w \in \mathbb{R}^D$.

Pros:

- We directly optimize for the *quantity of interest*, i.e. the expected loss.
- We do not need to compute the *partition function* Z.

Cons:

- There is no probabilistic reasoning to find w.
- We need to know the *loss function* already at training time.

Regularized loss minimization

Let us define the auxiliary function $g(x, y; w) := \langle w, \varphi(x, y) \rangle$. We aim to find the parameter w^* that minimizes

$$\mathbb{E}_{(x,y)\sim d(x,y)}[\Delta(y, \operatorname*{argmax}_{y \in \mathcal{Y}} g(x, y; w))] \ .$$

However, $d(x, y)$ is unknown, hence we apply *approximation*:

$$\mathbb{E}_{(x,y)\sim d(x,y)}[\Delta(y, \operatorname*{argmax}_{y \in \mathcal{Y}} g(x, y; w))] \approx \frac{1}{N} \sum_{n=1}^{N} \Delta(y^n, \operatorname*{argmax}_{y \in \mathcal{Y}} g(x^n, y^n; w)) \ .$$

Moreover, we add the **regularizer** $\lambda \|w\|^2$ in order to avoid *overfitting*.

Therefore, we get a new objective, that is

$$w^* \in \operatorname*{argmin}_{w \in \mathbb{R}^D} \lambda \|w\|^2 + \frac{1}{N} \sum_{n=1}^{N} \Delta(y^n, \operatorname*{argmax}_{y \in \mathcal{Y}} g(x^n, y^n; w)) \ .$$

Redefining the loss function

$$w^* \in \operatorname*{argmin}_{w \in \mathbb{R}^D} \lambda \|w\|^2 + \frac{1}{N} \sum_{n=1}^{N} \Delta(y^n, \operatorname*{argmax}_{y \in \mathcal{Y}} g(x^n, y^n; w)) \ .$$

Note that the loss function $\Delta(y, \operatorname{argmax}_{y \in \mathcal{Y}} g(x, y; w))$ is piecewise constant, hence it is **discontinuous**, hence we cannot use gradient-based techniques.

As a remedy we will *replace* $\Delta(y, y')$ with well behaved $\ell(x, y; w)$, i.e. it is continuous and convex with respect to w.

Typically, ℓ is chosen such that it is an upper bound to Δ. Basically, by making use of ℓ instead of Δ, it is still possible to achieve an optimal prediction accuracy in the limit of infinite data.

Therefore, we get a new objective, that is

$$w^* \in \operatorname*{argmin}_{w \in \mathbb{R}^D} \lambda \|w\|^2 + \frac{1}{N} \sum_{n=1}^{N} \ell(x^n, y^n; w) \ .$$

Hinge loss

The **hinge loss** is defined as

$$\begin{aligned} \ell(x^n, y^n, w) &\stackrel{\Delta}{=} \max_{y \in \mathcal{Y}} (\Delta(y^n, y) + g(x^n, y; w) - g(x^n, y^n; w)) \\ &= \max_{y \in \mathcal{Y}} (\Delta(y^n, y) + \langle w, \varphi(x^n, y) \rangle - \langle w, \varphi(x^n, y^n) \rangle) \ . \end{aligned}$$

ℓ is continuous and convex, since it is a maximum over linear functions.

The *hinge loss* ℓ provides an upper bound for the *loss function* Δ. To see this, let $\bar{y} = \operatorname{argmax}_{y \in \mathcal{Y}} g(x^n, y; w)$, then

$$\begin{aligned} \Delta(y^n, \bar{y}) &\leqslant \Delta(y^n, \bar{y}) + g(x^n, \bar{y}; w) - g(x^n, y^n; w) \\ &\leqslant \max_{y \in \mathcal{Y}} (\Delta(y^n, y) + g(x^n, y; w) - g(x^n, y^n; w)) \\ &= \ell(x^n, y^n, w) \ . \end{aligned}$$

Structured Support Vector Machine

Let $g(x, y; w) = \langle w, \varphi(x, y)\rangle$ be an auxiliary function parameterized by $w \in \mathbb{R}^D$. For any $C > 0$, **structured support vector machine** (S-SVM) training chooses the parameter

$$w^* \in \operatorname*{argmin}_{w \in \mathbb{R}^D} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \ell(x^n, y^n, w)$$

with

$$\ell(x^n, y^n, w) = \max_{y \in \mathcal{Y}} (\Delta(y^n, y) + \langle w, \varphi(x^n, y)\rangle - \langle w, \varphi(x^n, y^n)\rangle) \ .$$

Both CRF and S-SVM do *regularized risk minimization*. For CRF models, the *regularized conditional log-likelihood function* can be written as:

$$w^* \in \operatorname*{argmin}_{w \in \mathbb{R}^D} \frac{\|w\|^2}{2\sigma^2} + \sum_{n=1}^{N} \log \sum_{y \in \mathcal{Y}} \exp\left(\langle w, \varphi(x^n, y)\rangle - \langle w, \varphi(x^n, y^n)\rangle\right) \ .$$

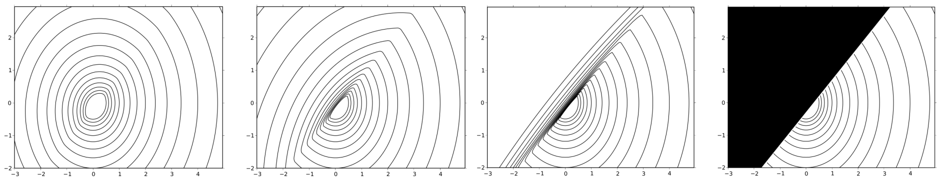
S-SVM: Toy example

Consider a simple CRF model with a single variable, where $\mathcal{Y} = \{-1, +1\}$. We define the energy function as

$$E(x, y, w) = w_1\varphi_1(x, y) + w_2\varphi_2(x, y) \ .$$

Assuming a training set $\mathcal{D} = \{(-10, +1), (-4, +1), (6, -1), (5, -1)\}$ with

$$\varphi_1(x, y) = \begin{cases} 0, & \text{if } y = -1 \\ x, & \text{if } y = +1 \end{cases} \quad \text{and} \quad \varphi_2(x, y) = \begin{cases} x, & \text{if } y = -1 \\ 0, & \text{if } y = +1 \end{cases} \ .$$



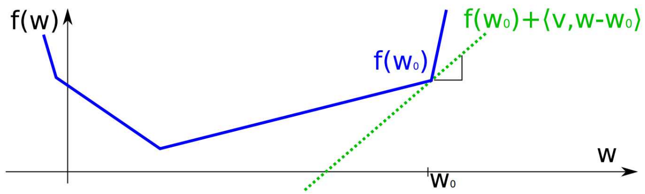
(a) $C = 0.01$ (b) $C = 0.10$ (c) $C = 1.00$ (d) $C \to \infty$

$\frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \max_{y \in \mathcal{Y}}(\Delta(y^n, y) + \langle w, \varphi(x^n, y)\rangle - \langle w, \varphi(x^n, y^n)\rangle)$.

Subgradient

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

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



Note that for differentiable f, the gradient $v = \nabla f(w_0)$ is the **only** subgradient.

Subgradient descent minimization

Subgradient descent methods work basically like gradient descent ones.

Input: Tolerance $\epsilon > 0$ and step-sizes η_t.
Output: The minimizer w of F.

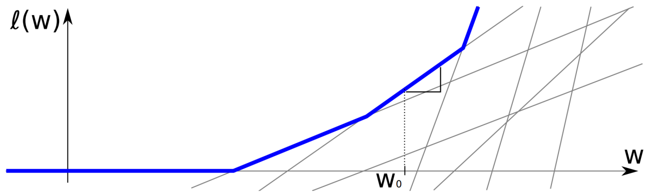
$w \leftarrow \mathbf{0}$
repeat
    $v \in \nabla_w^{\mathsf{sub}} F(w)$
    $w \leftarrow w - \eta_t v$
until F changed less than ϵ
return w

Converges to global minimum, but rather inefficient if the objective function F is non-differentiable.

Numerical solution

$$\operatorname*{argmin}_{w\in\mathbb{R}^D} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \max_{y\in\mathcal{Y}}(\Delta(y^n,y) + \langle w, \varphi(x^n,y)\rangle - \langle w, \varphi(x^n,y^n)\rangle)\ .$$

As we have discussed, this function is non-differentiable. Therefore, we cannot use gradient descent directly, so we have to use subgradients.



For each $y \in \mathcal{Y}$, ℓ is a linear function, since it is the maximum over all $y \in \mathcal{Y}$. In order to calculate the subgradient at w_0, one may find the maximal (active) y, and then use $v = \nabla\ell(w_0)$.

Calculating the subgradient

$$\operatorname*{argmin}_{w\in\mathbb{R}^D} \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \max_{y\in\mathcal{Y}}(\Delta(y^n,y) + \langle w, \varphi(x^n,y)\rangle - \langle w, \varphi(x^n,y^n)\rangle)\ .$$

Let $\hat{y} \in \operatorname{argmax}_{y\in\mathcal{Y}} \Delta(y^n,y) + \langle w, \varphi(x^n,y)\rangle$.
A subgradient v is given by

$$\begin{aligned}
&\nabla_w^{\mathsf{sub}}\left(\frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \max_{y\in\mathcal{Y}}(\Delta(y^n,y) + \langle w, \varphi(x^n,y)\rangle - \langle w, \varphi(x^n,y^n)\rangle)\right) \\
&\ni \nabla_w\left(\frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} (\Delta(y^n,\hat{y}) + \langle w, \varphi(x^n,\hat{y})\rangle - \langle w, \varphi(x^n,y^n)\rangle)\right) \\
&= w + \frac{C}{N}\sum_{n=1}^{N} \varphi(x^n,\hat{y} - \varphi(x^n,y^n) =: v\ .
\end{aligned}$$

Subgradient descent S-SVM learning

Input: Training set $\mathcal{D} = \{(x^1, y^1), \dots, (x^N, y^N)\}$, energies $\varphi(x, y)$, loss function $\Delta(y, y')$, regularizer C and step-sizes $\eta_1, \dots, \eta_T$ for all the T iterations.
Output: the weight vector w for the prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \varphi(x, y) \rangle$.

$w \leftarrow \mathbf{0}$
for $t = 1, \dots, T$ **do**
  for $n = 1, \dots, N$ **do**
    $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \varphi(x^n, y) \rangle$
    $v^n \leftarrow \varphi(x^n, \hat{y}) - \varphi(x^n, y^n)$
  end for
  $w \leftarrow w - \eta_t \left(w + \frac{C}{N} \sum_{n=1}^{N} v^n \right)$
end for

The step-size can be chosen as $\eta_t = \frac{1}{t}$ for all $t = 1, \dots, T$.

Note that each update of w needs only one argmax-prediction.

Stochastic subgradient descent S-SVM learning

Input: Training set $\mathcal{D} = \{(x^1, y^1), \dots, (x^N, y^N)\}$, energies $\varphi(x, y)$, loss function $\Delta(y, y')$, regularizer C and step-sizes $\eta_1, \dots, \eta_T$ for all the T iterations.
Output: The weight vector w for the prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \varphi(x, y) \rangle$.

$w \leftarrow \mathbf{0}$
for $t = 1, \dots, T$ **do**
  $(x^n, y^n) \leftarrow$ randomly chosen training example pair
  $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \varphi(x^n, y) \rangle$
  $w \leftarrow w - \eta_t \left(w + \frac{C}{N} \sum_{n=1}^{N} (\varphi(x^n, \hat{y}) - \varphi(x^n, y^n)) \right)$
end for

Note that each update step of w needs only one argmax-prediction, however we will generally need **many** iterations until convergence.

Summary of S-SVM learning

We are given a *training set* $\mathcal{D} = \{(x^1, y^1), \ldots, (x^N, y^N)\} \subset \mathcal{X} \times \mathcal{Y}$ and a problem specific *loss function* $\Delta : \mathcal{Y} \times \mathcal{Y} \leftarrow \mathbb{R}$.

The task is to *learn* parameter w for prediction function

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

that minimizes *expected loss* on *test data*.

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

$$\langle w, \varphi(x^n, y^n) \rangle \geqslant \Delta(y^n, y) + \langle w, \varphi(x^n, y) \rangle \,,$$

that is the correct output is enforced to be better than others by a margin.

We have seen that S-SVM learning ends up a convex optimization problem, but it is non-differentiable. Furthermore it requires repeated *argmax prediction*.

Literature

- Sebastian Nowozin and Christoph H. Lampert. **Structured Prediction and Learning in Computer Vision**. In *Foundations and Trends in Computer Graphics and Vision*, Volume 6, Number 3-4. Note: Chapter 5, 6.
- Christopher Bishop. **Pattern Recognition and Machine Learning**. Springer, 2006. Note: Chapter 11.