Loss-minimizing parameter learning

Loss-minimizing parameter learning
Regularized loss minimization
Digression: Support Vector Machine
Digression: Support Vector Machine
Redefining the loss function
Structured hinge loss
Structured Support Vector Machine
Subgradient
Pseudo-code of subgradient descent minimization
Subgradient descent minimization
Numerical solution
Calculating the subgradient
Subgradient descent S-SVM learning
Stochastic subgradient descent S-SVM learning
Summary of S-SVM learning
Next lecture: Summary of the course
Literature
11. Parameter learning

Agenda for today’s lecture *

Probabilistic parameter learning is the task of estimating the parameter \( w \) that minimizes the expected dissimilarity of a parameterized model distribution \( p(y \mid x, w) \) and the (unknown) conditional data distribution \( d(y \mid x) \):

\[
\text{KL}_{\text{tot}}(d \parallel p) = \sum_{x \in \mathcal{X}} d(x) \sum_{y \in \mathcal{Y}} d(y \mid x) \log \frac{d(y \mid x)}{p(y \mid x, w)}.
\]

The loss function \( \Delta : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^{+} \) measures the cost of predicting \( y' \) when the correct label is \( y \). Loss minimizing parameter learning is the task of estimating the parameter \( w \) that minimizes the expected loss:

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

where \( f(x) = \arg\max_{y \in \mathcal{Y}} p(y \mid x, w) \) is a prediction function, \( d(y \mid x) \) is the (unknown) conditional data distribution, and \( \Delta : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^{+} \) is a loss function.

Recap: 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 \( D = \{(x^{n}, y^{n})\}_{n=1}^{N} \) be a set of i.i.d. training samples. For any \( \lambda > 0 \), regularized maximum conditional likelihood training chooses the parameter \( w^{*} \), such that

\[
w^{*} \in \arg\min_{w \in \mathbb{R}^{D}} L(w)
= \arg\min_{w \in \mathbb{R}^{D}} \lambda \|w\|^{2} + \sum_{n=1}^{N} \langle w, \varphi(x^{n}, y^{n}) \rangle + \sum_{n=1}^{N} \log Z(x^{n}, w).
\]
Numerical solution

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

In a naïve way, the complexity of the gradient computation is \(O(K|V|ND)\), where

- \(N\) is the number of data samples,
- \(D\) is the dimension of weight vector,
- \(K = \max_{i \in V} |Y_i|\) is the (maximal) number of possible labels of each output variable (\(i \in V\)).

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

In a naïve way, the complexity of line search is \(O(K|V|ND)\) (for each evaluation of \(L\)).

Stochastic gradient descent

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

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

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 \(\eta_t\) for each iteration \((t = 1, \ldots, T)\).
Pseudo-code of Stochastic gradient descent

**Input:** Training set \( \{(x^n, y^n)\}_{n=1}^N \), number of iterations \( T \) and step-sizes \( \{\eta_t\}_{t=1}^T \).

**Output:** The learned weight vector \( \mathbf{w} \in \mathbb{R}^D \).

1: \( \mathbf{w} \leftarrow 0 \)
2: for \( t = 1, \ldots, T \) do
3: \((x^n, y^n) \leftarrow \) a randomly chosen training example
4: \( \mathbf{v} \leftarrow -\nabla_{\mathbf{w}} (x^n, y^n) L(\mathbf{w}) \)
5: \( \mathbf{w} \leftarrow \mathbf{w} + \eta_t \mathbf{v} \)
6: end for
7: return \( \mathbf{w} \)

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

Using of the output structure

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

\[
E_{y \sim p(y | x, \mathbf{w})}[\varphi(x, y)] = [E_{y \sim p(y | x, \mathbf{w})}[\varphi_F(x_F, y_F)]]_{F \in \mathcal{F}} \\
= [E_{y_F \sim p(y_F | x_F, \mathbf{w}_F)}[\varphi_F(x_F, y_F)]]_{F \in \mathcal{F}},
\]

where

\[
E_{y_F \sim p(y_F | x_F, \mathbf{w}_F)}[\varphi_F(x_F, y_F)] = \sum_{y_F \in y_F} p(y_F | x_F, \mathbf{w}_F) \varphi_F(x_F, y_F).
\]

Factor marginals \( \mu_F = p(y_F | x_F, \mathbf{w}_F) \) are generally (much) easier to calculate than the complete conditional distribution \( p(y | x, \mathbf{w}) \).

They can be either computed exactly (e.g., by applying belief propagation yielding complexity \( O(K^{\max} |\mathcal{V}| N D) \), where \( \max_{F \in \mathcal{F}} \mu_F = \max_{F \in \mathcal{F}} |N(F)| \) is the maximal factor size) or approximated.
Two-stage learning

The idea here is to split learning of energy functions into two steps:

1. learning unary energies via classifiers, and
2. learning their importance and the weighting factors of pairwise (and higher-order) energy functions.

\[ E(y; x) = \sum_{i \in V} w_i E_i(y_i; x_i) + \sum_{(i,j) \in E'} w_{ij} E_{ij}(y_i, y_j). \]

As an advantage, it results in a faster learning method. However, if local classifiers for \( E_i \) perform badly, then CRF learning cannot fix it.
**End-to-end training: CRF as RNN**


Evaluation on the PASCAL VOC 2012 Segmentation dataset:

<table>
<thead>
<tr>
<th></th>
<th>Unaries only</th>
<th>Fully connected CRF</th>
<th>End-to-end training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean IoU (%)</td>
<td>61.3</td>
<td>63.7</td>
<td>69.6</td>
</tr>
</tbody>
</table>

IN2329 - Probabilistic Graphical Models in Computer Vision
Piecewise learning

Assume a set of factors $F$ in a factor graph model, such that $\varphi(x, y) = [\varphi_F(x_F, y_F)]_{F \in F}$.

We now approximate $p(y \mid x, w)$ by a distribution that is a product over the factors:

$$p_{PW}(y \mid x, w) := \prod_{F \in F} p_F(y_F \mid x_F, w_F) = \prod_{F \in F} \frac{\exp(-\langle w_F, \varphi_F(x_F, y_F) \rangle)}{Z_F(x_F, w_F)}.$$  

By minimizing the negative conditional log-likelihood function $L(w)$, we get

$$w^* \in \arg\min_{w \in \mathbb{R}^D} \sum_{F \in F} \left( \lambda \|w_F\|^2 + \sum_{n=1}^{N} \langle w_F, \varphi_F(x^n_F, y^n_F) \rangle + \sum_{n=1}^{N} \log Z_F(x^n_F, w_F) \right).$$  

Consequently, piecewise training chooses the parameters $w^* = [w^*_F]_{F \in F}$ as

$$w^*_F \in \arg\min_{w_F \in \mathbb{R}} \left( \lambda \|w_F\|^2 + \sum_{n=1}^{N} \langle w_F, \varphi_F(x^n_F, y^n_F) \rangle + \sum_{n=1}^{N} \log Z_F(x^n_F, w_F) \right).$$  

One can perform gradient-based training for each factor as long as the individual factors remain small.

Comparing $p_{PW}(y \mid x, w)$ with the exact $p(y \mid x, w)$, we see that the exact $Z(w)$ does not factorize into a product of simpler terms, whereas its piecewise approximation $Z_{PW}(w)$ factorizes over the set of factors.

The simplification made by piece-wise training of CRFs resembles two-stage learning.
Piecewise learning: Deep Structured Model

Deep structured model: contextual deep CRF

Unary potential net: Multi-scale CNN
Pairwise potential net: Multi-scale CNN

Feature map (low resolution)
Generate features
Construct CRF graph
One node in CRF graph
One connection in CRF graph
CRF graph
Edge feature vector
Pairwise potential output
Unary potential output
Unary-Net
Pairwise-Net

Prediction refinement stage: up-sample & boundary refine

Coarse-level prediction stage: inference on contextual CRF

Low-resolution prediction


75.3% mean IoU on the PASCAL VOC 2012 Segmentation dataset.
Summary

Regularized maximum conditional likelihood training chooses the parameter $w^*$ for $\lambda > 0$, such that

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

The gradient might be expensive to calculate

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

- **Stochastic gradient descent**: the gradient is estimated on the subset of training samples.
- **Using of the input structure**: Factor marginals $\mu_F = p(y_F | x_F, w_F)$ are generally (much) easier to calculate than the complete conditional distribution $p(y | x, w)$

$$\mathbb{E}_{y \sim p(y|x, w)}[\varphi(x, y)] = \mathbb{E}\left[ \sum_{y_F \in Y_F} p(y_F | x_F, w_F) \varphi_F(x_F, y_F) \right]_{F \in F}.$$
Two–stage learning: first learning and fix unary energies, and then learning the weighting factors for the energy functions.

Piecewise training chooses the parameters $w^* = [w^*_F]_{F \in F}$ as

$$w^*_F \in \arg\min_{w_F \in \mathbb{R}} \lambda \|w_F\|^2 + \sum_{n=1}^N \langle w_F, \varphi_F(x^n_F, y^n_F) \rangle + \sum_{n=1}^N \log Z_F(x^n_F, w_F).$$

Loss function

The goal is to make prediction $y \in \mathcal{Y}$, as good as possible, about unobserved properties (e.g., class label) for a given data instance $x \in \mathcal{X}$.

In order to measure quality of prediction $f : \mathcal{X} \to \mathcal{Y}$ we define a loss function

$$\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+_0,$$

that is $\Delta(y, y')$ measures the cost of predicting $y'$ when the correct label is $y$.

Let us denote the model distribution by $p(y \mid x)$ and the true (conditional) data distribution by $d(y \mid x)$. The quality of prediction can be expressed by the expected loss (a.k.a. risk):

$$R^\Delta_f(x) := \mathbb{E}_{y \sim d(y \mid x)}[\Delta(y, f(x))] = \sum_{y \in \mathcal{Y}} d(y \mid x) \Delta(y, f(x))$$

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

assuming that $p(y \mid x) \approx d(y \mid x)$. 
0/1 loss *

In general, the loss function is application dependent. Arguably one of the most common loss functions for labelling tasks is the 0/1 loss, that is

\[ \Delta_{0/1}(y, y') = [y \neq y'] = \begin{cases} 0, & \text{if } y = y' \\ 1, & \text{otherwise.} \end{cases} \]

Minimizing the expected loss of the 0/1 loss yields

\[ y^* \in \arg\min_{y' \in Y} \mathbb{E}_{y \sim p(y \mid x)}[\Delta_{0/1}(y, y')] = \arg\min_{y' \in Y} \sum_{y \in Y} p(y \mid x) \Delta_{0/1}(y, y') \]

\[ = \arg\min_{y' \in Y} \sum_{y \in Y, y \neq y'} p(y \mid x) = \arg\min_{y' \in Y} (1 - p(y' \mid x)) = \arg\max_{y' \in Y} p(y' \mid x) \]

\[ = \arg\min_{y' \in Y} E(y' \mid x). \]

This shows that the optimal prediction \( f(x) = y^* \) in this case is given by MAP inference.

Hamming-loss *

Another popular choice of loss function is the Hamming-loss, which counts the percentage of mis-labeled variables:

\[ \Delta_H(y, y') = \frac{1}{|V|} \sum_{i \in V} [y_i \neq y'_i]. \]

For example, in semantic image segmentation, the Hamming-loss is proportional to the number of mis-classified pixels, whereas the 0/1 loss assigns the same cost to every labeling that is not pixel–by–pixel identical to the correct one.

The expected loss of the Hamming-loss takes the form (see Exercise)

\[ R_H^f(x) = 1 - \frac{1}{|V|} p(Y_i = f(x)_i \mid x), \]

which is minimized by predicting with \( f(x)_i = \arg\max_{y_i \in Y_i} p(Y_i = y_i \mid x). \)

To evaluate this prediction rule, we rely on probabilistic inference.
Loss-minimizing parameter learning

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

\[
    w^* \in \arg\min_{w \in \mathbb{R}^D} \mathbb{E}_{y \sim d(y \mid x)}[\Delta(y, f(x))]
\]

for a prediction function \( f(x) = \arg\max_{y \in \mathcal{Y}} g(x, y; w) \), where \( g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R} \) is an auxiliary function, which 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 as
\[ g(x, y; w) := -E(y; x, w) \overset{\Delta}{=} -\langle w, \varphi(x, y) \rangle. \]

We aim to find the parameter \( w^* \) that minimizes
\[ E_{y \sim d(y|x)}[\Delta(y, f(x))] = E_{y \sim d(y|x)}[\Delta(y, \text{argmax}_{y \in y} g(x, y; w))]. \]

However, \( d(y \mid x) \) is unknown, hence we use approximation:
\[ E_{y \sim d(y|x)}[\Delta(y, \text{argmax}_{y \in y} g(x, y; w))] \approx \frac{1}{N} \sum_{n=1}^{N} \Delta(y^n, \text{argmax}_{y \in y} g(x^n, y^n; w)). \]

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

Therefore, we get the objective
\[ w^* \in \argmin_{w \in \mathbb{R}^D} \lambda \|w\|^2 + \frac{1}{N} \sum_{n=1}^{N} \Delta(y^n, \text{argmax}_{y \in y} g(x^n, y^n; w))]. \]
Digression: Support Vector Machine

Let us consider the binary classification problem. Suppose we are given a set of labeled points \( \{(x^1, t^1), \ldots, (x^N, t^N)\} \) (i.e. a training set), where \( x^n \in \mathbb{R}^D \) and \( t^n \in \{-1, 1\} \) for all \( n = 1, \ldots, N \).

The goal is to find a hyperplane \( y(x) := \langle w, x \rangle + w_0 \) separating the input data according to their labels.

More precisely, \( y(x^n) > 0 \) for points having \( t^n = 1 \) and \( y(x^n) < 0 \) for points having \( t^n = -1 \), that is \( t^n \cdot y(x^n) \geq 1 \) for all training points.

If such a hyperplane exists, then we say the training set is linearly separable.
Digression: Support Vector Machine

We want to solve the following minimization problem:

\[ \mathbf{w}^* \in \arg\min_{\mathbf{w}} \| \mathbf{w} \|^2, \text{ subject to } t_n(\langle \mathbf{w}, \mathbf{x}^n \rangle + w_0) \geq 1, \text{ for all } n = 1, \ldots, N. \]

Since the training set is not necessarily linearly separable, instead, we consider the following minimization for \( \lambda > 0 \)

\[ \mathbf{w}^* \in \arg\min_{\mathbf{w}} \lambda \| \mathbf{w} \|^2 + \frac{1}{N} \sum_{n=1}^{N} \max(0, 1 - t_n(\langle \mathbf{w}, \mathbf{x}^n \rangle + w_0)). \]

where \( \ell(y) = \max(0, 1 - ty) \) is called the hinge loss function.
Redefining the loss function

\[ w^* \in \arg\min_{w \in \mathbb{R}^D} \lambda \|w\|^2 + \frac{1}{N} \sum_{n=1}^{N} \Delta(y^n, \arg\max_{y \in Y} g(x^n, y^n; w)) . \]

Note that the loss function \( \Delta(y, \arg\max_{y \in Y} g(x, y; w)) \) is piecewise constant, hence it is discontinuous, therefore we cannot use gradient-based techniques.

As a remedy we will replace \( \Delta(y, y') \) with a well behaved function \( \ell(x, y; w) \), which is continuous and convex with respect to \( w \).

Typically, \( \ell \) is chosen such that it is an upper bound to \( \Delta \).

Therefore, we will get a new objective, that is

\[
\begin{align*}
    w^* & \in \arg\min_{w \in \mathbb{R}^D} \lambda \|w\|^2 + \frac{1}{N} \sum_{n=1}^{N} \ell(x^n, y^n; w) \\
    & = \arg\min_{w \in \mathbb{R}^D} \frac{1}{2} \|w\|^2 + C \sum_{n=1}^{N} \ell(x^n, y^n; w) , \quad \text{with } C = \frac{1}{2\lambda} .
\end{align*}
\]
Structured hinge loss

Let $\hat{y} = \text{argmax}_{y \in Y} g(x^n, y; w)$, then we get

$$
\Delta(y^n, \hat{y}) \leq \Delta(y^n, \hat{y}) + g(x^n, \hat{y}; w) - g(x^n, y^n; w)
\leq \max_{y \in Y} \left( \Delta(y^n, y) + g(x^n, y; w) - g(x^n, y^n; w) \right)
\leq \Delta \ell(x^n, y^n, w),
$$

which is called the structured hinge loss. Note that $\ell$ provides an upper bound for the loss function $\Delta$. Moreover $\ell$ is continuous and convex, since it is a maximum over affine functions.

We remark that

$$
\ell(x^n, y^n, w) \overset{\Delta}{=} \max_{y \in Y} \left( \Delta(y^n, y) + g(x^n, y; w) - g(x^n, y^n; w) \right)
= \max \left( 0, \max_{y \in Y} \left( \Delta(y^n, y) + g(x^n, y; w) - g(x^n, y^n; w) \right) \right)
= \max_{y \in Y} \left( \Delta(y^n, y) - \langle w, \varphi(x^n, y) \rangle + \langle w, \varphi(x^n, y^n) \rangle \right).
$$
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

\[
\begin{align*}
    w^* \in \arg \min_{w \in \mathbb{R}^D} & \quad L(w) = \arg \min_{w \in \mathbb{R}^D} \frac{1}{2} \|w\|^2 + C \sum_{n=1}^{N} \ell(x^n, y^n, w) \\
\end{align*}
\]

with

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

Both probabilistic parameter learning and S-SVM do regularized risk minimization. For probabilistic parameter learning, the regularized conditional log-likelihood function can be written as \((C = \sigma^2)\):

\[
\begin{align*}
    w^* \in \arg \min_{w \in \mathbb{R}^D} & \quad \frac{1}{2} \|w\|^2 + C \sum_{n=1}^{N} \log \sum_{y \in \mathcal{Y}} \exp \left( \langle w, \varphi(x^n, y) \rangle - \langle w, \varphi(x^n, y^n) \rangle \right).
\end{align*}
\]

Subgradient

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

\[
\begin{align*}
    f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.
\end{align*}
\]

Note that for differentiable \( f \), the gradient \( v = \nabla f(w_0) \) is the only subgradient.
Pseudo-code of subgradient descent minimization *

Input: Tolerance $\epsilon > 0$ and step-sizes $\eta_t := \eta(t)$.
Output: The minimizer $w$ of $L$.

1: $w \leftarrow 0$
2: $t \leftarrow 0$
3: repeat
4: $t \leftarrow t + 1$
5: $v \in \nabla_w \text{sub} L(w)$
6: $w \leftarrow w - \eta(t)v$
7: until $L$ changed less than $\epsilon$
8: return $w$

This method converges to global minimum, but rather inefficient if the objective function $L$ is non-differentiable.

For step sizes satisfying diminishing step size conditions:

$$\lim_{t \to \infty} \eta_t = 0, \text{ and } \sum_{t=0}^{\infty} \eta_t \to \infty$$

convergence is guaranteed.

Example:

$$\eta_t = \eta(t) := \frac{1 + m}{t + m} \text{ for any } m \geq 0.$$
Numerical solution

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

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

For each \( y \in \mathcal{Y} \), \( \ell \) 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

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

Let \( \bar{y} = \arg\max_{y \in \mathcal{Y}} \Delta(y^n, y) - \langle w, \varphi(x^n, y) \rangle \). A subgradient \( v \) of \( L(w) \) is given by

\[
\nabla_{w}^{\text{sub}} \left( \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^{N} \max_{y \in \mathcal{Y}} (\Delta(y^n, y) - \langle w, \varphi(x^n, y) \rangle + \langle w, \varphi(x^n, y^n) \rangle) \right) \\
\equiv \nabla_{w} \left( \frac{1}{2} \|w\|^2 + \frac{C}{N} \sum_{n=1}^{N} (\Delta(y^n, \bar{y}) - \langle w, \varphi(x^n, \bar{y}) \rangle + \langle w, \varphi(x^n, y^n) \rangle) \right) \\
= w + \frac{C}{N} \sum_{n=1}^{N} (-\varphi(x^n, \bar{y}) + \varphi(x^n, y^n)) \\
= w + \frac{C}{N} \sum_{n=1}^{N} \varphi(x^n, y^n) - \varphi(x^n, \arg\max_{y \in \mathcal{Y}} \Delta(y^n, y) - \langle w, \varphi(x^n, y) \rangle) =: v.
\]
**Subgradient descent S-SVM learning**

**Input:** Training set \( \mathcal{D} = \{(x^n, y^n)\}_{n=1}^{N} \), energies \( \varphi(x, y) \), loss function \( \Delta : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_0^+ \), regularizer \( C \), and step-sizes \( \{\eta_t\}_{t=1}^{T} \).

**Output:** the weight vector \( w \) for the prediction function \( f(x) = \text{argmax}_{y \in \mathcal{Y}} \langle w, \varphi(x, y) \rangle \).

1. \( w \leftarrow 0 \)
2. for \( t = 1, \ldots, T \) do
3.    for \( n = 1, \ldots, N \) do
4.      \( \hat{y} \leftarrow \text{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) - \langle w, \varphi(x^n, y) \rangle \)
5.      \( v^n \leftarrow -\varphi(x^n, \hat{y}) + \varphi(x^n, y^n) \)
6.    end for
7. \( w \leftarrow w - \eta_t \left( w + \frac{C}{N} \sum_{n=1}^{N} v^n \right) \)
8. end for

The step-size can be chosen as \( \eta_t := \eta(t) = \frac{1}{t} \) for all \( t = 1, \ldots, T \). Note that each update of \( w \) needs an argmax-prediction for each training sample.

---

**Stochastic subgradient descent S-SVM learning**

**Input:** Training set \( \mathcal{D} = \{(x^1, y^1), \ldots, (x^n, y^n)\} \), energies \( \varphi(x, y) \), loss function \( \Delta : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_0^+ \), regularizer \( C \), number of iterations \( T \) and step-sizes \( \{\eta_t\}_{t=1}^{T} \).

**Output:** The weight vector \( w \) for the prediction function \( f(x) = \text{argmax}_{y \in \mathcal{Y}} \langle w, \varphi(x, y) \rangle \).

1. \( w \leftarrow 0 \)
2. for \( t = 1, \ldots, T \) do
3.    \( (x^n, y^n) \leftarrow \text{a randomly chosen training example} \)
4.    \( \hat{y} \leftarrow \text{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) - \langle w, \varphi(x^n, y) \rangle \)
5.    \( w \leftarrow w - \eta_t \left( w + \frac{C}{N} (\varphi(x^n, \hat{y}) - \varphi(x^n, y^n)) \right) \)
6. 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 $D = \{(x^1, y^1), \ldots, (x^n, y^n)\} \subset X \times Y$ and a problem specific loss function $\Delta : Y \times Y \to \mathbb{R}_0^+$. The task is to learn parameter $w$ for a prediction function

$$f(x) = \arg\max_{y \in Y} -\langle w, \varphi(x, y) \rangle = \arg\min_{y \in Y} \langle w, \varphi(x, y) \rangle$$

that minimizes expected loss on the training set.

S-SVM solution derived by the maximum margin framework:

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

that is the predicted output is enforced to be not worse than the correct one by a margin.

We have seen that S-SVM training ends up a convex optimization problem, but it is non-differentiable. Furthermore it requires repeated argmax predictions.
IN2329 - Probabilistic Graphical Models in Computer Vision

Next lecture: Summary of the course

IN2329

MRF

CRF

Bayesian network

Graphical models

Probabilistic inference

Inference

MAP inference

Human-pose estimation

Computer Vision

Object detection

Stereo matching

Image segmentation

Loss-minimizing parameter learning

Proba-bilistic parameter learning

Learning

IN2329

11. Parameter learning – 37 / 38
<table>
<thead>
<tr>
<th>Literature</th>
</tr>
</thead>
</table>