Probabilistic Graphical Models in Computer Vision (IN2329)

Csaba Domokos

Summer Semester 2017

11. Parameter learning

Agenda for today's lecture $*$

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

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

The **loss function** $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_0^+$ measures the cost of predicting $\mathbf{y}'$ when the correct label is $\mathbf{y}$. **Loss minimizing parameter learning** is the task of estimating the parameter $\mathbf{w}$ that minimizes the **expected loss**:

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

where $f(\mathbf{x}) = \operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}} p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$ is a *prediction function*, $d(\mathbf{y} \mid \mathbf{x})$ is the *(unknown)* conditional *data distribution*, and $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_0^+$ is a *loss function*.

Probabilistic parameter learning

Recap: Regularized maximum conditional likelihood training $*$

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

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

Numerical solution

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

In a naïve way, the complexity of the *gradient computation* is $\mathcal{O}(K^{|\mathcal{V}|}ND)$, where

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

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

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

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 $\nabla_{\mathbf{w}} L(\mathbf{w})$ on $\mathcal{D}'$ only. In an extreme case, one may *randomly select* only **one** sample and calculate the gradient

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

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 ($t = 1, \ldots, T$).

Pseudo-code of Stochastic gradient descent *

Input: Training set $\{(\mathbf{x}^n, \mathbf{y}^n)\}_{n=1}^N$, number of iterations T and step-sizes $\{\eta_t\}_{t=1}^T$.
Output: The learned weight vector $\mathbf{w} \in \mathbb{R}^D$.

```
1: $\mathbf{w} \leftarrow \mathbf{0}$
2: for $t = 1, \dots, T$ do
3:     $(\mathbf{x}^n, \mathbf{y}^n) \leftarrow$ a randomly chosen training example
4:     $\mathbf{v} \leftarrow -\tilde{\nabla}_{\mathbf{w}}^{(\mathbf{x}^n, \mathbf{y}^n)} L(\mathbf{w})$
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., $\eta_t := \eta(t) = \frac{\eta}{t}$), then SGD converges to $\operatorname{argmin}_{\mathbf{w} \in \mathbb{R}^D} L(\mathbf{w})$. 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 $\varphi(\mathbf{x}, \mathbf{y})$ decomposes as $\varphi(\mathbf{x}, \mathbf{y}) = [\varphi_F(\mathbf{x}_F, \mathbf{y}_F)]_{F \in \mathcal{F}}$. Thus

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

where

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

Factor marginals $\mu_F = p(\mathbf{y}_F \mid \mathbf{x}_F, \mathbf{w}_F)$ are generally (much) easier to calculate than the complete conditional distribution $p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$.
They can be either computed *exactly* (e.g., by applying *belief propagation* yielding complexity $\mathcal{O}(K^{|F_{\max}|}|\mathcal{V}|ND)$, where $|F_{\max}| = \max_{F \in \mathcal{F}} |N(F)|$ is the maximal factor size) or *approximated*.

Two-stage learning

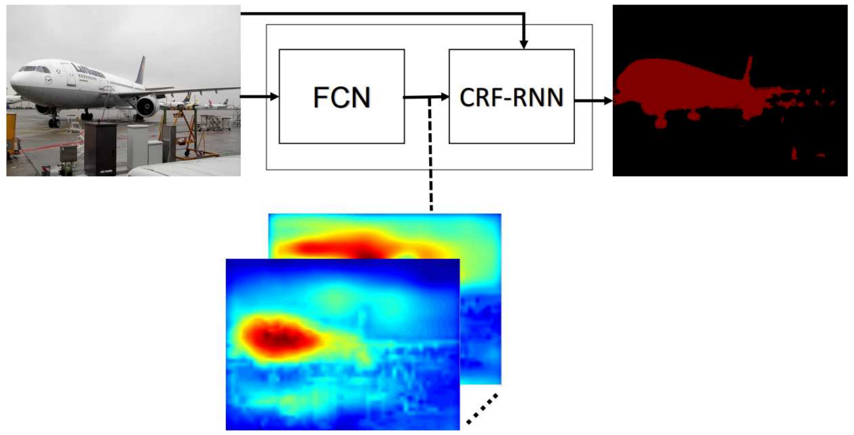
The idea here is to split learning of energy functions into two steps:

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

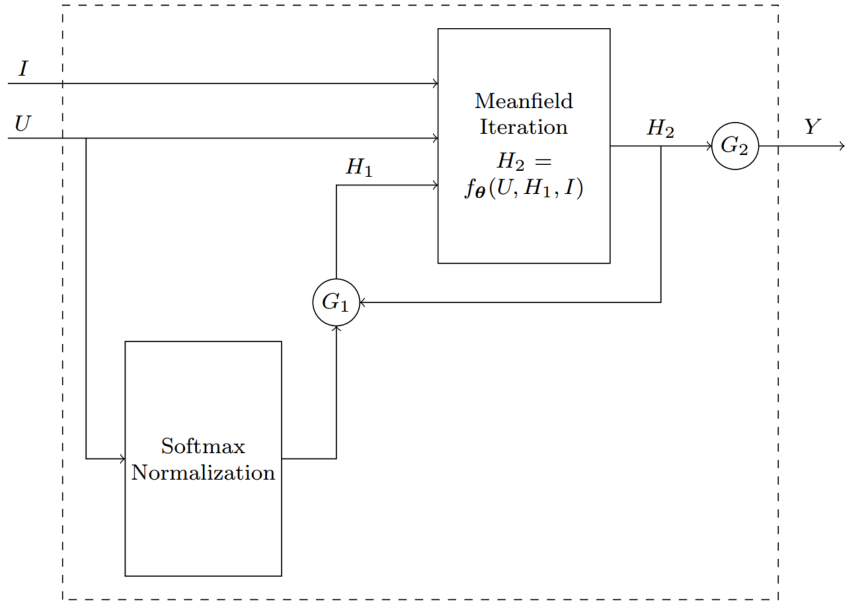
$$E(\mathbf{y};\mathbf{x}) = \sum_{i\in\mathcal{V}} w_i E_i(y_i;x_i) + \sum_{(i,j)\in\mathcal{E}'} w_{ij} E_{ij}(y_i,y_j) \ .$$

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 *



CRF as RNN network



CRF-RNN

Source: Zheng et al. Conditional Random Fields as Recurrent Neural Networks, ICCV'15.

Evaluation on the PASCAL VOC 2012 Segmentation dataset:

	Unaries only	Fully connected CRF	End-to-end training
Mean IoU (%)	61.3	63.7	69.6

Piecewise learning

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

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

$$p_{\mathsf{PW}}(\mathbf{y} \mid \mathbf{x}, \mathbf{w}) := \prod_{F \in \mathcal{F}} p_F(\mathbf{y}_F \mid \mathbf{x}_F, \mathbf{w}_F) = \prod_{F \in \mathcal{F}} \frac{\exp(-\langle \mathbf{w}_F, \varphi_F(\mathbf{x}_F, \mathbf{y}_F)\rangle)}{Z_F(\mathbf{x}_F, \mathbf{w}_F)} .$$

By minimizing the *negative conditional log-likelihood function* $L(\mathbf{w})$, we get

$$\begin{aligned}\mathbf{w}^* \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^D} L(\mathbf{w}) &\approx \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^D} \lambda \|\mathbf{w}\|^2 - \sum_{n=1}^N \log \prod_{F \in \mathcal{F}} p_F(\mathbf{y}_F^n \mid \mathbf{x}_F^n, \mathbf{w}_F) \\ &= \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^D} \sum_{F \in \mathcal{F}} \Big(\lambda \|\mathbf{w}_F\|^2 + \sum_{n=1}^N \langle \mathbf{w}_F, \varphi_F(\mathbf{x}_F^n, \mathbf{y}_F^n)\rangle + \sum_{n=1}^N \log Z_F(\mathbf{x}_F^n, \mathbf{w}_F) \Big) .\end{aligned}$$

Piecewise learning

$$\mathbf{w}^* \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^D} \sum_{F \in \mathcal{F}} \Big(\lambda \|\mathbf{w}_F\|^2 + \sum_{n=1}^N \langle \mathbf{w}_F, \varphi_F(\mathbf{x}_F^n, \mathbf{y}_F^n)\rangle + \sum_{n=1}^N \log Z_F(\mathbf{x}_F^n, \mathbf{w}_F) \Big) .$$

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

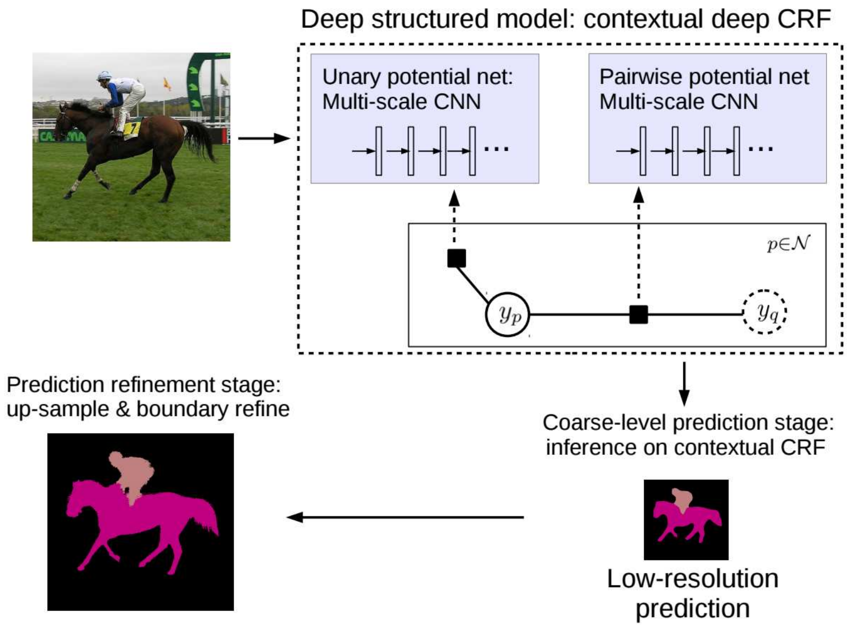
$$\mathbf{w}_F^* \in \operatorname*{argmin}_{\mathbf{w}_F \in \mathbb{R}} \lambda \|\mathbf{w}_F\|^2 + \sum_{n=1}^N \langle \mathbf{w}_F, \varphi_F(\mathbf{x}_F^n, \mathbf{y}_F^n)\rangle + \sum_{n=1}^N \log Z_F(\mathbf{x}_F^n, \mathbf{w}_F) .$$

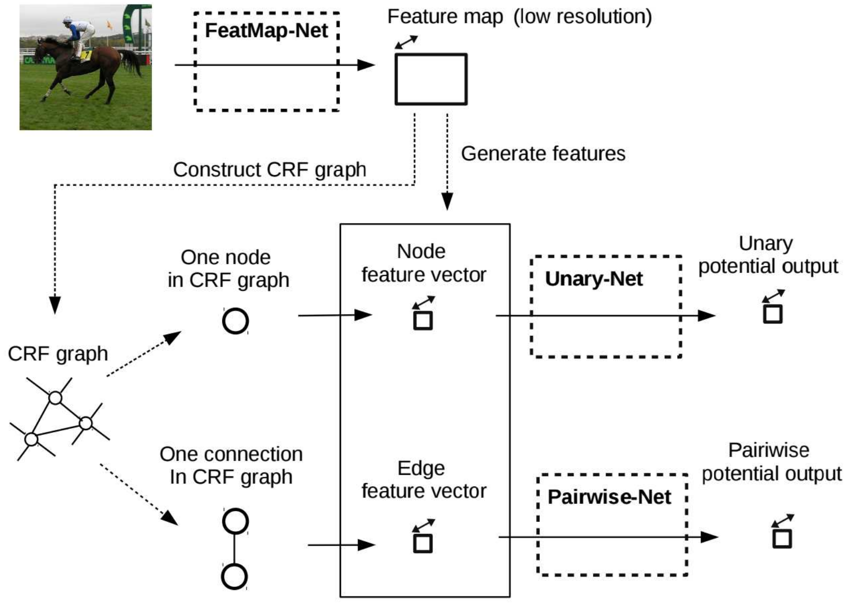
One can perform gradient-based training for each factor as long as the individual factors remain small.

Comparing $p_{\mathsf{PW}}(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$ with the exact $p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$, we see that the exact $Z(\mathbf{w})$ does not factorize into a product of simpler terms, whereas its piecewise approximation $Z_{\mathsf{PW}}(\mathbf{w})$ 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
$p \in \mathcal{N}$
y_p
y_q
Prediction refinement stage: up-sample & boundary refine
Coarse-level prediction stage: inference on contextual CRF
Low-resolution prediction



Source: Lin et al.. Efficient Piecewise Training of Deep Structured Models for Semantic Segmentation, CVPR'16.

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

Summary *

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

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

The gradient might be expensive to calculate

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

- **Stochastic gradient descent**: the gradient is estimated on the **subset** of *training samples*.
- **Using of the input structure**: *Factor marginals* $\mu_F = p(\mathbf{y}_F \mid \mathbf{x}_F, \mathbf{w}_F)$ are generally (much) easier to calculate than the complete conditional distribution $p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})$ $$\mathbb{E}_{\mathbf{y} \sim p(\mathbf{y} \mid \mathbf{x}, \mathbf{w})} [\varphi(\mathbf{x}, \mathbf{y})] = \mathbb{E}[\sum_{\mathbf{y}_F \in \mathcal{Y}_F} p(\mathbf{y}_F \mid \mathbf{x}_F, \mathbf{w}_F) \varphi_F(\mathbf{x}_F, \mathbf{y}_F)]_{F \in \mathcal{F}} \; .$$

Summary cont'd *

- **Two–stage learning**: first learning and fix unary energies, and then learning the weighting factors for the energy functions.
- **Piecewise training** chooses the parameters $\mathbf{w}^* = [\mathbf{w}_F^*]_{F \in \mathcal{F}}$ as

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

Loss function

Loss function

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

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

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

$$\begin{aligned} \mathcal{R}_f^{\Delta}(\mathbf{x}) :=& \mathbb{E}_{\mathbf{y} \sim d(\mathbf{y}|\mathbf{x})}[\Delta(\mathbf{y}, f(\mathbf{x}))] = \sum_{\mathbf{y} \in \mathcal{Y}} d(\mathbf{y} \mid \mathbf{x}) \Delta(\mathbf{y}, f(\mathbf{x})) \\ \approx& \mathbb{E}_{\mathbf{y} \sim p(\mathbf{y}|\mathbf{x})}[\Delta(\mathbf{y}, f(\mathbf{x}))] \ , \end{aligned}$$

assuming that $p(\mathbf{y} \mid \mathbf{x}) \approx d(\mathbf{y} \mid \mathbf{x})$.

$0/1$ **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

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

Minimizing the *expected loss* of the $0/1$ *loss* yields

$$\begin{aligned} \mathbf{y}^* &\in \operatorname*{argmin}_{\mathbf{y}' \in \mathcal{Y}} \mathbb{E}_{\mathbf{y} \sim p(\mathbf{y}|\mathbf{x})}[\Delta_{0/1}(\mathbf{y}, \mathbf{y}')] = \operatorname*{argmin}_{\mathbf{y}' \in \mathcal{Y}} \sum_{\mathbf{y} \in \mathcal{Y}} p(\mathbf{y} \mid \mathbf{x}) \Delta_{0/1}(\mathbf{y}, \mathbf{y}') \\ &= \operatorname*{argmin}_{\mathbf{y}' \in \mathcal{Y}} \sum_{\mathbf{y} \in \mathcal{Y},\ \mathbf{y} \neq \mathbf{y}'} p(\mathbf{y} \mid \mathbf{x}) = \operatorname*{argmin}_{\mathbf{y}' \in \mathcal{Y}} \big(1 - p(\mathbf{y}' \mid \mathbf{x})\big) = \operatorname*{argmax}_{\mathbf{y}' \in \mathcal{Y}} p(\mathbf{y}' \mid \mathbf{x}) \\ &= \operatorname*{argmin}_{\mathbf{y}' \in \mathcal{Y}} E(\mathbf{y}'; \mathbf{x}) \ . \end{aligned}$$

This shows that the *optimal* prediction $f(\mathbf{x}) = \mathbf{y}^*$ 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:

$$\Delta_{\mathsf{H}}(\mathbf{y}, \mathbf{y}') = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} [\![y_i \neq y_i']\!] \ .$$

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)

$$\mathcal{R}_f^{\mathsf{H}}(\mathbf{x}) = 1 - \frac{1}{|\mathcal{V}|} p(Y_i = f(\mathbf{x})_i \mid \mathbf{x}) \ ,$$

which is minimized by predicting with $f(\mathbf{x})_i = \operatorname{argmax}_{y_i \in \mathcal{Y}_i} p(Y_i = y_i \mid \mathbf{x})$.
To evaluate this prediction rule, we rely on **probabilistic inference**.

Loss-minimizing parameter learning

Loss-minimizing parameter learning

Let $\mathcal{D} = \{(\mathbf{x}^1, \mathbf{y}^1), \dots, (\mathbf{x}^N, \mathbf{y}^N)\} \subseteq \mathcal{X} \times \mathcal{Y}$ be a set of *i.i.d.* samples from the (unknown) *data distribution* $d(\mathbf{y} \mid \mathbf{x})$ and $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_0^+$ be a *loss function*. The task is to find a weight vector $\mathbf{w}^*$ that leads to **minimal expected loss**, that is

$$\mathbf{w}^* \in \operatorname*{argmin}_{\mathbf{w} \in \mathbb{R}^D} \mathbb{E}_{\mathbf{y} \sim d(\mathbf{y}|\mathbf{x})}[\Delta(\mathbf{y}, f(\mathbf{x}))]$$

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

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 $\mathbf{w}$.
- We need to know the *loss function* already at training time.

Regularized loss minimization

Let us define the *auxiliary function* as

$$g(\mathbf{x},\mathbf{y};\mathbf{w}) := -E(\mathbf{y};\mathbf{x},\mathbf{w}) \triangleq -\langle \mathbf{w}, \varphi(\mathbf{x},\mathbf{y})\rangle \ .$$

We aim to find the parameter $\mathbf{w}^*$ that minimizes

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

However, $d(\mathbf{y} \mid \mathbf{x})$ is unknown, hence we use *approximation*:

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

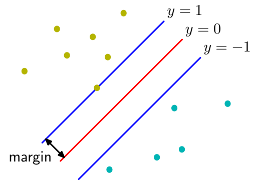
Moreover, we add the **regularizer** $\lambda\|\mathbf{w}\|^2$ in order to avoid *overfitting*.

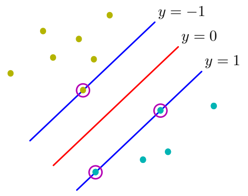
Therefore, we get the objective

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

Digression: Support Vector Machine *

Let us consider the **binary classification** problem. Suppose we are given a set of labeled points $\{(\mathbf{x}^1, t^1), \dots, (\mathbf{x}^N, t^N)\}$ (i.e. a *training set*), where $\mathbf{x}^n \in \mathbb{R}^D$ and $t^n \in \{-1, 1\}$ for all $n = 1, \dots, N$.
The *goal* is to find a hyperplane $y(\mathbf{x}) := \langle \mathbf{w}, \mathbf{x} \rangle + w_0$ separating the input data according to their labels.



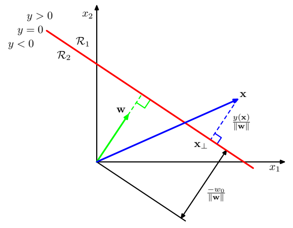


Source: C. Bishop. PRML, 2016

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

If such a hyperplane exists, then we say the *training set* is **linearly separable**.

Digression: Support Vector Machine *



Source: C. Bishop. PRML, 2016

We want to solve the following minimzation problem:

$$\mathbf{w}^* \in \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{w}\|^2 \ , \ \text{subject to } t^n(\langle \mathbf{w}, \mathbf{x}^n \rangle + w_0) \geqslant 1 \ , \ \text{for all } n = 1, \dots, N \ .$$

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

$$\mathbf{w}^* \in \underset{\mathbf{w}}{\operatorname{argmin}} \lambda \|\mathbf{w}\|^2 + \frac{1}{N} \sum_{n=1}^{N} \max(0, 1 - t^n(\langle \mathbf{w}, \mathbf{x}^n \rangle + w_0)) \ .$$

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

Redefining the loss function

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

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

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

Typically, ℓ is chosen such that it is an **upper bound** to Δ.

Therefore, we will get a new objective, that is

$$\begin{aligned}\mathbf{w}^* &\in \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^D} \lambda\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ell(\mathbf{x}^n, \mathbf{y}^n; \mathbf{w}) \\ &= \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^D} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{N}\sum_{n=1}^{N} \ell(\mathbf{x}^n, \mathbf{y}^n; \mathbf{w}) \ , \quad \text{with } C = \frac{1}{2\lambda} \ .\end{aligned}$$

Structured hinge loss

Let $\bar{\mathbf{y}} = \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} g(\mathbf{x}^n, \mathbf{y}; \mathbf{w})$, then we get

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

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

We remark that

$$\begin{aligned}\ell(\mathbf{x}^n, \mathbf{y}^n, \mathbf{w}) &\triangleq \max_{\mathbf{y}\in\mathcal{Y}} (\Delta(\mathbf{y}^n, \mathbf{y}) + g(\mathbf{x}^n, \mathbf{y}; \mathbf{w}) - g(\mathbf{x}^n, \mathbf{y}^n; \mathbf{w})) \\ &= \max\left(0, \max_{\mathbf{y}\in\mathcal{Y}} (\Delta(\mathbf{y}^n, \mathbf{y}) + g(\mathbf{x}^n, \mathbf{y}; \mathbf{w}) - g(\mathbf{x}^n, \mathbf{y}^n; \mathbf{w}))\right) \\ &= \max_{\mathbf{y}\in\mathcal{Y}} (\Delta(\mathbf{y}^n, \mathbf{y}) - \langle \mathbf{w}, \varphi(\mathbf{x}^n, \mathbf{y})\rangle + \langle \mathbf{w}, \varphi(\mathbf{x}^n, \mathbf{y}^n)\rangle) \ .\end{aligned}$$

Structured Support Vector Machine

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

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

with

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

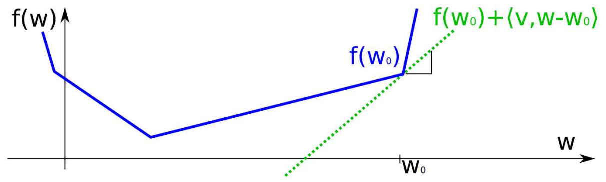
Both probabilistic parameter learning and S-SVM do **regularized risk minimization**. For probabilistic parameter learning, the *regularized conditional log-likelihood function* can be written as $(C = \sigma^2)$:

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

Subgradient *

Let $f : \mathbb{R}^D \to \mathbb{R}$ be a convex, but not necessarily differentiable, function. A vector $\mathbf{v} \in \mathbb{R}^D$ is called a **subgradient** of f at $\mathbf{w}_0$, if

$$f(\mathbf{w}) \geqslant f(\mathbf{w}_0) + \langle \mathbf{v}, \mathbf{w} - \mathbf{w}_0 \rangle \quad \text{for all } \mathbf{w} \, .$$



Source: http://www.nowozin.net/sebastian/cvpr2011tutorial/slides/talk-ssvm.pdf

Note that for differentiable f, the gradient $\mathbf{v} = \nabla f(\mathbf{w}_0)$ is the **only** subgradient.

Pseudo-code of subgradient descent minimization *

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

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

Subgradient descent minimization *

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.

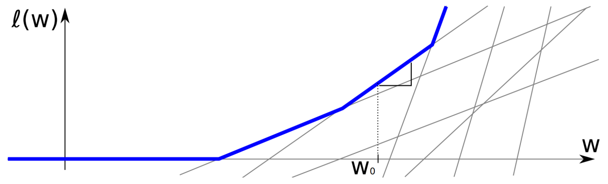
Example:

$$\eta_t = \eta(t) := \frac{1+m}{t+m} \quad \text{for any } m \geqslant 0 \, .$$

Numerical solution

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

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



Source: `http://www.nowozin.net/sebastian/cvpr2011tutorial/slides/talk-ssvm.pdf`

For each $\mathbf{y} \in \mathcal{Y}$, ℓ is a linear function, since it is the maximum over all $\mathbf{y} \in \mathcal{Y}$. In order to calculate the subgradient at $\mathbf{w}_0$, one may find the maximal (active) $\mathbf{y}$, and then use $\mathbf{v} = \nabla\ell(\mathbf{w}_0)$.

Calculating the subgradient

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

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

$$\begin{aligned}
&\nabla^{\text{sub}}_{\mathbf{w}}\Big(\frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{N}\sum_{n=1}^{N} \max_{\mathbf{y}\in\mathcal{Y}}(\Delta(\mathbf{y}^n,\mathbf{y}) - \langle \mathbf{w}, \varphi(\mathbf{x}^n,\mathbf{y})\rangle + \langle \mathbf{w}, \varphi(\mathbf{x}^n,\mathbf{y}^n)\rangle)\Big)\\
&\ni \nabla_{\mathbf{w}}\Big(\frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{N}\sum_{n=1}^{N} (\Delta(\mathbf{y}^n,\bar{\mathbf{y}}) - \langle \mathbf{w}, \varphi(\mathbf{x}^n,\bar{\mathbf{y}})\rangle + \langle \mathbf{w}, \varphi(\mathbf{x}^n,\mathbf{y}^n)\rangle)\Big)\\
&= \mathbf{w} + \frac{C}{N}\sum_{n=1}^{N} -\varphi(\mathbf{x}^n,\bar{\mathbf{y}}) + \varphi(\mathbf{x}^n,\mathbf{y}^n)\\
&= \mathbf{w} + \frac{C}{N}\sum_{n=1}^{N} \varphi(\mathbf{x}^n,\mathbf{y}^n) - \varphi(\mathbf{x}^n, \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}^n,\mathbf{y}) - \langle \mathbf{w}, \varphi(\mathbf{x}^n,\mathbf{y})\rangle) =: \mathbf{v}\ .
\end{aligned}$$

Subgradient descent S-SVM learning *

Input: Training set $\mathcal{D} = \{(\mathbf{x}^n, \mathbf{y}^n)\}_{n=1}^N$, energies $\varphi(\mathbf{x}, \mathbf{y})$, loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_0^+$, regularizer C, and step-sizes $\{\eta_t\}_{t=1}^T$.
Output: the weight vector $\mathbf{w}$ for the prediction function $f(\mathbf{x}) = \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} -\langle \mathbf{w}, \varphi(\mathbf{x}, \mathbf{y})\rangle$.

```
1: $\mathbf{w} \leftarrow \mathbf{0}$
2: for $t = 1, \ldots, T$ do
3:     for $n = 1, \ldots, N$ do
4:         $\bar{\mathbf{y}} \leftarrow \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}^n, \mathbf{y}) - \langle \mathbf{w}, \varphi(\mathbf{x}^n, \mathbf{y})\rangle$
5:         $\mathbf{v}^n \leftarrow -\varphi(\mathbf{x}^n, \bar{\mathbf{y}}) + \varphi(\mathbf{x}^n, \mathbf{y}^n)$
6:     end for
7:     $\mathbf{w} \leftarrow \mathbf{w} - \eta_t \Big( \underbrace{\mathbf{w} + \frac{C}{N} \sum_{n=1}^N \mathbf{v}^n}_{\mathbf{v}} \Big)$
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 $\mathbf{w}$ needs an argmax-prediction for each training sample.

Stochastic subgradient descent S-SVM learning *

Input: Training set $\mathcal{D} = \{(\mathbf{x}^1, \mathbf{y}^1), \ldots, (\mathbf{x}^n, \mathbf{y}^n)\}$, energies $\varphi(\mathbf{x}, \mathbf{y})$, loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_0^+$, regularizer C, number of iterations T and step-sizes $\{\eta_t\}_{t=1}^T$.
Output: The weight vector w for the prediction function $f(\mathbf{x}) = \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} -\langle \mathbf{w}, \varphi(\mathbf{x}, \mathbf{y})\rangle$.

```
1: $\mathbf{w} \leftarrow \mathbf{0}$
2: for $t = 1, \ldots, T$ do
3:     $(\mathbf{x}^n, \mathbf{y}^n) \leftarrow$ a randomly chosen training example
4:     $\bar{\mathbf{y}} \leftarrow \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} \Delta(\mathbf{y}^n, \mathbf{y}) - \langle \mathbf{w}, \varphi(\mathbf{x}^n, \mathbf{y})\rangle$
5:     $\mathbf{w} \leftarrow \mathbf{w} - \eta_t \left(\mathbf{w} + \frac{C}{N}(-\varphi(\mathbf{x}^n, \bar{\mathbf{y}}) + \varphi(\mathbf{x}^n, \mathbf{y}^n))\right)$
6: end for
```

Note that each update step of $\mathbf{w}$ 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* $\mathcal{D} = \{(\mathbf{x}^1, \mathbf{y}^1), \dots, (\mathbf{x}^n, \mathbf{y}^n)\} \subset \mathcal{X} \times \mathcal{Y}$ and a problem specific *loss function* $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_0^+$. The task is to *learn* parameter $\mathbf{w}$ for a *prediction function*

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

that minimizes *expected loss* on the *training set*.

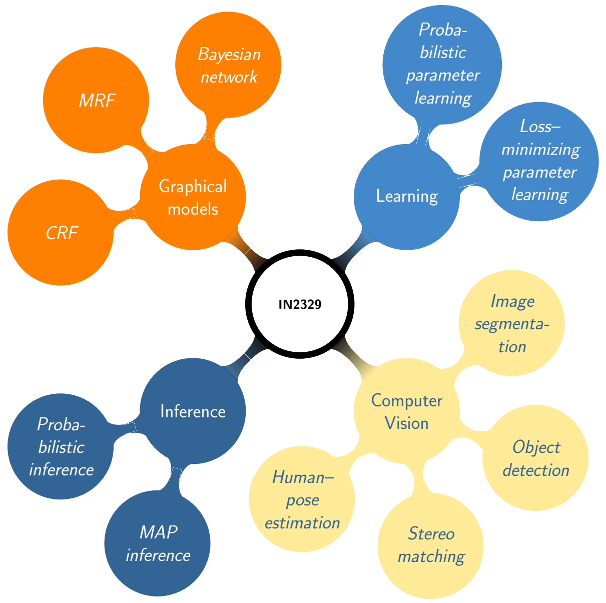
S-SVM solution derived by the *maximum margin framework*:

$$\langle \mathbf{w}, \varphi(\mathbf{x}^n, \mathbf{y}) \rangle \leqslant \langle \mathbf{w}, \varphi(\mathbf{x}^n, \mathbf{y}^n) \rangle + \Delta(\mathbf{y}^n, \mathbf{y}) \ ,$$

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*.

Next lecture: Summary of the course *



Literature *

1. Sebastian Nowozin and Christoph H. Lampert. Structured prediction and learning in computer vision. *Foundations and Trends in Computer Graphics and Vision*, 6(3–4), 2010
2. Christopher Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006
3. Shuai Zheng, Sadeep Jayasumana, Bernardino Romera-Paredes, Vibhav Vineet, Zhizhong Su, Dalong Du, Chang Huang, and Philip Torr. Conditional random fields as recurrent neural networks. In *Proceedings of International Conference on Computer Vision*, Santiago, Chile, December 2015. IEEE, IEEE
4. Guosheng Lin, Chunhua Shen, Anton van den Hengel, and Ian Reid. Efficient piecewise training of deep structured models for semantic segmentation. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas, NV, USA, 2016. IEEE