Combinatorial Optimization in Computer Vision (IN2245)

Frank R. Schmidt
Csaba Domokos

Winter Semester 2015/2016

22. Probabilistic parameter learning

Parameterization

Conditional Random Fields revisited

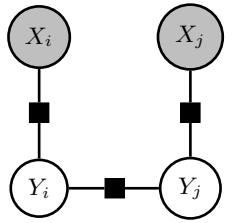
Let us suppose a set of random variables, denoted by $\{Y_i\}_{i \in \mathcal{V}}$, where $\mathcal{V}$ is a set of pixels. Moreover $Y_i \in \mathcal{Y}_i$ for all $i \in \mathcal{V}$, hence $\mathcal{Y} = \times_{i \in \mathcal{V}} \mathcal{Y}_i$. Let us assume that we have also access to *m*easurements $X = x \in \mathcal{X}$ (e.g., $\mathcal{X}$ is a set of images).

As we have already discussed (cf. Lecture 4), the *conditional probability distribution* $p(Y = y \mid X = x)$, expressed by an underlying **conditional random field** (CRF) model, can be directly modeled by a **factor graph** $G = (\mathcal{V}, \mathcal{F}, \mathcal{E})$:

$$p(y \mid x) = \frac{1}{Z(x)} \prod_{F \in \mathcal{F}} \psi_F(y_F; x_F)$$

with the **partition function** depending on x_F

$$Z(x) = \sum_{y \in \mathcal{Y}} \prod_{F \in \mathcal{F}} \psi_F(y_F; x_F) \ .$$



Shaded: the observations $X = x$.

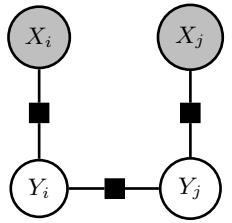
Conditional Random Fields revisited

$\mathcal{E} \subseteq \mathcal{V} \times \mathcal{F}$, and $\mathcal{F}$ encodes the *conditional independence assumption*.

$$p(y \mid x) = \frac{1}{Z(x)} \prod_{F \in \mathcal{F}} \psi_F(y_F; x_F)$$

with the **partition function** depending on x_F

$$Z(x) = \sum_{y \in \mathcal{Y}} \prod_{F \in \mathcal{F}} \psi_F(y_F; x_F) \ .$$



Shaded: The observations $X = x$.

Note that the potentials become also functions of (part of) x, i.e. $\psi_F(y_F; x_F)$ instead of just $\psi_F(y_F)$. Nevertheless, x is **not** part of the probability model, i.e. it is not treated as random variable.

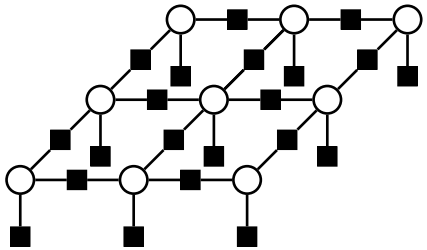
Example: Binary image segmentation

Assume that we are given a set of pixels $\mathcal{V}$. Consider the problem of *binary image segmentation* with a label set $\mathcal{L}$.

We have also defined the **energy function**

$$E(y; x) = \sum_{F \in \mathcal{F}} E_F(y_F; x_F)$$

corresponding to a CRF model $G = (\mathcal{V}, \mathcal{F}, \mathcal{E})$.



$p(y \mid x)$ is completely determined by $E(y; x)$:

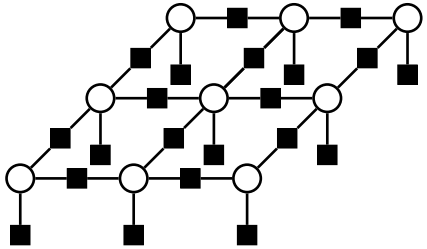
$$\begin{aligned} p(y \mid x) =& \frac{1}{Z(x)} \prod_{F \in \mathcal{F}} \psi_F(y_F; x_F) = \frac{1}{Z(x)} \exp(- \sum_{F \in \mathcal{F}} E_F(y_F; x_F)) \\ =& \frac{1}{Z(x)} \exp(-E(y; x)) \, . \end{aligned}$$

Example: Binary image segmentation

We are generally interested in a **MAP labelling** y^*:

$$y^* \in \operatorname*{argmin}_{y \in \mathcal{L}^{\mathcal{V}}} E(y; x) \ .$$

Now we assume that $\mathcal{F}$ consists of only *unary* and *pairwise* factors.



Let $\mathcal{E}'$ encode a *local neighborhood of pixels*, one can write the energy function as

$$\begin{aligned} E(y; x) = \sum_{F \in \mathcal{F}} E_F(y_F; x_F)) &= \sum_{i \in \mathcal{V}} E_i(y_i; x_i) + \sum_{(i,j) \in \mathcal{E}'} E_{ij}(y_i, y_j; x_i, x_j) \\ &= \sum_{i \in \mathcal{V}} E_i(y_i; x_i) + \sum_{(i,j) \in \mathcal{E}'} E_{ij}(y_i, y_j) \ , \end{aligned}$$

where $E_i(y_i; x_i)$ corresponds to the **data term of the pixel i, and $E_{ij}(y_i, y_j; x_i, x_j) \equiv E_{ij}(y_i, y_j)$ is a smoothness term.**

Parameterization

Instead of

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

one may want to apply weighting factors $w_1, w_2 \in \mathbb{R}_+$:

$$E(y;x,w) = w_1 \sum_{i \in \mathcal{V}} E_i(y_i; x_i) + w_2 \sum_{(i,j) \in \mathcal{E}'} E_{ij}(y_i, y_j) = \left\langle \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, \begin{bmatrix} \sum_{i \in \mathcal{V}} E_i(y_i; x_i)) \\ \sum_{(i,j \in \mathcal{E}')} E_{ij}(y_i, y_j) \end{bmatrix} \right\rangle .$$

In a more general form, one can write the *energy functions* as a **linear combination** for a **parameter vector** $w \in \mathbb{R}^D$, $D = |\mathcal{F}|$:

$$E(y;x,w) = \left\langle \begin{bmatrix} w_1 \\ \vdots \\ w_D \end{bmatrix}, \underbrace{\begin{bmatrix} E_{F_1}(y_{F_1}; x_{F_1})) \\ \vdots \\ E_{F_D}(y_{F_D}; x_{F_D})) \end{bmatrix}}_{\varphi(x,y)} \right\rangle = \langle w, \varphi(x,y) \rangle \ .$$

Loss function

The goal is to make predictions $y \in \mathcal{Y}$, *as good as possible*, about unobserved properties 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**

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

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

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

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

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

$0/1$ loss

The loss function is generally application dependent. Arguably the most common loss function for classification 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

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

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** counts the percentage of mislabeled variables:

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

For example, in pixel-wise 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 intended one.

The expected loss of the Hamming loss takes the form (see the exercise)

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

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

Learning tasks

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.

In principle each part of a graphical model (i.e. random variables, factors, and parameters) can be learned. However we assume that the model structure and parameterization are specified manually, and learning amounts to finding a vector of real-valued parameters.

Parameter Learning

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 makes $p(y \mid x, w^*)$ *closest* to $d(y \mid x)$.

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

Probabilistic parameter learning

We aim at identifying a weight vector w^* that makes $p(y \mid x, w)$ as close to the **true conditional label distribution** $d(y \mid x)$ as possible. The label distribution itself is unknown to us, but we have an *i.i.d.* sample set $\mathcal{D} = \{(x^n, y^n)\}_{n=1,\dots,N}$ from $d(x, y)$ that we can use for learning.

We now define what we mean by "closeness" between conditional distributions $p(y \mid x, w)$ and $d(x, y)$: for any $x \in \mathcal{X}$, we measure the dissimilarity by making use of **Kullback-Leibler (KL) divergence**:

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

From this we obtain a **total measure** of how much p differs from d by their **expected dissimilarity** over all $x \in \mathcal{X}$:

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

Probabilistic parameter learning

We choose the parameter w^* that minimizes expected dissimilarity, i.e.

$$\begin{aligned} w^* &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \mathsf{KL}_{\mathsf{tot}}(p \| d) = \operatorname*{argmin}_{w \in \mathbb{R}^D} \sum_{x \in \mathcal{X}} d(x) \sum_{y \in \mathcal{Y}} d(y \mid x) \log \frac{d(y \mid x)}{p(y \mid x, w)} \\ &= \operatorname*{argmax}_{w \in \mathbb{R}^D} \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} d(y \mid x) d(x) \log p(y \mid x, w) \\ &= \operatorname*{argmax}_{w \in \mathbb{R}^D} \mathbb{E}_{(x,y) \sim d(x,y)} [\log p(y \mid x, w)] \ . \end{aligned}$$

Unfortunately, we cannot compute this expression directly, because $d(x, y)$ is unknown to us. However, we can approximate it using the sample set $\mathcal{D}$.

$$\begin{aligned} &\approx \operatorname*{argmax}_{w \in \mathbb{R}^D} \sum_{(x^n, y^n) \in \mathcal{D}} \log p(y^n \mid x^n, w) = \operatorname*{argmax}_{w \in \mathbb{R}^D} \sum_{n=1}^{N} \log \frac{\exp(-\langle w, \phi(x^n, y^n) \rangle)}{Z(x^n, w)} \\ &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \sum_{n=1}^{N} \langle w, \phi(x^n, y^n) \rangle + \sum_{n=1}^{N} \log Z(x^n, w) \ . \end{aligned}$$

Maximum conditional likelihood

By making use of *i.i.d.* assumption of the sample set $\mathcal{D}$, we can write that

$$\begin{aligned}
&\operatorname*{argmax}_{w \in \mathbb{R}^D} \mathbb{E}_{(x,y)\sim d(x,y)}[\log p(y \mid x, w)] \\
\approx &\operatorname*{argmax}_{w \in \mathbb{R}^D} \sum_{(x^n,y^n)\in\mathcal{D}} \log p(y^n \mid x^n, w) \\
= &\operatorname*{argmax}_{w \in \mathbb{R}^D} \log \prod_{n=1}^{N} p(y^n \mid x^n, w) \\
= &\operatorname*{argmax}_{w \in \mathbb{R}^D} \prod_{n=1}^{N} p(y^n \mid x^n, w) \\
= &\operatorname*{argmax}_{w \in \mathbb{R}^D} p(y^1, \ldots, y^N \mid x^1, \ldots, x^N, w) \ .
\end{aligned}$$

from which the name **maximum conditional likelihood** (MCL) stems.

Prior belief on $p(w)$

When the number of training instances is *small* compared to the number of degrees (D) of freedom in w, then the approximation

$$\operatorname*{argmax}_{w \in \mathbb{R}^D} \mathbb{E}_{(x,y)\sim d(x,y)}[\log p(y \mid x, w)] \approx \operatorname*{argmax}_{w \in \mathbb{R}^D} \sum_{(x^n,y^n)\in\mathcal{D}} \log p(y^n \mid x^n, w)$$

becomes *unreliable*, and w^* will vary strongly with respect to the training set $\mathcal{D}$, which means MCL training is prone to **overfitting**.

To overcome this limitation, we treat w not as a deterministic parameter but as yet another random variable. For any prior distribution $p(w)$ over the space of weight vectors, the posterior probability of w for given observations $\mathcal{D} = \{(x^n, y^n)\}_{n=1,\ldots,N}$ is given by (see exercise):

$$p(w \mid \mathcal{D}) = p(w) \prod_{n=1}^{N} \frac{p(y^n \mid x^n, w)}{p(y^n \mid x^n)} \ .$$

Negative conditional log-likelihood

Assume a prior distribution of $p(w)$, then we can get

$$\begin{aligned} w^* &= \operatorname*{argmax}_{w \in \mathbb{R}^D} p(w \mid \mathcal{D}) = \operatorname*{argmin}_{w \in \mathbb{R}^D} \{-\log p(w \mid \mathcal{D})\} \\ &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \left\{ -\log \left(p(w) \prod_{n=1}^{N} \frac{p(y^n \mid x^n, w)}{p(y^n \mid x^n)} \right) \right\} \\ &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \left\{ -\log p(w) - \sum_{n=1}^{N} \log p(y^n \mid x^n, w) + \sum_{n=1}^{N} \log p(y^n \mid x^n) \right\} \\ &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \left\{ -\log p(w) - \sum_{n=1}^{N} \log p(y^n \mid x^n, w) \right\} . \end{aligned}$$

Regularized conditional log-likelihood

$$w^* = \operatorname*{argmin}_{w \in \mathbb{R}^D} \left\{ -\log p(w) - \sum_{n=1}^{N} \log p(y^n \mid x^n, w) \right\}$$

Assuming a zero-mean Gaussian prior $p(w) \propto \exp(-\frac{\|w\|^2}{2\sigma^2})$

$$\begin{aligned} w^* &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \left\{ \frac{\|w\|^2}{2\sigma^2} - \sum_{n=1}^{N} \log p(y^n \mid x^n, w) \right\} \\ &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \left\{ \lambda \|w\|^2 + \sum_{n=1}^{N} \langle w, \varphi(x^n, y^n) \rangle + \sum_{n=1}^{N} \log Z(x^n, w) \right\} , \end{aligned}$$

where $\lambda = \frac{1}{2\sigma^2}$.

The parameter λ is generally considered as a free hyper-parameter that determines the regularization strength. Unregularized situation can be seen as the limit case for $\lambda \to 0$.

Regularized Maximum Conditional Likelihood Training

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

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

For $\lambda = 0$ the simplified rule

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

results in **maximum conditional likelihood** (MCL) training.

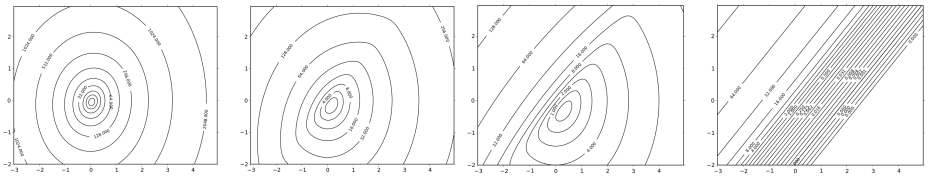
Negative conditional log-likelihood: Toy example

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) $\lambda = 100$ (b) $\lambda = 10$ (c) $\lambda = 1$ (d) $\lambda = 0$

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

Steepest Descent Minimization

1: $w_{\text{cur}} \leftarrow 0$
2: **repeat**
3: $\quad d \leftarrow -\nabla_w L(w_{\text{cur}})$
4: $\quad \eta \leftarrow \operatorname{argmin}_{\eta>0} \mathcal{L}(w_{\text{cur}} + \eta d)$
5: $\quad w_{\text{cur}} \leftarrow w_{\text{cur}} + \eta d$
6: **until** $\|d\| < \epsilon$
7: **return** w_{cur}

Let us consider the *negative conditional log-likelihood* function

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

Obviously, L is C^∞-differentiable, i.e. smooth function, on all $\mathbb{R}^D$.

Gradient Based Optimization

The Jacobian vector (cf. Analysis I/II) of $L(w)$ is given by

$$\begin{aligned}
\nabla_w L(w) =& \nabla_w \left(\lambda \|w\|^2 + \sum_{n=1}^{N} \langle w, \phi(x^n, y^n) \rangle + \sum_{n=1}^{N} \log Z(x^n, w) \right) \\
=& 2\lambda w + \sum_{n=1}^{N} \left(\phi(x^n, y^n) + \sum_{y \in \mathcal{Y}} \frac{\exp(-\langle w, \phi(x^n, y) \rangle)}{\sum_{y' \in \mathcal{Y}} \exp(-\langle w, \phi(x^n, y') \rangle)} (-\phi(x^n, y)) \right) \\
=& 2\lambda w + \sum_{n=1}^{N} \left(\phi(x^n, y^n) - \sum_{y \in \mathcal{Y}} p(y \mid x^n, w) \phi(x^n, y) \right) \\
=& 2\lambda w + \sum_{n=1}^{N} \left(\phi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n, w)}[\phi(x^n, y)] \right) \ .
\end{aligned}$$

Interpretation: we aim for *expectation matching*, i.e. $\phi(x^n, y^n) = \mathbb{E}_{y \sim p(y|x^n, w)}[\phi(x^n, y)]$ for $x^1, \ldots, x^n$.

Hessian of $L(w)$

By differentiating of $\nabla_w L(w)$, the Hessian matrix (cf. Analysis I/II) of $L(w)$ is given by (see exercise):

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

Reminder: for any random vector X the covariance $\mathrm{Cov}(X,X)$ can be written as:

$$Cov(X,X) \triangleq \mathbb{E}[(X - \mathbb{E}(X))(X - \mathbb{E}(X))^T] = \mathbb{E}[XX^T] - \mathbb{E}[X]\mathbb{E}[X]^T .$$

Note that $\Delta_w L(w)$ is a **covariance matrix**, hence it is *positive semidefinite*. Therefore, $L(w)$ is **convex**, which guarantees that every local minimum will also be a global one minimum of $L(w)$.

Numerical solution

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

In a naive way, computing the gradient takes $\mathcal{O}(K^M ND)$.

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

In a naive way, line search takes $\mathcal{O}(K^M ND)$ operations per evaluation of L, where

- N is the number of samples,
- D is the dimension of feature space,
- $M = |\mathcal{V}|$ 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.

Piecewise learning

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

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

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

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

$$\begin{aligned} w^* &= \operatorname*{argmin}_{w \in \mathbb{R}^D} L(w) = \operatorname*{argmin}_{w \in \mathbb{R}^D} \lambda \|w\|^2 - \sum_{n=1}^{N} \log \prod_{F \in \mathcal{F}} p(y_F^n \mid x_F^n, w_F) \\ &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \sum_{F \in \mathcal{F}} \lambda \|w_F\|^2 + \sum_{n=1}^{N} \langle w_F, \varphi_F(x_F^n, y_F^n) \rangle + \sum_{n=1}^{N} \log Z_F(x_F^n, w_F) \ . \end{aligned}$$

Piecewise learning

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

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

$$w_F^* = \operatorname*{argmin}_{w_F \in \mathbb{R}} \lambda \|w_F\|^2 + \sum_{n=1}^{N} \langle w_F, \varphi_F(x_F^n, y_F^n) \rangle + \sum_{n=1}^{N} \log Z_F(x_F^n, w_F) \ .$$

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

Comparing $p_{\mathsf{PW}}(y \mid x, w)$ 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 $Z_{\mathsf{PW}}(w)$ factorizes over the set of factors.

The simplification made by piece-wise training of CRFs resemble **two-stage learning**.

Two-stage learning

As a special case of *piecewise learning*, the idea here is to split learning of energy functions into two steps:

1. learning of unary energies via local classifiers, and
2. learning of their importance, pairwise and higher-order energies.

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

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.

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.