Learning for structured prediction

J. Serrat, Oriol Ramos

Master in CV, module M3, course 2019-20

Computer Vision Center, {oriolrt | joans}@cvc.uab.es

Index

Why and what to learn

Graphical models are a way to represent a the probability distribution of a potentially large number of variables and their dependencies or interactions.

We are interested in predicting the unknown variables y given known variables x.

Why and what to learn

Graphical models are a way to represent a the probability distribution of a potentially large number of variables and their dependencies or interactions.

We are interested in predicting the unknown variables y given known variables x.

In modern works

- GM renamed as *structured prediction*
- MRF (prior, likelihood) models $\longrightarrow$ energy minimization, more general formulation with CRF
- **hand selected parameters $\longrightarrow$ properly learned**

Why and what to learn

In the Nowozim & Lampert tutorial a CRF is a graphical model representing $p(y \mid x, w)$ as

$$\begin{aligned} p(y \mid x, w) &= \frac{1}{Z(x, w)} \exp[-E(x, y)] \\ Z(x, w) &= \sum_{y \in \mathcal{Y}} \exp[-E(x, y)] \\ E(x, y, w) &= \langle\, w, \phi(x, y) \,\rangle \end{aligned}$$

where E can or not decompose in terms of just y's and x_i, y (prior, likelihood) or not.

Then, the prediction task is

$$y^\star = \arg\max_{y \in \mathcal{Y}} \ \langle\, w, \phi(x, y) \,\rangle$$

Why and what to learn

For example, in the binary image filtering case,

$$E(x,y) = \sum_i \alpha\, y_i + \sum_{i,\ j \in \mathsf{Ne}_i} \beta\, y_i\, y_j + \sum_i \gamma\, x_i\, y_i$$

$$y^\star = \arg\min_{y \in \mathcal{Y}} E(x,y) = \arg\max_{y \in \mathcal{Y}} \ \langle\, w, \phi(x,y)\,\rangle$$

with (n number of pixels)

$$\begin{aligned} \phi(x,y) &= [y_1 \dots y_n, (y_i y_j)_{i=1\dots n, y_j \in \mathsf{Ne}_i}, x_1 y_1, \dots x_n y_n], \\ w &= -[\underbrace{\alpha \dots \alpha}_{n}, \underbrace{\beta \dots \beta}_{4n}, \underbrace{\gamma \dots \gamma}_{n}] \end{aligned}$$

Why and what to learn

What to learn from a CRF ?

1. the vector of parameters w given the model $\phi(x, y)$
2. the structure of the model: which are the interesting interactions (eg neighborhood) among the variables to represent them —
   how to decompose $\phi(x, y)$ into factors, each with a group of variables
3. the specific form of these interactions (not its parameters) — the concrete functions for these factors

1 is the easiest one, though still a difficult problem. We'll see how to solve it in a supervised way, that is, with samples $(x^k, y^k), k = 1 \dots N$

Why and what to learn

Why learn parameters ?

It does not make much sense to spend effort devising/applying good inference algorithms when the model is wrong.

Structure and shape of the model $\phi(x, y)$ are too difficult, we'll try to make reasonable assumptions.

But if they are right we can estimate the parameters that better explain the observed data = pairs of observation, labeling (x^k, y^k).

How to learn

Two styles of learning:

1. Probabilistic
   Find parameters $w^\star$ that make modeled posterior $p(y|x, w)$ closest to the *unknown*, real distribution $d(y|x)$.

   Since $d(y|x)$ is unknown we'll have to rely on the samples and maximize their likelihood

$$w^\star = \arg\max_w p(y^1, \dots y^m | x^1 \dots x^m, w)$$

How to learn

2. Loss minimizing
   Find $w^\star$ such that the risk $=$ average cost of predictions is minimum.

   Let $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ cost function, $\Delta(y, y')$ cost of predicting y' when the real label is y.

   If $f_p(x) = \arg\max_{y \in \mathcal{Y}} p(y|x, w^\star)$ is the predicting function (or inference algorithm), minimize

$$\mathbb{E}_{(x,y) \sim d(x,y)} \Delta(y, f_p(x)) = \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} p(x, y) \Delta(y, f_p(x))$$

Probabilistic learning

Starting point

- we have a training set with N samples, $(x^i, y^i),\ i = 1 \ldots N$
- have defined the form of $p(y|x, w)$ through some concrete, problem dependent $\psi(x, y)$

$$
\begin{aligned}
p(y|x,w) &= \frac{1}{Z(x,w)} \exp[-E(x,y,w)] \\
E(x,y,w) &= \langle\, w, \psi(x,y) \,\rangle \\
Z(x,y) &= \sum_{y \in \mathcal{Y}} \exp[-E(x,y,w)]
\end{aligned}
$$

  but not how to set its parameters w
- don't know the true conditional $d(y|x)$ or joint distribution $d(x, y)$

Probabilistic learning

Suppose $d(y|x)$ is known.

Then we want to estimate w such that $d(y|x)$ and $p(y|x, w)$ are the most close possible.

Need a measure of similarity between two (conditional) distributions in order to optimize it wrt w.

The Kullback-Leibler divergence (not distance) will be very convenient:

Probabilistic learning

In probability and information theory KL divergence is a non-symmetric measure of the difference between to probability distributions P and Q.

$KL(P||Q)$ the divergence of Q from P is a measure of the information lost when Q is used to approximate P.

Probabilistic learning

In probability and information theory KL divergence is a non-symmetric measure of the difference between to probability distributions P and Q.

$KL(P||Q)$ the divergence of Q from P is a measure of the information lost when Q is used to approximate P.

The expected number of extra bits required to code samples (messages) following P when using a code based on Q instead of P.

Q = model, theory, approximation. P = true distribution.

$$\mathrm{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}$$

Probabilistic learning

For a certain x,

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

a function of x and w.

Probabilistic learning

For a certain x,

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

a function of x and w.

The total divergence between p and d is summing over all x :

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

Goal: if $\phi(x,y)$ in $E(x,y,w) = \langle\, w, \phi(x,y) \,\rangle$ is a D-dimensional vector, $w^\star = \underset{w \in \mathbb{R}^D}{\arg\min}\, \mathsf{KL}_{\mathsf{total}}(d \,||\, p)$.

Probabilistic learning

$$
\begin{aligned}
w^\star &= \arg\min_{w \in \mathbb{R}^D} \mathsf{KL}_{\mathsf{total}}(d \,||\, p) \\
&= \arg\min_{w} \sum_{x \in \mathcal{X}} d(x) \sum_{y \in \mathcal{Y}} d(y|x) \big[\log d(y|x) - \log p(y|x, w) \big] \\
&= \arg\max_{w} \sum_{x \in \mathcal{X}} d(x) \sum_{y \in \mathcal{Y}} d(y|x) \log p(y|x, w) \\
&= \arg\max_{w} \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} d(x, y) \log p(y|x, w) \\
&= \arg\max_{w} \mathop{\mathbb{E}}_{(x,y) \sim d(x,y)} \log p(y|x, w) \quad \text{(another way to write it)} \\
&\approx \arg\max_{w} \sum_{(x^i, y^i),\, i=1 \ldots N} \log p(y^i|x^i, w) \\
&= \arg\min_{w} \sum_{i=1}^{N} \langle\, w, \psi(x^i, y^i) \,\rangle + \sum_{i=1}^{N} \log Z(x^i, w)
\end{aligned}
$$

Probabilistic learning

What's this ?

$$
\begin{aligned}
w^\star &= \arg\max_w \sum_{(x^i,y^i),\, i=1\ldots N} \log p(y^i|x^i,w) \\
&= \arg\max_w \prod_{(x^i,y^i),\, i=1\ldots N} p(y^i|x^i,w) \quad \text{(log increasing)} \\
&= \arg\max_w p(y^1 \ldots y^N|x^1 \ldots x^N,w) \quad \text{(independent samples)}
\end{aligned}
$$

Maximum conditional likelihood of the training set.

Probabilistic learning

Maximum Conditional Likelihood

$$w^\star = \arg\min_w \sum_{i=1}^{N} \left\langle w, \psi(x^i, y^i) \right\rangle + \sum_{i=1}^{N} \log Z(x^i, w)$$

Problems:

1. $Z(x^i, w) = \sum_{y \in \mathcal{Y}} \exp[-\left\langle w, \psi(x^i, y) \right\rangle]$ impossible to evaluate in practice because $|\mathcal{Y}|$ huge (in the example $2^n, n$ number of pixels)

Probabilistic learning

Maximum Conditional Likelihood

$$w^\star = \arg\min_w \sum_{i=1}^{N} \langle\, w, \psi(x^i, y^i) \,\rangle + \sum_{i=1}^{N} \log Z(x^i, w)$$

Problems:

2. Number of samples N can be large: good for approximating well but bad for computation of $\sum_{i=1}^{N} \log Z(x^i, w)$.

   We'll see that actually $Z(x^i, w)$ can be computed/approximated but has high cost. Having to do it N times is problematic if N large.

Probabilistic learning

Maximum Conditional Likelihood

$$w^\star = \arg\min_w \sum_{i=1}^{N} \langle\, w, \psi(x^i, y^i) \,\rangle + \sum_{i=1}^{N} \log Z(x^i, w)$$

Problems:

3. If N is small compared to length of w then overfitting occurs: approximation

$$\mathop{\mathbb{E}}_{(x,y) \sim d(x,y)} \log p(y|x,w) \approx \frac{1}{N} \sum_{(x^i,y^i),\, i=1\ldots N} \log p(y^i|x^i,w)$$

becomes unreliable when we move away from a sample.
A different training set drawn from the same distribution produces a very different $w^\star$.

Probabilistic learning

To address 3, think of w as a random variable with some nice prior distribution governed by a few meta-parameters: *regularization*.

Prior can not be estimated from data. It means how do we think vector w should be:

- zero-mean Gaussian prior : $p(w) \propto \exp[-\frac{||w||^2}{2\sigma^2}]$
  - all features mattter the same, no ones dominate over the others
  - good guess in practice if we have no idea
  - gives rise to easy equations

Probabilistic learning

To address 3, think of w as a random variable with some nice prior distribution governed by a few meta-parameters: *regularization*.

Prior can not be estimated from data. It means how do we think vector w should be:

- zero-mean Gaussian prior : $p(w) \propto \exp[-\frac{||w||^2}{2\sigma^2}]$
  - all features mattter the same, no ones dominate over the others
  - good guess in practice if we have no idea
  - gives rise to easy equations
- Laplacian : $p(w) \propto \exp[-\frac{||w||}{\sigma}]$
  - w sparse, only a few features matter
  - more difficult to handle $\rightarrow$ less common

Probabilistic learning

If $\mathcal{D} = (x^i, y^i), i = 1 \dots N$ training set, a sensible *new optimization goal* is to maximize the posterior probability

$$w^\star = \arg\max_w \; p(w|D)$$

Using Baye's rule

$$\begin{aligned} p(w|D) &= \frac{p(D|w)p(w)}{p(D)} \quad \text{samples are i.i.d.} \\ &= \prod_{i=1}^{N} \frac{p(x^i, y^i|w)p(w)}{p(x^i, y^i)} = \prod_{i=1}^{N} \frac{p(y^i|x^i, w)p(x^i|w)p(w)}{p(y^i|x^i)p(x^i)} \\ &= p(w) \prod_{i=1}^{N} \frac{p(y^i|x^i, w)}{p(y^i|x^i)} \quad x, w \text{ independent}, p(x^i|w) = p(x^i) \end{aligned}$$

$$\log p(w|D) = \log p(w) + \sum_{i=1}^{N} \log p(y^i|x^i, w) - \log p(y^i|x^i)$$

Probabilistic learning

$$
\begin{aligned}
w^\star &= \arg\max_w \; p(w|D) = \arg\max_w \; \log p(w|D) \\
&= \arg\max_w \; \log p(w) + \sum_{i=1}^{N} \log p(y^i|x^i, w)
\end{aligned}
$$

$p(y^i|x^i)$ independent of$w$

We have already seen

$$\log p(w) \propto -\frac{||w||^2}{2\sigma^2}$$

$$\sum_{i=1}^{N} \log p(y^i|x^i, w) = -\sum_{i=1}^{N} \left\langle\, w, \psi(x^i, y^i) \,\right\rangle - \sum_{i=1}^{N} \log Z(x^i, w)$$

$$w^\star = \arg\min_w \; \lambda||w||^2 + \sum_{i=1}^{N} \left\langle\, w, \psi(x^i, y^i) \,\right\rangle + \sum_{i=1}^{N} \log Z(x^i, w)$$

Probabilistic learning

Maximum Regularized Conditional Likelihood

$$
\begin{aligned}
w^\star &= \arg\min_w \ \lambda||w||^2 + \sum_{i=1}^{N} \left\langle\, w, \psi(x^i, y^i) \,\right\rangle + \sum_{i=1}^{N} \log Z(x^i, w) \\
&= \arg\min_w \ \lambda||w||^2 + \sum_{i=1}^{N} \left\langle\, w, \psi(x^i, y^i) \,\right\rangle \\
&\quad + \sum_{i=1}^{N} \log \left(\sum_{y \in \mathcal{Y}} \exp[- \left\langle\, w, \psi(x^i, y) \,\right\rangle] \right) = \arg\min_w \ \mathcal{L}(w)
\end{aligned}
$$

$\mathcal{L}(w)$ negative regularized log-likelihood

$\lambda > 0$ regularization strength. Penalizes large $|w_j|$.

For $\lambda = 0$ we get Maximum Conditional Likelihood.

Probabilistic learning: gradient descent

Having solved problem 3, back to problems 1 (N may be/need to be large) and 2 (impossibility to compute Z).

In the meantime . . . **good news!** $\mathcal{L}(w)$ can be shown[1] is a differentiable and convex function:

- can obtain an expression for the gradient $\nabla_w \mathcal{L}$
- following gradient descent could find global optimum

Let's derive $\mathcal{L}(w)$. Then we'll manage to sort out these two problems.

[1] Hessian is semi-positive definite, see why at p. 313, it's easy

Probabilistic learning: gradient descent

$$\nabla_w \mathcal{L}(w) = 2\lambda w + \sum_{i=1}^{N} \Big(\psi(x^i, y^i) + \underbrace{\nabla_w \log Z(x^i, w)}_{\frac{1}{Z(x^i,w)} \nabla_w Z(x^i, w) =} \Big)$$

$$\frac{1}{Z(x^i, w)} \sum_{y \in \mathcal{Y}} -\psi(x^i, y) \exp^{-\langle w, \psi(x^i, y) \rangle} =$$

$$\sum_{y \in \mathcal{Y}} -\psi(x^i, y) \underbrace{\frac{1}{Z(x^i, w)} \exp^{-\langle w, \psi(x^i, y) \rangle}}_{p(y|x^i, w)}$$

$$= 2\lambda w + \sum_{i=1}^{N} \Big(\psi(x^i, y^i) - \mathbb{E}_{y \sim p(y|x^i, w)} \psi(x^i, y) \Big)$$

Probabilistic learning: gradient descent

We can not perform simple steepest gradient descent because of the two former problems:

1. for each x^i and present w we need to sample y from $p(y|x^i, w)$ and this involves computing $Z(x^i, w)$
2. need to do this, or an approximation, for $i = 1 \ldots N$, N potentially large

Probabilistic learning: gradient descent

Let be

- N number of samples
- d dimension of feature space $\phi(y_i, y_j) \approx$ 100s, $\phi(x, y_i) \approx$ 100s-10,000s
- M number of output nodes $\approx$ 100s to 1,000,000s
- K number of possible labels of each output nodes $\approx$ 2 to 100s

naive gradient computation

$$\nabla_w \mathcal{L}(w) = 2\lambda w + \sum_{i=1}^{N} \Big(\psi(x^i, y^i) - \mathbb{E}_{y \sim p(y|x^i, w)} \psi(x^i, y) \Big)$$

takes $O(K^M N d)$.

Probabilistic learning: gradient descent

However, $p(y|x^i, w)$ and $\mathbb{E}_{y\sim p(y|x^i,w)}\psi(x^i, y)$ **can be computed !**

Key: $\psi(x, y)$ in $p(y|x, w) \propto \exp[-\langle w, \psi(x, y)\rangle]$ **decomposes in factors of few variables** y_j (low order factors: unary, pairwise, rarely higher orders)

$$\begin{aligned} \psi(x, y) &= \left(\psi_F(x_F, y_F) \right)_{F\in\mathcal{F}} \\ w &= \left(w_F \right)_{F\in\mathcal{F}} \end{aligned}$$

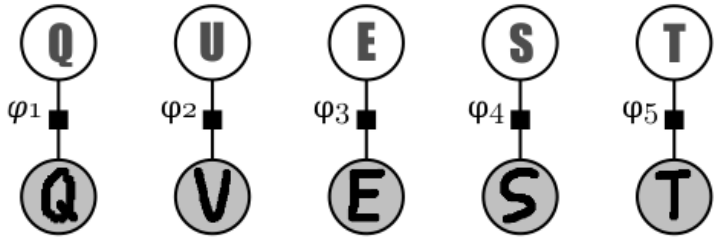
You've already used this to make marginals or MAP inference possible in belief propagation.

Let's illustrate it showing how comes it is possible to compute Z.

Probabilistic learning: gradient descent

Case I: only unary factors

$\mathcal{F} = \big\{\{x_1, y_1\}, \ldots, \{x_M, y_M\}\big\}.$



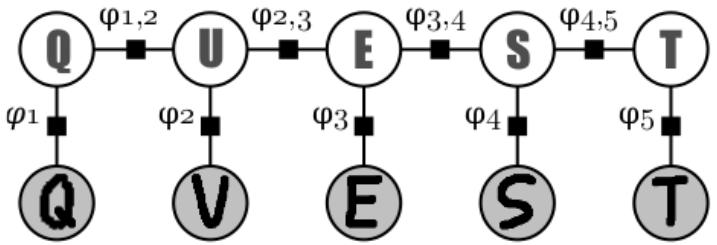
$$
\begin{aligned}
Z &= \sum_{y \in \mathcal{Y}} \prod_{F \in \mathcal{F}} \Psi_F(y_F) \\
&= \sum_{y \in \mathcal{Y}} \prod_{i=1}^{M} \Psi_i(y_i) \\
&= \sum_{y_1 \in Y} \sum_{y_2 \in Y} \cdots \sum_{y_M \in Y} \Psi_1(y_1) \cdots \Psi_M(y_M) \\
&= \sum_{y_1 \in Y} \Psi_1(y_1) \sum_{y_2 \in Y} \Psi_2(y_2) \cdots \sum_{y_M \in Y} \Psi_M(y_M) \\
&= \Big[\sum_{y_1 \in Y} \Psi_1(y_1)\Big] \cdot \Big[\sum_{y_2 \in Y} \Psi_2(y_2)\Big] \cdots \Big[\sum_{y_M \in Y} \Psi_M(y_M)\Big]
\end{aligned}
$$

Case I: $O(K^M nd) \;\rightarrow\; \boldsymbol{O}(MKnd)$

Learning with Structured Inputs and Outputs. Christoph Lampert. Slides INRIA Summer School 2010.

Probabilistic learning: gradient descent

Case II: chain/tree with unary and pairwise factors



$$\mathcal{F} = \Big\{\{x_1, y_1\}, \{y_1, y_2\}, \ldots, \{y_{M-1}, y_M\}, \{x_M, y_M\}\Big\}.$$

$$Z = \sum_{y \in \mathcal{Y}} \prod_{F \in \mathcal{F}} \Psi_F(y_F) = \sum_{y \in \mathcal{Y}} \prod_{i=1}^{M} \Psi_i(y_i) \prod_{i=2}^{M} \Psi_{i-1,i}(y_{i-1,i})$$

$$= \sum_{y_1 \in Y} \Psi_1 \sum_{y_2 \in Y} \Psi_{1,2}\Psi_2 \cdots \sum_{y_{M-1} \in Y} \Psi_{M-2,M-1}\Psi_{M-1} \sum_{y_M \in Y} \Psi_{M-1,M}\Psi_M$$

$$= \begin{bmatrix} \square \\ \square \\ \square \\ \square \end{bmatrix}^t \cdot \begin{bmatrix} \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \end{bmatrix} \cdots \begin{bmatrix} \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \end{bmatrix} \cdot \begin{bmatrix} \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \\ \square & \square & \square & \square \end{bmatrix} \begin{bmatrix} \square \\ \square \\ \square \\ \square \end{bmatrix}$$

Case II: $O(MK^2nd)$ independent was $O(MKnd)$, naive $O(K^Mnd)$

Probabilistic learning: gradient descent

Back to the gradient computation problem

$$\nabla_w \mathcal{L}(w) \quad = \quad 2\lambda w + \sum_{i=1}^{N} \Big(\psi(x^i, y^i) - \mathbb{E}_{y \sim p(y|x^i, w)} \psi(x^i, y) \Big)$$

Probabilistic learning: gradient descent

Back to the gradient computation problem

$$\nabla_w \mathcal{L}(w) = 2\lambda w + \sum_{i=1}^{N} \Big(\psi(x^i, y^i) - \mathbb{E}_{y \sim p(y|x^i, w)} \psi(x^i, y) \Big)$$

$$\psi(x, y) = \big(\psi_F(x_F, y_F) \big)_{F \in \mathcal{F}}$$
$$w = \big(w_F \big)_{F \in \mathcal{F}}$$

Probabilistic learning: gradient descent

Back to the gradient computation problem

$$\nabla_w \mathcal{L}(w) = 2\lambda w + \sum_{i=1}^{N} \Big(\psi(x^i, y^i) - \mathbb{E}_{y \sim p(y|x^i,w)} \psi(x^i, y) \Big)$$

$$\psi(x, y) = \big(\psi_F(x_F, y_F) \big)_{F \in \mathcal{F}}$$
$$w = \big(w_F \big)_{F \in \mathcal{F}}$$

$$\mathbb{E}_{y \sim p(y|x^i,w)} \psi(x^i, y) = \Big(\mathbb{E}_{y_F \sim p(y_F|x^i,w)} \psi_F(x^i, y_F) \Big)_{F \in \mathcal{F}}$$

and for each factor F,

$$\mathbb{E}_{y_F \sim p(y_F|x^i,w)} \psi(x^i, y_F) = \underbrace{\sum_{y_F \in \mathcal{Y}_F}}_{K^{|F|} \text{ terms}} \underbrace{p(y_F|x, w)}_{\text{factor marginals}} \psi_F(x^i, y_F)$$

Probabilistic learning: gradient descent

- unary terms: $|F| = 1$, pairwise $|F| = 2$.

Probabilistic learning: gradient descent

- unary terms: $|F| = 1$, pairwise $|F| = 2$.
- factor marginals $p(y_F|x, w)$ are much smaller than the complete joint distribution $p(y|x, w)$

Probabilistic learning: gradient descent

- unary terms: $|F| = 1$, pairwise $|F| = 2$.
- factor marginals $p(y_F|x, w)$ are much smaller than the complete joint distribution $p(y|x, w)$
- computed by exact (chains, trees) or loopy (graphs with cycles) sum-product belief propagation

Probabilistic learning: gradient descent

- unary terms: $|F| = 1$, pairwise $|F| = 2$.
- factor marginals $p(y_F|x, w)$ are much smaller than the complete joint distribution $p(y|x, w)$
- computed by exact (chains, trees) or loopy (graphs with cycles) sum-product belief propagation
- gradient computation is as costly as inference with belief propagation

Probabilistic learning: gradient descent

- unary terms: $|F| = 1$, pairwise $|F| = 2$.
- factor marginals $p(y_F|x, w)$ are much smaller than the complete joint distribution $p(y|x, w)$
- computed by exact (chains, trees) or loopy (graphs with cycles) sum-product belief propagation
- gradient computation is as costly as inference with belief propagation
- but this is *for each* $x^i, i = 1 \dots N$, still a considerable cost $\rightarrow$ our last problem

Probabilistic learning: gradient descent

At last, we can apply a first learning algorithm:

Algorithm 10: Steepest Descent Minimization

$w^* = \text{SteepestDescentMinimization}(\varepsilon)$
Input:
    $\varepsilon > 0$ tolerance
Output:
    $w^* \in \mathbb{R}^D$ learned weight vector
Algorithm:
$w_{\text{cur}} \leftarrow 0$
repeat
    $d \leftarrow -\nabla_w \mathcal{L}(w_{\text{cur}})$ {descent direction}
    $\eta \leftarrow \text{argmin}_{\eta>0} \mathcal{L}(w_{\text{cur}} + \eta d)$ {univariate line search}
    $w_{\text{cur}} \leftarrow w_{\text{cur}} + \eta d$
until $\|p\| < \varepsilon$
$w^* \leftarrow w_{\text{cur}}$

Probabilistic learning: stochastic gradient descent

$$\nabla_w \mathcal{L}(w) = 2\lambda w + \sum_{i=1}^{N} \psi(x^i, y^i) - \Big(\underbrace{\sum_{y_F \in \mathcal{Y}_F} p(y_F|x, w)\psi_F(x^i, y_F)}_{\text{belief propagation}} \Big)_{F \in \mathcal{F}}$$

At each step of gradient descent need to compute
$\nabla_w \mathcal{L}(w_{\text{cur}}) \Longrightarrow N$ inferences, one per sample in the training set $\mathcal{D} = \{(x^i, y^i), i = 1 \dots N\}$.

$O(K^{\max|F|} N d)$ *per step*, often $\max|F| = 2$.

Having many samples is good to avoid overfitting, but bad for this computation. Subsampling $\mathcal{D}$ therefore is not a good idea.

Probabilistic learning: stochastic gradient descent

Key: introduce subsampling not before but when running the algorithm of gradient descent. How:

- Compute an approximation of $\nabla_w \mathcal{L}(w)$ with a few samples (< 10) or even just one, randomly selected.

Probabilistic learning: stochastic gradient descent

Key: introduce subsampling not before but when running the algorithm of gradient descent. How:

- Compute an approximation of $\nabla_w \mathcal{L}(w)$ with a few samples (< 10) or even just one, randomly selected.
- Takes many more steps to converge than exact gradient

Probabilistic learning: stochastic gradient descent

Key: introduce subsampling not before but when running the algorithm of gradient descent. How:

- Compute an approximation of $\nabla_w \mathcal{L}(w)$ with a few samples (< 10) or even just one, randomly selected.
- Takes many more steps to converge than exact gradient
- But it pays: each step is *orders of magnitude faster*

Probabilistic learning: stochastic gradient descent

Key: introduce subsampling not before but when running the algorithm of gradient descent. How:

- Compute an approximation of $\nabla_w \mathcal{L}(w)$ with a few samples (< 10) or even just one, randomly selected.
- Takes many more steps to converge than exact gradient
- But it pays: each step is *orders of magnitude faster*
- Before, once we knew the gradient, performed a line search $\Rightarrow$ compute $\mathcal{L}(w_{\text{cur}} + \eta \nabla_w \mathcal{L}(w_{\text{cur}}))$ $\Rightarrow$ again perform one inference per sample, all samples:

$$\mathcal{L}(w) = \lambda ||w||^2 + \sum_{i=1}^{N} \left\langle\, w, \psi(x^i, y^i) \,\right\rangle + \sum_{i=1}^{N} \log Z(x^i, w)$$

Probabilistic learning: stochastic gradient descent

Key: introduce subsampling not before but when running the algorithm of gradient descent. How:

- Compute an approximation of $\nabla_w \mathcal{L}(w)$ with a few samples (< 10) or even just one, randomly selected.
- Takes many more steps to converge than exact gradient
- But it pays: each step is *orders of magnitude faster*
- Before, once we knew the gradient, performed a line search $\Rightarrow$ compute $\mathcal{L}(w_{\text{cur}} + \eta \nabla_w \mathcal{L}(w_{\text{cur}})) \Rightarrow$ again perform one inference per sample, all samples:

$$\mathcal{L}(w) = \lambda ||w||^2 + \sum_{i=1}^{N} \left\langle\, w, \psi(x^i, y^i) \,\right\rangle + \sum_{i=1}^{N} \log Z(x^i, w)$$

- Now, to avoid it, do simple first-order descent with a fixed sequence of learning rates $\eta_1, \eta_2, \ldots$ with $\eta_t = \eta / t$

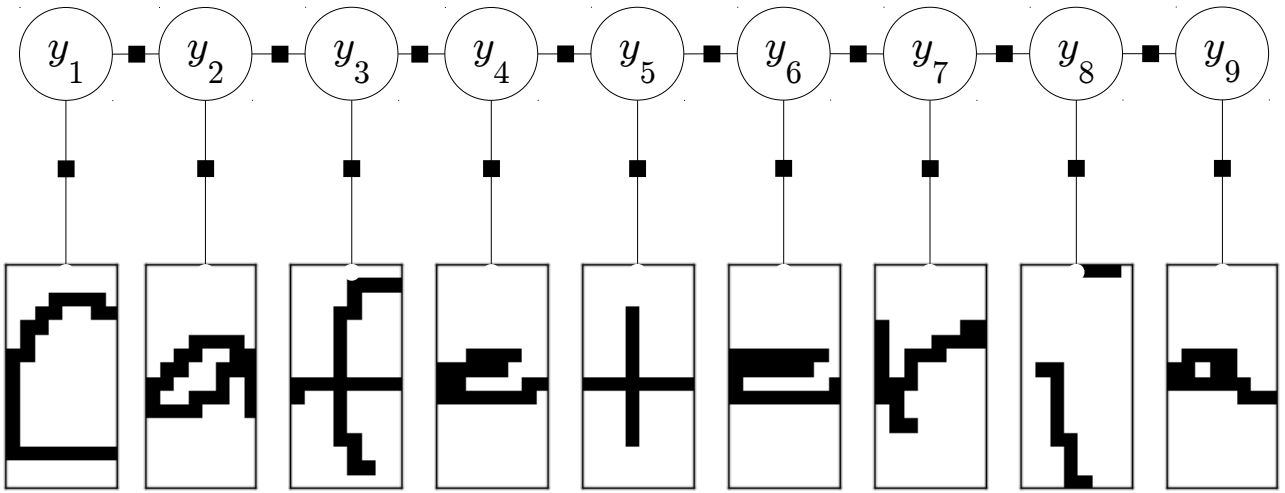
Probabilistic learning: stochastic gradient descent

Algorithm 12: Stochastic Gradient Descent

$w^* = \text{StochasticGradientDescent}(T, \eta)$
Input:
  T number of iterations
  $\eta_1, \ldots, \eta_T$ sequence of learning rates
Output:
  $w^* \in \mathbb{R}^D$ learned weight vector
Algorithm:
$w_{\text{cur}} \leftarrow 0$
for t=1,...,T **do**
  $(x^n, y^n) \leftarrow$ randomly chosen training pair
  $d \leftarrow -\widetilde{\nabla}_w^{(x^n, y^n)} \mathcal{L}(w_{\text{cur}})$
  $w_{\text{cur}} \leftarrow w_{\text{cur}} + \eta_t d$
end for
$w^* \leftarrow w_{\text{cur}}$

Probabilistic learning: two stage training

Consider the following pairwise model with observations binary images $16 \times 8 = 128$ pixels:



- $y_i \in \{\mathsf{a}, \mathsf{b} \ldots \mathsf{z}\}$
- $x_1 \ldots x_{128} \in \{0, 1\}$

Probabilistic learning: two stage training

- features are binary image pixels

$$\phi(x_i) = [x_{i,1} \dots x_{i,128}] \ , \ i = 1 \dots 9 \text{ word letters}$$

Probabilistic learning: two stage training

- features are binary image pixels

$$\phi(x_i) = [x_{i,1} \dots x_{i,128}] \ , \ i = 1 \dots 9 \text{ word letters}$$

- unary terms

$$\psi(x_i, y_i) = \sum_{j=\mathsf{a}}^{\mathsf{z}} \sum_{k=1}^{128} w_{j,k} \ x_{i,k} \ , \ i = 1 \dots 9$$

$26 \cdot 128 = 3328$ parameters $w_{j,k}$: one per pixel-label combination

Probabilistic learning: two stage training

- features are binary image pixels

$$\phi(x_i) = [x_{i,1} \dots x_{i,128}] \ , \ i = 1 \dots 9 \text{ word letters}$$

- unary terms

$$\psi(x_i, y_i) = \sum_{j=\mathsf{a}}^{\mathsf{z}} \sum_{k=1}^{128} w_{j,k} \ x_{i,k} \ , \ i = 1 \dots 9$$

$26 \cdot 128 = 3328$ parameters $w_{j,k}$: one per pixel-label combination

- pairwise terms

$$\psi(y_i, y_{i+1}) = \sum_{p=\mathsf{a}}^{\mathsf{z}} \sum_{q=\mathsf{a}}^{\mathsf{z}} w_{p,q} \, 1_{y_i=p, y_{i+1}=q}$$

$26 \cdot 26 = 676$ pairwise parameters $w_{p,q}$

Probabilistic learning: two stage training

- features are binary image pixels

$$\phi(x_i) = [x_{i,1} \dots x_{i,128}] \ , \ i = 1 \dots 9 \text{ word letters}$$

- unary terms

$$\psi(x_i, y_i) = \sum_{j=\mathsf{a}}^{\mathsf{z}} \sum_{k=1}^{128} w_{j,k} \, x_{i,k} \ , \ i = 1 \dots 9$$

$26 \cdot 128 = 3328$ parameters $w_{j,k}$: one per pixel-label combination

- pairwise terms

$$\psi(y_i, y_{i+1}) = \sum_{p=\mathsf{a}}^{\mathsf{z}} \sum_{q=\mathsf{a}}^{\mathsf{z}} w_{p,q} \, 1_{y_i=p, y_{i+1}=q}$$

$26 \cdot 26 = 676$ pairwise parameters $w_{p,q}$

- unary and pairwise factors share parameters but that's not always necessary

Probabilistic learning: two stage training

Unary factors : kind of local classifiers based on high-dimensional feature vectors.

Pairwise (and higher order) factors : low-dimensional feature maps to encode label smoothness (=continuity in segmentation). Here frequency of successive pairs of letters.

CRF training becomes a very high-dimensional optimization problem but most parameters encode single-node (just one y_j) information: 3328 unary $w_{j,k}$ versus 676 pairwise $w_{p,q}$.

If binary images were $16 \times 8 \longrightarrow 32 \times 16$: 13312 vs 676.

Idea: learn them independently in order to simplify the problem.

Probabilistic learning: two stage training

1. Train *one* independent node classifier $f(x_i) = y_i$ with multiclass SVM or logistic regression

Probabilistic learning: two stage training

1. Train *one* independent node classifier $f(x_i) = y_i$ with multiclass SVM or logistic regression
   - or use any other classifier like random forest ...

Probabilistic learning: two stage training

1. Train *one* independent node classifier $f(x_i) = y_i$ with multiclass SVM or logistic regression
   - or use any other classifier like random forest ...
   - just one classifier because all nodes share unary parameters : we're assuming letter position in a word doesn't matter.

Probabilistic learning: two stage training

1. Train *one* independent node classifier $f(x_i) = y_i$ with multiclass SVM or logistic regression
   - or use any other classifier like random forest ...
   - just one classifier because all nodes share unary parameters : we're assuming letter position in a word doesn't matter.
2. Given a binary image, get from the classifier either a label = letter, or a vector of 26 probability values $p(y_i = \mathsf{a}|x_i), \ldots p(y_i = \mathsf{z}|x_i)$

Probabilistic learning: two stage training

1. Train *one* independent node classifier $f(x_i) = y_i$ with multiclass SVM or logistic regression
   - or use any other classifier like random forest ...
   - just one classifier because all nodes share unary parameters : we're assuming letter position in a word doesn't matter.
2. Given a binary image, get from the classifier either a label = letter, or a vector of 26 probability values $p(y_i = \mathsf{a}|x_i), \ldots p(y_i = \mathsf{z}|x_i)$
3. Use the low dimensional output vector of this classifier as feature vector for unary factors : from 3328 to just 26 $w_{j,k}$ parameters

Probabilistic learning: two stage training

1. Train *one* independent node classifier $f(x_i) = y_i$ with multiclass SVM or logistic regression
   - or use any other classifier like random forest ...
   - just one classifier because all nodes share unary parameters : we're assuming letter position in a word doesn't matter.
2. Given a binary image, get from the classifier either a label = letter, or a vector of 26 probability values $p(y_i = \mathsf{a}|x_i), \ldots p(y_i = \mathsf{z}|x_i)$
3. Use the low dimensional output vector of this classifier as feature vector for unary factors : from 3328 to just 26 $w_{j,k}$ parameters
4. pairwise parameters $w_{p,q}$ are the same as before

Probabilistic learning: two stage training

- significant reduction in model training time: much less free parameters to learn
- flexibility: can choose any (multiclass) classifier (that is fast and works $\pm$ well)
- loss of expressive power: once the new unary features are computed we can just change its weight / importance through parameters $w_{j,k}$ to be estimated

Probabilistic learning: pseudo-likelihood

In stochastic gradient descent we approximated the gradient with something easy to compute, gradient of 1 or a few samples (x_i, y_i) $\rightarrow$ 1 or a few inference computations $p(y_i|x, w)$.

Now we'll approximate the likelihood $p(y|x, w)$ with another likelihood for which gradient is easier to compute.

Idea: divide y into $y_1, \ldots y_s \ldots y_M$ and estimate w from the approximation

$$p(y|x, w) \approx \prod_s p(y_s|y_{t\neq s}, x, w)$$

which maybe reasonable for training data $x^n, y^n, n = 1 \ldots N$, and we know y.

Probabilistic learning: pseudo-likelihood

Pseudo-likelihood

For a graphical model with M nodes, let S set of nodes, s node or site with possible values $\mathcal{Y}_s$. Pseudo-likelihood is

$$
\begin{aligned}
p_{PL}(y|x,w) &= \prod_{s=1}^{M} p_{PL}(y_s|y_{S\setminus s},x,w) \\
p_{PL}(y_s|y_{S\setminus s},x,w) &= \frac{1}{Z_s(x,y_{S\setminus s},w)} \exp[-\langle w,\psi(x,y)\rangle] \\
Z_s(x,y_{S\setminus s},w) &= \sum_{y_s\in\mathcal{Y}_s} \exp[-\langle w,\psi(x,y_{S\setminus s},y_s)\rangle]
\end{aligned}
$$

So what's the difference with regular likelihood $p(y|x,w)$?

Probabilistic learning: pseudo-likelihood

$$
\begin{aligned}
p_{PL}(y|x,w) &= \prod_{s=1}^{M} p_{PL}(y_s|y_{S\setminus s},x,w) \\
p_{PL}(y_s|y_{S\setminus s},x,w) &= \frac{1}{Z_s(x,y_{S\setminus s},w)} \exp[-\langle w,\psi(x,y)\rangle] \\
Z_s(x,y_{S\setminus s},w) &= \sum_{y_s\in\mathcal{Y}_s} \exp[-\langle w,\psi(x,y_{S\setminus s},y_s)\rangle]
\end{aligned}
$$

- *At training time* we know $(x^n,y^n), n=1\dots N \rightarrow$ we treat $y_{S\setminus s}$ as observations, known values $\Rightarrow Z_s(x^n,y^n_{S\setminus s},w)$ easily computable!
- $p_{PL}(y^n|x^n,w)$ factorizes as product of functions over a single variable $p_{PL}(y^n_s|y^n_{S\setminus s},x^n,w)$

Probabilistic learning: pseudo-likelihood

(pseudo)likelihood gradient is now

$$\nabla_w \mathcal{L}(w) = 2\lambda w + \sum_{n=1}^{N}\sum_{s=1}^{M} \psi(x^n, y^n) - \mathbb{E}_{y_s \sim p_{PL}(y_s | y_{S\setminus s}, x, w)}\ \psi(x^n, y^n_{S\setminus n}, y_s)$$

much more easy to compute because expectation is only over one random variable y_s each time (for each n).

Nearly as efficient as training independent variables, dependent only on observations.

Tends to work when there are not strong dependencies between connected sites, but that's precisely what we want to exploit in graphical models!

See *piecewise training* for another approximation trying to solve this problem.

Probabilistic learning: summary

Summary – CRF Learning Given:

- training set $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}, \quad (x^n, y^n) \overset{i.i.d.}{\sim} d(x, y)$
- feature function $\phi : \mathcal{X} \times \mathbb{R}^D$.

Task: find parameter vector w such that $\frac{1}{Z} \exp(\langle w, \phi(x, y) \rangle) \approx d(y|x)$.

CRF solution derived by minimizing *negative conditional log-likelihood*:

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

- *convex* optimization problem $\rightarrow$ gradient descent works
- training needs repeated runs of *probabilistic inference*

Structured prediction. Sebastian Nowozin, Christoph Lampert. Slides CVPR tutorial 2011.

Probabilistic learning: summary

Solving the Training Optimization Problem Numerically CRF training methods is based on gradient-descent optimization.
The faster we can do it, the better (more realistic) models we can use:

$$\tilde{\nabla}_w \mathcal{L}(w) = \frac{w}{\sigma^2} - \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \sum_{y \in \mathcal{Y}} p(y|x^n, w)\, \phi(x^n, y)\right] \quad \in \mathbb{R}^D$$

A lot of research on accelerating CRF training:

problem	"solution"	method(s)
$\lvert\mathcal{Y}\rvert$ too large	exploit structure	(loopy) belief propagation
	smart sampling	contrastive divergence
	use approximate $\mathcal{L}$	e.g. pseudo-likelihood
N too large	mini-batches	stochastic gradient descent
D too large	trained ϕ_{unary}	two-stage training

Structured prediction. Sebastian Nowozin, Christoph Lampert. Slides CVPR tutorial 2011.

Libraries

How to represent a graphical model, train it and make inference ?
No need to program it from scratch, others have done.

Pystruct : structured learning in Python

`https://pystruct.github.io/`

> *PyStruct aims at being an easy-to-use structured learning and prediction library. Currently it implements only max-margin methods.*
> *The goal of PyStruct is to provide a well-documented tool for researchers as well as non-experts to make use of structured prediction algorithms. The design tries to stay as close as possible to the interface and conventions of scikit-learn*

Libraries

Highlights:

- general CRF models including chains and grids
- inference algorithms: AD3, QPBO, linear programming, max-product
- learning with SSVM
- just Python
- excellent gallery of examples, sklearn-style
- easy to install