Part 4: Conditional Random Fields

Sebastian Nowozin and Christoph H. Lampert

Colorado Springs, 25th June 2011

Problem (Probabilistic Learning)

Let $d(y|x)$ be the (unknown) true conditional distribution.
Let $\mathcal{D} = \{(x^1, y^1), \ldots, (x^N, y^N)\}$ be i.i.d. samples from $d(x, y)$.

- *Find a distribution $p(y|x)$ that we can use as a proxy for $d(y|x)$.*

or

- *Given a parametrized family of distributions, $p(y|x, w)$, find the parameter w^* making $p(y|x, w)$ closest to $d(y|x)$.*

Open questions:

- What do we mean by closest?
- What's a good candidate for $p(y|x, w)$?
- How to actually find w^*?
  - conceptually, and
  - numerically

Principle of Parsimony (Parsimoney, aka Occam's razor)

"Pluralitas non est ponenda sine neccesitate."

William of Ockham

"We are to admit no more causes of natural things than such as are both true and sufficient to explain their appearances."

Isaac Newton

"Make everything as simple as possible, but not simpler."

Albert Einstein

"Use the simplest explanation that covers all the facts."

what we'll use

- 1) Define what aspects we consider relevant facts about the data.
- 2) Pick the simplest distribution reflecting that.

Definition (Simplicity ≡ Entropy)

The *simplicity* of a distribution p is given by its *entropy*:

$$H(p) = -\sum_{z \in \mathcal{Z}} p(z) \log p(z)$$

Definition (Relevant Facts ≡ Feature Functions)

By $\phi_i : \mathcal{Z} \to \mathbb{R}$ for $i = 1, \ldots, D$ we denote a set of *feature functions* that express everything we want to be able to model about our data.

For example:

- the grayvalue of a pixel,
- a bag-of-words histogram of an image,
- the time of day an image was taken,
- a flag if a pixel is darker than half of its neighbors.

Principle (Maximum Entropy Principle)

Let $z^1, \ldots, z^N$ be samples from a distribution $d(z)$. Let $\phi_1, \ldots, \phi_D$ be feature functions, and denote by $\mu_i := \frac{1}{N} \sum_n \phi_i(z^n)$ their means over the sample set.
The maximum entropy distribution, p, is the solution to

$$\max_{p \text{ is a prob.distr.}} H(p) \quad \text{subject to} \quad \mathbb{E}_{z \sim p(z)}\{\phi_i(z)\} = \mu_i.$$

be as simple as possible

be faithful to what we know

Theorem (Exponential Family Distribution)

Under some very reasonable conditions, the maximum entropy distribution has the form

$$p(z) = \frac{1}{Z} \exp\left(\sum_i w_i \phi_i(z)\right)$$

for some parameter vector $w = (w_1, \ldots, w_D)$ and constant Z.

Example:

- Let $\mathcal{Z} = \mathbb{R}$, $\phi_1(z) = z$, $\phi_2(z) = z^2$.
- The exponential family distribution is

$$p(z) = \frac{1}{Z(w)} \exp(\ w_1 z + w_2 z^2)$$
$$= \frac{b^2 a}{Z(a,b)} \exp(\ a(z-b)^2\ ) \quad \text{for } a = w_2,\ b = -\frac{w_1}{w_2}.$$

It's a Gaussian!

- Given examples $z^1, \ldots, z^N$, we can compute a and b, and derive w.

Example:

- Let $\mathcal{Z} = \{1, \ldots, K\}$, $\phi_k(z) = [\![z = k]\!]$, for $k = 1, \ldots, K$.
- The exponential family distribution is

$$p(z) = \frac{1}{Z(w)} \exp\Big(\sum_k w_k \phi_k(z) \Big)$$

$$= \begin{cases} \exp(w_1)/Z & \text{for } z = 1, \\ \exp(w_2)/Z & \text{for } z = 2, \\ \ldots & \\ \exp(w_K)/Z & \text{for } z = K. \end{cases}$$

$$\text{with } Z = \exp(w_1) + \cdots + \exp(w_K).$$

It's a Multinomial!

Example:

- Let $\mathcal{Z} = \{0,1\}^{N \times M}$ image grid,
  $\phi_i(y) := y_i$ for each pixel i,
  $\phi_{NM}(y) = \sum_{i \sim j} [\![y_i \neq y_j]\!]$ (summing over all 4-neighbor pairs)
- The exponential family distribution is

$$p(z) = \frac{1}{Z(w)} \exp\Big(\langle w, \phi(y) \rangle + \tilde{w} \sum_{i,j} [\![y_i \neq y_j]\!] \Big)$$

It's a (binary) Markov Random Field!

Conditional Random Field Learning

Assume:

- a set of i.i.d. samples $\mathcal{D} = \{(x^n, y^n)\}_{n=1,\ldots,N}, \quad (x^n, y^n) \sim d(x, y)$
- feature functions $(\,\phi_1(x, y), \ldots, \phi_D(x, y)\,) \equiv: \phi(x, y)$
- parametrized family $p(y|x, w) = \frac{1}{Z(x,w)} \exp(\,\langle w, \phi(x, y)\rangle\,)$

Task:

- adjust w of $p(y|x, w)$ based on $\mathcal{D}$.

Many possible technique to do so:

- Expectation Matching
- Maximum Likelihood
- Best Approximation
- MAP estimation of w

Punchline: they all turn out to be (almost) the same!

Maximum Likelihood Parameter Estimation **Idea:** maximize conditional likelihood of observing outputs $y^1, \ldots, y^N$ for inputs $x^1, \ldots, x^N$

$$
\begin{aligned}
w^* &= \operatorname*{argmax}_{w \in \mathbb{R}^D} \; p(y^1, \ldots, y^N | x^1, \ldots, x^N, w) \\
&\overset{i.i.d.}{=} \operatorname*{argmax}_{w \in \mathbb{R}^D} \; \prod_{n=1}^{N} p(y^n | x^n, w) \\
&\overset{-\log(\cdot)}{=} \operatorname*{argmin}_{w \in \mathbb{R}^D} \; \underbrace{-\sum_{n=1}^{N} \log p(y^n | x^n, w)}_{\text{negative conditional log-likelihood (of } \mathcal{D})}
\end{aligned}
$$

Best Approximation

Idea: find $p(y|x,w)$ that is closest to $d(y|x)$

Definition (Similarity between conditional distributions)

For fixed $x \in \mathcal{X}$: KL-divergence measure similarity

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

For $x \sim d(x)$, compute expectation:

$$\begin{aligned} \mathrm{KL}_{tot}(p||d) &:= \mathbb{E}_{x \sim d(x)} \Big\{ \mathsf{KL}_{\mathsf{cond}}(p||d)(x) \Big\} \\ &= \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} d(x,y) \log \frac{d(y|x)}{p(y|x,w)} \end{aligned}$$

Best Approximation

Idea: find $p(y|x,w)$ of minimal KL_{tot}-distance to $d(y|x)$

$$
\begin{aligned}
w^* &= \operatorname*{argmin}_{w \in \mathbb{R}^D} \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} d(x,y) \log \frac{d(y|x)}{p(y|x,w)} \\
&\overset{\text{drop const.}}{=} \operatorname*{argmin}_{w \in \mathbb{R}^D} - \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} d(x,y) \log p(y|x,w) \\
&\overset{(x^n,y^n) \sim d(x,y)}{\approx} \operatorname*{argmin}_{w \in \mathbb{R}^D} \underbrace{- \sum_{n=1}^{N} \log p(y^n|x^n,w)}_{\text{negative conditional log-likelihood (of } \mathcal{D})}
\end{aligned}
$$

MAP Estimation of w **Idea:** Treat w as random variable; maximize posterior probability $p(w|\mathcal{D})$

$$p(w|\mathcal{D}) \overset{\text{Bayes}}{=} \frac{p(x^1, y^1, \ldots, x^n, y^n|w)p(w)}{p(\mathcal{D})} \overset{i.i.d.}{=} p(w) \prod_{n=1}^{N} \frac{p(y^n|x^n, w)}{p(y^n|x^n)}$$

$p(w)$: *prior belief* on w (cannot be estimated from data).

$$\begin{aligned}
w^* &= \operatorname*{argmax}_{w \in \mathbb{R}^D} \; p(w|\mathcal{D}) = \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[-\log p(w|\mathcal{D}) \Big] \\
&= \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[-\log p(w) - \sum_{n=1}^{N} \log p(y^n|x^n, w) + \underbrace{\log p(y^n|x^n)}_{\text{indep. of } w} \Big] \\
&= \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[-\log p(w) - \sum_{n=1}^{N} \log p(y^n|x^n, w) \Big]
\end{aligned}$$

$$w^* = \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[-\log p(w) - \sum_{n=1}^{N} \log p(y^n | x^n, w) \Big]$$

Choices for $p(w)$:

- $p(w) :\equiv$ const. (uniform; in $\mathbb{R}^D$ not really a distribution)

$$w^* = \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[\underbrace{-\sum_{n=1}^{N} \log p(y^n | x^n, w)}_{\text{negative conditional log-likelihood}} + \text{const.} \Big]$$

- $p(w) := \textit{const.} \cdot e^{-\frac{1}{2\sigma^2}\|w\|^2}$ (Gaussian)

$$w^* = \operatorname*{argmin}_{w \in \mathbb{R}^D} \Big[\underbrace{-\frac{1}{2\sigma^2}\|w\|^2 + \sum_{n=1}^{N} \log p(y^n | x^n, w)}_{\text{regularized negative conditional log-likelihood}} + \text{const.} \Big]$$

Probabilistic Models for Structured Prediction - Summary

Negative (Regularized) Conditional Log-Likelihood (of $\mathcal{D}$)

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

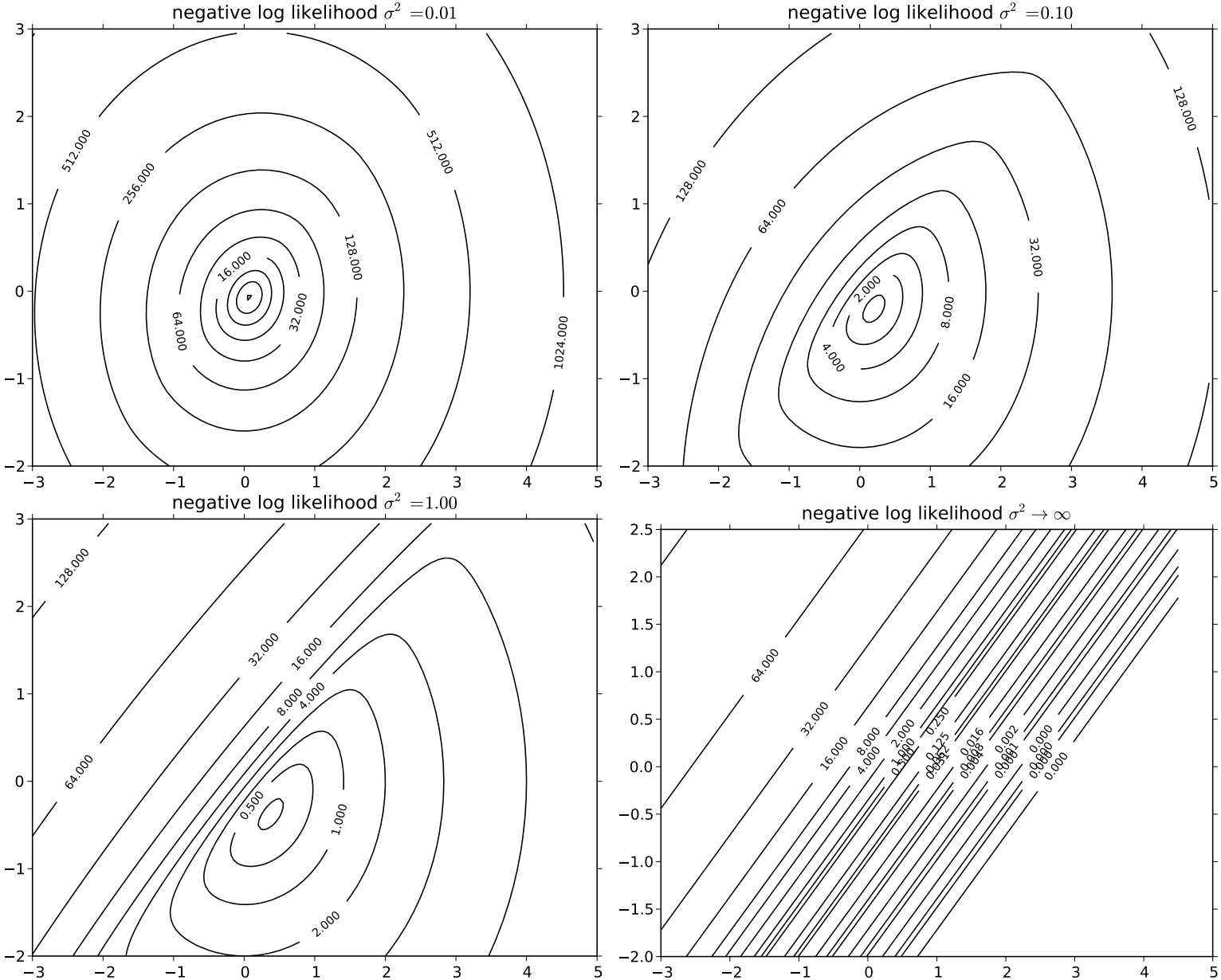
($\sigma^2 \to \infty$ makes it *unregularized*)

Probabilistic parameter estimation or *training* means solving

$$w^* = \operatorname*{argmin}_{w\in\mathbb{R}^D} \mathcal{L}(w).$$

Same optimization problem as for multi-class **logistic regression**.

Negative Conditional Log-Likelihood (Toy Example)



Steepest Descent Minimization – minimize $\mathcal{L}(w)$

input tolerance $\epsilon > 0$

1: $w_{cur} \leftarrow 0$
2: **repeat**
3: $\quad v \leftarrow \nabla_w \mathcal{L}(w_{cur})$
4: $\quad \eta \leftarrow \operatorname{argmin}_{\eta \in \mathbb{R}} \mathcal{L}(w_{cur} - \eta v)$
5: $\quad w_{cur} \leftarrow w_{cur} - \eta v$
6: **until** $\|v\| < \epsilon$

output w_{cur}

Alternatives:

- L-BFGS (second-order descent without explicit Hessian)
- Conjugate Gradient

We always need (at least) the gradient of $\mathcal{L}$.

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

$$\begin{aligned}
\nabla_w \mathcal{L}(w) &= \frac{1}{\sigma^2} w - \sum_{n=1}^{N}\Big[\phi(x^n, y^n) - \frac{\sum_{y\in\mathcal{Y}} e^{\langle w, \phi(x^n, y)\rangle}\phi(x^n, y)}{\sum_{\bar{y}\in\mathcal{Y}} e^{\langle w, \phi(x^n, \bar{y})\rangle}}\Big] \\
&= \frac{1}{\sigma^2} w - \sum_{n=1}^{N}\Big[\phi(x^n, y^n) - \sum_{y\in\mathcal{Y}} p(y|x^n, w)\phi(x^n, y)\Big] \\
&= \frac{1}{\sigma^2} w - \sum_{n=1}^{N}\Big[\phi(x^n, y^n) - \mathbb{E}_{y\sim p(y|x^n, w)}\phi(x^n, y)\Big]
\end{aligned}$$

$$\Delta\mathcal{L}(w) = \frac{1}{\sigma^2} Id_{D\times D} + \sum_{n=1}^{N}[\mathbb{E}_{y\sim p(y|x^n, w)}\phi(x^n, y)][\mathbb{E}_{y\sim p(y|x^n, w)}\phi(x^n, y)]^\top$$

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

- C^∞-differentiable on all $\mathbb{R}^D$.

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

- For $\sigma \to \infty$:

$$\mathbb{E}_{y \sim p(y|x^n, w)} \phi(x^n, y) = \phi(x^n, y^n) \qquad \Rightarrow \quad \nabla_w \mathcal{L}(w) = 0,$$

  criticial point of $\mathcal{L}$ (local minimum/maximum/saddle point).

Interpretation:

- We aim for *expectation matching*: $\mathbb{E}_{y \sim p} \phi(x, y) = \phi(x, y^{\text{obs}})$ but discriminatively: only for $x \in \{x^1, \ldots, x^n\}$.

$$\Delta\mathcal{L}(w) = \frac{1}{\sigma^2} Id_{D\times D} + \sum_{n=1}^{N} [\mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n,y)][\mathbb{E}_{y\sim p(y|x^n,w)}\phi(x^n,y)]^\top$$

- positive definite Hessian matrix $\rightarrow \mathcal{L}(w)$ is *convex*
  $\rightarrow \nabla_w \mathcal{L}(w) = 0$ implies *global minimum*.

Milestone I: Probabilistic Training (Conditional Random Fields)

- $p(y|x, w)$ log-linear in $w \in \mathbb{R}^D$.
- Training: many probabilistic derivations lead to same optimization problem $\rightarrow$ minimize negative conditional log-likelihood, $\mathcal{L}(w)$
- $\mathcal{L}(w)$ is differentiable and *convex*,
  $\rightarrow$ gradient descent will find global optimum with $\nabla_w \mathcal{L}(w) = 0$
- Same structure as multi-class *logistic regression*.

For logistic regression: this is where the textbook ends. we're done.

For conditional random fields: we're not in safe waters, yet!

Task: Compute $v = \nabla_w \mathcal{L}(w_{cur})$, evaluate $\mathcal{L}(w_{cur} + \eta v)$:

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

$$\nabla_w \mathcal{L}(w) = \frac{1}{\sigma^2} w - \sum_{n=1}^{N}\left[\phi(x^n, y^n) - \sum_{y\in\mathcal{Y}} p(y|x^n, w)\phi(x^n, y)\right]$$

Problem: $\mathcal{Y}$ typically is very (exponentially) large:

- binary image segmentation: $|\mathcal{Y}| = 2^{640\times480} \approx 10^{92475}$
- ranking N images: $|\mathcal{Y}| = N!$, e.g. $N = 1000$: $|\mathcal{Y}| \approx 10^{2568}$.

We must use the **structure** in $\mathcal{Y}$, or we're lost.

Solving the Training Optimization Problem Numerically

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

Computing the Gradient (naive): $O(K^M N D)$

$$\mathcal{L}(w) = \frac{1}{2\sigma^2} \|w\|^2 - \sum_{n=1}^{N} \left[\langle w, \phi(x^n, y^n) \rangle + \log Z(x^n, w) \right]$$

Line Search (naive): $O(K^M N D)$ per evaluation of $\mathcal{L}$

- N: number of samples
- D: dimension of feature space
- M: number of output nodes $\approx$ 100s to 1,000,000s
- K: number of possible labels of each output nodes $\approx$ 2 to 100s

Probabilistic Inference to the Rescue Remember: in a graphical model with factors $\mathcal{F}$, the features decompose:

$$\phi(x, y) = \Big(\phi_F(x, y_F)\Big)_{F \in \mathcal{F}}$$

$$\begin{aligned}\mathbb{E}_{y \sim p(y|x,w)} \phi(x, y) &= \Big(\mathbb{E}_{y \sim p(y|x,w)} \phi_F(x, y_F)\Big)_{F \in \mathcal{F}} \\ &= \big(\mathbb{E}_{y_F \sim p(y_F|x,w)} \phi_F(x, y_F)\big)_{F \in \mathcal{F}}\end{aligned}$$

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

Factor marginals $\mu_F = p(y_F|x, w)$

- are much smaller than complete joint distribution $p(y|x, w)$,
- can be computed/approximated, e.g., with *(loopy) belief propagation*.

Solving the Training Optimization Problem Numerically

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

Computing the Gradient: ~~$O(K^M nd)$~~, $O(MK^{|F_{max}|} ND)$:

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

Line Search: ~~$O(K^M nd)$~~, $O(MK^{|F_{max}|} ND)$ per evaluation of $\mathcal{L}$

- N: number of samples $\approx$ 10s to 1,000,000s
- D: dimension of feature space
- M: number of output nodes
- K: number of possible labels of each output nodes

What, if the training set $\mathcal{D}$ is too large (e.g. millions of examples)?

Simplify Model

- Train simpler model (smaller factors)

Less expressive power $\Rightarrow$ results might get *worse* rather than better ☹

Subsampling

- Create random subset $\mathcal{D}' \subset \mathcal{D}$. Train model using $\mathcal{D}'$

Ignores all information in $\mathcal{D} \setminus \mathcal{D}'$ ☹

Parallelize

- Train multiple models in parallel. Merge the models.

Follows the multi-core/GPU trend ☺
How to actually merge? ☹ (or ☺?)
Doesn't *really* save computation. ☹

What, if the training set $\mathcal{D}$ is too large (e.g. millions of examples)?

Stochastic Gradient Descent (SGD)

- Keep maximizing $p(w|\mathcal{D})$. ☺
- In each gradient descent step:
  - Create random subset $\mathcal{D}' \subset \mathcal{D}$, ← **often just 1–3 elements!**
  - Follow approximate gradient

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

- *Line search* no longer possible. Extra parameter: stepsize η
- SGD converges to $\operatorname{argmin}_w \mathcal{L}(w)$! (if η chosen right)
- SGD needs more iterations, but each one is much faster

---

more: see L. Bottou, O. Bousquet: "*The Tradeoffs of Large Scale Learning*", NIPS 2008.
also: `http://leon.bottou.org/research/largescale`

Solving the Training Optimization Problem Numerically

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

Computing the Gradient: ~~$O(K^M nd)$~~, $O(MK^2ND)$ (if BP is possible):

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

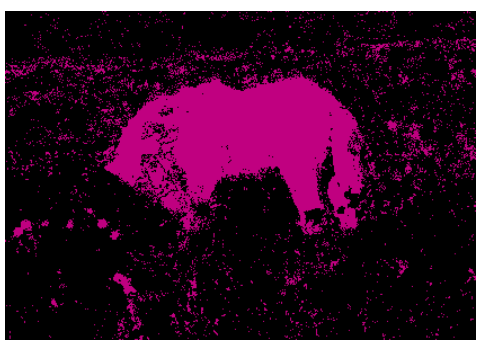
Line Search: ~~$O(K^M nd)$~~, $O(MK^2ND)$ per evaluation of $\mathcal{L}$

- N: number of samples
- D: dimension of feature space: $\approx \phi_{i,j}$ 1–10s, ϕ_i: 100s to 10000s
- M: number of output nodes
- K: number of possible labels of each output nodes

Typical feature functions in **image segmentation:**

- $\phi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: local image features, e.g. bag-of-words
  $\rightarrow$ $\langle w_i, \phi_i(y_i, x) \rangle$: local classifier (like logistic-regression)
- $\phi_{i,j}(y_i, y_j) = [\![y_i = y_j]\!] \in \mathbb{R}^1$: test for same label
  $\rightarrow$ $\langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle$: penalizer for label changes (if $w_{ij} > 0$)
- combined: $\operatorname{argmax}_y p(y|x)$ is smoothed version of local cues

original

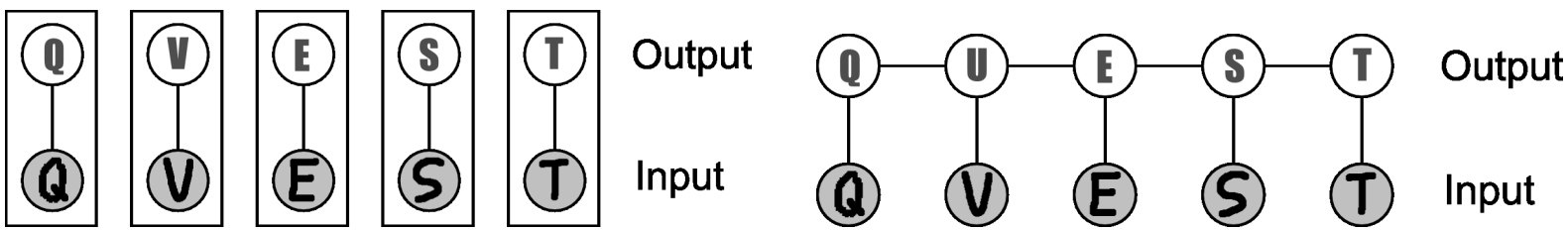


local classification

local + smoothness

Typical feature functions in **handwriting recognition:**

- $\phi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: image representation (pixels, gradients)
  $\rightarrow$ $\langle w_i, \phi_i(y_i, x) \rangle$: local classifier if x_i is letter y_i
- $\phi_{i,j}(y_i, y_j) = e_{y_i} \otimes e_{y_j} \in \mathbb{R}^{26 \cdot 26}$: letter/letter indicator
  $\rightarrow$ $\langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle$: encourage/suppress letter combinations
- combined: $\operatorname{argmax}_y p(y|x)$ is "corrected" version of local cues



local classification local + "correction"

Typical feature functions in **pose estimation:**

- $\phi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: local image representation, e.g. HoG
  $\rightarrow$ $\langle w_i, \phi_i(y_i, x) \rangle$: local confidence map
- $\phi_{i,j}(y_i, y_j) = \textit{good\_fit}(y_i, y_j) \in \mathbb{R}^1$: test for geometric fit
  $\rightarrow$ $\langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle$: penalizer for unrealistic poses
- together: $\operatorname{argmax}_y p(y|x)$ is sanitized version of local cues

original

local classification

local + geometry

[V. Ferrari, M. Marin-Jimenez, A. Zisserman: "Progressive Search Space Reduction for Human Pose Estimation", CVPR 2008.]

Typical feature functions for **CRFs in Computer Vision**:

- $\phi_i(y_i, x)$: local representation, high-dimensional
  $\rightarrow$ $\langle w_i, \phi_i(y_i, x) \rangle$: local classifier
- $\phi_{i,j}(y_i, y_j)$: prior knowledge, low-dimensional
  $\rightarrow$ $\langle w_{ij}, \phi_{ij}(y_i, y_j) \rangle$: penalize outliers
- learning adjusts parameters:
  - unary w_i: learn local classifiers and their importance
  - binary w_{ij}: learn importance of smoothing/penalization
- $\operatorname{argmax}_y p(y|x)$ is cleaned up version of local prediction

Solving the Training Optimization Problem Numerically **Idea:** split learning of unary potentials into two parts:

- local classifiers,
- their importance.

Two-Stage Training

- pre-train $f_i^y(x) \hat{=} \log p(y_i|x)$
- use $\tilde{\phi}_i(y_i, x) := f_i^y(x) \in \mathbb{R}^K$ (low-dimensional)
- keep $\phi_{ij}(y_i, y_j)$ are before
- perform CRF learning with $\tilde{\phi}_i$ and ϕ_{ij}

Advantage:

- lower dimensional feature space during inference $\rightarrow$ faster
- $f_i^y(x)$ can be stronger classifiers, e.g. non-linear SVMs

Disadvantage:

- if local classifiers are bad, CRF training cannot fix that.

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

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 $\phi : \mathcal{X} \times \mathbb{R}^D$.

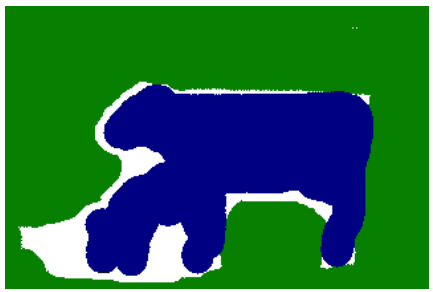
Task: find parameter vector w such that $\frac{1}{Z} \exp(\langle w, \phi(x, y) \rangle) \approx d(y|x)$.

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

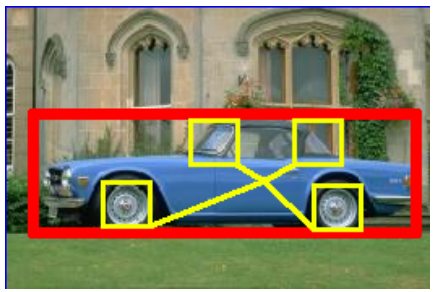
$$w^* = \operatorname*{argmin}_w \frac{1}{2\sigma^2} \|w\|^2 - \sum_{n=1}^{N} \left[\langle w, \phi(x^n, y^n) \rangle - \log \sum_{y \in \mathcal{Y}} e^{\langle w, \phi(x^n, y) \rangle} \right]$$

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

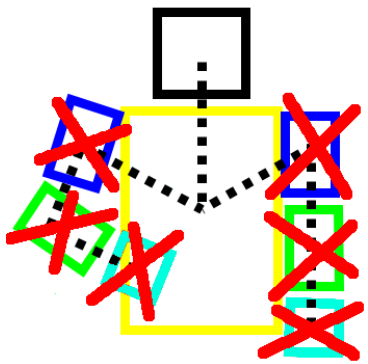
Extra I: Beyond Fully Supervised Learning So far, training was *fully supervised*, all variables were observed.
In real life, some variables can be *unobserved* even during training.

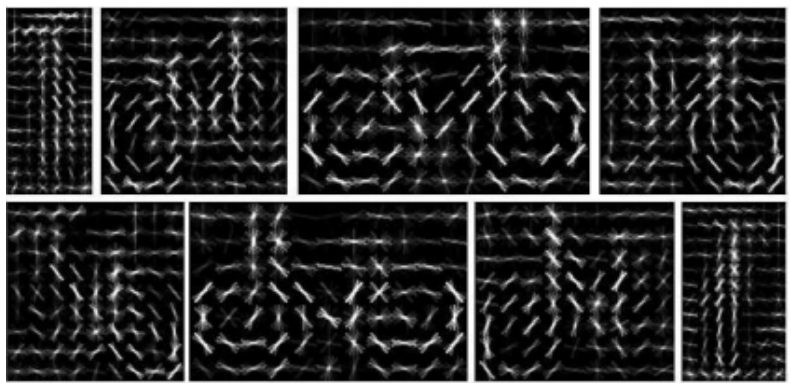
missing labels in training data



latent variables, e.g. part location

latent variables, e.g. part occlusion



latent variables, e.g. viewpoint

Graphical model: three types of variables

- $x \in \mathcal{X}$ always observed,
- $y \in \mathcal{Y}$ observed only in training,
- $z \in \mathcal{Z}$ never observed (latent).

Marginalization over Latent Variables

Construct conditional likelihood as usual:

$$p(y, z|x, w) = \frac{1}{Z(x, w)} \exp(\langle w, \phi(x, y, z) \rangle)$$

Derive $p(y|x, w)$ by marginalizing over z:

$$p(y|x, w) = \frac{1}{Z(x, w)} \sum_{z \in \mathcal{Z}} p(y, z|x, w)$$

Negative regularized conditional log-likelihood:

$$\begin{aligned}
\mathcal{L}(w) &= \lambda\|w\|^2 - \sum_{n=1}^{N} \log p(y^n|x^n, w) \\
&= \lambda\|w\|^2 - \sum_{n=1}^{N} \log \sum_{z\in\mathcal{Z}} p(y^n, z|x^n, w) \\
&= \lambda\|w\|^2 - \sum_{n=1}^{N} \log \sum_{z\in\mathcal{Z}} \exp(\langle w, \phi(x^n, y^n, z)\rangle) \\
&\qquad + \sum_{n=1}^{N} \log \sum_{\substack{z\in\mathcal{Z} \\ y\in\mathcal{Y}}} \exp(\langle w, \phi(x^n, y, z)\rangle)
\end{aligned}$$

- $\mathcal{L}$ is *not convex* in $w \rightarrow$ can have local minima
- no agreed on best way for treating latent variables