Introduction to Probabilistic Graphical Models

Christoph Lampert

IST Austria (Institute of Science and Technology Austria)

Schedule

- Refresher of Probabilities
- Introduction to Probabilistic Graphical Models
- Probabilistic Inference
- Learning Conditional Random Fields
- MAP Prediction / Energy Minimization
- Learning Structured Support Vector Machines

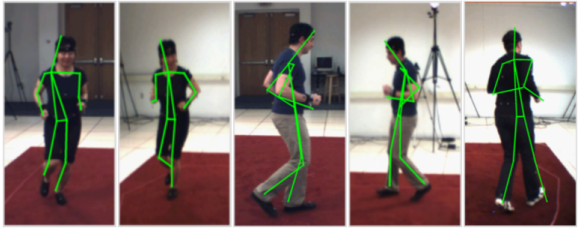
Links to slide download: `http://pub.ist.ac.at/~chl/courses/PGM_W16/`

Password for ZIP files (if any): `pgm2016`

Email for questions, suggestions or typos that you found: `chl@ist.ac.at`

Supervised Learning Problem

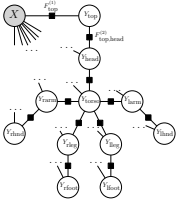
- Given training examples $(x^1, y^1), \ldots, (x^N, y^N) \in \mathcal{X} \times \mathcal{Y}$
  - X: collection of interacting random variables, *e.g.* image pixels
  - $x \in \mathcal{X}$: one joint assignment, *e.g.* a specific image, out of all possible joint assignments
  - Y: collection of interacting random variables, *e.g.* part locations
  - $y \in \mathcal{Y}$: one joint assignment, *e.g.* a specific pose, out of all possible joint assignments



Images: HumanEva dataset

Goal: be able to make predictions for new inputs, i.e. learn a function $f : \mathcal{X} \to \mathcal{Y}$.

Supervised Learning Problem



Step 1: define a proper graph structure of X and Y

Step 2: define a proper parameterization of $p(y|x;\theta)$

$$p(y|x;\theta) = \frac{1}{Z} e^{\sum_{i=1}^{d} \theta_i \phi_i(x,y)}$$

Step 3: learn parameters θ^* from training data

e.g. maximum likelihood
($\rightarrow$ today)

Step 4: for new $x \in \mathcal{X}$, make prediction

e.g. $y^* = \underset{y \in \mathcal{Y}}{\operatorname{argmax}}\, p(y|x;\theta^*)$
($\rightarrow$ next lecture)

Parameterization

Goal: Define feature functions, $\phi_i : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$, for $i = 1, \ldots, d$, or just $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$

Main step:

- For each factor $F \in \mathcal{F}$ we define some $\phi_F(x_F, y_F) : \mathcal{X}_F \times \mathcal{Y}_F \to \mathbb{R}^{d_F}$,

  where (x_F, y_F) are those random variables of (x, y) that appear in F.

Example: pose estimation (X: image pixels, Y: part locations)

- $\phi_{\texttt{head}}(x, y_i) =$ "the pixels in the image x at the location specified by y_i"
- $\phi_{\texttt{head-torso}}(y_i, y_j) =$ "the distance between locations y_i and y_j"

Example: image segmentation (X: image pixels, Y: per-pixel foreground/background flag)

- "unary terms" $\quad \phi_i(x_i, y_i) = \begin{pmatrix} x_i [\![y_i = 0]\!] \\ (1 - x_i) [\![y_i = 1]\!] \end{pmatrix}$
- "pairwise terms" $\quad \phi_{ij}(y_i, y_j) = [\![y_i \neq y_j]\!]$

Parameterization

Goal: Define feature functions, $\phi_i : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$, for $i = 1, \ldots, d$, or just $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$

Two common options (combinations of them also work):

- 1) combine factors by stacking their vectors $\phi(x, y) = \big(\phi_F(x_F, y_F)\big)_{F \in \mathcal{F}}$

$$p(y|x;\theta) = \frac{1}{Z(x;\theta)} e^{-\langle \theta, \phi(x,y) \rangle} \quad \text{with} \quad \langle \theta, \phi(x,y) \rangle = \sum_{F \in \mathcal{F}} \langle \theta_F, \phi_F(x_F, y_F) \rangle$$

- 2) combine factors by summing their vectors $\phi(x, y) = \sum_{F \in \mathcal{F}} \phi_F(x_F, y_F)$

$$p(y|x;\theta) = \frac{1}{Z(x;\theta)} e^{-\langle \theta, \phi(x,y) \rangle} \quad \text{with} \quad \langle \theta, \phi(x,y) \rangle = \sum_{F \in \mathcal{F}} \langle \theta, \phi_F(x_F, y_F) \rangle$$

Result: log-linear model with parameter vector θ (sometimes: w, for weight vector)

Conditional Random Field

Maximum Likelihood Parameter Estimation

Maximize likelihood of outputs $y^1, \ldots, y^N$ for inputs $x^1, \ldots, x^N$

$$\theta^* = \operatorname*{argmax}_{\theta \in \mathbb{R}^D} \; p(y^1, \ldots, y^N | x^1, \ldots, x^N; \theta) \overset{i.i.d.}{=} \operatorname*{argmax}_{\theta \in \mathbb{R}^D} \; \prod_{n=1}^{N} p(y^n | x^n; \theta)$$

$$\overset{-\log(\cdot)}{=} \operatorname*{argmin}_{\theta \in \mathbb{R}^D} \underbrace{- \sum_{n=1}^{N} \log p(y^n | x^n; \theta)}_{\text{negative conditional log-likelihood (of } \mathcal{D})}$$

Maximum Likelihood Parameter Estimation

Maximize likelihood of outputs $y^1, \ldots, y^N$ for inputs $x^1, \ldots, x^N$

$$\theta^* = \operatorname*{argmax}_{\theta \in \mathbb{R}^D} \; p(y^1, \ldots, y^N | x^1, \ldots, x^N; \theta) \overset{i.i.d.}{=} \operatorname*{argmax}_{\theta \in \mathbb{R}^D} \; \prod_{n=1}^{N} p(y^n | x^n; \theta)$$

$$\overset{-\log(\cdot)}{=} \operatorname*{argmin}_{\theta \in \mathbb{R}^D} \; \underbrace{-\sum_{n=1}^{N} \log p(y^n | x^n; \theta)}_{\text{negative conditional log-likelihood (of } \mathcal{D})}$$

$$= \operatorname*{argmin}_{\theta \in \mathbb{R}^D} \; -\sum_{n=1}^{N} \left[\log e^{-\langle \theta, \phi(x^n, y) \rangle} - \log Z(x^n; \theta) \right]$$

$$= \operatorname*{argmin}_{\theta \in \mathbb{R}^D} \; \sum_{n=1}^{N} \left[\langle \theta, \phi(x^n, y^n) \rangle + \underbrace{\log \sum_{y \in \mathcal{Y}} e^{-\langle \theta, \phi(x^n, y) \rangle}}_{\text{log-partition function}} \right]$$

MAP Estimation of θ

Treat θ as random variable; maximize posterior $p(\theta|\mathcal{D})$

MAP Estimation of θ

Treat θ as random variable; maximize posterior $p(\theta|\mathcal{D})$

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

$p(\theta)$: *prior belief* on θ (cannot be estimated from data).

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

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

Choices for $p(\theta)$:

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

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

- $p(\theta) := \mathit{const.} \cdot e^{-\frac{\lambda}{2}\|\theta\|^2}$ (Gaussian)

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

Probabilistic Models for Structured Prediction - Summary

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

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

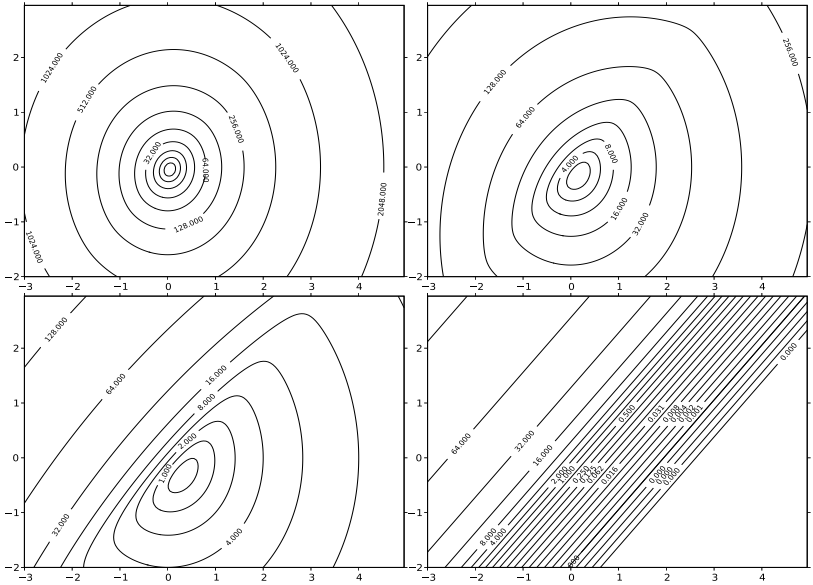
($\lambda \to 0$ makes it *unregularized*)

Probabilistic parameter estimation or *training* means solving

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

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

Negative Conditional Log-Likelihood (Toy Example)



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

input tolerance $\epsilon > 0$

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

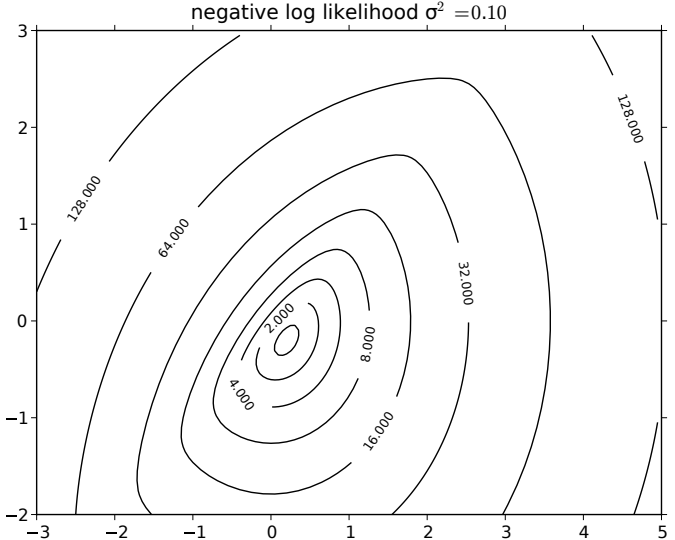
output θ_{cur}

Alternatives:

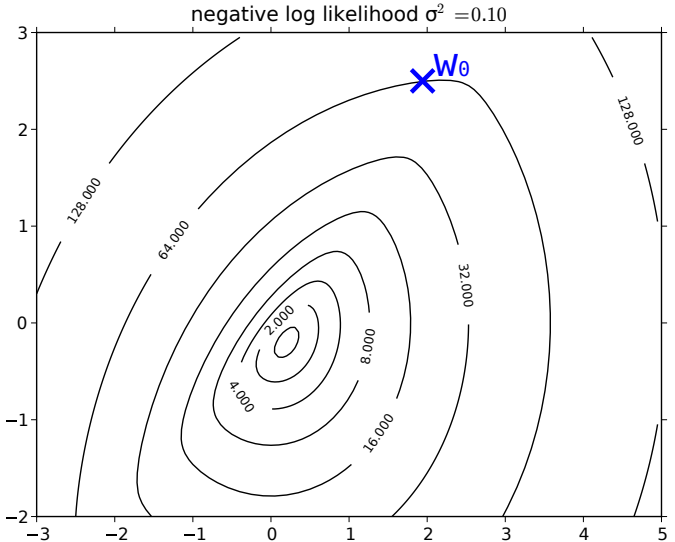
- L-BFGS (second-order descent without explicit Hessian)
- Conjugate Gradient

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

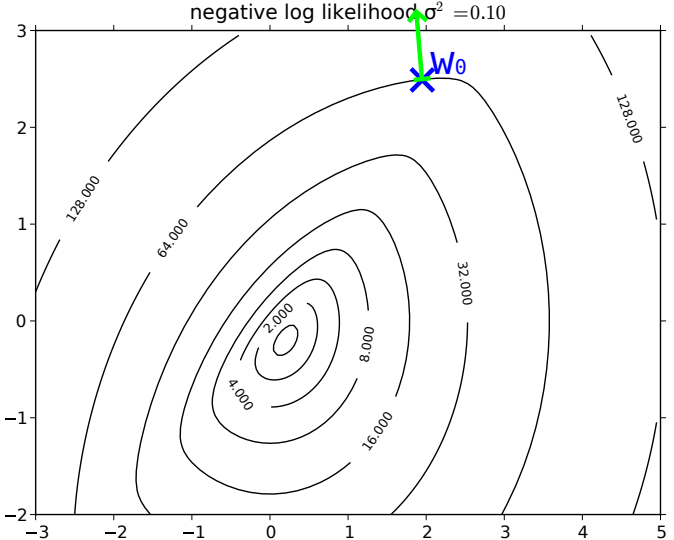
Steepest Descent Minimization – minimize $\mathcal{L}(\theta)$



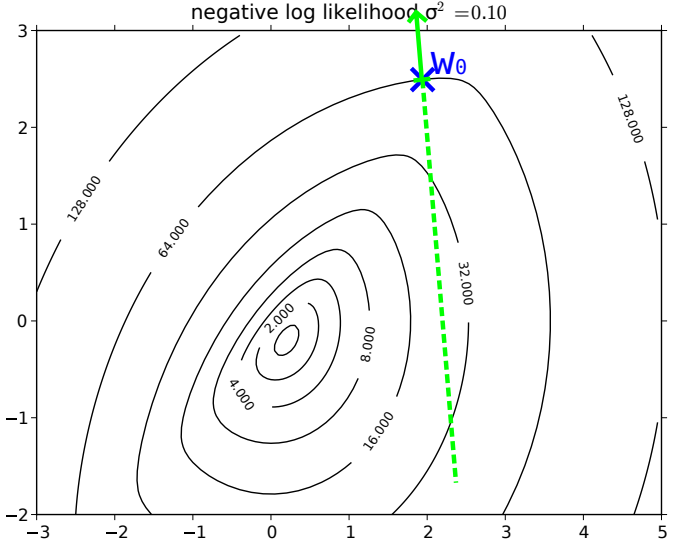
Steepest Descent Minimization – minimize $\mathcal{L}(\theta)$



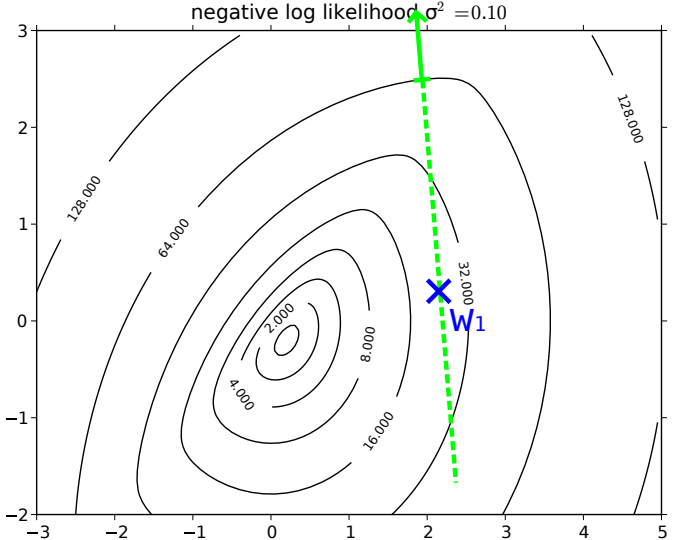
Steepest Descent Minimization – minimize $\mathcal{L}(\theta)$



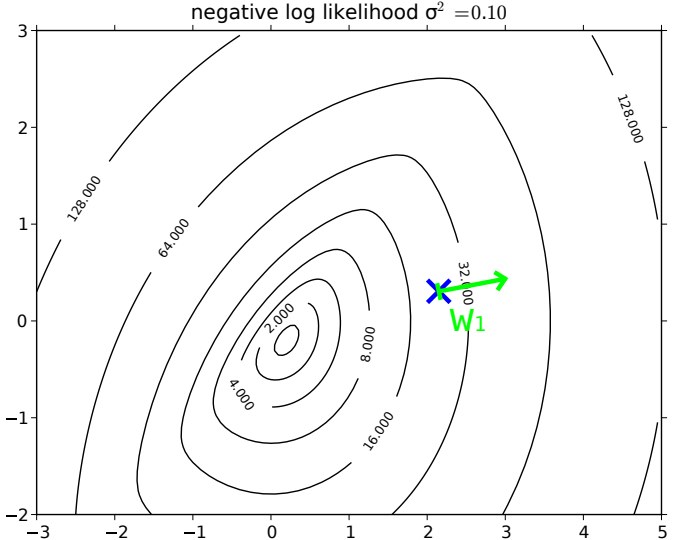
Steepest Descent Minimization – minimize $\mathcal{L}(\theta)$



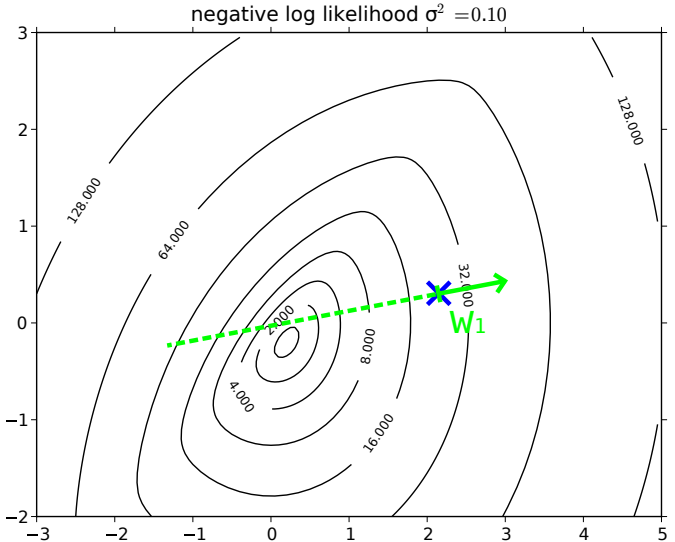
Steepest Descent Minimization – minimize $\mathcal{L}(\theta)$



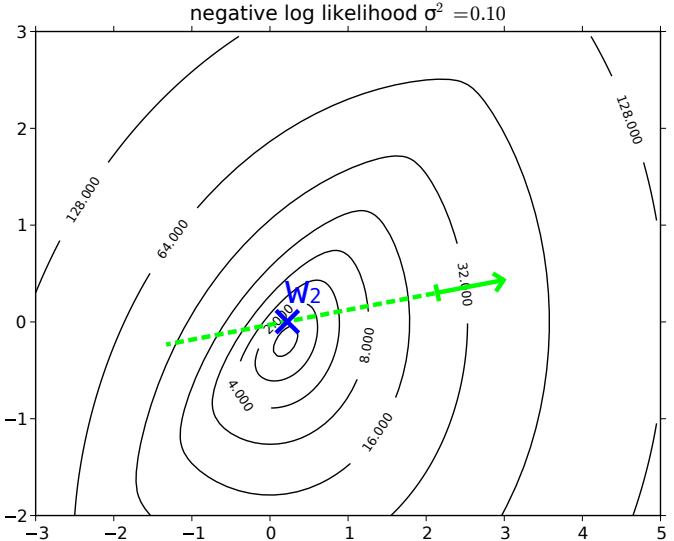
Steepest Descent Minimization – minimize $\mathcal{L}(\theta)$



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



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



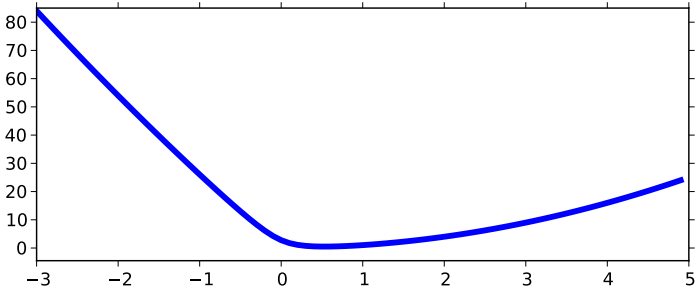
$$\mathcal{L}(\theta) = \frac{\lambda}{2}\|\theta\|^2 + \sum_{n=1}^{N} \left[\langle \theta, \phi(x^n, y^n)\rangle + \log \sum_{y \in \mathcal{Y}} e^{-\langle \theta, \phi(x^n, y)\rangle}\right]$$

$$\begin{aligned}
\nabla_\theta \mathcal{L}(\theta) &= \lambda\theta + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \frac{\sum_{y \in \mathcal{Y}} e^{-\langle \theta, \phi(x^n, y)\rangle} \phi(x^n, y)}{\sum_{\bar{y} \in \mathcal{Y}} e^{-\langle \theta, \phi(x^n, \bar{y})\rangle}}\right] \\
&= \lambda\theta + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \sum_{y \in \mathcal{Y}} p(y|x^n; \theta)\phi(x^n, y)\right] \\
&= \lambda\theta + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n;\theta)}\phi(x^n, y)\right]
\end{aligned}$$

$$\Delta\mathcal{L}(\theta) = \lambda Id_{D \times D} + \sum^{N} \mathbb{E}_{y \sim p(y|x^n;\theta)} \Big\{ \ \phi(x^n, y)\phi(x^n, y)^\top \Big\}$$

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

- continuous (not discrete), C^∞-differentiable on all $\mathbb{R}^D$.



slice through objective value ($\theta_x \in [-3, 5]$, $\theta_y = 0$)

$$\nabla_\theta \mathcal{L}(\theta) = \lambda\theta + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y\sim p(y|x^n;\theta)}\phi(x^n, y)\right]$$

- For $\lambda = 0$:

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

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

Interpretation:

- We want the model distribution to match the empirical one:

$$\mathbb{E}_{y\sim p(y|x;\theta)}\phi(x, y) \stackrel{!}{=} \phi(x, y^{\text{obs}})$$

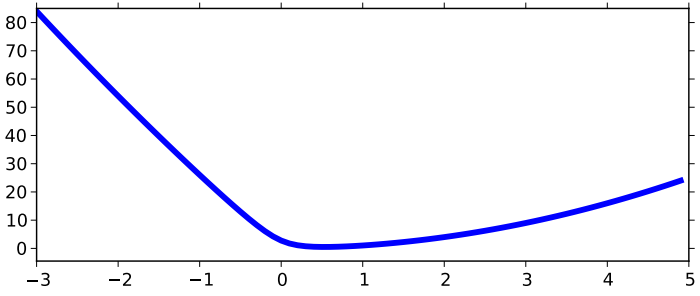
- *E.g.* image segmentation

  ϕ_{unary}: correct amount of foreground vs. background

  ϕ_{pairwise}: correct amount of fg/bg transitions $\rightarrow$ smoothness

$$\Delta\mathcal{L}(\theta) = \lambda Id_{D\times D} + \sum_{n=1}^{N} \mathbb{E}_{y\sim p(y|x^n;\theta)}\Big\{\ \phi(x^n, y)\phi(x^n, y)^\top\Big\}$$

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



slice through objective value ($\theta_x \in [-3, 5]$, $\theta_y = 0$)

Milestone I: Probabilistic Training (Conditional Random Fields)

- $p(y|x;\theta)$ log-linear in $\theta \in \mathbb{R}^D$.
- Training: minimize (regularized) negative conditional log-likelihood, $\mathcal{L}(\theta)$
- $\mathcal{L}(\theta)$ is differentiable and *convex*,
  $\rightarrow$ gradient descent will find global optimum with $\nabla_\theta \mathcal{L}(\theta) = 0$
- Same structure as multi-class *logistic regression* ($\hat{=}$ no structure and $\mathcal{Y} = \{1, \ldots, K\}$).

Milestone I: Probabilistic Training (Conditional Random Fields)

- $p(y|x;\theta)$ log-linear in $\theta \in \mathbb{R}^D$.
- Training: minimize (regularized) negative conditional log-likelihood, $\mathcal{L}(\theta)$
- $\mathcal{L}(\theta)$ is differentiable and *convex*,
  $\rightarrow$ gradient descent will find global optimum with $\nabla_\theta \mathcal{L}(\theta) = 0$
- Same structure as multi-class *logistic regression* ($\hat{=}$ no structure and $\mathcal{Y} = \{1, \ldots, K\}$).

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

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

Solving the Training Optimization Problem Numerically

Task: Compute $v = \nabla_\theta \mathcal{L}(\theta_{cur})$, evaluate $\mathcal{L}(\theta_{cur} + \eta v)$:

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

$$\nabla_\theta \mathcal{L}(\theta) = \frac{\lambda}{2}\theta + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \sum_{y \in \mathcal{Y}} p(y|x^n; \theta)\phi(x^n, y)\right]$$

Solving the Training Optimization Problem Numerically

Task: Compute $v = \nabla_\theta \mathcal{L}(\theta_{cur})$, evaluate $\mathcal{L}(\theta_{cur} + \eta v)$:

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

$$\nabla_\theta \mathcal{L}(\theta) = \frac{\lambda}{2}\theta + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \sum_{y \in \mathcal{Y}} p(y|x^n; \theta)\phi(x^n, y)\right]$$

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

- binary image segmentation: $|\mathcal{Y}| = 2^{640 \times 480} \approx 10^{92475}$
- 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

$$\nabla_\theta \mathcal{L}(\theta) = \lambda\theta + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n;\theta)} \phi(x^n, y)\right]$$

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

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

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

- N: number of samples
- D: dimension of feature space
- M: number of output variables
- K: number of possible labels of each output variables

Solving the Training Optimization Problem Numerically

$$\nabla_\theta \mathcal{L}(\theta) = \lambda\theta + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n;\theta)} \phi(x^n, y)\right]$$

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

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

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

- N: number of samples
- D: dimension of feature space
- M: number of output variables $\approx$ 10s to 1,000,000s
- K: number of possible labels of each output variables $\approx$ 2 to 1000s

Solving the Training Optimization Problem Numerically

For a graphical model with factors $\mathcal{F}$, the features decompose:

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

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

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

Factor marginals $\mu_F = p(y_F|x;\theta)$

- are much smaller than complete joint distribution $p(y|x;\theta)$,
- compute them by probabilistic inference.

Solving the Training Optimization Problem Numerically

For a graphical model with factors $\mathcal{F}$, the features decompose:

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

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

Again, we need only the factor marginals $\mu_F = p(y_F|x;\theta)$

Solving the Training Optimization Problem Numerically

$$\nabla_\theta \mathcal{L}(\theta) = \lambda\theta + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y\sim p(y|x^n;\theta)}\phi(x^n, y)\right]$$

Computing the Gradient (if marginal inference is possible): ~~$O(K^M ND)$~~, $O(MK^{|F_{max}|}ND)$:

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

- N: number of samples
- D: dimension of feature space
- M: number of output variables
- K: number of possible labels of each output variables

Solving the Training Optimization Problem Numerically

$$\nabla_\theta \mathcal{L}(\theta) = \lambda\theta + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y\sim p(y|x^n;\theta)}\phi(x^n, y)\right]$$

Computing the Gradient (if marginal inference is possible): ~~$O(K^M ND)$~~, $O(MK^{|F_{max}|}ND)$:

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

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

Solving the Training Optimization Problem Numerically

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

Stochastic Gradient Descent (SGD)

- Minimize $\mathcal{L}(\theta)$, but without ever computing $\mathcal{L}(\theta)$ or $\nabla\mathcal{L}(\theta)$ exactly
- In each gradient descent step:
  - Pick random subset $\mathcal{D}' \subset \mathcal{D}$, $\quad \leftarrow$ **often just 1–3 elements**
  - Compute approximate gradient

$$\tilde{\nabla}\mathcal{L}(\theta) = \lambda\theta + \frac{|\mathcal{D}|}{|\mathcal{D}'|} \sum_{(x^n, y^n) \in \mathcal{D}'} \left[\phi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n;\theta)} \phi(x^n, y)\right]$$

  - Update parameter in negative direction of approximate gradient

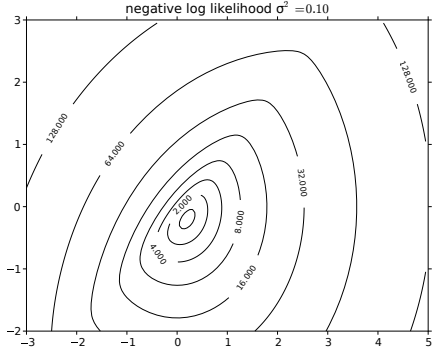
Line search would still be too slow $\rightarrow$ use fixed stepsize rule, η_t, instead (new parameter)

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

Stochastic Gradient Descent (SGD)

Important property of SGD:

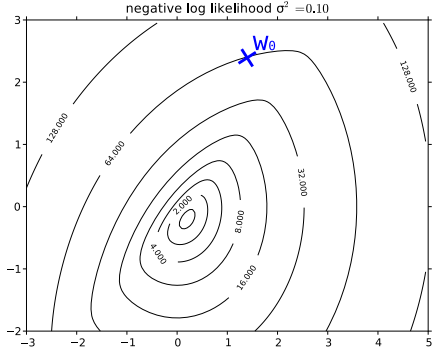
- $v = \tilde{\nabla}\mathcal{L}(\theta)$ is random, let's call it's distribution q



Stochastic Gradient Descent (SGD)

Important property of SGD:

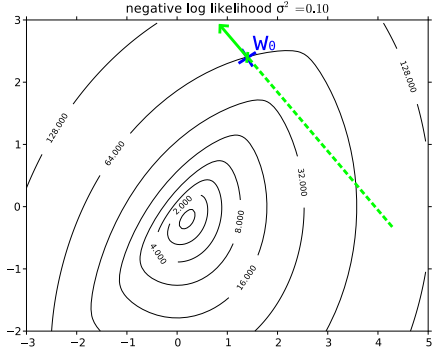
- $v = \tilde{\nabla}\mathcal{L}(\theta)$ is random, let's call it's distribution q



Stochastic Gradient Descent (SGD)

Important property of SGD:

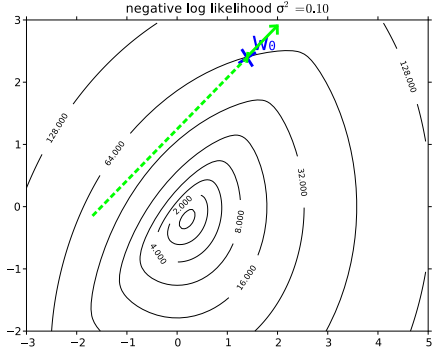
- $v = \tilde{\nabla}\mathcal{L}(\theta)$ is random, let's call it's distribution q



Stochastic Gradient Descent (SGD)

Important property of SGD:

- $v = \tilde{\nabla}\mathcal{L}(\theta)$ is random, let's call it's distribution q



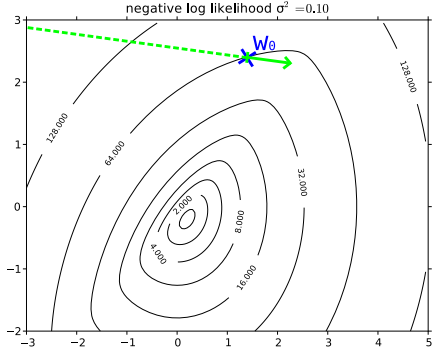
Stochastic Gradient Descent (SGD)

Important property of SGD:

- $v = \tilde{\nabla}\mathcal{L}(\theta)$ is random, let's call it's distribution q
- v is unbiased estimate of true gradient:

$$\mathbb{E}_{v \sim q}[v] = \nabla \mathcal{L}(\theta)$$

- "on average", parameters updates point in the right direction



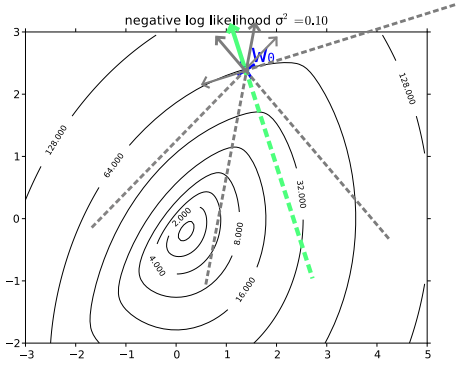
Stochastic Gradient Descent (SGD)

Important property of SGD:

- $v = \tilde{\nabla}\mathcal{L}(\theta)$ is random, let's call it's distribution q
- v is unbiased estimate of true gradient:

$$\mathbb{E}_{v \sim q}[v] = \nabla \mathcal{L}(\theta)$$

- "on average", parameters updates point in the right direction



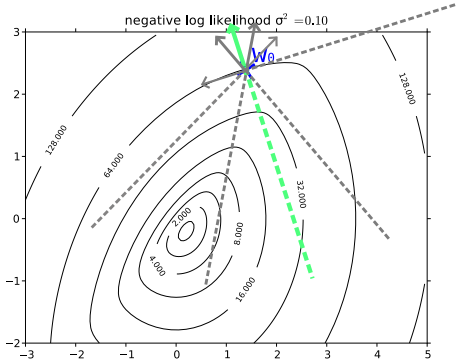
Stochastic Gradient Descent (SGD)

Important property of SGD:

- $v = \tilde{\nabla}\mathcal{L}(\theta)$ is random, let's call it's distribution q
- v is unbiased estimate of true gradient:

$$\mathbb{E}_{v\sim q}[v] = \nabla\mathcal{L}(\theta)$$

- "on average", parameters updates point in the right direction



- SGD converges to $\text{argmin}_\theta\, \mathcal{L}(\theta)$
  - η_t must be chosen appropriately: $\sum_t \eta_t = \infty, \sum_t \eta_t^2 < \infty$, e.g. $\eta_t \propto \frac{1}{t}$
- SGD needs more iterations, but each one is much faster

Solving the Training Optimization Problem Numerically

$$\nabla_\theta \mathcal{L}(\theta) = \lambda\theta + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y\sim p(y|x^n;\theta)}\phi(x^n, y)\right]$$

Computing the Gradient (if marginal inference is possible): $O(MK^{|F_{max}|}ND)$:

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

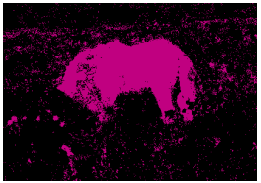
- N: number of samples
- D: dimension of feature space: $\approx \phi_{i,j}$ 1–10s, ϕ_i: 10s to 10000s
- M: number of output variables
- K: number of possible labels of each output variables

Solving the Training Optimization Problem Numerically

Typical feature functions in **image segmentation:**

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

original



local confidence

local + smoothness

Solving the Training Optimization Problem Numerically

Typical feature functions in **pose estimation:**

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

original

local confidence

local + geometry

Solving the Training Optimization Problem Numerically

Idea: split learning of unary potentials into two parts:

- local classifiers,
- their importance.

Two-Stage Training

- pre-train $f_i^y(x) \mathrel{\hat{=}} \log p(y_i|x)$
- 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}

Solving the Training Optimization Problem Numerically

Idea: split learning of unary potentials into two parts:

- local classifiers,
- their importance.

Two-Stage Training

- pre-train $f_i^y(x) \mathrel{\hat{=}} \log p(y_i|x)$
- 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 any classifiers, *e.g.* deep network, random forest, ...

Disadvantage:

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

Solving the Training Optimization Problem Numerically

$$\nabla_\theta \mathcal{L}(\theta) = \lambda\theta + \sum_{n=1}^{N} \left[\phi(x^n, y^n) - \mathbb{E}_{y\sim p(y|x^n;\theta)}\phi(x^n, y)\right]$$

Computing the Gradient: $O(K^M ND)$

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

- N: number of samples
- D: dimension of feature space
- M: number of output variables
- K: number of possible labels of each output variables

What, if exact marginal inference is not possible?

Training with Approximate Inference

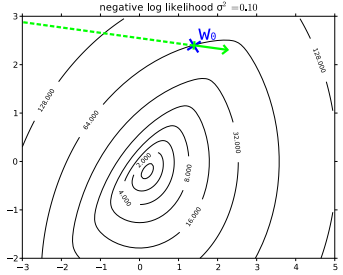
Problem: exact marginal inference might be computationally too costly

Idea: just use approximate marginals, *e.g.* from loopy BP

Possibility 1: everything works fine

Possibility 2: nothing works anymore

- approximate gradient might not be a descent direction (objective does not decrease along its negative)
- then, gradient descent might
  - terminate too early (line search chooses $\eta = 0$), or
  - not convergence at all, or
  - convergence to a bad solution, ...



In general: no way to tell which of the possibilities.

Training with Approximate Likelihood

Question: how to train if (approximate) maximum likelihood learning does not work

Idea: optimize a simpler quantity instead of $\mathcal{L}$

Training with Approximate Likelihood

Question: how to train if (approximate) maximum likelihood learning does not work

Idea: optimize a simpler quantity instead of $\mathcal{L}$

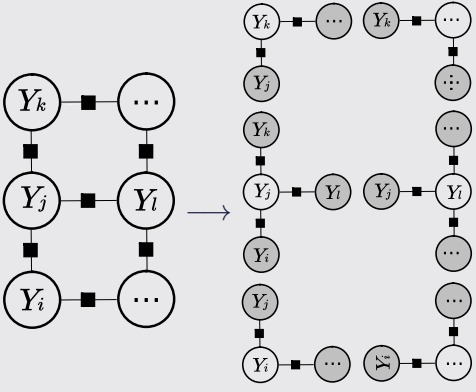
Pseudolikelihood [Besag, 1987]

Pretend:

$$p(y) \approx p_{\mathrm{PL}}(y)$$

for

$$p_{\mathrm{PL}}(y) = \prod_{i \in V} p(y_i | y_{V \setminus \{i\}})$$
$$= \prod_{i \in V} p(y_i | y_{N(i)})$$



Training with Approximate Likelihood – Pseudolikelihood (PL)

$$p(y|x;\theta) \approx p_{\text{PL}}(y|x;\theta) = \prod_{i \in V} p(y_i|y_{N(i)}, x;\theta)$$

For training data $\{(x^1, y^1), \ldots, (x^N, y^N)\}$:

$$\begin{aligned}
\mathcal{L}_{PL}(\theta) &= \log \prod_{n=1}^{N} p_{\text{PL}}(y^n|x^n;\theta) \\
&= \sum_{n=1}^{N} \sum_{i \in V} \log p(y_i^n|y_{N(i)}^n, x;\theta) \\
&= \sum_{n=1}^{N} \sum_{i \in V} \left[\langle \theta, \phi(y^n, x^n) \rangle - \log \sum_{\mathbf{z} \in \mathcal{Y}_i} e^{\langle \theta, \phi(y_1^n, \ldots, y_{i-1}^n, \mathbf{z}, y_{i+1}^n, \ldots, y_{|V|}^n, x^n) \rangle} \right]
\end{aligned}$$

Training with Approximate Likelihood – Pseudolikelihood (PL)

$$p(y|x;\theta) \approx p_{\text{PL}}(y|x;\theta) = \prod_{i \in V} p(y_i|y_{N(i)}, x;\theta)$$

For training data $\{(x^1, y^1), \ldots, (x^N, y^N)\}$:

$$\begin{aligned}
\mathcal{L}_{PL}(\theta) &= \log \prod_{n=1}^{N} p_{\text{PL}}(y^n|x^n;\theta) \\
&= \sum_{n=1}^{N} \sum_{i \in V} \log p(y_i^n|y_{N(i)}^n, x;\theta) \\
&= \sum_{n=1}^{N} \sum_{i \in V} \left[\langle \theta, \phi(y^n, x^n) \rangle - \log \sum_{\mathbf{z} \in \mathcal{Y}_i} e^{\langle \theta, \phi(y_1^n, \ldots, y_{i-1}^n, \mathbf{z}, y_{i+1}^n, \ldots, y_{|V|}^n, x^n) \rangle} \right]
\end{aligned}$$

Partition functions sum only over one variable at a time $\rightarrow$ tractable

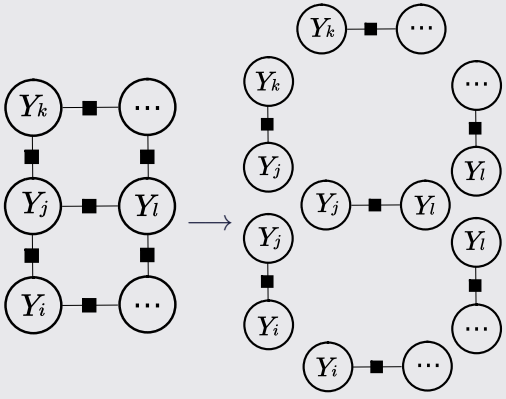
Training with Approximate Likelihood – Piecewise Training (PW)

Idea: optimize a simpler quantity instead of $\mathcal{L}$

Piecewise Training [Sutton, McCallum, 2005]

$$p(y|x) \approx \prod_{F \in \mathcal{F}} p_F(y_F|x) \quad \text{for}$$

$$p_F(y_F|x) = \frac{1}{Z_F(x;\theta)} e^{-\langle \theta_F, \phi_F(y_F,x) \rangle}$$



Training with Approximate Likelihood – Piecewise Training (PW)

$$p(y|x) \approx \prod_{F \in \mathcal{F}} p_F(y_F|x;\theta_F) \quad \text{for} \quad p_F(y_F|x) \propto e^{-\langle \theta_F, \phi_F(y_F,x)\rangle}$$

For training data $\{(x^1,y^1),\ldots,(x^N,y^N)\}$:

$$\mathcal{L}_{PW}(\theta) = \log \prod_{n=1}^{N} p_{\text{PW}}(y_F^n|x^n;\theta) = \sum_{n=1}^{N} \sum_{F \in \mathcal{F}} \log p_F(y_F^n|x)$$

$$= \sum_{n=1}^{N} \sum_{F \in \mathcal{F}} \left[\langle \theta_F, \phi_F(y_F^n, x^n)\rangle - \log \sum_{\bar{y}_F \in \mathcal{Y}_F} e^{\langle \theta_F, \phi_F(\bar{y}_F, x^n)\rangle} \right]$$

Training with Approximate Likelihood – Piecewise Training (PW)

$$p(y|x) \approx \prod_{F \in \mathcal{F}} p_F(y_F|x;\theta_F) \quad \text{for} \quad p_F(y_F|x) \propto e^{-\langle \theta_F, \phi_F(y_F,x)\rangle}$$

For training data $\{(x^1, y^1), \ldots, (x^N, y^N)\}$:

$$\mathcal{L}_{PW}(\theta) = \log \prod_{n=1}^{N} p_{\text{PW}}(y_F^n|x^n;\theta) = \sum_{n=1}^{N} \sum_{F \in \mathcal{F}} \log p_F(y_F^n|x)$$

$$= \sum_{n=1}^{N} \sum_{F \in \mathcal{F}} \left[\langle \theta_F, \phi_F(y_F^n, x^n)\rangle - \log \sum_{\bar{y}_F \in \mathcal{Y}_F} e^{\langle \theta_F, \phi_F(\bar{y}_F, x^n)\rangle} \right]$$

Partition functions sum over $|F|$ variables at a time $\rightarrow$ usually tractable

Training with Approximate Likelihood – Piecewise Training (PW)

$$p(y|x) \approx \prod_{F \in \mathcal{F}} p_F(y_F|x;\theta_F) \quad \text{for} \quad p_F(y_F|x) \propto e^{-\langle \theta_F, \phi_F(y_F,x)\rangle}$$

For training data $\{(x^1, y^1), \ldots, (x^N, y^N)\}$:

$$\mathcal{L}_{PW}(\theta) = \log \prod_{n=1}^{N} p_{\text{PW}}(y_F^n|x^n;\theta) = \sum_{n=1}^{N} \sum_{F \in \mathcal{F}} \log p_F(y_F^n|x)$$

$$= \sum_{F \in \mathcal{F}} \sum_{n=1}^{N} \left[\langle \theta_F, \phi_F(y_F^n, x^n)\rangle - \log \sum_{\bar{y}_F \in \mathcal{Y}_F} e^{\langle \theta_F, \phi_F(\bar{y}_F, x^n)\rangle} \right]$$

Partition functions sum over $|F|$ variables at a time $\rightarrow$ usually tractable

Optimization decomposes into a sum over the θ_F $\rightarrow$ easy to parallelize

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:

$$\tilde{\nabla}_\theta \mathcal{L}(\theta) = \lambda\theta - \sum_{\mathbf{n=1}}^{\mathbf{N}} \left[\phi(x^n, y^n) - \sum_{\mathbf{y}\in\mathcal{Y}} \mathbf{p(y|x^n;\theta)}\, \phi(x^n, y)\right] \quad \in \mathbb{R}^{\mathbf{D}}$$

A lot of research on accelerating CRF training:

problem	**"solution"**	**method(s)**
$\lvert\mathcal{Y}\rvert$ too large	exploit structure	(loopy) belief propagation
	fast sampling	contrastive divergence
	surrogate $\mathcal{L}$	pseudo-likelihood, piecewise training
N too large	mini-batches	stochastic gradient descent
D too large	pretrained ϕ_{unary}	two-stage training

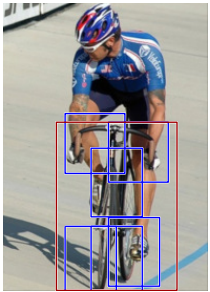
CRFs with Latent Variables

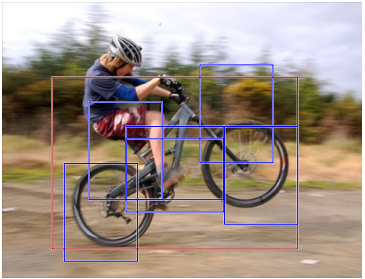
Three types of variables in graphical model:

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

Example:

- x : image
- y : part positions
- $z \in \{0, 1\}$: flag
  front-view or *side-view*





CRFs with Latent Variables

Idea 1: Gradient Descent

Construct conditional probability as usual:

$$p(y, z|x; \theta) = \frac{1}{Z(x; \theta)} e^{-\langle \theta, \phi(x,y,z) \rangle}$$

Derive $p(y|x; \theta)$ by marginalizing over z:

$$p(y|x; \theta) = \sum_{z \in \mathcal{Z}} p(y, z|x; \theta) \; = \; \frac{1}{Z(x; \theta)} \sum_{z \in \mathcal{Z}} e^{-\langle \theta, \phi(x,y,z) \rangle}$$

Maximize conditional log-likelihood by gradient descent w.r.t. θ.

CRFs with Latent Variables

Negative regularized conditional log-likelihood:

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

- $\mathcal{L}$ is *not convex* in $\theta \rightarrow$ local minima possible

CRFs with Latent Variables

Gradient of one log-likelihood component:

$$\nabla_\theta \log p(y^n|x^n;\theta) = \nabla_\theta \log \sum_{z\in\mathcal{Z}} e^{-\langle\theta,\phi(x^n,y^n,z)\rangle} - \nabla_\theta \log \sum_{\substack{z\in\mathcal{Z}\\ y\in\mathcal{Y}}} e^{-\langle\theta,\phi(x^n,y,z)\rangle}$$

$$= \mathbb{E}_{z\sim p(z|x^n,y^n;\theta)}\phi(x^n,y^n,z) - \mathbb{E}_{(y,z)\sim p(y,z|x^n;\theta)}\phi(x^n,y,z)$$

If feature function decompose over factor graph:

- gradient computable using probability inference w.r.t. y, z

CRFs with Latent Variables

Idea 2: Expectation Maximization

Alternate between estimating a distribution over z and parameter updates:

repeat
  for $n = 1, \ldots, N$ **do**
    $q^n(z) \leftarrow p(z|x^n, y^n; \hat{\theta})$
  end for
  $\hat{\theta} \leftarrow \operatorname{argmin}_{\theta} \frac{\lambda}{2}\|\theta\|^2 - \sum_{n=1}^{N} \mathbb{E}_{z\sim q^n} \log p(x^n, y^n, z; \theta)$ // variational bound
until convergence

Variational bound is *convex* $\rightarrow$ unique minimizer $\hat{\theta}$

Expectation Maximization

Gradient of one component of the variational lower bound:

$$
\begin{aligned}
-\nabla_\theta \mathbb{E}_{z\sim q^n} \log p(x^n, y^n, z) &= -\mathbb{E}_{z\sim q^n} \nabla_\theta \log p(x^n, y^n, z) \\
&= \mathbb{E}_{z\sim q^n} \nabla_\theta \Big[\langle \theta, \phi(x^n, y^n, z) \rangle - \log \sum_y e^{-\langle \theta, \phi(x^n, y, z) \rangle} \Big] \\
&= \mathbb{E}_{z\sim q^n} \phi(x^n, y^n, z) - \mathbb{E}_{z\sim q^n} \mathbb{E}_{y\sim \phi(y|x^n, z; \theta)} \phi(x^n, y, z)
\end{aligned}
$$

If feature function decompose over factor graph:

- only marginal/factor probabilities of q^n need $\rightarrow$ can be stored explicitly,
- gradient computable using probability inference w.r.t. y

Training CRFs with Latent Variables

Idea 2b: Hard Expectation Maximization

Alternate between estimating the 'most likely' z's and parameter updates:

repeat
  for $n = 1, \ldots, N$ **do**
    $\hat{z}^n \leftarrow \operatorname{argmax}_{z \in \mathcal{Z}} p(z|x^n, y^n; \theta) = \operatorname{argmin}_{z \in \mathcal{Z}} \langle \theta, \phi(x^n, y^n, z)$
  end for
  $\hat{\theta} \leftarrow \operatorname*{argmin}_{\theta} \frac{\lambda}{2}\|\theta\|^2 - \sum_{n=1}^{N} \log p(x^n, y^n, z^n; \theta)$ // as if all data observed
until convergence

Like EM when only δ-peaks are allowed only for q^n.

Summary – CRF Learning

- Given: training set $\{(x^1, y^1), \ldots, (x^N, y^N)\} \subset \mathcal{X} \times \mathcal{Y}$
- Choose: feature functions $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$
  that decompose over factors, $\phi_F : \mathcal{X} \times \mathcal{Y}_F \to \mathbb{R}^d$ for $F \in \mathcal{F}$

Energy is linear in parameter vector $\theta = (\theta_F)_{F \in \mathcal{F}}$

$$E(x, y; \theta) = \langle \theta, \phi(x, y) \rangle = \sum_{F \in \mathcal{F}} \langle \theta_F, \phi_F(y_F, x) \rangle$$

Overall model is log-linear: $p(y|x; \theta) \propto e^{-\langle \theta, \phi(x,y) \rangle}$

Summary – CRF Learning

- Given: training set $\{(x^1, y^1), \dots, (x^N, y^N)\} \subset \mathcal{X} \times \mathcal{Y}$
- Choose: feature functions $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$
  that decompose over factors, $\phi_F : \mathcal{X} \times \mathcal{Y}_F \to \mathbb{R}^d$ for $F \in \mathcal{F}$

Energy is linear in parameter vector $\theta = (\theta_F)_{F \in \mathcal{F}}$

$$E(x, y; \theta) = \langle \theta, \phi(x, y) \rangle = \sum_{F \in \mathcal{F}} \langle \theta_F, \phi_F(y_F, x) \rangle$$

Overall model is log-linear: $p(y|x; \theta) \propto e^{-\langle \theta, \phi(x,y) \rangle}$

CRF training requires minimizing negative conditional log-likelihood:

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

- *convex* optimization problem → (stochastic) gradient descent finds optimum
- training needs repeated runs of probabilistic inference

Latent variables are possible, but make training harder (in particular non-convex).