Learning with Structured Inputs and Outputs

Christoph Lampert
Institute of Science and Technology Austria
Vienna (Klosterneuburg), Austria

INRIA Summer School 2010

Learning with Structured Inputs and Outputs

Overview...

- 15:30–16:30 Intro + Conditional Random Fields
- 16:30–16:40 — mini-break —
- 16:40–17:30 Structured Support Vector Machines

Slides and References: `http://www.christoph-lampert.org`

"Normal" Machine Learning:

$$f : \mathcal{X} \to \mathbb{R}.$$

- inputs $\mathcal{X}$ can be any kind of objects
- output y is a real number
  - classification, regression, density estimation, ...

Structured Output Learning:

$$f : \mathcal{X} \to \mathcal{Y}.$$

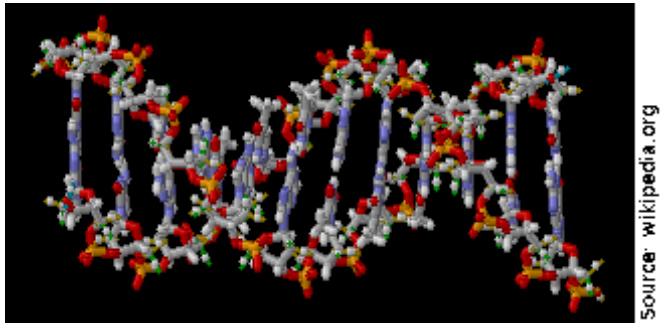
- inputs $\mathcal{X}$ can be any kind of objects
- outputs $y \in \mathcal{Y}$ are complex (structured) objects
  - images, text, audio, folds of a protein

What is structured data?

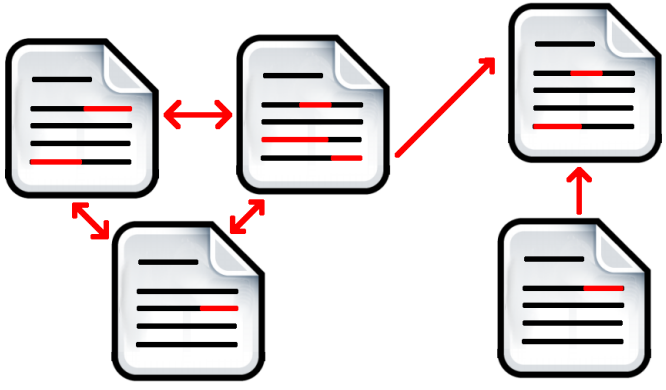
Ad hoc definition: data that consists of several parts, and not only the parts themselves contain information, but also the way in which the parts belong together.

Jemand musste Josef K. verleumdet haben, denn ohne dass er etwas Böses getan hätte, wurde er eines Morgens verhaftet. »Wie ein Hund! « sagte er, es war, als sollte die Scham ihn überleben. Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu einem ungeheueren Ungeziefer verwandelt. Und es war ihnen wie eine Bestätigung ihrer neuen Träume und guten Absichten, als am Ziele ihrer Fahrt die Tochter als erste sich erhob und ihren jungen Körper dehnte. »Es ist ein eigentümlicher Apparat«, sagte der Offizier zu dem Forschungsreisenden und überblickte mit einem gewissermaßen

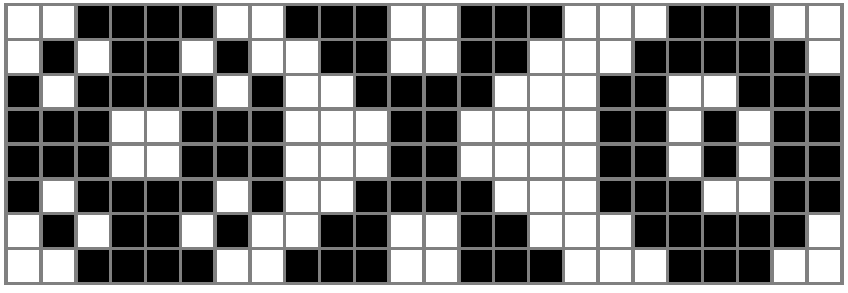
Text



Molecules / Chemical Structures



Documents/HyperText



Images

What is structured output prediction?

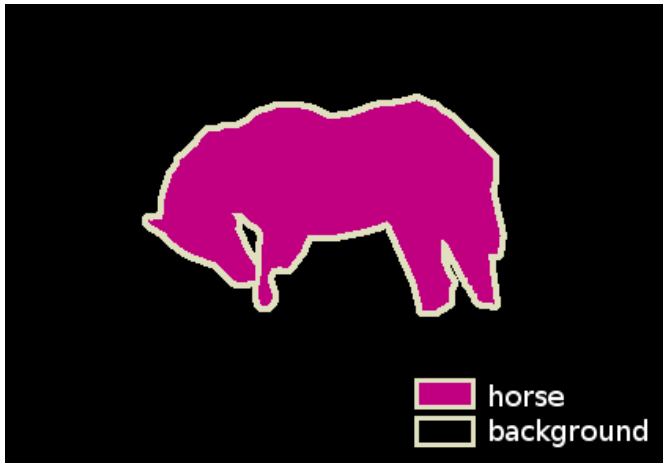
Ad hoc definition: predicting *structured* outputs from input data
(in contrast to predicting just a single number, like in classification or regression)

- Natural Language Processing:
  - Automatic Translation (output: sentences)
  - Sentence Parsing (output: parse trees)
- Bioinformatics:
  - Secondary Structure Prediction (output: bipartite graphs)
  - Enzyme Function Prediction (output: path in a tree)
- Speech Processing:
  - Automatic Transcription (output: sentences)
  - Text-to-Speech (output: audio signal)
- Robotics:
  - Planning (output: sequence of actions)

This lecture: Structured Prediction in Computer Vision

Computer Vision Example: Semantic Image Segmentation

$\mapsto$



input: images output: segmentation masks

- input space $\mathcal{X} = \{\textit{images}\} \mathrel{\hat{=}} [0, 255]^{3 \cdot M \cdot N}$
- output space $\mathcal{Y} = \{\textit{segmentation masks}\} \mathrel{\hat{=}} \{0, 1\}^{M \cdot N}$
- (structured output) prediction function: $f : \mathcal{X} \to \mathcal{Y}$

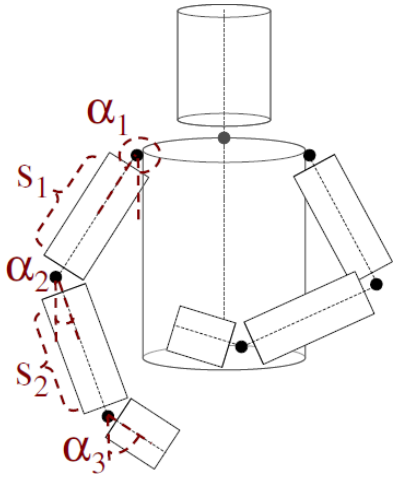
$$f(x) := \operatorname*{argmin}_{y \in \mathcal{Y}} E(x, y)$$

- energy function $E(x, y) = \sum_i w_i^\top \varphi_u(x_i, y_i) + \sum_{i,j} w_{ij}^\top \varphi_p(y_i, y_j)$

Images: [M. Everingham et al. "The PASCAL Visual Object Classes (VOC) challenge", IJCV 2010]

Computer Vision Example: Human Pose Estimation

input: image



body model

output: model fit

- input space $\mathcal{X} = \{\textit{images}\}$
- output space $\mathcal{Y} = \{\textit{positions/angle of } K \textit{ body parts}\} \hat{=} \mathbb{R}^{3K}$.
- prediction function: $f : \mathcal{X} \to \mathcal{Y}$

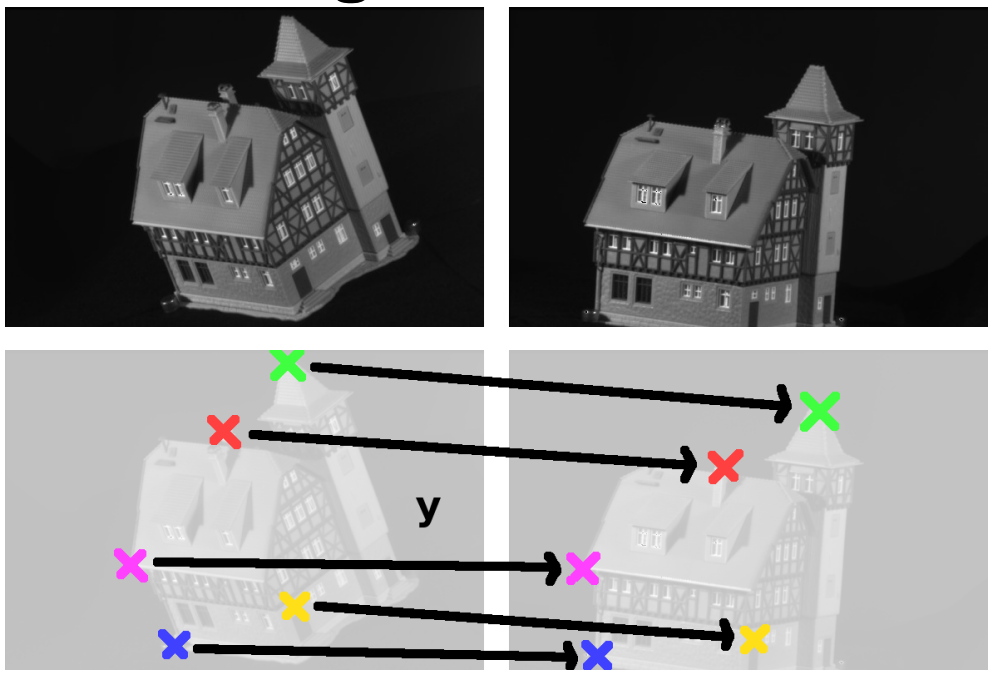
$$f(x) := \operatorname*{argmin}_{y \in \mathcal{Y}} E(x, y)$$

- energy $E(x, y) = \sum_i w_i^\top \varphi_{\textit{fit}}(x_i, y_i) + \sum_{i,j} w_{ij}^\top \varphi_{\textit{pose}}(y_i, y_j)$

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

Computer Vision Example: Point Matching

input: image pairs



output: mapping $y : x_i \leftrightarrow y(x_i)$

- prediction function: $f : \mathcal{X} \to \mathcal{Y}$

$$f(x) := \operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y)$$

- scoring function $F(x, y) =$
  $\sum_i w_i^\top \varphi_{\textit{sim}}(x_i, y(x_i)) + \sum_{i,j} w_{ij}^\top \varphi_{\textit{geom}}(x_i, x_j, y(x_i), y(x_j))$

[J. McAuley et al.: "Robust Near-Isometric Matching via Structured Learning of Graphical Models", NIPS, 2008]

Computer Vision Example: Object Localization

input:
image

output:
object position
$(left, top$
$right, bottom)$

- input space $\mathcal{X} = \{images\}$
- output space $\mathcal{Y} = \mathbb{R}^4$ bounding box coordinates
- scoring function $F(x, y) = w^\top \varphi(x, y)$ where $\varphi(x, y) = h(x|_y)$ is a feature vector for an image region, e.g. bag-of-visual-words.
- prediction function: $f : \mathcal{X} \to \mathcal{Y}$

$$f(x) := \operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y)$$

[M. Blaschko, C. Lampert: "Learning to Localize Objects with Structured Output Regression", ECCV, 2008]

Computer Vision Examples: Summary

Image Segmentation

$$y = \underset{y \in \{0,1\}^N}{\operatorname{argmin}} E(x, y) \qquad E(x, y) = \sum_i w_i^\top \varphi(x_i, y_i) + \sum_{i,j} w_{ij}^\top \varphi(y_i, y_j)$$

Pose Estimation

$$y = \underset{y \in \mathbb{R}^{3K}}{\operatorname{argmin}} E(x, y) \qquad E(x, y) = \sum_i w_i^\top \varphi(x_i, y_i) + \sum_{i,j} w_{ij}^\top \varphi(y_i, y_j)$$

Point Matching

$$y = \underset{y \in \Pi_n}{\operatorname{argmax}} F(x, y) \qquad F(x, y) = \sum_i w_i^\top \varphi(x_i, y_i) + \sum_{i,j} w_{ij}^\top \varphi(y_i, y_j)$$

Object Localization

$$y = \underset{y \in \mathbb{R}^4}{\operatorname{argmax}} F(x, y) \qquad F(x, y) = w^\top \varphi(x, y)$$

Unified Setup

Predict structured output by maximization

$$y = \operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y)$$

of a compatiblity function

$$F(x, y) = \langle w, \varphi(x, y) \rangle$$

that is linear in a parameter vector w.

A generic structured prediction problem

- $\mathcal{X}$: arbitrary input domain
- $\mathcal{Y}$: structured output domain, decompose $y = (y_1, \ldots, y_k)$
- Prediction function $f : \mathcal{X} \to \mathcal{Y}$ by

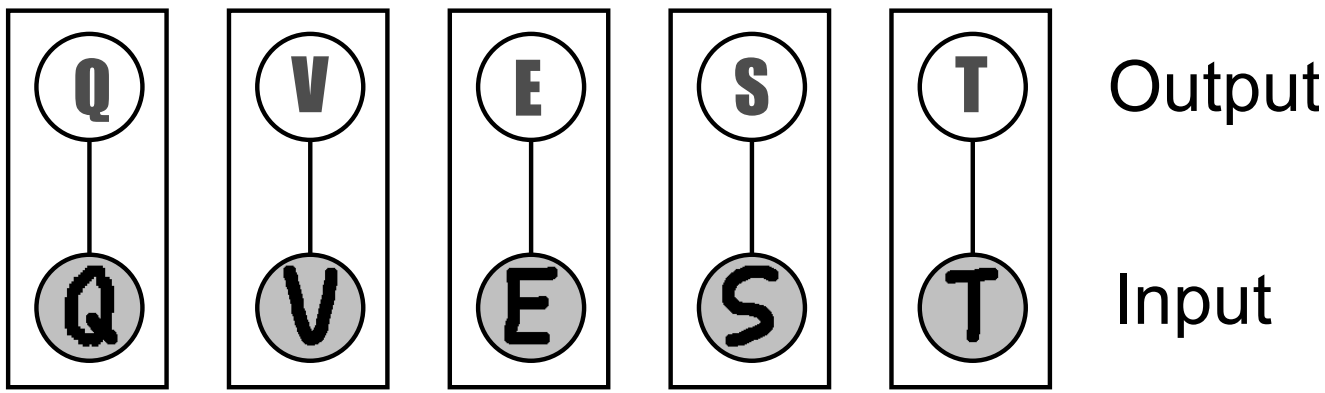
$$f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y)$$

- Compatiblity function (or negative of "energy")

$$\begin{aligned} F(x, y) &= \langle w, \varphi(x, y) \rangle \\ &= \sum_{i=1}^{k} w_i^\top \varphi_i(y_i, x) \qquad \textbf{unary terms} \\ &\quad + \sum_{i,j=1}^{k} w_{ij}^\top \varphi_{ij}(y_i, y_j, x) \qquad \textbf{binary terms} \\ &\quad + \quad \ldots \quad \textbf{higher order terms} \text{ (sometimes)} \end{aligned}$$

Example: Sequence Prediction – Handwriting Recognition

- $\mathcal{X} =$ *5-letter word images* , $x = (x_1, \ldots, x_5)$, $x_j \in \{0,1\}^{300\times 80}$
- $\mathcal{Y} =$ *ASCII translation* , $y = (y_1, \ldots, y_5)$, $y_j \in \{A, \ldots, Z\}$.
- feature functions has only unary terms
  $\varphi(x, y) = \Big(\varphi_1(x, y_1), \ldots, \varphi_5(x, y_5) \Big)$.
- $F(x, y) = \langle w_1, \varphi_1(x, y_1) \rangle + \cdots + \langle w, \varphi_5(x, y_5) \rangle$



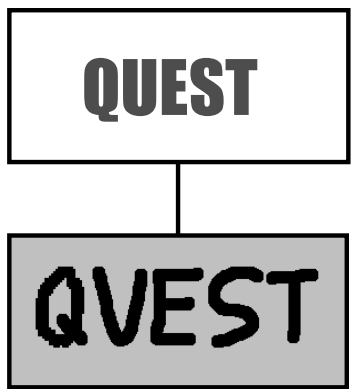
Advantage: computing $y^* = \operatorname{argmax}_y F(x, y)$ is easy.
We can find each y_i^* independently, check $5 \cdot 26 = 130$ values.

Problem: only local information, we can't correct errors.

Example: Sequence Prediction – Handwriting Recognition

- $\mathcal{X} =$ *5-letter word images* , $x = (x_1, \ldots, x_5)$, $x_j \in \{0,1\}^{300\times 80}$
- $\mathcal{Y} =$ *ASCII translation* , $y = (y_1, \ldots, y_5)$, $y_j \in \{A, \ldots, Z\}$.
- one *global* feature function

$$\varphi(x,y) = \begin{cases} (0,\ldots,0,\overbrace{\Phi(x)}^{y\text{th pos.}},0,\ldots,0) & \text{if } y \in \mathcal{D} \text{ dictionary,} \\ (0,\ldots,0,\quad 0\quad ,0,\ldots,0) & \text{otherwise.} \end{cases}$$



Output

Input

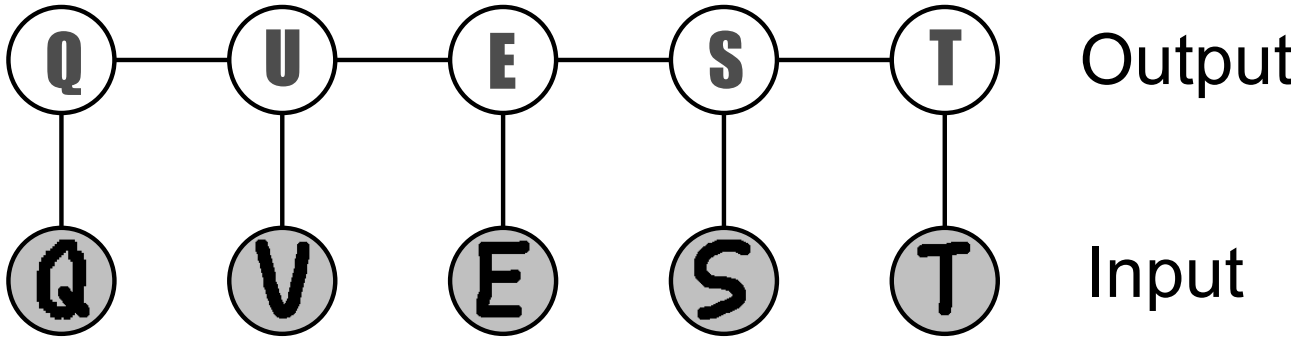
Advantage: access to global information, e.g. from dictionary $\mathcal{D}$.

Problem: $\operatorname{argmax}_y \langle w, \varphi(x,y)\rangle$ has to check $26^5 = 11881376$ values.
We need separate training data for each word.

Example: Sequence Prediction – Handwriting Recognition

- $\mathcal{X} =$ *5-letter word images* , $x = (x_1, \ldots, x_5)$, $x_j \in \{0,1\}^{300\times 80}$
- $\mathcal{Y} =$ *ASCII translation* , $y = (y_1, \ldots, y_5)$, $y_j \in \{A, \ldots, Z\}$.
- feature function with unary and pairwise terms

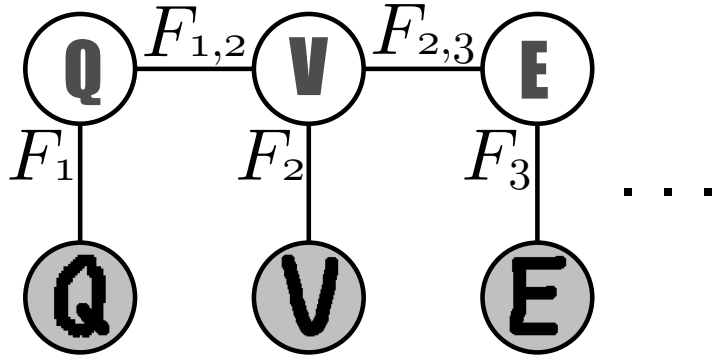
$$\varphi(x,y) = \Big(\varphi_1(y_1, x),\ \varphi_2(y_2, x), \ldots, \varphi_5(y_5, x),$$
$$\varphi_{1,2}(y_1, y_2),\ \ldots,\ \varphi_{4,5}(y_4, y_5) \Big)$$



Compromise: computing y^* is still efficient (Viterbi best path)
Compromise: neighbor information allows correction of local errors.

Example: Sequence Prediction – Handwriting Recognition

- $\varphi(x, y) = (\varphi_1(y_1, x), \ldots, \varphi_5(y_5, x), \varphi_{1,2}(y_1, y_2), \ldots, \varphi_{4,5}(y_4, y_5))$.
- $w = (w_1, , \ldots, w_5, w_{1,2}, \ldots, w_{4,5})$
- $F(x, y) = \langle w, \varphi(x, y) \rangle =$
  $F_1(x_1, y_1) + \cdots + F_5(x_5, y_5) + F_{1,2}(y_1, y_2) + \ldots, F_{4,5}(y_4, y_5)$.



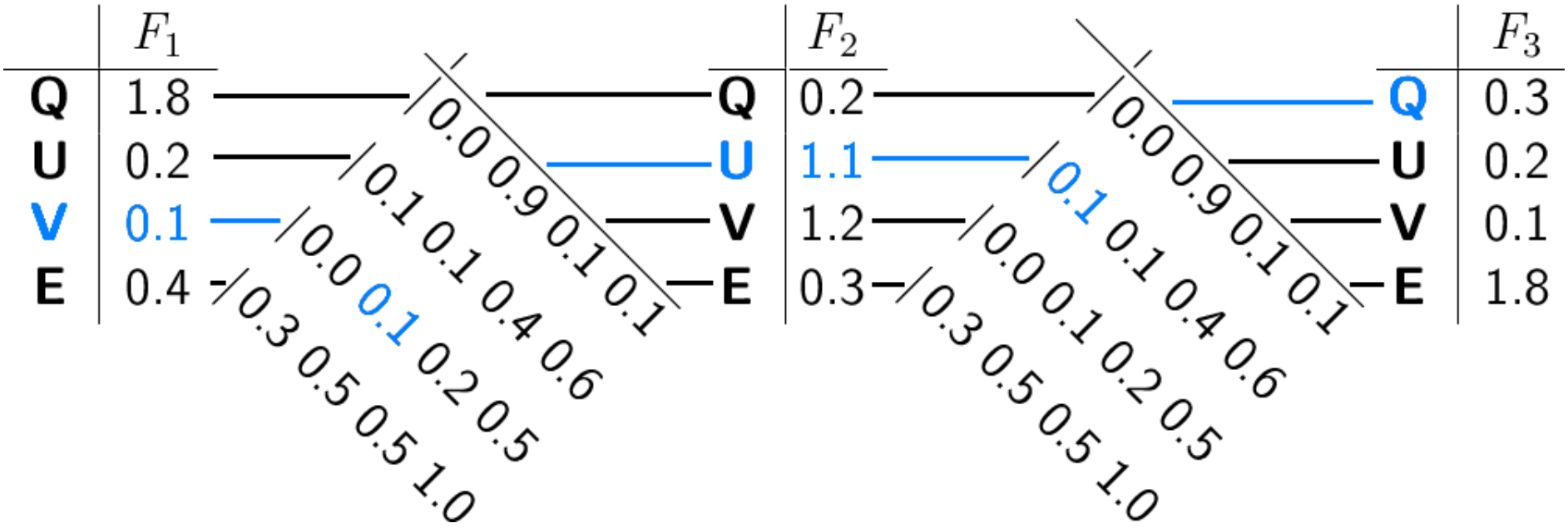
$F_{i,i+1}$	Q	U	V	E
Q	0.0	0.9	0.1	0.1
U	0.1	0.1	0.4	0.6
V	0.0	0.1	0.2	0.5
E	0.3	0.5	0.5	1.0

	F_1
Q	1.8
U	0.2
V	0.1
E	0.4

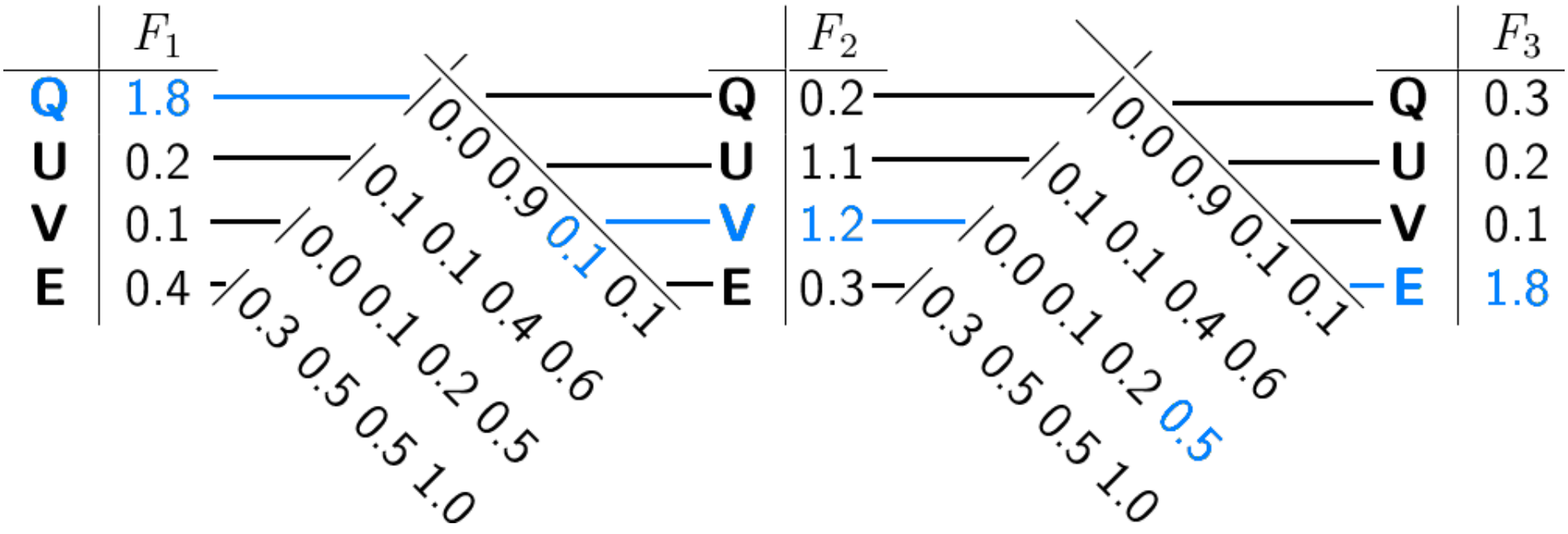
	F_2
Q	0.2
U	1.1
V	1.2
E	0.3

	F_3
Q	0.3
U	0.2
V	0.1
E	1.8

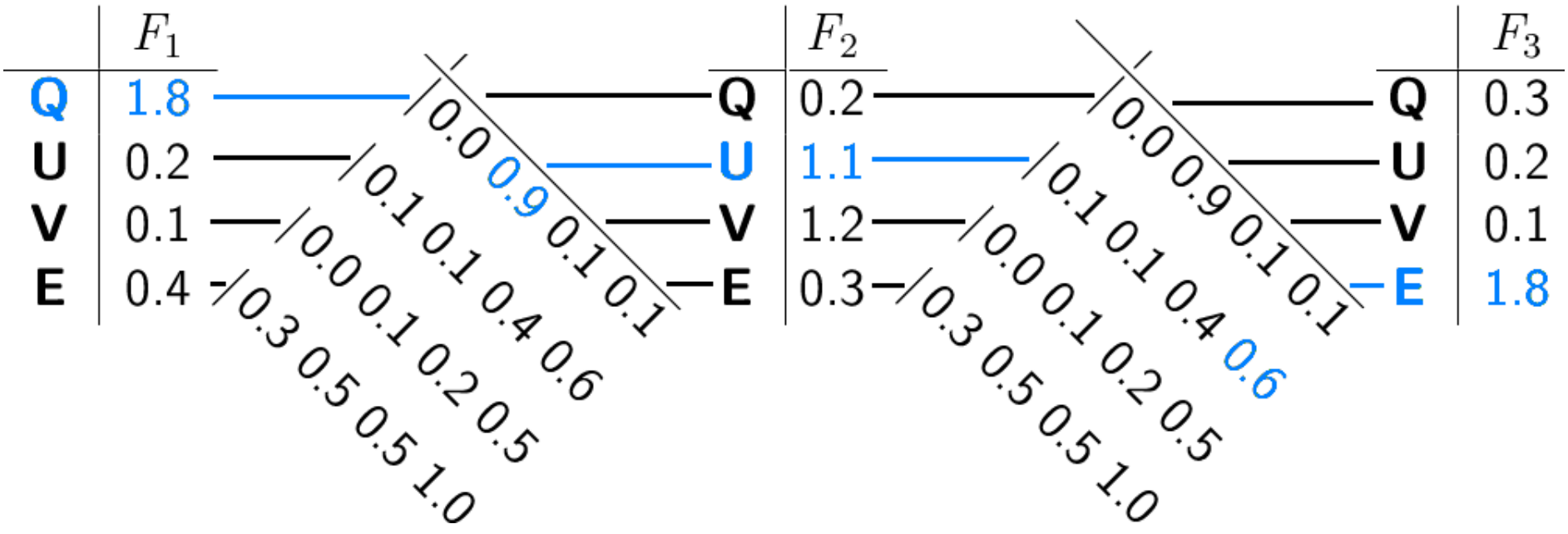
Every $y \in \mathcal{Y}$ corresponds to a path. $\operatorname{argmax}_{y \in \mathcal{Y}}$ is *best path*.



$$F(\texttt{VUQ}, x) = F_1(\texttt{V}, x_1) + F_{12}(\texttt{V}, \texttt{U}) + F_2(\texttt{U}, x_2) + F_{23}(\texttt{U}, \texttt{Q}) + F_3(\texttt{Q}, x_3)$$
$$= 0.1 + 0.1 + 1.1 + 0.1 + 0.3 = \mathbf{1.7}$$



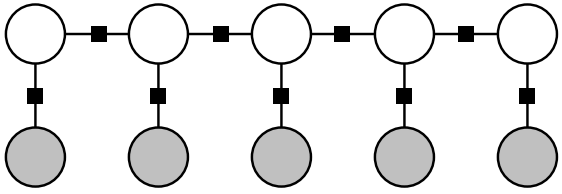
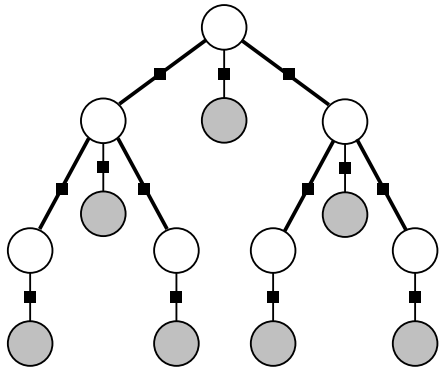
Maximal per-letter scores. Total: $1.8 + 0.1 + 1.2 + 0.5 + 1.8 = \mathbf{5.4}$



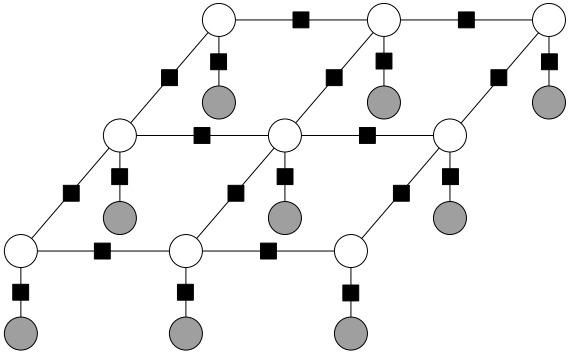
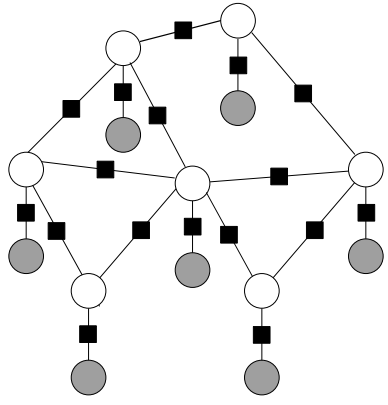
Best (=Viterbi) path. Total: $1.8 + 0.9 + 1.1 + 0.6 + 1.8 = \mathbf{6.2}$

Many popular models have **unary** and **pairwise** terms.

Yesterday's lecture: how to evaluate $\operatorname{argmax}_y F(x, y)$.

chain   tree

- loop-free graphs: Viterbi decoding / dynamic programming

grid   arbitrary graph

- loopy graphs: approximate inference (e.g. loopy BP)

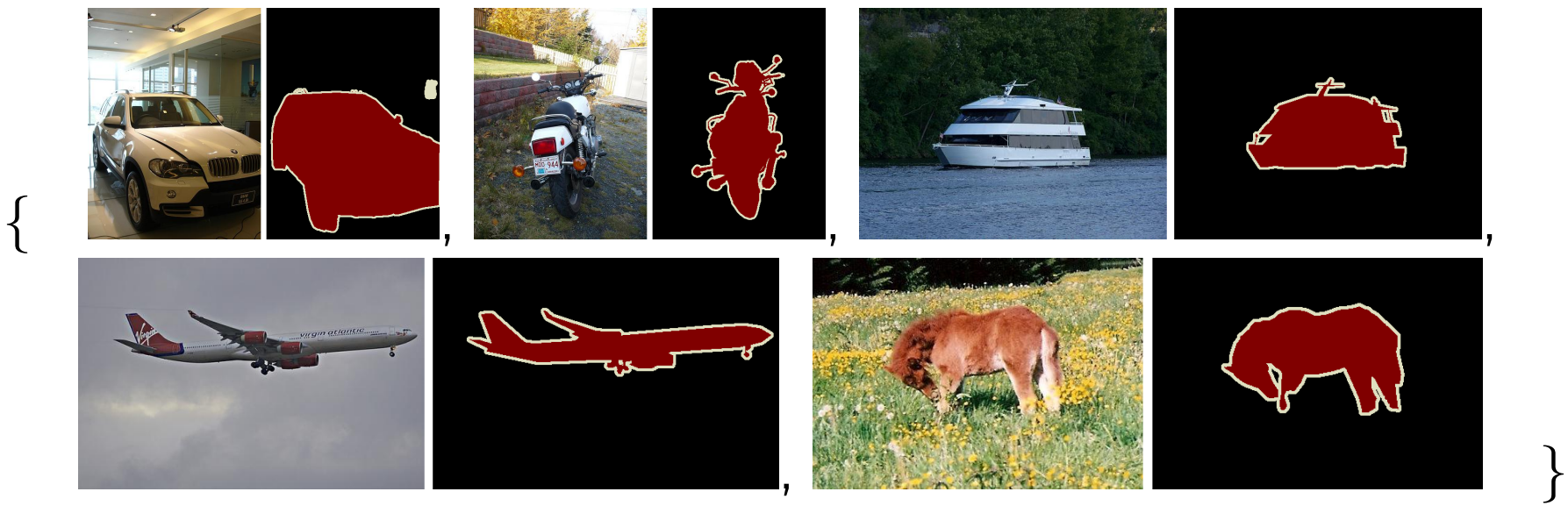
Today: how to learn a good function $F(x, y)$ from training data.

Parameter Learning in Structured Models

- Given: parametric model (family): $F(x, y) = \langle w, \varphi(x, y) \rangle$
- Given: prediction method: $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} F(x, y)$
- Not given: parameter vector w (high-dimensional)

Supervised Training:

- Given: example pairs $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$. typical inputs with "the right" outputs for them.
- Task: determine "good" w



- Task: determine "good" w

Probabilistic Training of Structured Models
(Conditional Random Fields)

Probabilistic Models for Structured Prediction

Establish a one-to-one correspondence between

- compatibility function $F(x, y)$ and
- conditional probability distribution $p(y|x)$,

such that

$$\operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y) = \operatorname*{argmax}_{y \in \mathcal{Y}} p(y|x)$$

= maximize the posterior probability = Bayes optimal decision

Gibbs Distribution

$$p(y|x) = \frac{1}{Z(x)} e^{F(x,y)} \qquad \text{with } Z(x) = \sum_{y \in \mathcal{Y}} e^{F(x,y)}$$

$$\operatorname*{argmax}_{y \in \mathcal{Y}} p(y|x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \frac{e^{F(x,y)}}{Z(x)} = \operatorname*{argmax}_{y \in \mathcal{Y}} e^{F(x,y)} = \operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y)$$

For learning, we make the dependence on w explicit:

$$F(x, y, w) = \langle w, \varphi(x, y) \rangle$$

Parametrized Gibbs Distribution – Loglinear Model

$$p(y|x, w) = \frac{1}{Z(x, w)} e^{F(x,y,w)} \qquad \text{with } Z(x, w) = \sum_{y \in \mathcal{Y}} e^{F(x,y,w)}$$

Probabilistic Models for Structured Prediction

Supervised training:

- Given: i.i.d. example pairs $\mathcal{D} = \{(x^1, y^1), \ldots, (x^n, y^n)\}$.
- Task: determine w.
- Idea: Maximize the posterior probability $p(w|\mathcal{D})$

Use Bayes' rule to infer posterior distribution for w:

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

assuming x does not depend on w: $p(x|w) \equiv p(x)$

$$= p(w) \prod_{i=1}^{n} \frac{p(y^i|x^i, w)}{p(y^i|x^i)}$$

Posterior probability:

$$p(w|\mathcal{D}) = p(w) \prod_i \frac{p(y^i|x^i, w)}{p(y^i|x^i)}$$

Prior Data Term (loglinear)

Typical Prior choice:

- **Gaussian** $p(w) := const. \cdot e^{-\frac{1}{2\sigma^2}\|w\|^2}$
  - Gaussianity is often a good guess if you know nothing else
  - easy to handle, analytically und numerically

Less common:

- **Laplacian** $p(w) := const. \cdot e^{-\frac{1}{\sigma}\|w\|}$
  - if we expect *sparsity* in w
  - but: properties of the optimization problem change

Learning w from $\mathcal{D}$ $\equiv$ Maximizing posterior probability for w

$$w^* = \operatorname*{argmax}_{w \in \mathbb{R}^d} \; p(w|\mathcal{D}) = \operatorname*{argmin}_{w \in \mathbb{R}^d} \Big[-\log p(w|\mathcal{D}) \Big]$$

$$\begin{aligned}
-\log p(w|\mathcal{D}) &= -\log \Big[p(w) \prod_i \frac{p(y^i|x^i, w)}{p(y^i|x^i)} \Big] \\
&= -\log p(w) - \sum_{i=1}^n \log \underbrace{p(y^i|x^i, w)}_{= \frac{1}{Z} e^{F(x^i, y^i, w)}} + \underbrace{\sum_{i=1}^n \log p(y^i|x^i)}_{\text{indep. of } w} \\
&= \frac{\|w\|^2}{2\sigma^2} - \sum_{i=1}^n \Big[F(x^i, y^i, w) - \log Z(x^i, w) \Big] + \text{const.} \\
&= \underbrace{\frac{\|w\|^2}{2\sigma^2} - \sum_{i=1}^n \Big[\langle w, \varphi(x^i, y^i) \rangle - \log \sum_{y \in \mathcal{Y}} e^{\langle w, \varphi(x^i, y) \rangle} \Big]}_{=: \mathcal{L}(w)} + \text{c.}
\end{aligned}$$

Commercial Break: Objective Functions

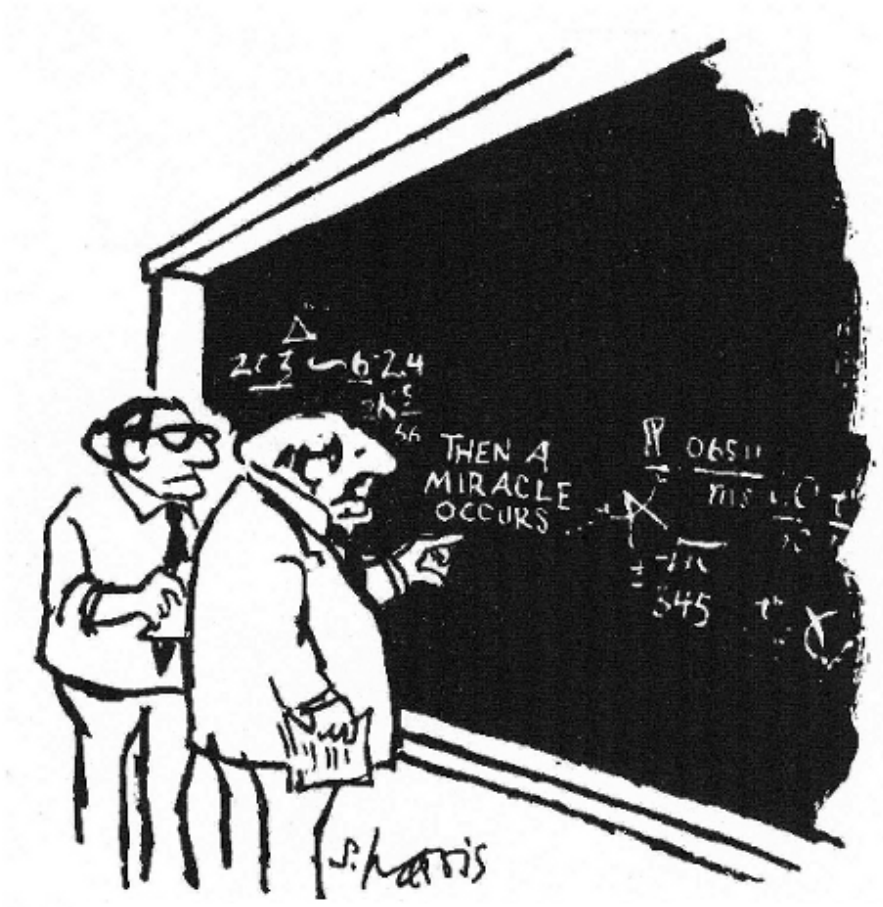
Why is everything an optimization problem?

Why all the formulas?

Why not simply teach algorithms?

Because...

- we want to separate between:
  - what is our ideal goal?
    = objective function
  - (how) do we achieve it?
    = optimization method
- defining a goal helps in **understanding** the problem
- mathematical formulation allows **re-using existing algorithms** (developed for different tasks)



"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

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

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

Probabilistic parameter estimation or *training* means solving

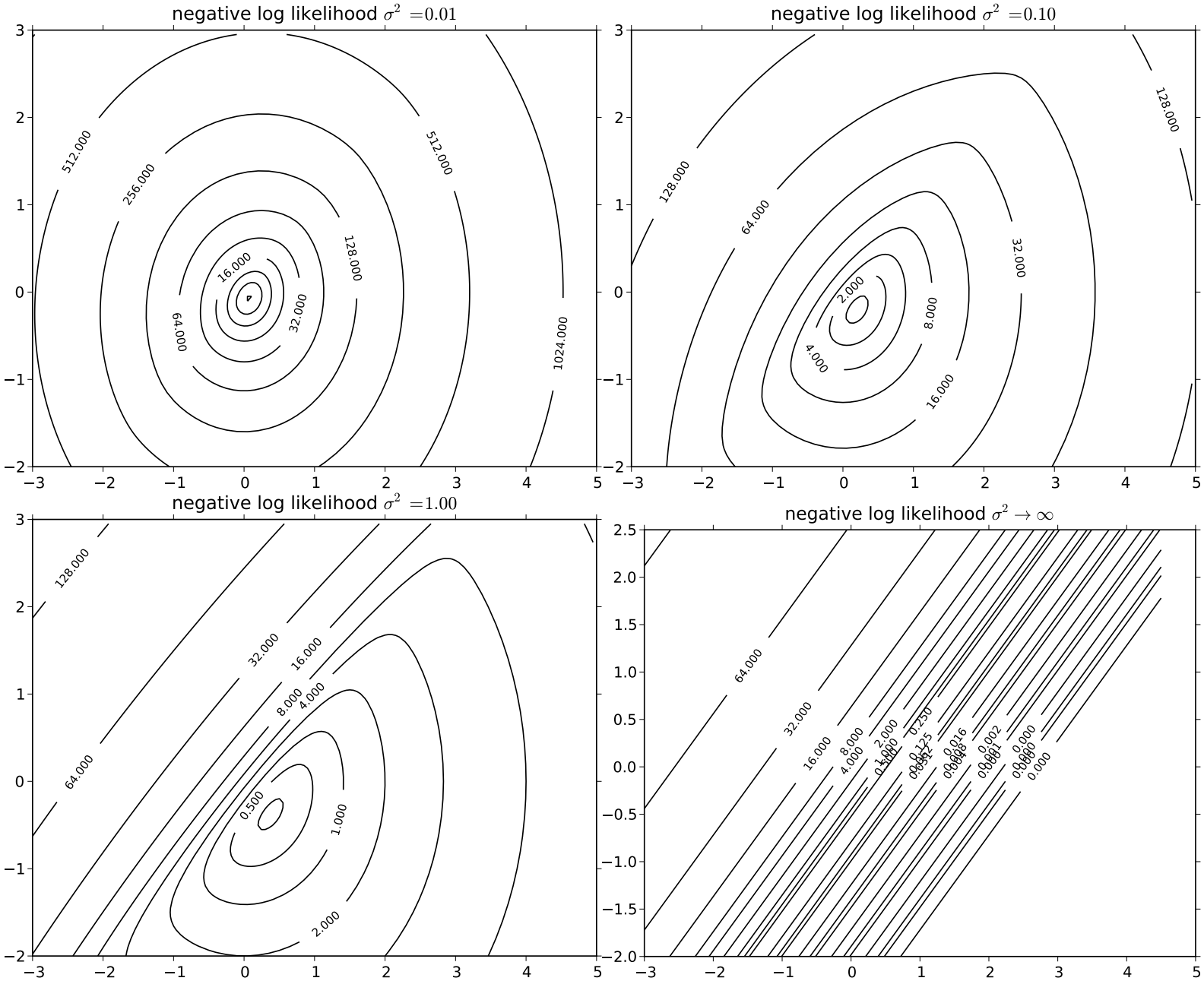
$$w^* = \operatorname*{argmin}_{w\in\mathbb{R}^d} \mathcal{L}(w).$$

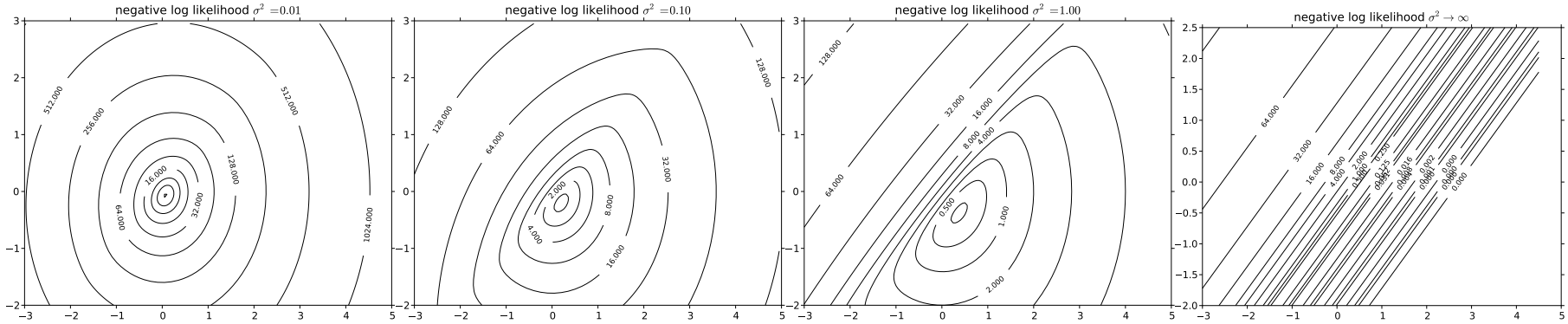
Same optimization problem as for multi-class **logistic regression**.

$$\mathcal{L}_{\textit{logreg}}(w) = \frac{1}{2\sigma^2}\|w\|^2 - \sum_{i=1}^{n}\left[\langle w_{y^i}, \varphi(x^i)\rangle - \log \sum_{y\in\mathcal{Y}} e^{\langle w_y, \varphi(x^i)\rangle}\right]$$

with $w = (w_1, \ldots, w_K)$ for K classes.

Negative Conditional Log-Likelihood for $w \in \mathbb{R}^2$:





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

- $w \in \mathbb{R}^d$ continuous (not discrete) ☺
- $w \in \mathbb{R}^d$ high-dimensional (often $d \gg 1000$) ☹
- optimization has no side-constraints ☺
- $\mathcal{L}(w)$ differentiable ☺ $\rightarrow$ can apply gradient descent
- $\mathcal{L}(w)$ **convex!** ☺ $\rightarrow$ g.d. will find **global optimum**

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

- **require:** tolerance $\epsilon > 0$
- $w_{cur} \leftarrow 0$
- **repeat**
  - $v \leftarrow -\nabla_w \mathcal{L}(w_{cur})$ descent direction
  - $\eta \leftarrow \operatorname{argmin}_{\eta \in \mathbb{R}} \mathcal{L}(w_{cur} + \eta v)$ stepsize
  - $w_{cur} \leftarrow w_{cur} + \eta v$ update
- **until** $\|v\| < \epsilon$
- **return** w_{cur}

Alternatives:

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

Summary I: Probabilistic Training (Conditional Random Fields)

- Well-defined probabilistic setting.
- $p(y|x, w)$ log-linear in $w \in \mathbb{R}^d$.
- Training: maximize $p(w|\mathcal{D})$ for data set $\mathcal{D}$.
- Differentiable, *convex* optimization,
- Same structure as *logistic regression*.
  $\Rightarrow$ gradient descent will find global optimum.

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

For conditional random fields: the worst lies still ahead of us!

Problem: Computing $\nabla_w \mathcal{L}(w_{cur})$ and evaluating $\mathcal{L}(w_{cur} + \eta v)$:

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

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

- $\mathcal{Y}$ typically is very (exponentially) large:
  - N parts, each with label from a set Y: $|\mathcal{Y}| = |Y|^N$,
    binary image segmentation: $|\mathcal{Y}| = 2^{640 \times 480} \approx 10^{92475}$,
  - ranking N images: $|\mathcal{Y}| = N!$, e.g. $N = 1000$: $|\mathcal{Y}| \approx 10^{2568}$.

Without **structure** in $\mathcal{Y}$, we're lost.

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

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

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

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 nodes $\approx$ 10s to 1,000,000s
- K: number of possible labels of each output nodes $\approx$ 2 to 100s

How Does Structure in $\mathcal{Y}$ Help?

Computing

$$Z(x, w) = \sum_{y \in \mathcal{Y}} e^{\langle w, \varphi(x,y) \rangle}$$

with φ and w decomposing over the factors

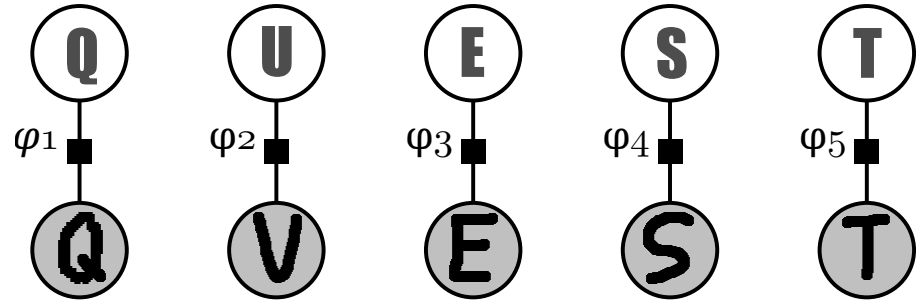
$$\varphi(x, y) = \Big(\varphi_F(x_F, y_F)\Big)_{F \in \mathcal{F}} \qquad \text{and} \qquad w = \Big(w_F\Big)_{F \in \mathcal{F}}$$

$$\begin{aligned} Z(x, w) &= \sum_{y \in \mathcal{Y}} e^{\sum_{F \in \mathcal{F}} \langle w_F, \varphi_F(x_F, y_F) \rangle} \\ &= \sum_{y \in \mathcal{Y}} \prod_{F \in \mathcal{F}} \underbrace{e^{\langle w_F, \varphi_F(x_F, y_F) \rangle}}_{=: \Psi_F(y_F)} \end{aligned}$$

A lot of research goes into analyzing this expression.

Case I: only unary factors

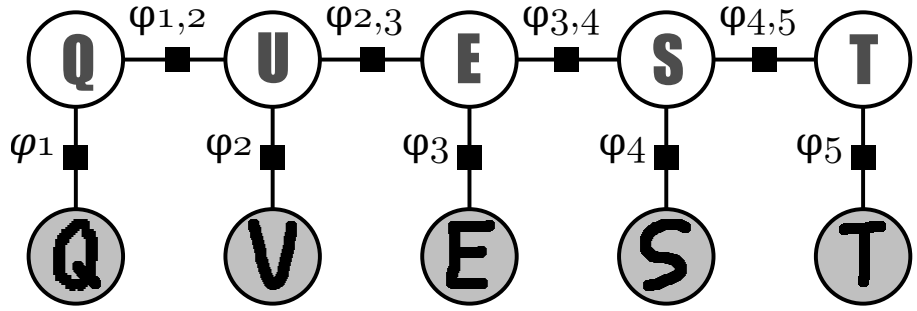
$\mathcal{F} = \Big\{\{x_1, y_1\}, \ldots, \{x_M, y_M\}\Big\}.$



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

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

Case II: chain/tree with unary and pairwise factors



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

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

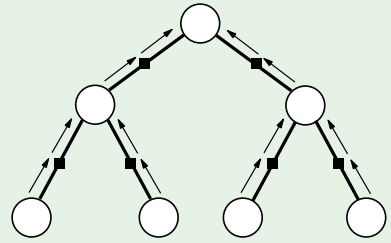
$$= \sum_{y_1 \in Y} \Psi_1 \sum_{y_2 \in Y} \Psi_{1,2}\Psi_2 \cdots \sum_{y_{M-1} \in Y} \Psi_{M-2,M-1}\Psi_{M-1} \sum_{y_M \in Y} \Psi_{M-1,M}\Psi_M$$

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

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

Message Passing in Trees

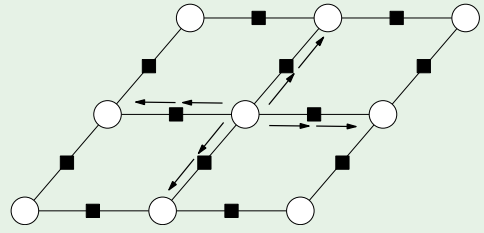
Rephrase matrix multiplication as *sending messages* from node to node:
Belief Propagation (BP)



Can also computate expectations: $\mathbb{E}_{y \sim p(y|x_i,w)} \varphi(x_i, y)$.

Message Passing in Loopy Graphs

For loopy graph, send the same messages and iterate: **Loopy Belief Propagation (LBP)**



Results only in *approximation* to $Z(x, w)$, $\mathbb{E}_{y \sim p(y|x_i,w)} \varphi(x_i, y)$.
No converge guarantee, but often used in practice.

More:

- Carsten's lecture yesterday (Wednesday)
- Pawan Kumar's thesis *"Combinatorial and Convex Optimization for Probabilistic Models in Computer Vision"* `http://ai.stanford.edu/~pawan/kumar-thesis.pdf`

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

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

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

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

- n: number of samples $\approx$ 1,000s 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 our training set $\mathcal{D}$ is too large (e.g. millions of examples)?

Switch to Simpler Classifier

- Train independent per-node classifiers

We lose all label dependences ☹. Still slow ☹.

Subsampling

- Create random subset $\mathcal{D}' \subset \mathcal{D}$
- Perform gradient descent to maximize $p(w|\mathcal{D}')$

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

Parallelize

- Train several smaller models on different computers.
- Merge the models.

Follows "multi-core" trend ☺. Unclear how to merge models ☹ ☺?
Doesn't reduce computation, just distributes it ☹.

What, if our 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 $\mathcal{D}' \subset \mathcal{D}$, $\quad \leftarrow$ **often just 1–3 elements!**
  - Follow approximate gradient

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

- *Line search* no longer possible. Extra parameter: stepsize η
- SGD converges to $\operatorname{argmin}_w p(w|\mathcal{D})$! (with 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`

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

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

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

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

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

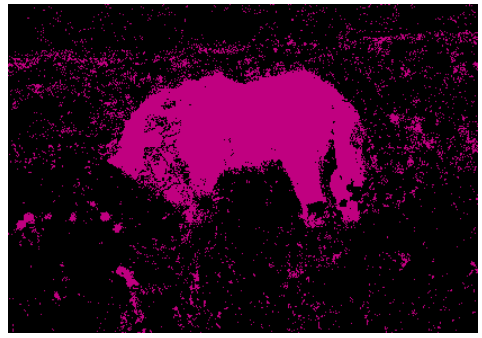
Typical feature functions in **image segmentation:**

- $\varphi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: local image features, e.g. bag-of-words
  $\rightarrow \quad \langle w_i, \varphi_i(y_i, x) \rangle$: local classifier (like logistic-regression)

- $\varphi_{i,j}(y_i, y_j) = [\![y_i = y_j]\!] \in \mathbb{R}^1$: test for same label
  $\rightarrow \quad \langle w_{ij}, \varphi_{ij}(y_i, y_j) \rangle$: penalizer for label changes (if $w_{ij} > 0$)

- combined: $\operatorname{argmax}_y p(y|x)$ is smoothed version of local cues

original



local classification

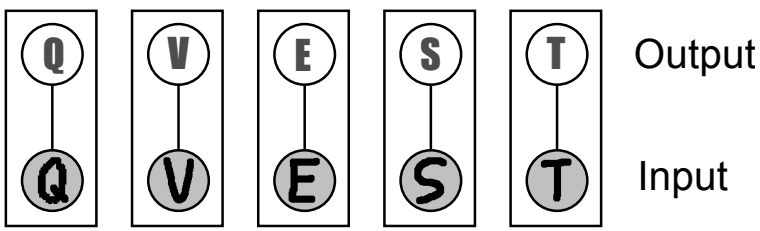
local + smoothness

Typical feature functions in **handwriting recognition:**

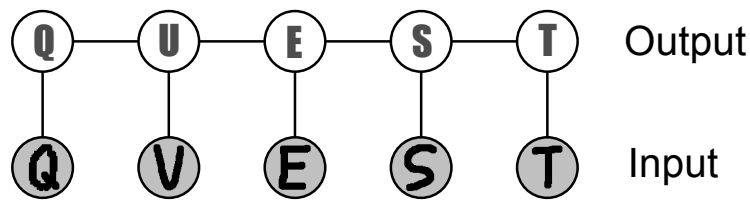
- $\varphi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: image representation (pixels, gradients)
  $\rightarrow$ $\langle w_i, \varphi_i(y_i, x) \rangle$: local classifier if x_i is letter y_i

- $\varphi_{i,j}(y_i, y_j) = e_{y_i} \otimes e_{y_j} \in \mathbb{R}^{26 \cdot 26}$: letter/letter indicator
  $\rightarrow$ $\langle w_{ij}, \varphi_{ij}(y_i, y_j) \rangle$: 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:**

- $\varphi_i(y_i, x) \in \mathbb{R}^{\approx 1000}$: local image representation, e.g. HoG
  $\rightarrow \quad \langle w_i, \varphi_i(y_i, x) \rangle$: local confidence map

- $\varphi_{i,j}(y_i, y_j) = \textit{good\_fit}(y_i, y_j) \in \mathbb{R}^1$: test for geometric fit
  $\rightarrow \quad \langle w_{ij}, \varphi_{ij}(y_i, y_j) \rangle$: penalizer for unrealistic poses

- together: $\operatorname{argmax}_y p(y|x)$ is sanitized version of local cues

original

local classification

local + geometry

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

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

- $\varphi_i(y_i, x)$: local representation, high-dimensional
  $\rightarrow \quad \langle w_i, \varphi_i(y_i, x) \rangle$: local classifier

- $\varphi_{i,j}(y_i, y_j)$: prior knowledge, low-dimensional
  $\rightarrow \quad \langle w_{ij}, \varphi_{ij}(y_i, y_j) \rangle$: penalize outliers

- learning adjusts parameters:
  - unary w_i: learn local classifiers and their importance
  - pairwise w_{ij}: learn importance of smoothing/penalization

- $\operatorname{argmax}_y p(y|x)$ is cleaned up version of local prediction

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 $\varphi_i(y_i, x) := f_i^y(x) \in \mathbb{R}^K$ (low-dimensional)
- keep $\varphi_{ij}(y_i, y_j)$ are before
- perform CRF learning with φ_i and φ_{ij}

Advantage:

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

Disadvantage:

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

Summary II — Numeric Training

(All?) Current CRF training methods rely on gradient descent.

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

The faster we can do it, the better (more realistic) models we can use.

A lot of research on speeding up CRF training:

problem	**"solution"**	**method(s)**
$\lvert\mathcal{Y}\rvert$ too large	exploit structure	(loopy) belief propagation
	smart sampling	*contrastive divergence*
n too large	mini-batches	stochastic gradient descent
d too large	discriminative φ_{unary}	two-stage training

Conditional Random Fields Summary

CRFs model *conditional probability distribution* $p(y|x) = \frac{1}{Z} e^{\langle w, \varphi(x,y) \rangle}$.

- $x \in \mathcal{X}$ **is arbitrary input**
  - we need a feature function $\varphi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^d$,
  - we don't need to know/estimate $p(x)$!
- $y \in \mathcal{Y}$ **is (almost arbitrary) output**
  - we need structure in $\mathcal{Y}$ to make training/inference feasible
  - e.g. graph labeling: local properties plus neighborhood relations
- $w \in \mathbb{R}^d$ **is parameter vector**
  - *CRF training* is solving $w = \operatorname{argmax}_w p(w|\mathcal{D})$ for training set $\mathcal{D}$.
- **after learning, we can predict:** $f(x) := \operatorname{argmax}_y p(y|x)$**.**

Many many many more details in Machine Learning literature, e.g.:

- [M. Wainwright, M. Jordan: "Graphical Models, Exponential Families, and Variational Inference", now publishers, 2008]. Free PDF at `http://www.nowpublishers.com`
- [D. Koller, N. Friedman: "Probabilistic Graphical Models", MIT Press, 2010] 60 EUR (i.e. less than 5 cents per page)

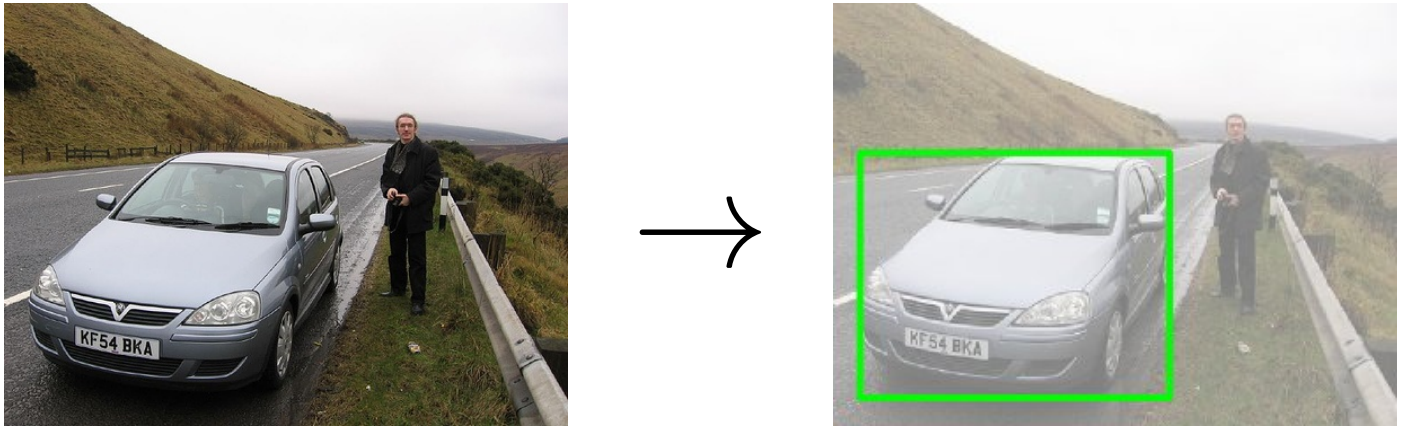
Maximum Margin Training of Structured Models

(Structure Output SVMs)

Example: Object Localization

Learn a **localization function**:

$$f : \mathcal{X} \to \mathcal{Y} \qquad \text{with } \mathcal{X} = \{\text{images}\},\ \mathcal{Y} = \{\text{bounding boxes}\}.$$



Compatibility function:

$$F : \mathcal{X} \times \mathcal{Y} \to \mathbb{R} \qquad \text{with} \quad F(x, y) = \langle w, \varphi(x, y) \rangle$$

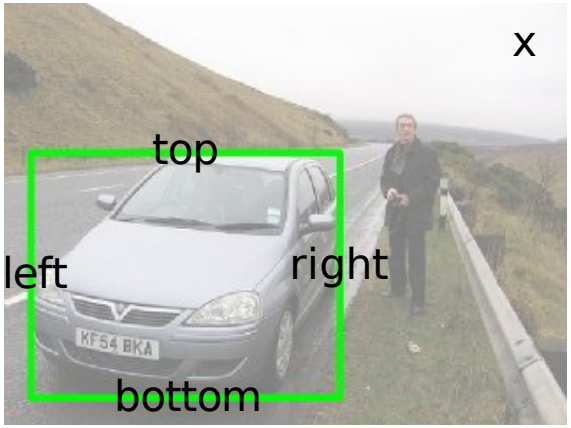
with $\varphi(x, y)$, e.g., the BoW histogram of the region y in x.

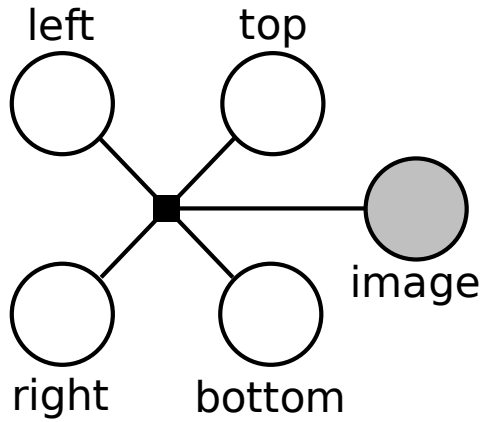
$$f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y)$$

Learn a **localization function**:

$$f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} F(x, y) \qquad \text{with } F(x, y) = \langle w, \varphi(x, y) \rangle.$$

$\varphi(x, y) =$ BoW histogram of region $y = [\textit{left,top,right,bottom}]$ in x.





$$p(y|x, w) = \frac{1}{Z} \exp\Big(\langle w, \varphi(x, y) \rangle\Big) \quad \text{with } Z = \sum_{y' \in \mathcal{Y}} \exp\Big(\langle w, \varphi(x, y') \rangle\Big)$$

We can't train probabilistically: φ doesn't decompose, Z too hard.
We **can do** MAP prediction: sliding window or branch-and-bound.

Can we learn structured prediction functions without "Z"?

Reminder: general form of regularized learning

Find w for decision function $f = \langle w, \varphi(x, y) \rangle$ by

$$\min_{w \in \mathbb{R}^d} \quad \lambda \|f\|^2 \quad + \quad \sum_{i=1}^{n} \ell(y^i, f(x^i))$$

Regularization + *Error on data*

Probabilistic Training: logistic loss

$$\ell(y^i, f(x^i)) := \log \sum_{y \in \mathcal{Y}} \exp\Big(\langle w, \varphi(x^i, y) \rangle - \langle w, \varphi(x^i, y^i) \rangle\Big)$$

Maximum-Margin Training: Hinge-loss

$$\ell(y^i, f(x^i)) := \max\Big\{0, \max_{y \in \mathcal{Y}} \ \Delta(y^i, y) + \langle w, \varphi(x^i, y) \rangle - \langle w, \varphi(x^i, y^i) \rangle\Big\}$$

$\Delta(y, y') \geq 0$ with $\Delta(y, y) = 0$ is *loss function* between outputs.

Structured Output Support Vector Machine

Conditional Random Fields:

$$\min_{w \in \mathbb{R}^d} \frac{\|w\|^2}{2\sigma^2} + \log \sum_{y \in \mathcal{Y}} \exp\Big(\langle w, \varphi(x^i, y)\rangle - \langle w, \varphi(x^i, y^i)\rangle\Big)$$

Unconstrained optimization, convex, differentiable objective.

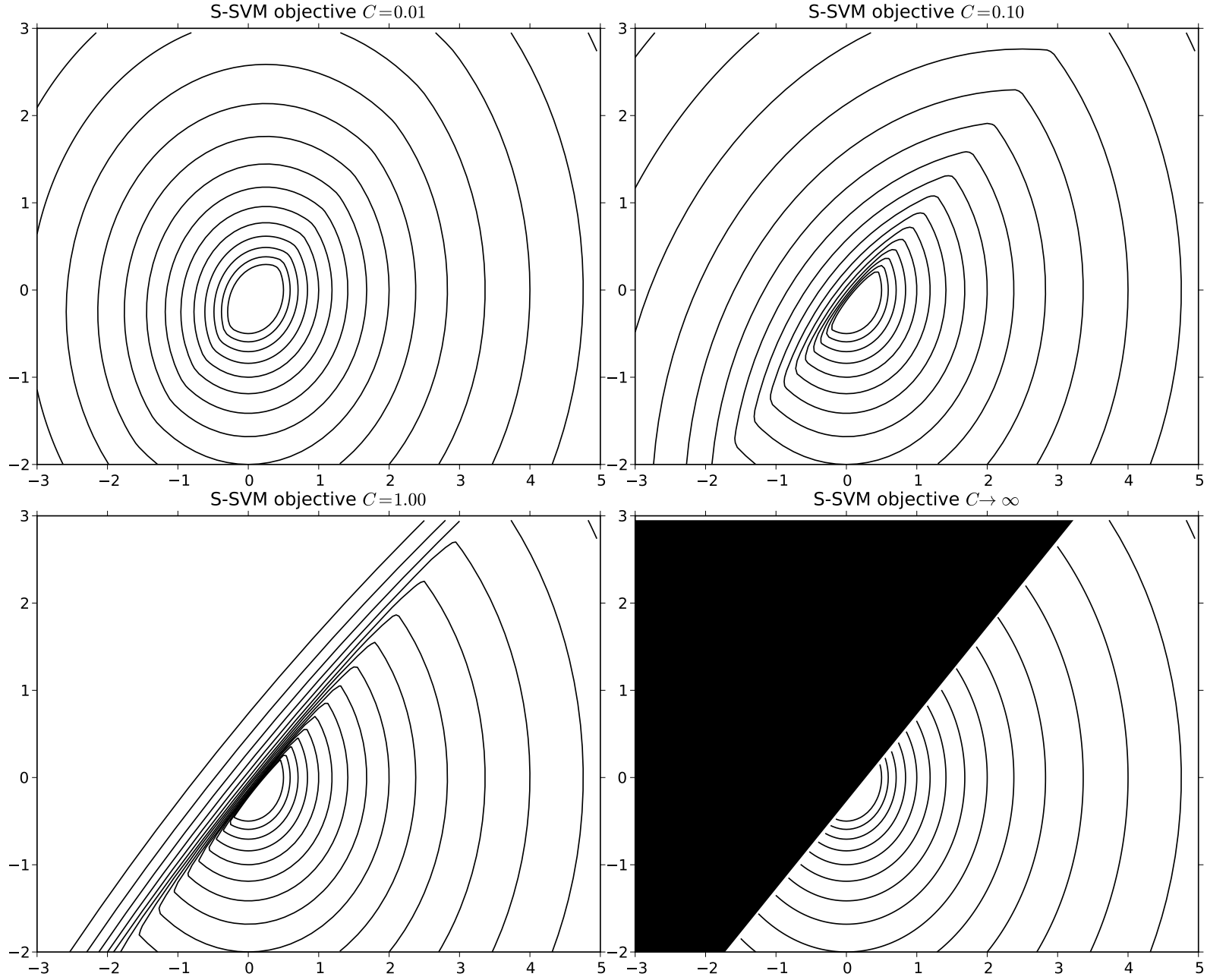
Structured Output Support Vector Machine:

$$\min_{w \in \mathbb{R}^d} \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n}\Big[\max_{y \in \mathcal{Y}} \Delta(y^i, y) + \langle w, \varphi(x^i, y)\rangle - \langle w, \varphi(x^i, y^i)\rangle\Big]_+$$

where $[t]_+ := \max\{0, t\}$.

Unconstrained optimization, convex, non-differentiable objective.

S-SVM Objective Function for $w \in \mathbb{R}^2$:



Structured Output SVM (equivalent formulation):

$$\min_{w \in \mathbb{R}^d, \xi \in \mathbb{R}^n_+} \quad \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi^i$$

subject to, for $i = 1, \ldots, n$,

$$\max_{y \in \mathcal{Y}} \left[\Delta(y^i, y) + \langle w, \varphi(x^i, y) \rangle - \langle w, \varphi(x^i, y^i) \rangle \right] \leq \xi^i$$

n non-linear contraints, convex, differentiable objective.

Structured Output SVM (also equivalent formulation):

$$\min_{w \in \mathbb{R}^d, \xi \in \mathbb{R}^n_+} \quad \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi^i$$

subject to, for $i = 1, \ldots, n$,

$$\Delta(y^i, y) + \langle w, \varphi(x^i, y)\rangle - \langle w, \varphi(x^i, y^i)\rangle \leq \xi^i, \quad \text{for all } y \in \mathcal{Y}$$

$|\mathcal{Y}|n$ linear constraints, convex, differentiable objective.

Example: A "True" Multiclass SVM

- $\mathcal{Y} = \{1, 2, \ldots, K\}, \quad \Delta(y, y') = \begin{cases} 1 & \text{for } y \neq y' \\ 0 & \text{otherwise.} \end{cases}$.
- $\varphi(x, y) = \Big([\![y = 1]\!]\Phi(x),\ [\![y = 2]\!]\Phi(x),\ \ldots,\ [\![y = K]\!]\Phi(x) \Big)$
  $= \Phi(x) e_y^\top$ with e_y=y-th unit vector

Solve:

$$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi^i$$

subject to, for $i = 1, \ldots, n$,

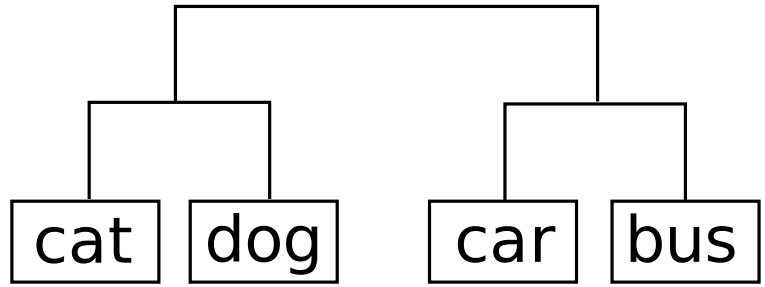
$$\langle w, \varphi(x^i, y^i)\rangle - \langle w, \varphi(x^i, y)\rangle \geq 1 - \xi^i \quad \text{for all } y \in \mathcal{Y} \setminus \{y^i\}.$$

Classification: MAP $\quad f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \ \langle w, \varphi(x, y)\rangle$

Crammer-Singer Multiclass SVM

Hierarchical Multiclass Classification

Loss function can reflect hierarchy:



$$\Delta(y, y') := \frac{1}{2}(\text{distance in tree})$$

$$\Delta(\texttt{cat}, \texttt{cat}) = 0, \quad \Delta(\texttt{cat}, \texttt{dog}) = 1, \quad \Delta(\texttt{cat}, \texttt{bus}) = 2, \quad etc.$$

Solve:

$$\min_{w,\xi} \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n} \xi^i$$

subject to, for $i = 1, \ldots, n$,

$$\langle w, \varphi(x^i, y^i)\rangle - \langle w, \varphi(x^i, y)\rangle \geq \Delta(y^i, y) - \xi^i \quad \text{for all } y \in \mathcal{Y} \setminus \{y^i\}.$$

Numeric Solution: Solving an S-SVM like a CRF.

How to solve?

$$\min_{w \in \mathbb{R}^d} \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n}\left[\max_{y \in \mathcal{Y}} \Delta(y^i, y) + \langle w, \varphi(x^i, y)\rangle - \langle w, \varphi(x^i, y^i)\rangle\right]_+$$

with $[t]_+ := \max\{0, t\}$.

Unconstrained, convex, non-differentiable:

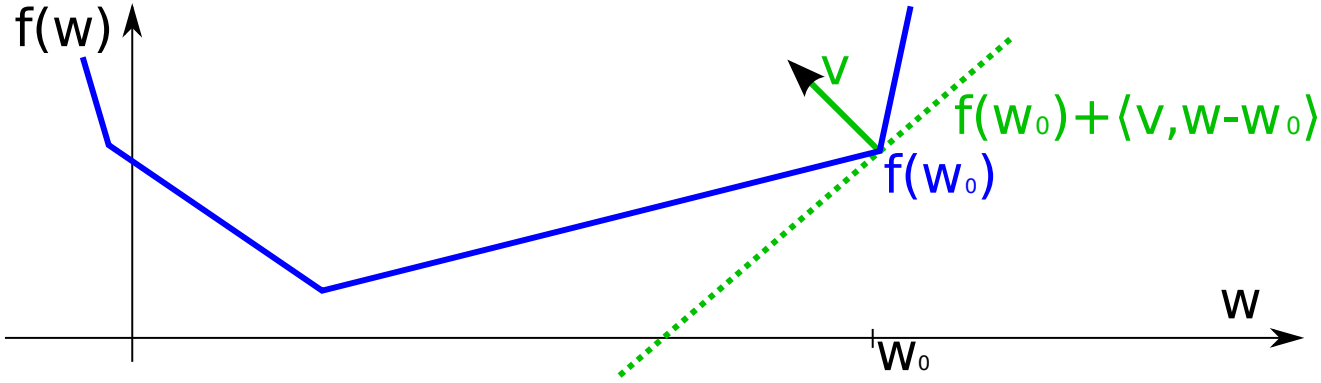
- unconstrained ☺
- convex ☺
- non-differentiable ☹ $\rightarrow$ we can't use gradient descent directly.

Idea: use **sub-gradient descent**.

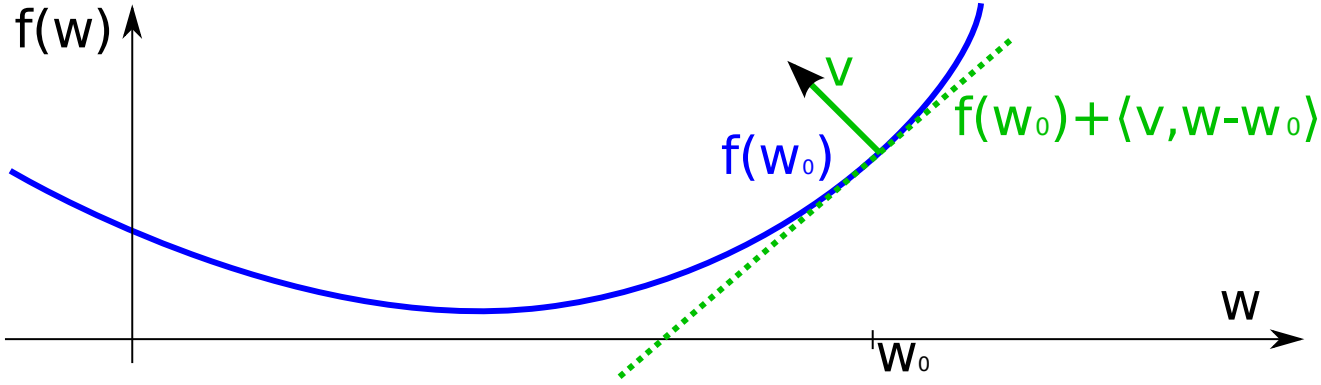
Sub-Gradients

Definition: Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex function. A vector $v \in \mathbb{R}^d$ is called a **sub-gradient** of f at w_0, if

$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



For differentiable f, the gradient $v = \nabla f(w_0)$ is the only subgradient.

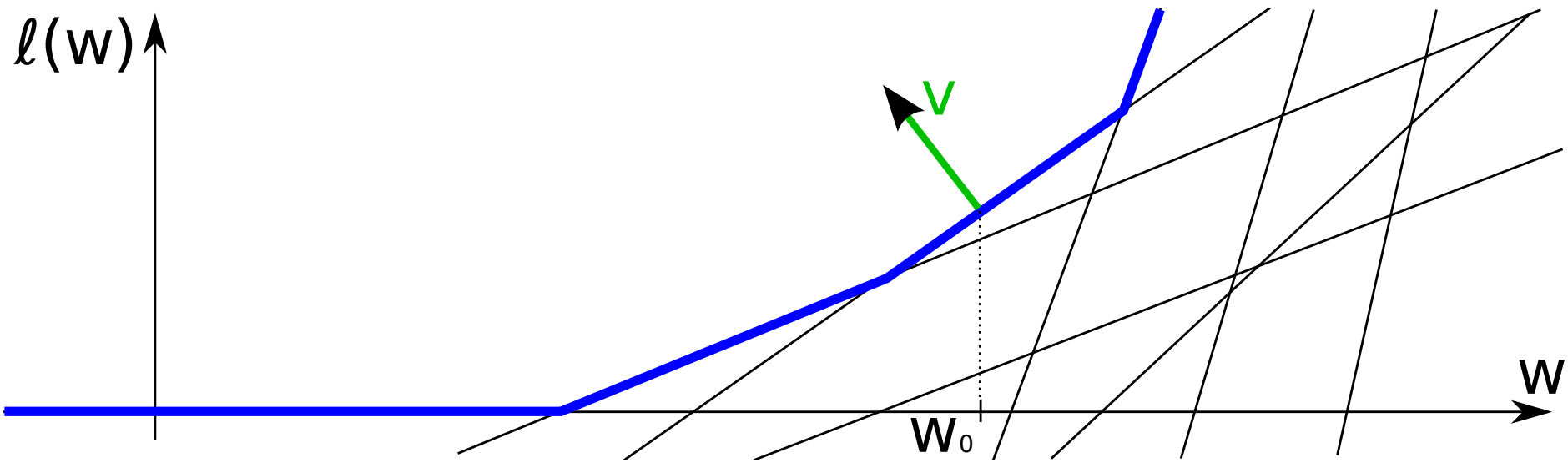


Computing a subgradient:

$$\min_w \ \frac{1}{2}\|w\|^2 + \frac{C}{n}\sum_{i=1}^{n} \ell^i(w)$$

with $\ell^i(w) = \max\{0, \max_y \ell^i_y(w)\}$, and

$$\ell^i_y(w) := \Delta(y^i, y) + \langle w, \varphi(x^i, y)\rangle - \langle w, \varphi(x^i, y^i)\rangle$$



Subgradient of ℓ^i at w_0: find maximal (active) y, use $v = \nabla \ell^i_y(w_0)$.

Subgradient Descent S-SVM Training

input training pairs $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
input feature map $\varphi(x, y)$, loss function $\Delta(y, y')$, regularizer C,
input number of iterations T, stepsizes η_t for $t = 1, \ldots, T$

1: $w \leftarrow \vec{0}$
2: **for** t=1,. . . ,T **do**
3: **for** i=1,. . . ,n **do**
4: $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^i, y) + \langle w, \varphi(x^i, y) \rangle - \langle w, \varphi(x^i, y^i) \rangle$
5: $v^i \leftarrow \varphi(x^i, \hat{y}) - \varphi(x^i, y^i)$
6: **end for**
7: $w \leftarrow w - \eta_t (w - \frac{C}{n} \sum_i v^i)$
8: **end for**

output prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \varphi(x, y) \rangle$.

Obs: each update of w needs 1 MAP-like prediction per example.

We can use the same tricks as for CRFs, e.g. **stochastic updates**:

Stochastic Subgradient Descent S-SVM Training

input training pairs $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
input feature map $\varphi(x, y)$, loss function $\Delta(y, y')$, regularizer C,
input number of iterations T, stepsizes η_t for $t = 1, \ldots, T$

1: $w \leftarrow \vec{0}$
2: **for** t=1,. . . ,T **do**
3: $(x^i, y^i) \leftarrow$ randomly chosen training example pair
4: $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^i, y) + \langle w, \varphi(x^i, y) \rangle - \langle w, \varphi(x^i, y^i) \rangle$
5: $v^i \leftarrow \varphi(x^i, \hat{y}) - \varphi(x^i, y^i)$
6: $w \leftarrow w - \eta_t(w - Cv^i)$
7: **end for**

output prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \varphi(x, y) \rangle$.

Obs: each update of w needs only 1 MAP-like prediction
(but we'll need more iterations until convergence)

Numeric Solution: Solving an S-SVM like a linear SVM.

Other, equivalent, S-SVM formulation was

$$\min_{w\in\mathbb{R}^d,\xi\in\mathbb{R}^n_+} \|w\|^2 \;+\; \frac{C}{n}\sum_{i=1}^{n}\xi^i$$

subject to, for $i = 1, \dots n$,

$$\langle w, \varphi(x^i, y^i)\rangle - \langle w, \varphi(x^i, y)\rangle \geq \Delta(y^i, y) \;-\; \xi^i, \quad \text{for all } y \in \mathcal{Y} \setminus \{y^i\}.$$

With feature vectors $\delta\varphi(x^i, y^i, y) := \varphi(x^i, y^i) - \varphi(x^i, y)$, this is

$$\min_{w\in\mathbb{R}^d,\xi\in\mathbb{R}^n_+} \|w\|^2 \;+\; \frac{C}{n}\sum_{i=1}^{n}\xi^i$$

subject to, for $i = 1, \dots n$, for all $y \in \mathcal{Y} \setminus \{y^i\}$,

$$\langle w, \delta\varphi(x^i, y^i, y)\rangle \geq \Delta(y^i, y) \;-\; \xi^i.$$

Solve

$$\min_{w \in \mathbb{R}^d, \xi \in \mathbb{R}^n_+} \|w\|^2 \; + \; \frac{C}{n} \sum_{i=1}^{n} \xi^i \qquad \text{(QP)}$$

subject to, for $i = 1, \ldots n$, for all $y \in \mathcal{Y} \setminus \{y^i\}$,

$$\langle w, \delta\varphi(x^i, y^i, y) \rangle \geq \Delta(y^i, y) \; - \; \xi^i.$$

This has the same structure as an ordinary SVM!

- quadratic objective ☺
- linear constraints ☺

Question: Can't we use a ordinary QP solver to solve this?

Answer: Almost! We could, if there weren't $n(|\mathcal{Y}| - 1)$ constraints.

- E.g. 100 binary 16×16 images: 10^{79} constraints ☹

Solution: working set training

- It's enough if we enforce the **active constraints**.
  The others will be fulfilled automatically.
- We don't know which ones are active for the optimal solution.
- But it's likely to be only a small number ← can of course be formalized.

Keep a set of potentially active constraints and update it iteratively:

- Start with working set $S = \emptyset$ (no contraints)
- Repeat until convergence:
  - Solve S-SVM training problem with constraints from S
  - Check, if solution violates any of the *full* constraint set
    - ⋆ if no: we found the optimal solution, *terminate*.
    - ⋆ if yes: add most violated constraints to S, *iterate*.

Good *practical performance* and *theoretic guarantees*:

- polynomial time convergence ϵ-close to the global optimum

[Tsochantaridis et al. "Large Margin Methods for Structured and Interdependent Output Variables", JMLR, 2005.]

Working Set S-SVM Training

input training pairs $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
input feature map $\varphi(x, y)$, loss function $\Delta(y, y')$, regularizer C

$S \leftarrow \emptyset$
repeat
  $(w, \xi) \leftarrow$ *solution to QP only with constraints from* S
  for i=1,. . . ,n **do**
    $\hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^i, y) + \langle w, \varphi(x^i, y) \rangle$
    if $\hat{y} \neq y^i$ **then**
      $S \leftarrow S \cup \{(x^i, \hat{y})\}$
    end if
  end for
until S doesn't change anymore.

output prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \varphi(x, y) \rangle$.

Obs: each update of w needs 1 MAP-like prediction per example.
(but we solve globally for w, not locally)

Specialized modifications are possible: In object localization solving

$$\operatorname*{argmax}_{i=1,\ldots,n,\, y\in\mathcal{Y}} \ell_y^i(w)$$

can be done much faster than by exhaustive evaluation of

$$\operatorname{argmax}_{y\in\mathcal{Y}} \ \ell_y^1(w), \ldots, \operatorname{argmax}_y \ \ell_y^n(w).$$

Single Element Working Set S-SVM Training

input ...

1: $S \leftarrow \emptyset$
2: **repeat**
3: $(w, \xi) \leftarrow$ *solution to QP only with constraints from* S
4: $(\hat{\imath}, \hat{y}) \leftarrow \operatorname{argmax}_{i,y} \Delta(y^i, y) + \langle w, \varphi(x^i, y)\rangle$
5: **if** $\hat{y} \neq y^{\hat{\imath}}$ **then**
6: $S \leftarrow S \cup \{(x^{\hat{\imath}}, \hat{y})\}$
7: **end if**
8: **until** S doesn't change anymore.

Obs: each update of w needs only 1 MAP-like prediction.

"One-Slack" Structured Output SVM:
(equivalent to ordinary S-SVM by $\xi = \frac{1}{n}\sum_i \xi^i$)

$$\min_{w\in\mathbb{R}^d,\xi\in\mathbb{R}_+} \quad \frac{1}{2}\|w\|^2 + C\xi$$

subject to

$$\sum_{i=1}^{n}\Big[\Delta(y^i,\hat{y}^i) + \langle w, \varphi(x^i,\hat{y}^i)\rangle - \langle w, \varphi(x^i,y^i)\rangle\Big] \leq n\xi,$$

subject to, for all $(\hat{y}^1,\ldots,\hat{y}^n) \in \mathcal{Y}\times\cdots\times\mathcal{Y}$,

$$\sum_{i=1}^{n}\Delta(y^i,\hat{y}^i) + \langle w, \sum_{i=1}^{n}[\varphi(x^i,\hat{y}^i) - \varphi(x^i,y^i)]\rangle \leq n\xi,$$

$|\mathcal{Y}|^n$ linear constraints, convex, differentiable objective.

We have blown up the constraint set even further:

- 100 binary 16×16 images: 10^{177} constraints (instead of 10^{79}).

Working Set One-Slack S-SVM Training

input training pairs $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
input feature map $\varphi(x, y)$, loss function $\Delta(y, y')$, regularizer C

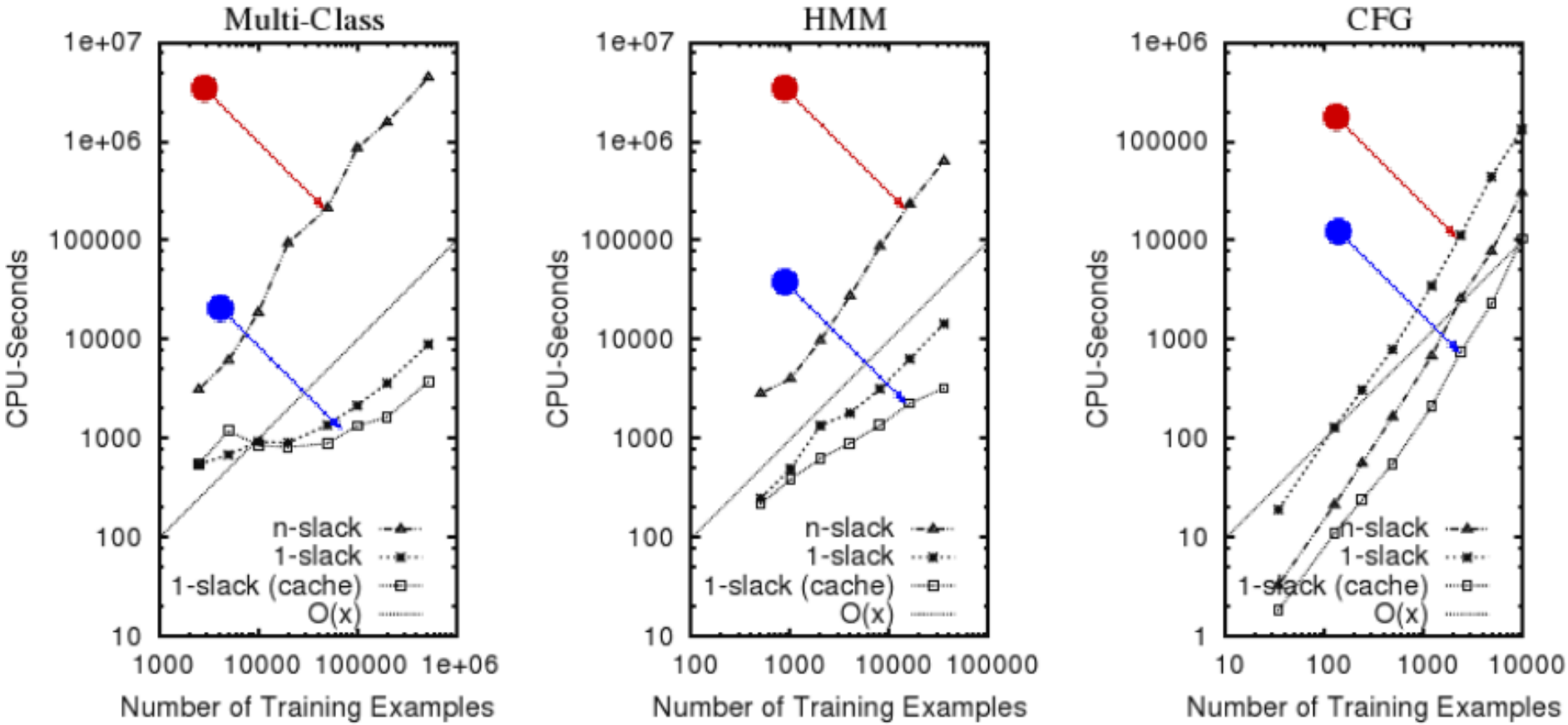
1: $S \leftarrow \emptyset$
2: **repeat**
3: $(w, \xi) \leftarrow$ *solution to QP only with constraints from* S
4: **for** i=1,. . . ,n **do**
5: $\hat{y}^i \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^i, y) + \langle w, \varphi(x^i, y) \rangle$
6: **end for**
7: $S \leftarrow S \cup \{\Big((x^1, \ldots, x^n), (\hat{y}^1, \ldots, \hat{y}^n)\Big)\}$
8: **until** S doesn't change anymore.

output prediction function $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \varphi(x, y) \rangle$.

Often faster convergence:
We add one strong constraint per iteration instead of n weak ones.

[Joachims, Finley, Yu: "Cutting-Plane Training of Structural SVMs", Machine Learning, 2009]

Training Times ordinary S-SVM versus one-slack S-SVM



Observation 1: One-slack training is usually faster than n-slack.

Observation 2: Training structured models is nevertheless **slow**.

Figure: [Joachims, Finley, Yu: "Cutting-Plane Training of Structural SVMs", Machine Learning, 2009]

Numeric Solution: Solving an S-SVM like a non-linear SVM.

Dual S-SVM problem

$$\max_{\alpha \in \mathbb{R}_+^{n|\mathcal{Y}|}} \sum_{\substack{i=1,\dots,n \\ y \in \mathcal{Y}}} \alpha_{iy} \Delta(y^i, y) - \frac{1}{2} \sum_{\substack{y,y' \in \mathcal{Y} \\ i,i'=1,\dots,n}} \alpha_{iy} \alpha_{i'y'} \langle \delta\varphi(x^i, y^i, y), \delta\varphi(x^{i'}, y^{i'}, y') \rangle$$

subject to, for $i = 1, \dots, n$,

$$\sum_{y \in \mathcal{Y}} \alpha_{iy} \leq \frac{C}{n}.$$

Like ordinary SVMs, we can *dualize* the S-SVMs optimization:

- $\min$ becomes $\max$,
- original (primal) variables w, ξ disappear,
- we get one dual variable α_{iy} for each primal constraint.

n linear contraints, convex, differentiable objective, $n|\mathcal{Y}|$ **variables**.

Data appears only inside inner products: **kernelize**

- Define joint kernel function $k : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$

$$k(\,(x, y)\,,\,(x', y')\,) = \langle \varphi(x, y), \varphi(x', y') \rangle.$$

- k measure similarity between two *(input,output)*-pairs.

- We can express the optimization in terms of k:

$$\begin{aligned}
&\langle \delta\varphi(x^i, y^i, y)\,, \delta\varphi(x^{i'}, y^{i'}, y') \rangle \\
&\qquad = \Big\langle\, \varphi(x^i, y^i) - \varphi(x^i, y)\ ,\ \varphi(x^{i'}, y^{i'}) - \varphi(x^{i'}, y')\, \Big\rangle \\
&\qquad = \langle\, \varphi(x^i, y^i), \varphi(x^{i'}, y^{i'})\, \rangle - \langle\, \varphi(x^i, y^i), \varphi(x^{i'}, y')\, \rangle \\
&\qquad\quad - \langle\, \varphi(x^i, y), \varphi(x^{i'}, y^{i'}) \rangle + \langle\, \varphi(x^i, y), \varphi(x^{i'}, y') \rangle \\
&\qquad = k(\,(x^i, y^i), (x^{i'}, y^{i'})\,) - k(\,(x^i, y^i), \varphi(x^{i'}, y')\,) \\
&\qquad\quad - k(\,(x^i, y), (x^{i'}, y^{i'})\,) + k(\,(x^i, y), \varphi(x^{i'}, y')\,) \\
&\qquad =: K_{ii'yy'}
\end{aligned}$$

Kernelized S-SVM problem:

$$\max_{\alpha \in \mathbb{R}_+^{n|\mathcal{Y}|}} \sum_{\substack{i=1,\dots,n \\ y \in \mathcal{Y}}} \alpha_{iy} \Delta(y^i, y) - \frac{1}{2} \sum_{\substack{y,y' \in \mathcal{Y} \\ i,i'=1,\dots,n}} \alpha_{iy} \alpha_{i'y'} K_{ii'yy'}$$

subject to, for $i = 1, \dots, n$,

$$\sum_{y \in \mathcal{Y}} \alpha_{iy} \leq \frac{C}{n}.$$

- too many variables: train with **working set** of α_{iy}.

Kernelized prediction function:

$$f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \sum_{iy'} \alpha_{iy'} k(\,(x^i, y^i), (x, y)\,)$$

Be careful in kernel design:

Evaluating f can easily become very slow / completely infeasible.

What do "joint kernel functions" look like?

$$k(\,(x,y)\,,\,(x',y')\,) = \langle \varphi(x,y), \varphi(x',y') \rangle.$$

Graphical Model: φ decomposes into components per factor:

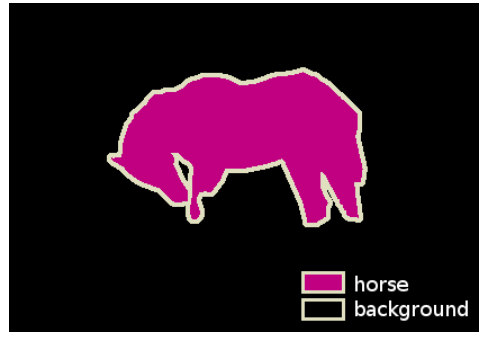
- $\varphi(x,y) = \Big(\varphi_f(x_f,y_f)\Big)_{f\in\mathcal{F}}$

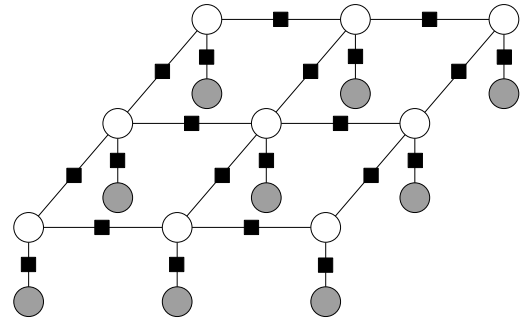
Kernel k decomposes into sum over *factors*:

$$\begin{aligned} k(\,(x,y)\,,\,(x',y')\,) &= \left\langle \Big(\varphi_f(x_f,y_f)\Big)_{f\in\mathcal{F}}, \Big(\varphi_f(x'_f,y'_f)\Big)_{f\in\mathcal{F}} \right\rangle \\ &= \sum_{f\in\mathcal{F}} \langle\, \varphi_f(x_f,y_f), \varphi_f(x'_f,y'_f)\,\rangle \\ &= \sum_{f\in\mathcal{F}} k_f(\,(x_f,y_f),(x'_f,y'_f)\,) \end{aligned}$$

We can define kernels for each factor (e.g. nonlinear).

Example: figure-ground segmentation with grid structure

$(x, y) \hat{=} ($  , 

$)$ 

Typical kernels: arbirary in x, linear (or at least simple) w.r.t. y:

- Unary factors:

$$k_p((x, y_p), (x', y'_p) = k(x_{N(p)}, x'_{N(p)}) [\![y_p = y'_p]\!]$$

with k image kernel on local neighborhood $N(p)$, e.g. χ^2 or *histogram intersection*

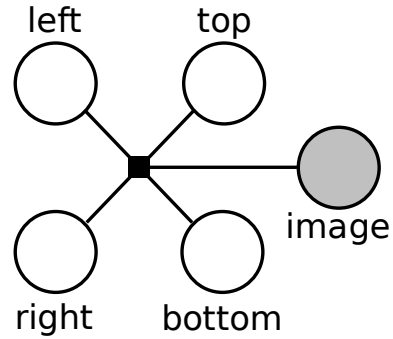
- Pairwise factors:

$$k_{pq}((y_p, y_q), (y'_p, y'_p) = [\![y_p = y'_p]\!] \, [\![y_q = y'_q]\!]$$

More powerful than all-linear, and MAP prediction still possible.

Example: object localization

$(x, y) \hat{=} ($  ,  $)$



Only one factor that includes all x and y:

$$k(\,(x, y)\,,\,(x', y')\,) = k_{image}(x|_y, x'|_{y'})$$

with k_{image} image kernel and $x|_y$ is image region within box y.

MAP prediction is just object localization with k_{image}-SVM.

$$f(x) = \operatorname*{argmax}_{y \in \mathcal{Y}} \sum_{iy'} \alpha_{iy'} k_{image}(x^i|_{y'}, x|_y)$$

Summary I – S-SVM Learning

Task: parameter learning

- given training set $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$, learn parameters w for prediction function

$$F(x, y) = \langle w, \varphi(x, y) \rangle.$$

- predict $f : \mathcal{X} \to \mathcal{Y}$ by $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} F(x, y)$ ($\hat{=}$ MAP)

Solution from maximum-margin principle:

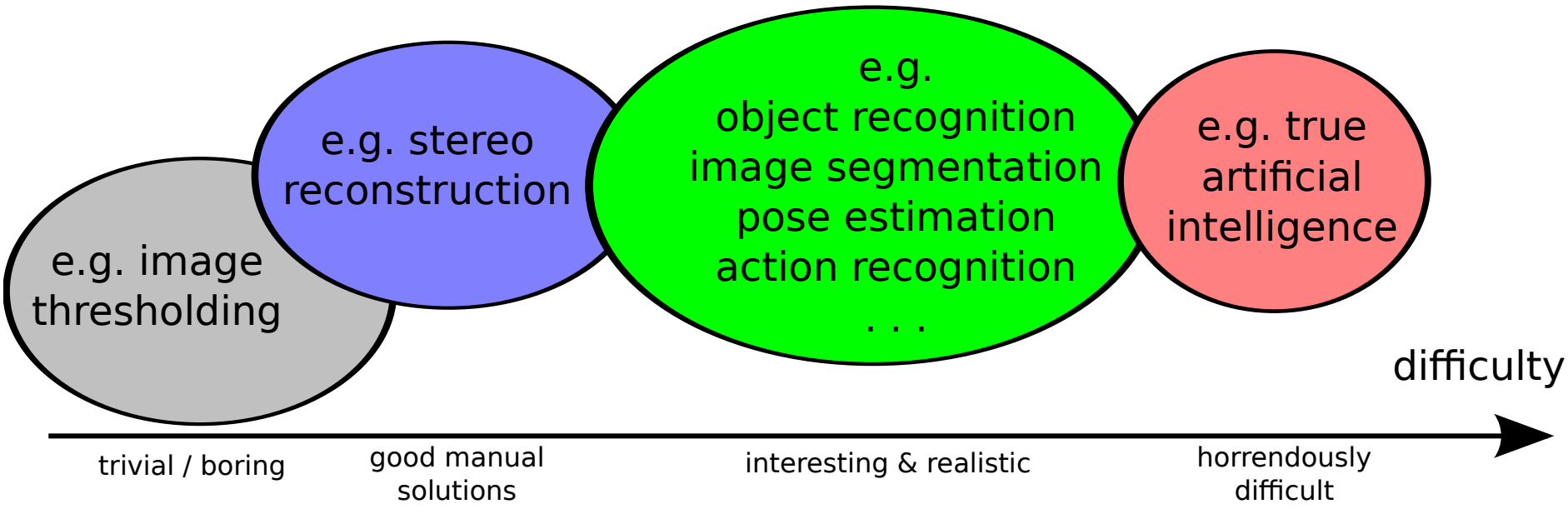
- For each example, correct output must be better than all others:

$$\text{"} \quad F(x^i, y^i) \geq \Delta(y^i, y) + F(x^i, y) \qquad \text{for all } y \in \mathcal{Y} \setminus \{y^i\}. \quad \text{"}$$

- convex optimization problem (similar to multiclass SVM)
- many equivalent formulations $\to$ different training algorithms
- **training calls MAP repeatedly, no probabilistic inference.**

For what problems can we learn solution?

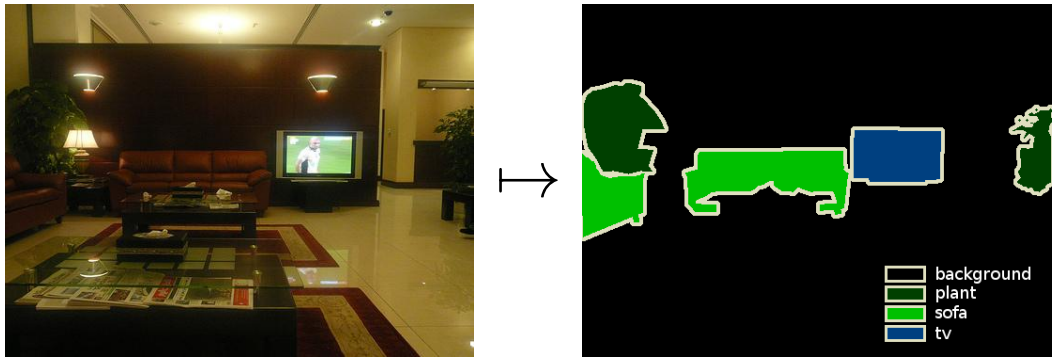
- we need training data,
- we need to be able to perform MAP/MAP-like prediction.



For many interesting problems, we cannot do exact MAP prediction.

The Need for Approximate Inference

- Image segmentation on pixel grid:
  - two labels and positive pairwise weights: exact MAP doable
  - more than two labels: MAP generally NP-hard



- Stereo reconstruction (grid) $\#labels = \#depth\ levels$:
  - convex penalization of disparities: exact MAP doable
  - non-convex (truncated) penalization: NP-hard

- Human Pose estimation
  - limbs independent given torso: exact MAP doable
  - limbs interact with each other: MAP NP-hard

Approximate inference $\hat{y} \approx \operatorname{argmax}_{y \in \mathcal{Y}} F(x, y)$:

- has been used for decades, e.g. for
  - classical combinatorial problems: traveling salesman, knapsack
  - energy minimization: Markov/Conditional Random Fields
- often yield good (=almost perfect) solutions y
- can have guarantees $F(x, y^*) \leq F(x, \hat{y}) \leq (1 + \epsilon) F(x, y^*)$
- typically much faster than exact inference

Can't we use that for S-SVM training? **Iterating it is problematic!**

- In each S-SVM training iteration
  - we solve $y^* = \operatorname{argmax}_y [\Delta(y^i, y) + F(x^i, y)]$,
  - we use y^* to update w,
  - we stop, when there are no more violated constraints / updates.
- With $\hat{y} \approx \operatorname{argmax}_y [\Delta(y^i, y) + F(x^i, y)]$
  - errors can accumulate,
  - we might stop too early.

Training S-SVMs with approximate MAP is active research question.

Summary – Learning for Structured Prediction

Task: Learn parameter w for a function $f : \mathcal{X} \to \mathcal{Y}$

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

Regularized risk minimization framework:

- loss function:
  - logistic loss $\rightarrow$ conditional random fields (CRF)
  - Hinge loss $\rightarrow$ structured support vector machine (S-SVM)
- regularization:
  - L^2 norm of $w \leftrightarrow$ Gaussian prior

Neither guarantees better prediction accuracy than the other.

see e.g. [Nowozin et al., ECCV 2010]

Difference is in numeric optimization:

- Use CRFs if you can do probabilistic inference.
- Use S-SVMs if you can do loss-augmented MAP prediction.

Open Research in Structured Learning

- Faster training
  - CRFs need many runs of probablistic inference,
  - SSVMs need many runs of MAP-like predictions.
- Reduce amount of training data necessary
  - semi-supervised learning? transfer learning?
- Understand competing training methods
  - when to use *probabilistic training*, when *maximum margin*?
- **Understand S-SVM with approximate inference**
  - very often, exactly solving $\operatorname{argmax}_y F(x, y)$ is infeasible.
  - can we guarantee convergence even with approximate inference?

Vacancies at I.S.T. Austria, Vienna

More info: **www.ist.ac.at** or **chl@ist.ac.at**.

- **PhD at I.S.T. Graduate School**
  - 1(2) + 3 yr PhD program
  - full scholarship
  - flexible starting dates
- **PostDoc in my group**
  - *computer vision*
    - object/attribute prediction.
  - *machine learning*
    - structured output learning.
  - curiosity driven basic research
    - no project deadlines,
    - no mandatory teaching, ...
- **Professor / Assistant Professor**
- **Visiting Scientist**
- **Internships**