Learning with Structured Inputs and Outputs

Christoph H. Lampert
IST Austria (Institute of Science and Technology Austria), Vienna

INRIA CVML Summer School, Grenoble, July 2012

IST AUSTRIA
Institute of Science and Technology

Slides: `http://www.ist.ac.at/~chl/`

Schedule

9:30-10:30	Introduction to Graphical Models
10:30-11:00	Conditional Random Fields
11:00-11:30	Structured Support Vector Machines

Slides available on my home page:

`http://www.ist.ac.at/~chl`

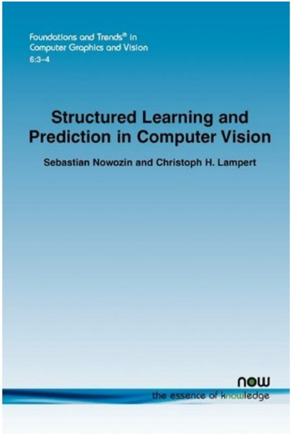
Extended version lecture in book form (180 pages)

Foundations and Trends in
Computer Graphics and Vision

now publisher

`http://www.nowpublishers.com/`

Available as PDF on my homepage



"Normal" Machine Learning:

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

Structured Output Learning:

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

"Normal" Machine Learning:

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

- inputs $\mathcal{X}$ can be any kind of objects
- output y is a real number

Structured Output Learning:

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

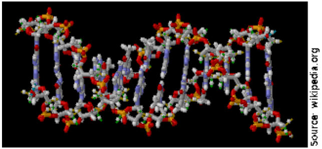
- inputs $\mathcal{X}$ can be any kind of objects
- outputs $y \in \mathcal{Y}$ are complex (structured) objects

What is structured data?

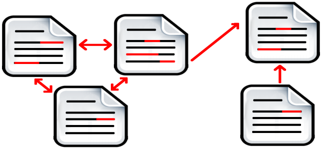
Ad hoc definition: data that consists of several parts, and not only the parts themselves contain information, but also the way in which the parts belong together.

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

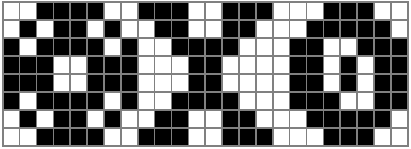
Text



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 tutorial: Applications and Examples from Computer Vision

Probabilistic Graphical Models

How to express $f : \mathcal{X} \to \mathcal{Y}$?

How to express $f : \mathcal{X} \to \mathcal{Y}$?

Scalar functions, $\mathcal{X} = \mathbb{R}^D$, $\mathcal{Y} = \mathbb{R}$

$x = (x_1, \dots, x_d)$, where $x_1, \dots, x_D$ are just numbers $\to$ do anything

$$e.g. \qquad f(x_1, \dots, x_4) = (x_1 + x_2)^2 + e^{\frac{1}{2\pi}(\sqrt{\sin(x_3 x_4)})}.$$

Application: predicting stock prices

How to express $f : \mathcal{X} \to \mathcal{Y}$?

Scalar functions, $\mathcal{X} = \mathbb{R}^D$, $\mathcal{Y} = \mathbb{R}$

$x = (x_1, \dots, x_d)$, where $x_1, \dots, x_D$ are just numbers $\to$ do anything

$$e.g. \qquad f(x_1, \dots, x_4) = (x_1 + x_2)^2 + e^{\frac{1}{2\pi}(\sqrt{\sin(x_3 x_4)})}.$$

Application: predicting stock prices

Boolean functions, $\mathcal{X} = \mathbb{R}^D$, $\mathcal{Y} = \{0, 1\}$

Compute real-valued function $\hat{f} : \mathbb{R}^D \to \mathbb{R}$ and threshold it:

$$f(x) = \operatorname{sign} \hat{f}(x)$$

Application: decide whether to buy a stock or not

Scalar functions, $\mathcal{X}$ = anything, $\mathcal{Y} = \mathbb{R}$

We can't *compute* directly with $x \in \mathcal{X}$.
But we can extract *features*, $\phi : \mathcal{X} \to \mathbb{R}^D$:

$$e.g. \qquad f(x) = \sum_{i=1}^{D} w_i \phi_i(x) + b$$

or use a *kernel function*, $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$:

$$e.g. \qquad f(x) = \sum_{j=1}^{n} \alpha_j k(x^j, x)$$

Application: image classification

- $x \equiv$ image
- $\phi(x) \equiv$ e.g. HoG features
- $k(x, x') \equiv$ e.g. χ^2-kernel of visual word histogram

Structured output function, $\mathcal{X}$ = anything, $\mathcal{Y}$ = anything

1) Define auxiliary function, $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, using *joint features* $\phi(x, y)$:

$$e.g. \qquad g(x,y) = \sum_i w_i \phi_i(x,y) + b,$$

or using a *joint kernel function* $k(\,(x,y),(x',y')\,)$:

$$e.g. \qquad g(x,y) = \sum_j \alpha_j k(\,(x^j, y^j),(x,y)\,)$$

2) Obtain $f : \mathcal{X} \to \mathcal{Y}$ by *maximimization*:

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

Structured output function, $\mathcal{X}$ = anything, $\mathcal{Y}$ = anything

1) Define auxiliary function, $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$, using *joint features* $\phi(x, y)$:

$$e.g. \qquad g(x, y) = \sum_i w_i \phi_i(x, y) + b,$$

or using a *joint kernel function* $k(\,(x, y), (x', y')\,)$:

$$e.g. \qquad g(x, y) = \sum_j \alpha_j k(\,(x^j, y^j), (x, y)\,)$$

2) Obtain $f : \mathcal{X} \to \mathcal{Y}$ by *maximimization*:

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

Construction familiar from one-vs-rest SVMs, $\mathcal{Y} = \{1, \dots, K\}$:

- Train classifiers $f_y : \mathcal{X} \to \mathbb{R}$ for each class $y \in \{1, \dots, K\}$.
- For new sample $x \in \mathcal{X}$, predict by $f(x) = \operatorname{argmax}_y f_y(x)$

A Probabilistic View

Computer Vision almost always deals with *uncertain* information

- Training examples are collected "randomly" (e.g. from the web)
- Annotation is "noisy" (there can be mistakes, or ambiguous cases)
- Tasks cannot be solved with 100 percent certainty, because of
  - incomplete information (*"guess what number I think of"*), or
  - inherent randomness (*"guess a coin toss"*)

Uncertainty is captured by (conditional) probability distributions: $p(y|x)$

- for input $x \in \mathcal{X}$, how *likely* is $y \in \mathcal{Y}$ the correct output?

We can also phrase this as

- what's the probability of observing y given x?
- how strong is our *belief* in y in we know x?

A Probabilistic View on $f : \mathcal{X} \to \mathcal{Y}$

Structured output function, $\mathcal{X}$ = anything, $\mathcal{Y}$ = anything

We need to define an auxiliary function, $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$.

$$e.g. \qquad g(x, y) := p(y|x).$$

Then *maximimization*

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

becomes *maximum a posteriori (MAP) prediction*.

Interpretation:
If you have to *decide* for a single output, $y \in \mathcal{Y}$, use the most probable one.

Probability Distributions

$$\forall y \in \mathcal{Y} \quad p(y) \geq 0 \qquad \text{(positivity)}$$

$$\sum_{y \in \mathcal{Y}} p(y) = 1 \qquad \text{(normalization)}$$

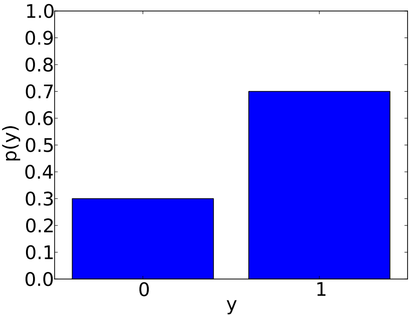
Probability Distributions

$$\forall y \in \mathcal{Y} \quad p(y) \geq 0 \qquad \text{(positivity)}$$

$$\sum_{y \in \mathcal{Y}} p(y) = 1 \qquad \text{(normalization)}$$

Example: binary ("Bernoulli") variable $y \in \mathcal{Y} = \{0, 1\}$

- 2 values,
- 1 degree of freedom



Conditional Probability Distributions

$$\forall x \in \mathcal{X}\ \forall y \in \mathcal{Y} \quad p(y|x) \geq 0 \qquad \text{(positivity)}$$

$$\forall x \in \mathcal{X}\ \sum_{y \in \mathcal{Y}} p(y|x) = 1 \qquad \text{(normalization w.r.t. } y\text{)}$$

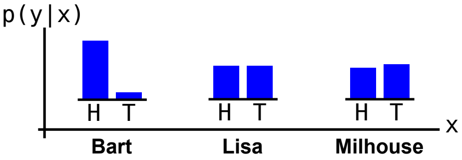
Conditional Probability Distributions

$$\forall x \in \mathcal{X}\ \forall y \in \mathcal{Y} \quad p(y|x) \geq 0 \qquad \text{(positivity)}$$

$$\forall x \in \mathcal{X}\ \sum_{y \in \mathcal{Y}} p(y|x) = 1 \qquad \text{(normalization w.r.t. } y\text{)}$$

For example: **binary** prediction
$\mathcal{X} = \{\text{coin owners}\}$, $\mathcal{Y} = \{0, 1\}$

- each x: 2 values, 1 d.o.f.



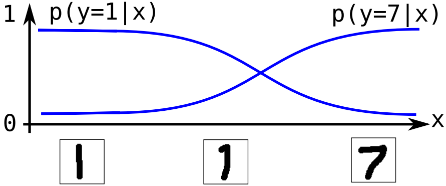
Conditional Probability Distributions

$$\forall x \in \mathcal{X}\ \forall y \in \mathcal{Y} \quad p(y|x) \geq 0 \qquad \text{(positivity)}$$

$$\forall x \in \mathcal{X}\ \sum_{y \in \mathcal{Y}} p(y|x) = 1 \qquad \text{(normalization w.r.t. } y\text{)}$$

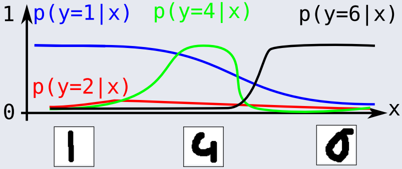
For example: **binary** prediction
$\mathcal{X} = \{\text{images}\}$, $y \in \mathcal{Y} = \{0, 1\}$

- each x: 2 values, 1 d.o.f.
  $\rightarrow$ one (or two) *function*



Multi-class prediction, $y \in \mathcal{Y} = \{1, \dots, K\}$

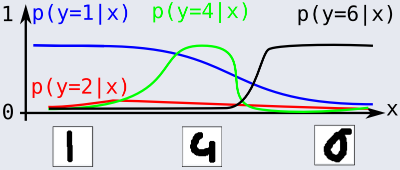
- each x: K values, $K-1$ d.o.f. $\rightarrow K-1$ *functions*
- or 1 vector-valued function with $K-1$ outputs



Typically: K functions, plus explicit normalization

Multi-class prediction, $y \in \mathcal{Y} = \{1, \dots, K\}$

- each x: K values, $K-1$ d.o.f.
  $\rightarrow K-1$ *functions*
- or 1 vector-valued function with $K-1$ outputs

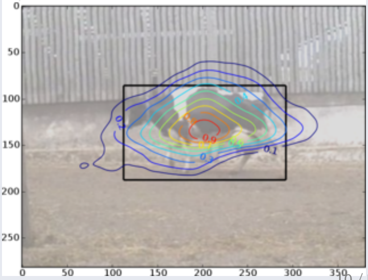


Typically: K functions, plus explicit normalization

Example: **predicting the center point of an object**

$y \in \mathcal{Y} = \{(1,1), \dots, (\textit{width}, \textit{height})\}$
- for each x: $|\mathcal{Y}| = W \cdot H$ values,

$y = (y_1, y_2) \in \mathcal{Y}_1 \times \mathcal{Y}_2$ with
$\mathcal{Y}_1 = \{(1, \dots, \textit{width}\}$ and
$\mathcal{Y}_2 = \{1, \dots, \textit{height}\}$.
- each x: $|\mathcal{Y}_1| \cdot |\mathcal{Y}_2| = W \cdot H$ values,

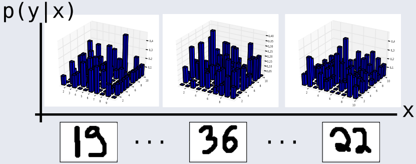


Structured objects: **predicting M variables jointly**

$\mathcal{Y} = \{1, K\} \times \{1, K\} \cdots \times \{1, K\}$

For each x:

- K^M values, $K^M - 1$ d.o.f.
  $\to K^M$ functions



Example: Object detection with **variable size bounding box**

$$\mathcal{Y} \subset \{1, \ldots, W\} \times \{1, \ldots, H\} \times \{1, \ldots, W\} \times \{1, \ldots, H\}$$

$y = ($*left*, *top*, *right*, *bottom*$)$

For each x:

- $\frac{1}{4} W(W-1)H(H-1)$ values
  (millions to billions...)

Example: image denoising

$\mathcal{Y} = \{640 \times 480 \text{ RGB images}\}$

For each x:

- 16777216^{307200} values in $p(y|x)$,
- $\geq 10^{2,000,000}$ functions

too much!

Example: image denoising

$\mathcal{Y} = \{640 \times 480 \text{ RGB images}\}$

For each x:

- 16777216^{307200} values in $p(y|x)$,
- $\geq 10^{2,000,000}$ functions

too much!

We cannot consider all possible distributions, we must impose **structure**.

Probabilistic Graphical Models

A **(probabilistic) graphical model** defines

- a *family of probability distributions* over a set of random variables, by means of a *graph*.

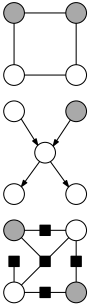
Probabilistic Graphical Models

A **(probabilistic) graphical model** defines

- a *family of probability distributions* over a set of random variables, by means of a *graph*.

Popular classes of graphical models,

- Undirected graphical models (Markov random fields),
- Directed graphical models (Bayesian networks),
- **Factor graphs,**
- Others: chain graphs, influence diagrams, etc.

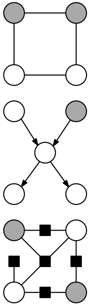


Probabilistic Graphical Models

A **(probabilistic) graphical model** defines

- a *family of probability distributions* over a set of random variables, by means of a *graph*.

Popular classes of graphical models,



- Undirected graphical models (Markov random fields),
- Directed graphical models (Bayesian networks),
- **Factor graphs,**
- Others: chain graphs, influence diagrams, etc.

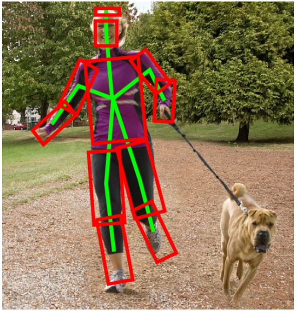
The graph encodes *conditional independence assumptions* between the variables:

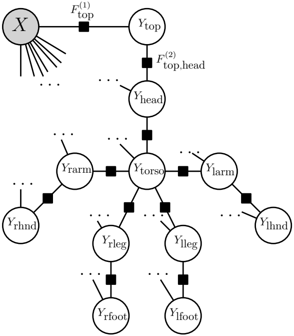
- for $N(i)$ are the neighbors of node i in the graph

$$p(y_i | y_{V\setminus\{i\}}) = p(y_i | y_N(i))$$

with $y_{V\setminus\{i\}} = (y_1, \ldots, y_{i-1}, y_{i+1}, y_n)$.

Example: Pictorial Structures for Articulated Pose Estimation



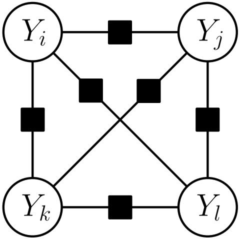


- In principle, all parts depend on each other.
  - Knowing where the head is puts constraints on where the feet can be.
- But **conditional independences** as specified by the graph:
  - If we *know* where the **left leg** is, the **left foot**'s position does not depend on the **torso** position anymore, etc.

$$p(y_{\text{lfoot}}|y_{\text{top}},\dots,y_{\text{torso}},\dots,y_{\text{rfoot}},x) = p(y_{\text{lfoot}}|y_{\text{lleg}},x)$$

Factor Graphs

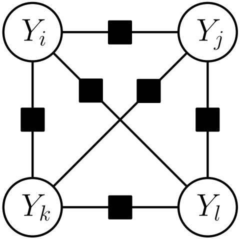
- Decomposable output $y = (y_1, \dots, y_{|V|})$
- Graph: $G = (V, \mathcal{F}, \mathcal{E})$, $\mathcal{E} \subseteq V \times \mathcal{F}$
  - variable nodes V,
  - factor nodes $\mathcal{F}$,
  - edges $\mathcal{E}$ between variable and factor nodes.
  - each factor $F \in \mathcal{F}$ connects a subset of nodes,
  - write $F = \{v_1, \dots, v_{|F|}\}$ and $y_F = (y_{v_1}, \dots, y_{v_{|F|}})$



Factor graph

Factor Graphs

- Decomposable output $y = (y_1, \dots, y_{|V|})$
- Graph: $G = (V, \mathcal{F}, \mathcal{E})$, $\mathcal{E} \subseteq V \times \mathcal{F}$
  - variable nodes V,
  - factor nodes $\mathcal{F}$,
  - edges $\mathcal{E}$ between variable and factor nodes.
  - each factor $F \in \mathcal{F}$ connects a subset of nodes,
  - write $F = \{v_1, \dots, v_{|F|}\}$ and $y_F = (y_{v_1}, \dots, y_{v_{|F|}})$



Factor graph

- Factorization into **potentials** ψ at **factors**:

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

- Z is a normalization constant, called **partition function**:

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

Conditional Distributions

How to model $p(y|x)$?

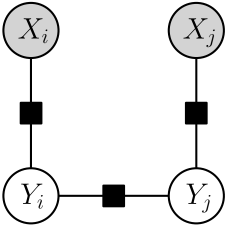
- Potentials become also functions of (part of) x: $\psi_F(y_F; x_F)$ instead of just $\psi_F(y_F)$

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

- Partition function depends on x_F

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

- Note: x is treated just as an argument, not as a random variable.



Factor graph

Conditional random fields (CRFs)

Conventions: Potentials and Energy Functions

Assume $\psi_F(y_F) > 0$. Then

- instead of *potentials*, we can also work with *energies*:

$$\psi_F(y_F; x_F) = \exp(-E_F(y_F; x_F)),$$

or equivalently

$$E_F(y_F; x_F) = -\log(\psi_F(y_F; x_F)).$$

Conventions: Potentials and Energy Functions

Assume $\psi_F(y_F) > 0$. Then

- instead of *potentials*, we can also work with *energies*:

$$\psi_F(y_F; x_F) = \exp(-E_F(y_F; x_F)),$$

  or equivalently

$$E_F(y_F; x_F) = -\log(\psi_F(y_F; x_F)).$$

- $p(y|x)$ can be written as

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

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

Conventions: Energy Minimization

$$\begin{aligned}
\operatorname*{argmax}_{y} p(y|x) &= \operatorname*{argmax}_{y\in\mathcal{Y}} \frac{1}{Z(x)} \exp(-E(y;x)) \\
&= \operatorname*{argmax}_{y\in\mathcal{Y}} \exp(-E(y;x)) \\
&= \operatorname*{argmax}_{y\in\mathcal{Y}} -E(y;x) \\
&= \operatorname*{argmin}_{y\in\mathcal{Y}} E(y;x).
\end{aligned}$$

MAP prediction can be performed by *energy minimization*.

Conventions: Energy Minimization

$$
\begin{aligned}
\operatorname*{argmax}_{y} \; p(y|x) &= \operatorname*{argmax}_{y\in\mathcal{Y}} \; \frac{1}{Z(x)} \exp(-E(y;x)) \\
&= \operatorname*{argmax}_{y\in\mathcal{Y}} \; \exp(-E(y;x)) \\
&= \operatorname*{argmax}_{y\in\mathcal{Y}} \; -E(y;x) \\
&= \operatorname*{argmin}_{y\in\mathcal{Y}} \; E(y;x).
\end{aligned}
$$

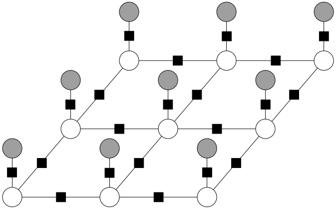
MAP prediction can be performed by *energy minimization*.

In practice, one typically models the energy function directly.
$\rightarrow$ the probability distribution is uniquely determined by it.

Example: An Energy Function for Image Segmentation

Foreground/background image segmentation

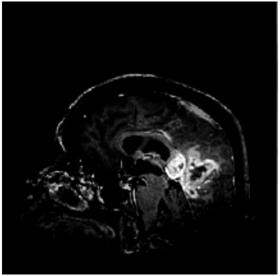
- $\mathcal{X} = [0, 255]^{WH}$, $\mathcal{Y} = \{0, 1\}^{WH}$
  foreground: $y_i = 1$, background: $y_i = 0$.
- graph: 4-connected grid
- Each output pixel depends on
  - local grayvalue (inputs)
  - neighboring outputs



Energy function components ("Ising" model):

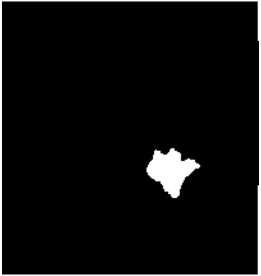
- $E_i(y_i = 1, x_i) = 1 - \frac{1}{255}x_i$ $\quad E_i(y_i = 0, x_i) = \frac{1}{255}x_i$
  x_i bright $\rightarrow$ y_i rather foreground, x_i dark $\rightarrow$ y_i rather background
- $E_{ij}(0, 0) = E_{ij}(1, 1) = 0$, $\quad E_{ij}(0, 1) = E_{ij}(1, 0) = \omega$ for $\omega > 0$
  prefer that neighbors have the same label $\rightarrow$ labeling *smooth*

$$E(y;x) = \sum_i \Big((1 - \frac{1}{255} x_i) [\![y_i = 1]\!] + \frac{1}{255} x_i [\![y_i = 0]\!] \Big) + \sum_{i \sim j} w [\![y_i \neq y_j]\!]$$



input image

segmentation from thresholding



segmentation from minimal energy

What to do with Structured Prediction Models?

Case 1) $p(y|x)$ **is known**

MAP Prediction

Predict $f : \mathcal{X} \to \mathcal{Y}$ by solving

$$y^* = \operatorname*{argmax}_{y \in \mathcal{Y}} \; p(y|y) \\ = \operatorname*{argmin}_{y \in \mathcal{Y}} \; E(y, x)$$

Probabilistic Inference

Compute *marginal probabilities*

$$p(y_F|x)$$

for any factor F, in particular, $p(y_i|x)$ for all $i \in V$.

What to do with Structured Prediction Models?

Case 2) $p(y|x)$ **is unknown, but we have training data**

Structure Learning

Learn graph structure from training data.

Variable Learning

Learn, whether to use additional (latent) variables, and which ones.
(input and output variables are fixed by the task we try to solve).

Parameter Learning

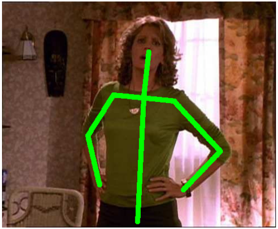
Assume fixed graph structure, **learn potentials/energies**.

Probabilistic Inference

Compute $p(y_F|x)$ and $Z(x)$.

Example: Pictorial Structures


input image

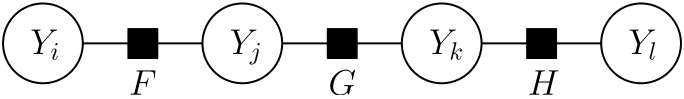

$\mathrm{argmax}_y \, p(y|x)$


$p(y_i|x)$

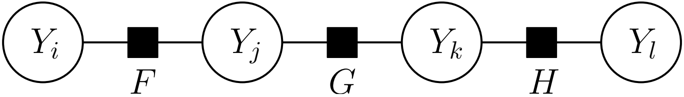
- MAP makes a single (structured) prediction (point estimate)
  - best overall pose
- Marginal probabilities $p(y_i|x)$ give us
  - potential positions
  - uncertainty

  of the individual body parts.

Assume $y = (y_i, y_j, y_k, y_l)$, $\mathcal{Y} = \mathcal{Y}_i \times \mathcal{Y}_j \times \mathcal{Y}_k \times \mathcal{Y}_l$, and an energy function $E(y; x)$ compatible with the following factor graph:



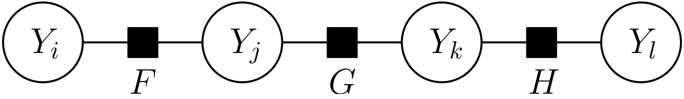
Assume $y = (y_i, y_j, y_k, y_l)$, $\mathcal{Y} = \mathcal{Y}_i \times \mathcal{Y}_j \times \mathcal{Y}_k \times \mathcal{Y}_l$, and an energy function $E(y; x)$ compatible with the following factor graph:



Task 1: for any $y \in \mathcal{Y}$, compute $p(y|x)$, using

$$p(y|x) = \frac{1}{Z(x)} \exp(-E(y; x)).$$

Assume $y = (y_i, y_j, y_k, y_l)$, $\mathcal{Y} = \mathcal{Y}_i \times \mathcal{Y}_j \times \mathcal{Y}_k \times \mathcal{Y}_l$, and an energy function $E(y; x)$ compatible with the following factor graph:



Task 1: for any $y \in \mathcal{Y}$, compute $p(y|x)$, using

$$p(y|x) = \frac{1}{Z(x)} \exp(-E(y; x)).$$

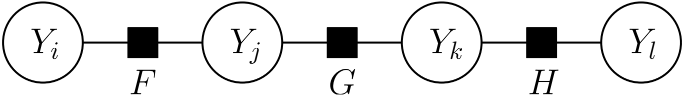
Problem: We don't know $Z(x)$, and computing it using

$$Z(x) = \sum_{y \in \mathcal{Y}} \exp(-E(y; x))$$

looks expensive (the sum has $|\mathcal{Y}_i| \cdot |\mathcal{Y}_j| \cdot |\mathcal{Y}_k| \cdot |\mathcal{Y}_l|$ terms).

A lot research has been done on how to **efficiently compute** $Z(x)$.

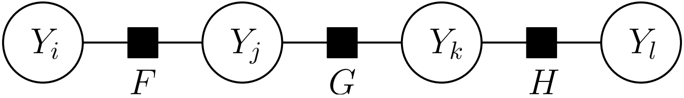
Probabilistic Inference – Belief Propagation / Message Passing



For notational simplicity, we drop the dependence on (fixed) x:

$$Z = \sum_{y \in \mathcal{Y}} \exp(-E(y))$$

Probabilistic Inference – Belief Propagation / Message Passing

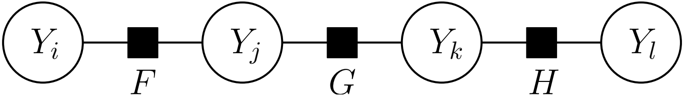


For notational simplicity, we drop the dependence on (fixed) x:

$$Z = \sum_{y \in \mathcal{Y}} \exp(-E(y))$$

$$= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-E(y_i, y_j, y_k, y_l))$$

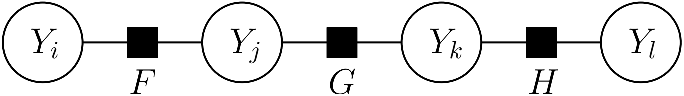
Probabilistic Inference – Belief Propagation / Message Passing



For notational simplicity, we drop the dependence on (fixed) x:

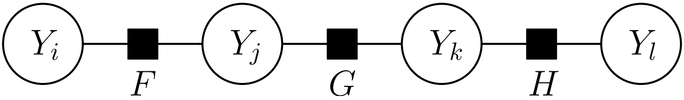
$$
\begin{aligned}
Z &= \sum_{y \in \mathcal{Y}} \exp(-E(y)) \\
&= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-E(y_i, y_j, y_k, y_l)) \\
&= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-(E_F(y_i, y_j) + E_G(y_j, y_k) + E_H(y_k, y_l)))
\end{aligned}
$$

Probabilistic Inference – Belief Propagation / Message Passing



$$Z = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-(E_F(y_i, y_j) + E_G(y_j, y_k) + E_H(y_k, y_l)))$$

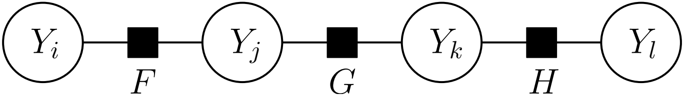
Probabilistic Inference – Belief Propagation / Message Passing



$$Z = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-(E_F(y_i, y_j) + E_G(y_j, y_k) + E_H(y_k, y_l)))$$

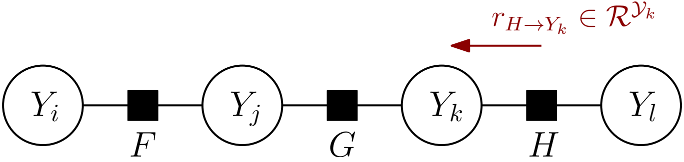
$$= \sum_{y_i} \sum_{y_j} \sum_{y_k} \sum_{y_l} \exp(-E_F(y_i, y_j)) \exp(-E_G(y_j, y_k)) \exp(-E_H(y_k, y_l))$$

Probabilistic Inference – Belief Propagation / Message Passing



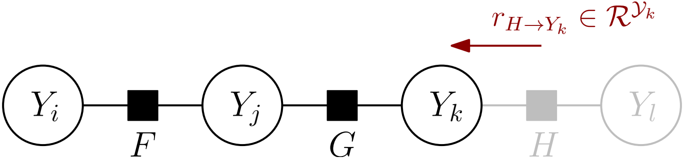
$$
\begin{aligned}
Z =& \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-(E_F(y_i, y_j) + E_G(y_j, y_k) + E_H(y_k, y_l))) \\
=& \sum_{y_i} \sum_{y_j} \sum_{y_k} \sum_{y_l} \exp(-E_F(y_i, y_j)) \exp(-E_G(y_j, y_k)) \exp(-E_H(y_k, y_l)) \\
=& \sum_{y_i} \sum_{y_j} \exp(-E_F(y_i, y_j)) \sum_{y_k} \exp(-E_G(y_j, y_k)) \sum_{y_l} \exp(-E_H(y_k, y_l))
\end{aligned}
$$

Probabilistic Inference – Belief Propagation / Message Passing



$$Z = \sum_{y_i} \sum_{y_j} \exp(-E_F(y_i, y_j)) \sum_{y_k} \exp(-E_G(y_j, y_k)) \underbrace{\sum_{y_l} \exp(-E_H(y_k, y_l))}_{r_{H\to Y_k}(y_k)}$$

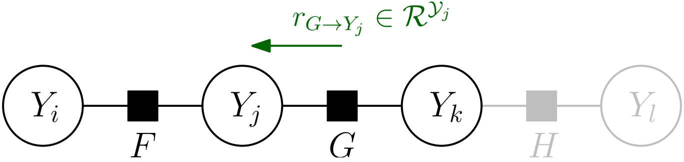
Probabilistic Inference – Belief Propagation / Message Passing



$$Z = \sum_{y_i}\sum_{y_j} \exp(-E_F(y_i,y_j)) \sum_{y_k} \exp(-E_G(y_j,y_k)) \underbrace{\sum_{y_l} \exp(-E_H(y_k,y_l))}_{r_{H\to Y_k}(y_k)}$$

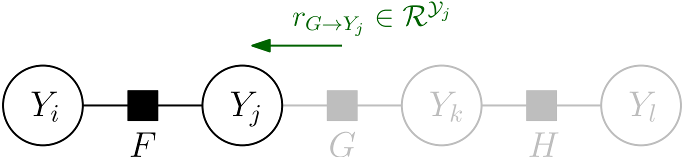
$$= \sum_{y_i}\sum_{y_j} \exp(-E_F(y_i,y_j)) \sum_{y_k} \exp(-E_G(y_j,y_k)) r_{H\to Y_k}(y_k)$$

Probabilistic Inference – Belief Propagation / Message Passing



$$Z = \sum_{y_i} \sum_{y_j} \exp(-E_F(y_i, y_j)) \underbrace{\sum_{y_k} \exp(-E_G(y_j, y_k)) r_{H\to Y_k}(y_k)}_{r_{G\to Y_j}(y_j)}$$

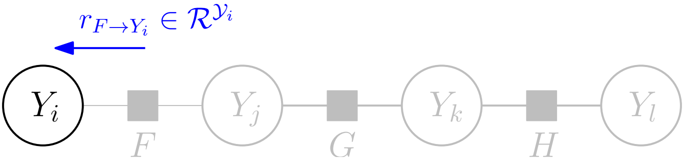
Probabilistic Inference – Belief Propagation / Message Passing



$$Z = \sum_{y_i} \sum_{y_j} \exp(-E_F(y_i, y_j)) \underbrace{\sum_{y_k} \exp(-E_G(y_j, y_k)) r_{H\to Y_k}(y_k)}_{r_{G\to Y_j}(y_j)}$$

$$= \sum_{y_i} \sum_{y_j} \exp(-E_F(y_i, y_j)) r_{G\to Y_j}(y_j)$$

Probabilistic Inference – Belief Propagation / Message Passing

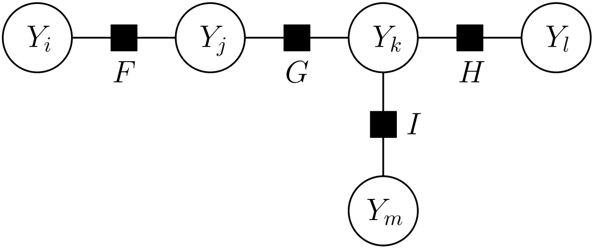


$$Z = \sum_{y_i} \sum_{y_j} \exp(-E_F(y_i, y_j)) \underbrace{\sum_{y_k} \exp(-E_G(y_j, y_k)) r_{H\to Y_k}(y_k)}_{r_{G\to Y_j}(y_j)}$$

$$= \sum_{y_i} \sum_{y_j} \exp(-E_F(y_i, y_j)) r_{G\to Y_j}(y_j)$$

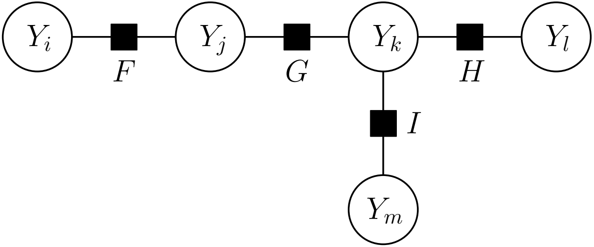
$$= \sum_{y_i} r_{F\to Y_i}(y_i)$$

Example: Inference on Trees



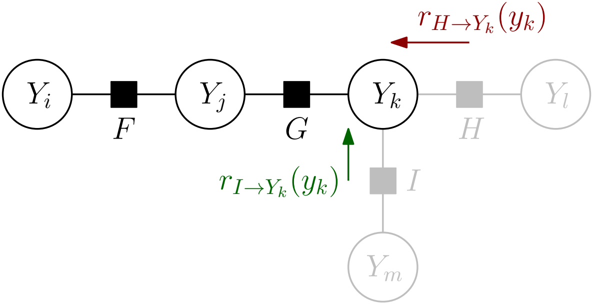
$$
\begin{aligned}
Z &= \sum_{y \in \mathcal{Y}} \exp(-E(y)) \\
&= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_i} \sum_{y_k \in \mathcal{Y}_i} \sum_{y_l \in \mathcal{Y}_i} \sum_{y_m \in \mathcal{Y}_m} \exp(-(E_F(y_i, y_j) + \cdots + E_I(y_k, y_m)))
\end{aligned}
$$

Example: Inference on Trees



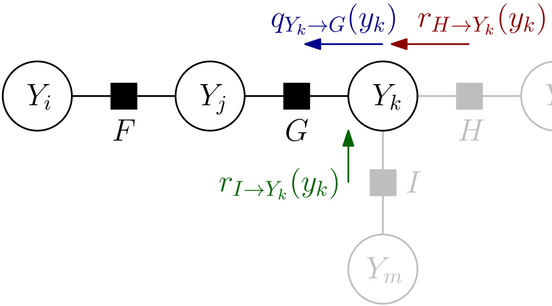
$$Z = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_F(y_i, y_j)) \sum_{y_k \in \mathcal{Y}_k} \exp(-E_G(y_j, y_k)) \cdot \left(\underbrace{\left(\sum_{y_l \in \mathcal{Y}_l} \exp(-E_H(y_k, y_l)) \right)}_{r_{H \to Y_k}(y_k)} \cdot \underbrace{\left(\sum_{y_m \in \mathcal{Y}_m} \exp(-E_I(y_k, y_m)) \right)}_{r_{I \to Y_k}(y_k)} \right)$$

Example: Inference on Trees



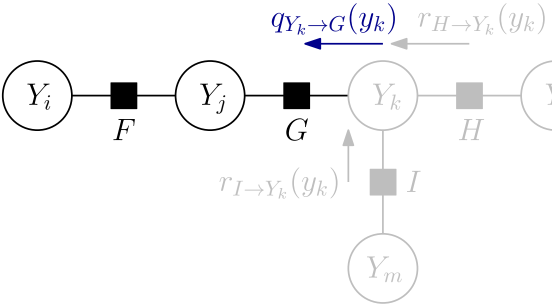
$$Z = \sum_{y_i\in\mathcal{Y}_i}\sum_{y_j\in\mathcal{Y}_j}\exp(-E_F(y_i,y_j))\sum_{y_k\in\mathcal{Y}_k}\exp(-E_G(y_j,y_k))\cdot \left(r_{H\to Y_k}(y_k)\cdot r_{I\to Y_k}(y_k)\right)$$

Example: Inference on Trees



$$Z = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_F(y_i, y_j)) \sum_{y_k \in \mathcal{Y}_k} \exp(-E_G(y_j, y_k)) \cdot \underbrace{\left(r_{H \to Y_k}(y_k) \cdot r_{I \to Y_k}(y_k)\right)}_{q_{Y_k \to G}(y_k)}$$

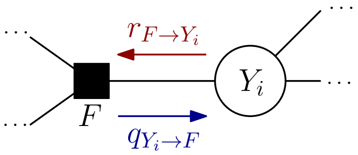
Example: Inference on Trees



$$Z = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_F(y_i, y_j)) \sum_{y_k \in \mathcal{Y}_k} \exp(-E_G(y_j, y_k)) q_{Y_k \to G}(y_k)$$

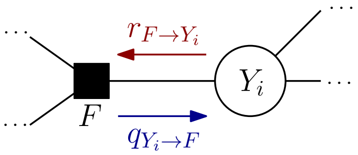
Factor Graph Sum-Product Algorithm

- "Message": pair of vectors at each factor graph edge $(i, F) \in \mathcal{E}$
  1. $r_{F \to Y_i} \in \mathbb{R}^{\mathcal{Y}_i}$: factor-to-variable message
  2. $q_{Y_i \to F} \in \mathbb{R}^{\mathcal{Y}_i}$: variable-to-factor message



Factor Graph Sum-Product Algorithm

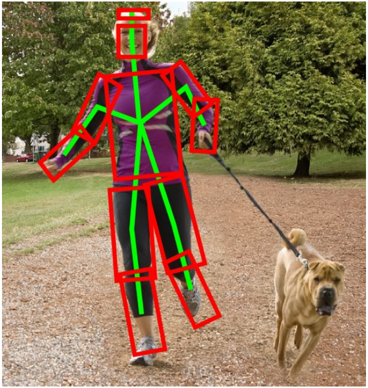
- "Message": pair of vectors at each factor graph edge $(i, F) \in \mathcal{E}$
    1. $r_{F \to Y_i} \in \mathbb{R}^{\mathcal{Y}_i}$: factor-to-variable message
    2. $q_{Y_i \to F} \in \mathbb{R}^{\mathcal{Y}_i}$: variable-to-factor message

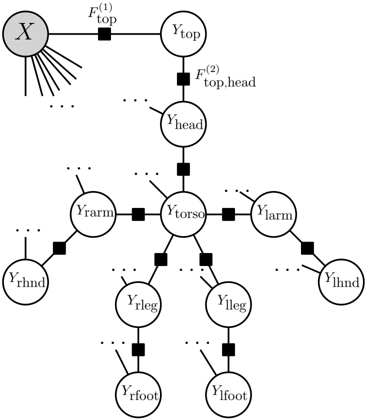


- Algorithm iteratively update messages
- After convergence: Z and $p(y_F)$ can be obtained from the messages.

Belief Propagation

Example: Pictorial Structures





- Tree-structured model for articulated pose (Felzenszwalb and Huttenlocher, 2000), (Fischler and Elschlager, 1973)
- Body-part variables, states: discretized tuple (x, y, s, θ)
- (x, y) position, s scale, and θ rotation

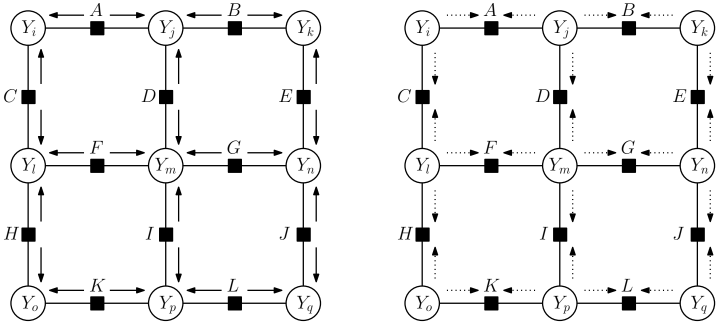
Example: Pictorial Structures

- Marginal probabilities $p(y_i|x)$ give us
  - potential positions
  - uncertainty

of the body parts.

Belief Propagation in Loopy Graphs

Can we do *message passing* also in graphs with loops?

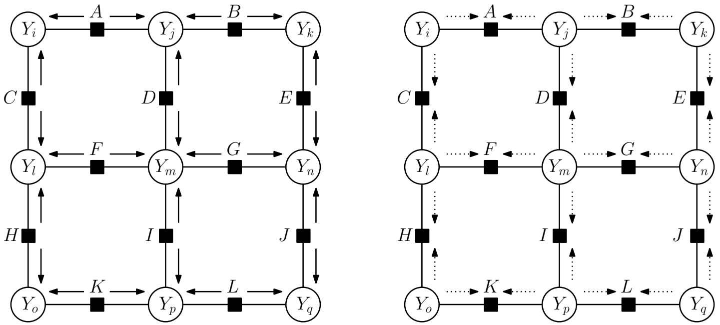


Problem: There is no well-define *leaf–to–root* order.

Suggested solution: Loopy Belief Propagation (LBP)

- initialize all messages as constant 1
- pass messages until convergence

Belief Propagation in Loopy Graphs



Loopy Belief Propagation is very popular, but has some problems:

- it might not converge (e.g. oscillate)
- even if it does, the computed probabilities are only *approximate*.

Many improved message-passing schemes exist (see tutorial book).

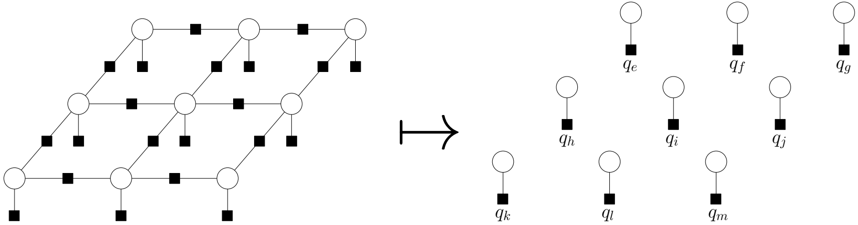
Probabilistic Inference – Variational Inference / Mean Field

Task: Compute marginals $p(y_F|x)$ for general $p(y|x)$

Idea: Approximate $p(y|x)$ by simpler $q(y)$ and use marginals from that.

$$q^* = \operatorname*{argmin}_{q \in \mathcal{Q}} D_{KL}(q(y) \| p(y|x))$$

E.g. **Naive Mean Field**: $\mathcal{Q}$ all distributions of the form $q(y) = \prod_{i \in V} q_i(y_i)$.



Probabilistic Inference – Sampling / Markov-Chain Monte Carlo

Task: Compute marginals $p(y_F|x)$ for general $p(y|x)$

Idea: Rephrase as computing the *expected value of a quantity*:

$$\mathbb{E}_{y\sim p(y|x,w)}[h(x,y)],$$

for some (well-behaved) function $h : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$.

For probabilistic inference, this step is easy. Set

$$h_{F,z}(x,y) := [\![y_F = z]\!],$$

then

$$\boxed{\mathbb{E}_{y\sim p(y|x,w)}[h_{F,z}(x,y)]} = \sum_{y\in\mathcal{Y}} p(y|x)[\![y_F = z]\!]$$
$$= \sum_{y_F\in\mathcal{Y}_F} p(y_F|x)[\![y_F = z]\!] = \boxed{p(y_F = z|x)}.$$

Probabilistic Inference – Sampling / Markov-Chain Monte Carlo

Expectations can be computed/approximated by **sampling**:

- For fixed x, let $y^{(1)}, y^{(2)}, \dots$ be i.i.d. samples from $p(y|x)$, then

$$\mathbb{E}_{y \sim p(y|x)}[h(x,y)] \approx \frac{1}{S}\sum_{s=1}^{S} h(x, y^{(s)}).$$

- The *law of large numbers* guarantees convergence for $S \to \infty$,
- For S independent samples, approximation error is $O(1/\sqrt{S})$, *independent* of the dimension of $\mathcal{Y}$.

Probabilistic Inference – Sampling / Markov-Chain Monte Carlo

Expectations can be computed/approximated by **sampling**:

- For fixed x, let $y^{(1)}, y^{(2)}, \dots$ be i.i.d. samples from $p(y|x)$, then

$$\mathbb{E}_{y\sim p(y|x)}[h(x,y)] \approx \frac{1}{S}\sum_{s=1}^{S} h(x,y^{(s)}).$$

- The *law of large numbers* guarantees convergence for $S \to \infty$,
- For S independent samples, approximation error is $O(1/\sqrt{S})$, *independent* of the dimension of $\mathcal{Y}$.

Problem:

- Producing i.i.d. samples, $y^{(s)}$, from $p(y|x)$ is *hard*.

Solution:

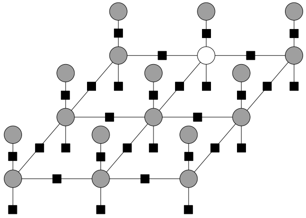
- We can get away with a sequence of *dependent* samples
  $\rightarrow$ Monte-Carlo Markov Chain (MCMC) sampling

Probabilistic Inference – Sampling / Markov-Chain Monte Carlo

One example how to do MCMC sampling: **Gibbs sampler**

- Initialize $y^{(0)} = (y_1, \dots, y_d)$ arbitrarily
- For $s = 1, \dots, S$:
  1. Select a variable y_i,
  2. Re-sample $y_i \sim p(y_i | y^{(s-1)}_{V\setminus\{i\}}, x)$.
  3. Output sample $y^{(s)} = (y_1^{(s-1)}, \dots, y_{i-1}^{(s-1)},\ y_i,\ y_{i+1}^{(s-1)}, \dots, y_d^{(s-1)})$

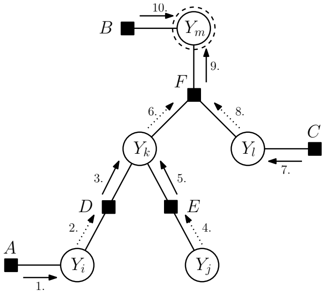
$$p(y_i | y^{(s)}_{V\setminus\{i\}}, x) = \frac{p(y_i, y^{(t)}_{V\setminus\{i\}} | x)}{\sum_{y_i \in \mathcal{Y}_i} p(y_i, y^{(t)}_{V\setminus\{i\}} | x)} = \frac{\exp(-E(y_i, y^{(t)}, x)}{\sum_{y_i \in \mathcal{Y}_i} \exp(-E(y_i, y^{(t)}, x)}$$

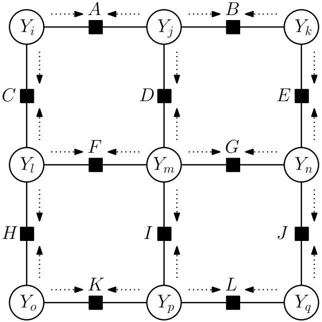


MAP Prediction

Compute $y^* = \operatorname{argmax}_y p(y|x)$.

MAP Prediction – Belief Propagation / Message Passing





One can also derive message passing algorithms for MAP prediction.

- In trees: guaranteed to converge to optimal solution.
- In loopy graphs: convergence not guaranteed, approximate solution.

MAP Prediction – Graph Cuts

For loopy graph, we can find the global optimum only in **special cases**:

- Binary output variables: $\mathcal{Y}_i = \{0, 1\}$ for $i = 1, \dots, d$,
- Energy function with only unary and pairwise terms

$$E(y; x, w) = \sum_i E_i(y_i; x) + \sum_{i \sim j} E_{i,j}(y_i, y_j; x)$$

MAP Prediction – Graph Cuts

For loopy graph, we can find the global optimum only in **special cases**:

- Binary output variables: $\mathcal{Y}_i = \{0,1\}$ for $i = 1, \dots, d$,
- Energy function with only unary and pairwise terms

$$E(y; x, w) = \sum_i E_i(y_i; x) + \sum_{i \sim j} E_{i,j}(y_i, y_j; x)$$

- Restriction 1 (positive unary potentials):

$$E_F(y_i; x, w_{t_F}) \geq 0 \quad \textit{(always achievable by reparametrization)}$$

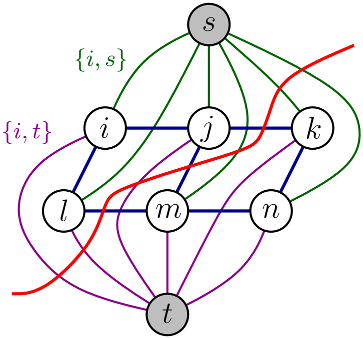
- Restriction 2 (regular/submodular/attractive pairwise potentials)

$$\begin{aligned} E_F(y_i, y_j; x, w_{t_F}) &= 0, \qquad \text{if } y_i = y_j, \\ E_F(y_i, y_j; x, w_{t_F}) &= E_F(y_j, y_i; x, w_{t_F}) \geq 0, \qquad \text{otherwise.} \end{aligned}$$

(not always achievable, depends on the task)

- Construct auxiliary undirected graph
- One node $\{i\}_{i \in V}$ per variable
- Two extra nodes: source s, sink t
- Edges

Edge	Graph cut weight
$\{i, j\}$	$E_F(y_i = 0, y_j = 1; x, w_{t_F})$
$\{i, s\}$	$E_F(y_i = 1; x, w_{t_F})$
$\{i, t\}$	$E_F(y_i = 0; x, w_{t_F})$



- Find linear s-t-mincut
  - Solution defines optimal binary labeling of the original energy minimization problem

GraphCuts algorithms

(Approximate) multi-class extensions exist, see tutorial book.

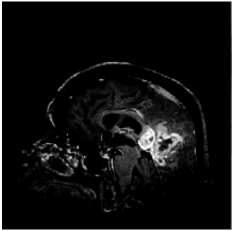
GraphCuts Example

Image segmentation energy:

$$E(y;x) = \sum_i \Big((1 - \frac{1}{255} x_i) [\![y_i = 1]\!] + \frac{1}{255} x_i [\![y_i = 0]\!] \Big) + \sum_{i \sim j} w [\![y_i \neq y_j]\!]$$

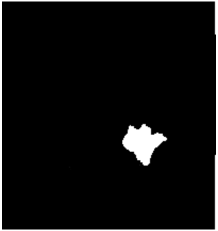
All conditions to apply GraphCuts are fulfilled.

- $E_i(y_i, x) \geq 0$,
- $E_{ij}(y_i, y_j) = 0 \quad$ for $y_i = y_j$,
- $E_{ij}(y_i, y_j) = w > 0 \quad$ for $y_i \neq y_j$.



input image

thresholding



GraphCuts

MAP Prediction – Linear Programming Relaxation

More general alternative, $\mathcal{Y}_i = \{1, \dots, K\}$:

$$E(y; x) = \sum_i E_i(y_i; x) + \sum_{ij} E_{ij}(y_i, y_j; x)$$

Linearize the energy using indicator functions:

$$E_i(y_i; x) = \sum_{k=1}^{K} \underbrace{E_i(k; x)}_{=: a_{ik}} [\![y_i = k]\!] = \sum_{k=1}^{K} a_{i;k} \mu_{i;k}$$

for new variables $\mu_{i;k} \in \{0, 1\}$ with $\sum_k \mu_{i;k} = 1$.

$$E_{ij}(y_i, y_j; x) = \sum_{k=1}^{K} \sum_{l=1}^{K} \underbrace{E_i(k; x)}_{=: a_{ij;kl}} [\![y_i = k \wedge y_j = l]\!] = \sum_{k=1}^{K} a_{ij;kl} \mu_{ij;kl}$$

for new variables $\mu_{ij;kl} \in \{0, 1\}$ with $\sum_l \mu_{ij;kl} = \mu_{i;k}$ and $\sum_k \mu_{ij;kl} = \mu_{j;l}$.

MAP Prediction – Linear Programming Relaxation

Energy minimization becomes

$$y^* \leftarrow \mu^* := \operatorname*{argmin}_{\mu} \sum_i a_{i;k}\mu_{i;k} + \sum_{ij} a_{ij;kl}\mu_{ij;kl} = \operatorname*{argmin}_{\mu} \boldsymbol{A}\boldsymbol{\mu}$$

subject to

$$\mu_{i;k} \in \{0,1\} \qquad \mu_{ij;kl} \in \{0,1\}$$

$$\sum_k \mu_{i;k} = 1, \qquad \sum_l \mu_{ij;kl} = \mu_{i;k}, \qquad \sum_k \mu_{ij;kl} = \mu_{j;l}$$

Integer variables, linear objective function, linear constraints:

Integer linear program (ILP)

Unfortunately, ILPs are –in general– NP-hard.

MAP Prediction – Linear Programming Relaxation

Energy minimization becomes

$$y^* \leftarrow \mu^* := \operatorname*{argmin}_{\mu} \sum_i a_{i;k}\mu_{i;k} + \sum_{ij} a_{ij;kl}\mu_{ij;kl} = \operatorname*{argmin}_{\mu} \boldsymbol{A\mu}$$

subject to

$$\mu_{i;k} \in [0,1]\cancel{\{0,1\}} \qquad \mu_{ij;kl} \in [0,1]\cancel{\{0,1\}}$$

$$\sum_k \mu_{i;k} = 1, \qquad \sum_l \mu_{ij;kl} = \mu_{i;k}, \qquad \sum_k \mu_{ij;kl} = \mu_{j;l}$$

~~Integer~~ real-values variables, linear objective function, linear constraints:

Linear program (LP) relaxation

LPs can be solved very efficiently, μ^* yields approximate solution for y^*.

MAP Prediction – Custom solutions: E.g. branch-and-bound

Note: we just try to solve an optimization problem

$$y^* = \operatorname*{argmin}_{y \in \mathcal{Y}} E(y; x)$$

We can use any optimization technique that fits the problem.

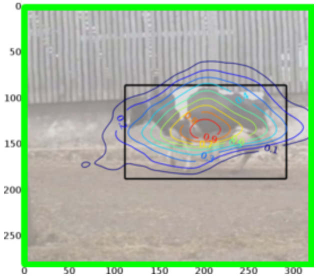
MAP Prediction – Custom solutions: E.g. branch-and-bound

Note: we just try to solve an optimization problem

$$y^* = \operatorname*{argmin}_{y \in \mathcal{Y}} E(y; x)$$

We can use any optimization technique that fits the problem.

For low-dimensional $\mathcal{Y}$, such as bounding boxes: **branch-and-bound**:



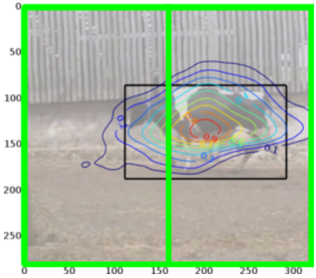
MAP Prediction – Custom solutions: E.g. branch-and-bound

Note: we just try to solve an optimization problem

$$y^* = \operatorname*{argmin}_{y \in \mathcal{Y}} E(y; x)$$

We can use any optimization technique that fits the problem.

For low-dimensional $\mathcal{Y}$, such as bounding boxes: **branch-and-bound**:



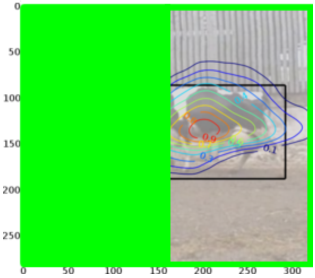
MAP Prediction – Custom solutions: E.g. branch-and-bound

Note: we just try to solve an optimization problem

$$y^* = \operatorname*{argmin}_{y \in \mathcal{Y}} E(y; x)$$

We can use any optimization technique that fits the problem.

For low-dimensional $\mathcal{Y}$, such as bounding boxes: **branch-and-bound**:



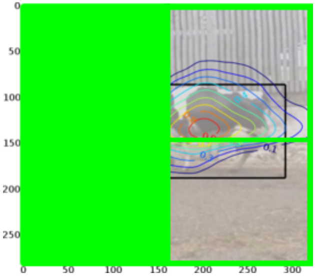
MAP Prediction – Custom solutions: E.g. branch-and-bound

Note: we just try to solve an optimization problem

$$y^* = \operatorname*{argmin}_{y \in \mathcal{Y}} E(y; x)$$

We can use any optimization technique that fits the problem.

For low-dimensional $\mathcal{Y}$, such as bounding boxes: **branch-and-bound**:



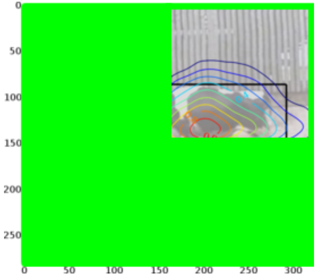
MAP Prediction – Custom solutions: E.g. branch-and-bound

Note: we just try to solve an optimization problem

$$y^* = \operatorname*{argmin}_{y \in \mathcal{Y}} E(y; x)$$

We can use any optimization technique that fits the problem.

For low-dimensional $\mathcal{Y}$, such as bounding boxes: **branch-and-bound**:



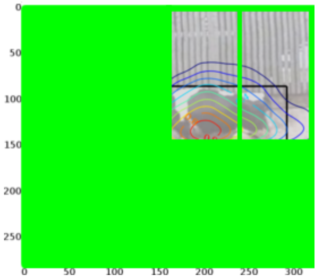
MAP Prediction – Custom solutions: E.g. branch-and-bound

Note: we just try to solve an optimization problem

$$y^* = \operatorname*{argmin}_{y \in \mathcal{Y}} E(y; x)$$

We can use any optimization technique that fits the problem.

For low-dimensional $\mathcal{Y}$, such as bounding boxes: **branch-and-bound**:



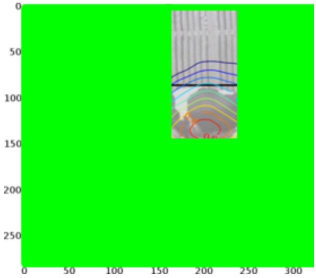
MAP Prediction – Custom solutions: E.g. branch-and-bound

Note: we just try to solve an optimization problem

$$y^* = \operatorname*{argmin}_{y \in \mathcal{Y}} E(y; x)$$

We can use any optimization technique that fits the problem.

For low-dimensional $\mathcal{Y}$, such as bounding boxes: **branch-and-bound**:



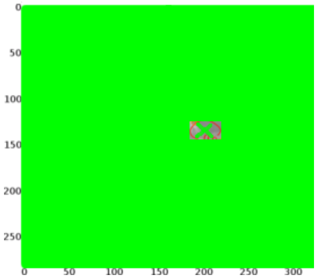
MAP Prediction – Custom solutions: E.g. branch-and-bound

Note: we just try to solve an optimization problem

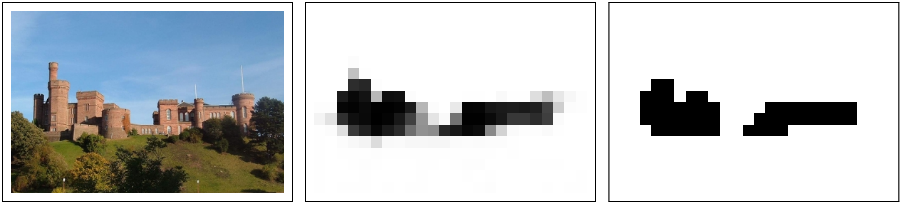
$$y^* = \operatorname*{argmin}_{y \in \mathcal{Y}} E(y; x)$$

We can use any optimization technique that fits the problem.

For low-dimensional $\mathcal{Y}$, such as bounding boxes: **branch-and-bound**:



Example: Man-made structure detection



- Left: input image x,
- Middle (probabilistic inference): visualization of the variable marginals $p(y_i = '' \textit{manmade}'' | x, w)$,
- Right (MAP inference): joint MAP labeling $y^* = \text{argmax}_{y \in \mathcal{Y}}\, p(y|x, w)$.

Loss function

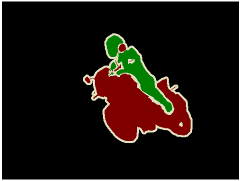
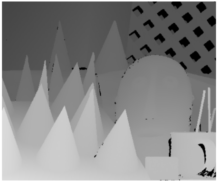
How to judge if a prediction is good?

- Define a *loss function*

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

$\Delta(y', y)$ measures the loss incurred by predicting y when y' is correct.

- The *loss function* is application dependent


Example 1: 0/1 loss

Loss is 0 for perfect prediction, 1 otherwise:

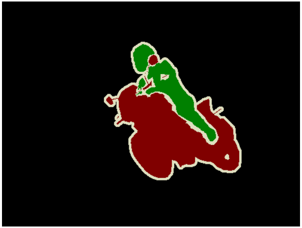
$$\Delta_{0/1}(y', y) = [\![y' \neq y]\!] = \begin{cases} 0 & \text{if } y' = y \\ 1 & \text{otherwise} \end{cases}$$

Every mistake is equally bad. Usually not very useful in *structured prediction*.

Example 2: Hamming loss

Count the number of mislabeled variables:

$$\Delta_H(y', y) = \frac{1}{|V|} \sum_{i \in V} I(y'_i \neq y_i)$$



Used, e.g., in image segmentation.

Example 3: Squared error

If we can add elements in $\mathcal{Y}_i$
(pixel intensities, optical flow vectors, etc.).

Sum of squared errors

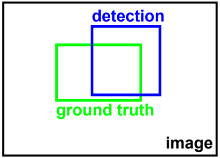
$$\Delta_Q(y', y) = \frac{1}{|V|} \sum_{i \in V} \|y'_i - y_i\|^2.$$

Used, e.g., in stereo reconstruction, part-based object detection.

Example 4: Task specific losses

Object detection

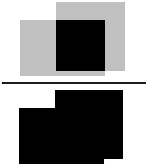
- bounding boxes, or
- arbitrary regions



Area overlap loss:

$$\Delta_{AO}(y', y) = 1 - \frac{\text{area}(y' \cap y)}{\text{area}(y' \cup y)}$$

$= 1 -$



Used, e.g., in PASCAL VOC challenges for object detection, because it scale-invariants (no bias for or against big objects).

Summary: Inference and Prediction

Two main tasks for a given probability distribution $p(y|x)$:

Probabilistic Inference

Compute $p(y_I|x)$ for a subset I of variables, in particular $p(y_i|x)$

- (Loopy) Belief Propagation, Variation Inference, Sampling, ...

MAP Prediciton

Identify $y^* \in \mathcal{Y}$ that maximizes $p(y|x)$ (minimizes energy)

- (Loopy) Belief Propagation, GraphCuts, LP-relaxation, custom, ...

The quality of a prediction is measured by a loss function, $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.

Loss Function

$\Delta(y', y)$ is loss (or cost) for predicting $y \in \mathcal{Y}$ if $y' \in \mathcal{Y}$ is correct.

- Task specific: use 0/1-loss, Hamming loss, area overlap, ...

Ad: PhD/PostDoc Positions at I.S.T. Austria, Vienna

I.S.T. Graduate School

- 1(2) + 3 yr PhD program
- full scholarship
- flexible starting dates

PostDoc Positions in my Group

- *computer vision*
  - object/attribute prediction
- *machine learning*
  - structured output learning
- curiosity driven basic research
  - no project deliverables/deadlines,
  - no teaching duties, ...

Internships: ask me!

More information: www.ist.ac.at or talk to me during a break

Part 2: Conditional Random Fields

What to do if $p(y|x)$ is unknown, but we have training data.

Assume that a probability distribution $d(x, y)$ *exists* that describes the relation between x and y, but we don't know it.

Approach 1) Probabilistic Parameter Estimation

1) Use training data to obtain an estimate $p(y|x)$ for $d(y|x)$.
2) Use $p(y|x)$ to make predictions.

Approach 2) Loss-minimizing Parameter Estimation

1) Use training data to learn an energy function $E(y, x)$ that results in "good" (low loss) predictions.
2) Use $E(y, x)$ to make predictions.

Problem (Probabilistic Learning)

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

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

or

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

Problem (Probabilistic Learning)

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

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

or

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

Open questions:

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

Conditional Random Field Learning

Assume:

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

Task:

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

Many possible technique to do so:

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

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

Maximum Likelihood Parameter Estimation

Idea: maximize conditional likelihood of observing outputs $y^1, \dots, y^N$ for inputs $x^1, \dots, x^N$

$$w^* = \operatorname*{argmax}_{w \in \mathbb{R}^D} \; p(y^1, \dots, y^N | x^1, \dots, x^N, w)$$

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

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

MAP Estimation of w

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

MAP Estimation of w

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

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

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

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

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

Choices for $p(w)$:

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

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

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

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

Probabilistic Models for Structured Prediction - Summary

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

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

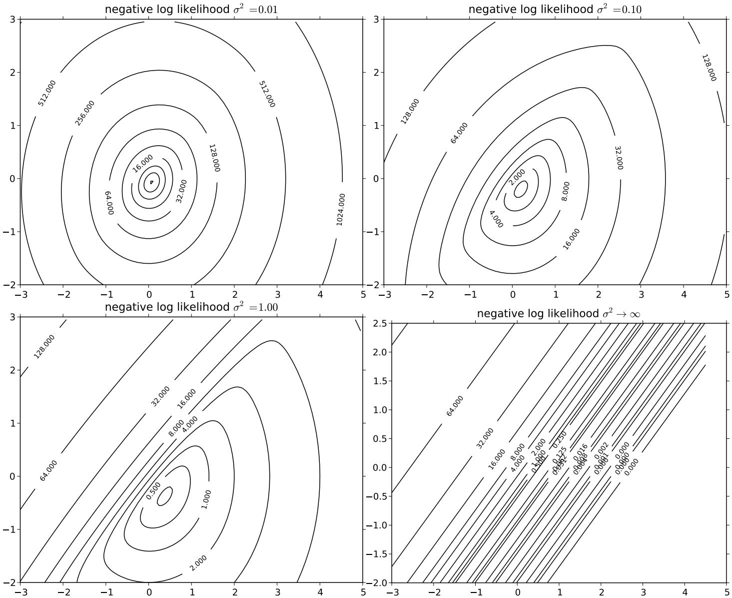
($\sigma^2 \to \infty$ makes it *unregularized*)

Probabilistic parameter estimation or *training* means solving

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

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

Negative Conditional Log-Likelihood (Toy Example)



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

input tolerance $\epsilon > 0$

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

output w_{cur}

Alternatives:

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

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

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

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

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

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

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

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

- For $\sigma \to \infty$:

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

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

Interpretation:

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

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

but discriminatively: only for $x \in \{x^1, \ldots, x^n\}$.

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

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

Milestone I: Probabilistic Training (Conditional Random Fields)

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

Milestone I: Probabilistic Training (Conditional Random Fields)

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

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

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

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

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

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

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

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

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

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

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

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

Solving the Training Optimization Problem Numerically

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

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

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

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 ≈ 100s to 1,000,000s
- K: number of possible labels of each output nodes ≈ 2 to 100s

Probabilistic Inference to the Rescue

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

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

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

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

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

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

Solving the Training Optimization Problem Numerically

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

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

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

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

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

Solving the Training Optimization Problem Numerically

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

Computing the Gradient: $\cancel{O(K^M nd)}$, $O(MK^{|F_{max}|} N D)$:

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

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

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

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

Stochastic Gradient Descent (SGD)

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

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

---

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

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

Stochastic Gradient Descent (SGD)

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

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

- Avoid *line search* by using fixed stepsize rule η (new parameter)

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

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

Stochastic Gradient Descent (SGD)

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

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

- Avoid *line search* by using fixed stepsize rule η (new parameter)
- SGD converges to $\operatorname{argmin}_w \mathcal{L}(w)$! (if η chosen right)

---

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

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

Stochastic Gradient Descent (SGD)

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

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

- Avoid *line search* by using fixed stepsize rule η (new parameter)
- SGD converges to $\operatorname{argmin}_w \mathcal{L}(w)$! (if η chosen right)
- SGD needs more iterations, but each one is much faster

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

Solving the Training Optimization Problem Numerically

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

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

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

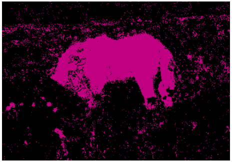
Line Search: $\cancel{O(K^M nd)}$, $O(MK^2ND)$ per evaluation of $\mathcal{L}$

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

Typical feature functions in **image segmentation:**

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

original



local classification

local + smoothness

Typical feature functions in **pose estimation:**

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

original

local classification

local + geometry

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

Solving the Training Optimization Problem Numerically

Idea: split learning of unary potentials into two parts:

- local classifiers,
- their importance.

Two-Stage Training

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

Solving the Training Optimization Problem Numerically

Idea: split learning of unary potentials into two parts:

- local classifiers,
- their importance.

Two-Stage Training

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

Advantage:

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

Disadvantage:

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

Solving the Training Optimization Problem Numerically

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

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

A lot of research on accelerating CRF training:

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

Summary – CRF Learning

Given:

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

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

Summary – CRF Learning

Given:

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

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

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

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

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

Part 3: Structured Support Vector Machines

Problem (Loss-Minimizing Parameter Learning)

Let $d(x, y)$ be the (unknown) true data distribution.
Let $\mathcal{D} = \{(x^1, y^1), \dots, (x^N, y^N)\}$ be i.i.d. samples from $d(x, y)$.
Let $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ be a feature function.
Let $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function.

- *Find a weight vector w^* that leads to minimal expected loss*

$$\mathbb{E}_{(x,y)\sim d(x,y)}\{\Delta(y, f(x))\}$$

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

Problem (Loss-Minimizing Parameter Learning)

Let $d(x, y)$ be the (unknown) true data distribution.
Let $\mathcal{D} = \{(x^1, y^1), \ldots, (x^N, y^N)\}$ be i.i.d. samples from $d(x, y)$.
Let $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ be a feature function.
Let $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function.

- *Find a weight vector w^* that leads to minimal expected loss*

$$\mathbb{E}_{(x,y)\sim d(x,y)}\{\Delta(y, f(x))\}$$

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

Pro:

- We directly optimize for the quantity of interest: expected loss.
- No expensive-to-compute partition function Z will show up.

Con:

- We need to know the loss function already at training time.
- We can't use probabilistic reasoning to find w^*.

Reminder: learning by regularized risk minimization

For compatibility function $g(x, y; w) := \langle w, \phi(x, y)\rangle$ find w^* that minimizes

$$\mathbb{E}_{(x,y)\sim d(x,y)}\ \Delta(\ y, \operatorname{argmax}_y g(x, y; w)\ ).$$

Two major problems:

- $d(x, y)$ is unknown
- $\operatorname{argmax}_y g(x, y; w)$ maps into a discrete space
  $\rightarrow \Delta(\ y, \operatorname{argmax}_y g(x, y; w))$ is discontinuous, piecewise constant

Task:

$$\min_{w} \quad \mathbb{E}_{(x,y)\sim d(x,y)}\, \Delta(\ y, \operatorname{argmax}_y g(x,y;w)\ ).$$

Problem 1:

- $d(x,y)$ is unknown

Solution:

- Replace $\mathbb{E}_{(x,y)\sim d(x,y)}\big(\,\cdot\,\big)$ with *empirical estimate* $\frac{1}{N}\sum_{(x^n,y^n)}\big(\,\cdot\,\big)$
- To avoid overfitting: add a *regularizer*, e.g. $\lambda\|w\|^2$.

New task:

$$\min_{w} \quad \lambda\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N}\Delta(\ y^n, \operatorname{argmax}_y g(x^n,y;w)\ ).$$

Task:

$$\min_{w} \quad \lambda\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \Delta(\ y^n, \mathrm{argmax}_y\, g(x^n, y; w)\ ).$$

Problem:

- $\Delta(\ y, \mathrm{argmax}_y\, g(x, y; w)\ )$ discontinuous w.r.t. w.

Solution:

- Replace $\Delta(y, y')$ with *well behaved* $\ell(x, y, w)$
- Typically: ℓ *upper bound* to Δ, *continuous* and *convex* w.r.t. w.

New task:

$$\min_{w} \quad \lambda\|w\|^2 + \frac{1}{N}\sum_{n=1}^{N} \ell(x^n, y^n, w))$$

Regularized Risk Minimization

$$\min_w \quad \lambda \|w\|^2 \quad + \quad \frac{1}{N}\sum_{n=1}^{N} \ell(x^n, y^n, w))$$

Regularization $+$ *Loss on training data*

Regularized Risk Minimization

$$\min_{w} \qquad \lambda\|w\|^2 \quad + \quad \frac{1}{N}\sum_{n=1}^{N}\ell(x^n, y^n, w))$$

Regularization + *Loss on training data*

Hinge loss: maximum margin training

$$\ell(x^n, y^n, w) := \max_{y\in\mathcal{Y}} \left[\; \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle \;\right]$$

Regularized Risk Minimization

$$\min_{w} \qquad \lambda\|w\|^2 \quad + \quad \frac{1}{N}\sum_{n=1}^{N} \ell(x^n, y^n, w))$$

Regularization + *Loss on training data*

Hinge loss: maximum margin training

$$\ell(x^n, y^n, w) := \max_{y\in\mathcal{Y}} \left[\ \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\ \right]$$

- ℓ is maximum over linear functions $\rightarrow$ *continuous*, *convex*.
- ℓ bounds Δ from above.
  Proof: Let $\bar{y} = \operatorname{argmax}_y g(x^n, y, w)$

$$\begin{aligned}\Delta(y^n, \bar{y}) &\leq \Delta(y^n, \bar{y}) + g(x^n, \bar{y}, w) - g(x^n, y^n, w)\\ &\leq \max_{y\in\mathcal{Y}} \left[\Delta(y^n, y) + g(x^n, y, w) - g(x^n, y^n, w)\right]\end{aligned}$$

Regularized Risk Minimization

$$\min_{w} \qquad \lambda\|w\|^2 \quad + \quad \frac{1}{N}\sum_{n=1}^{N} \ell(x^n, y^n, w))$$

Regularization + *Loss on training data*

Hinge loss: maximum margin training

$$\ell(x^n, y^n, w) := \max_{y\in\mathcal{Y}} \left[\; \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle \; \right]$$

Alternative:

Logistic loss: probabilistic training

$$\ell(x^n, y^n, w) := \log \sum_{y\in\mathcal{Y}} \exp\big(\langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\big)$$

Structured Output Support Vector Machine

$$\min_{w} \; \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N}\Big[\max_{y\in\mathcal{Y}} \; \Delta(y^n,y) + \langle w,\phi(x^n,y)\rangle - \langle w,\phi(x^n,y^n)\rangle\Big]$$

Conditional Random Field

$$\min_{w} \; \frac{\|w\|^2}{2\sigma^2} + \sum_{n=1}^{N}\Big[\log\sum_{y\in\mathcal{Y}}\exp\big(\langle w,\phi(x^n,y)\rangle - \langle w,\phi(x^n,y^n)\rangle\big)\Big]$$

CRFs and SSVMs have more in common than usually assumed.

- both do regularized risk minimization
- $\log\sum_y \exp(\cdot)$ can be interpreted as a *soft-max*

Solving the Training Optimization Problem Numerically

Structured Output Support Vector Machine:

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

Unconstrained optimization, convex, non-differentiable objective.

Structured Output SVM (equivalent formulation):

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

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

$$\max_{y\in\mathcal{Y}} \Big[\Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\Big] \leq \xi^n$$

N non-linear contraints, convex, differentiable objective.

Structured Output SVM (also equivalent formulation):

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

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

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

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

Example: Multiclass SVM

- $\mathcal{Y} = \{1, 2, \dots, K\}, \quad \Delta(y, y') = \begin{cases} 1 & \text{for } y \neq y' \\ 0 & \text{otherwise} \end{cases}.$
- $\phi(x, y) = \Big([\![y = 1]\!]\phi(x),\ [\![y = 2]\!]\phi(x),\ \dots,\ [\![y = K]\!]\phi(x) \Big)$

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

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

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

Classification: $f(x) = \operatorname{argmax}_{y \in \mathcal{Y}} \ \langle w, \phi(x, y)\rangle.$

Crammer-Singer Multiclass SVM

[K. Crammer, Y. Singer: "On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines", JMLR, 2001]

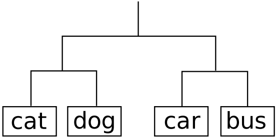
Example: Hierarchical SVM

Hierarchical Multiclass Loss:

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

$$\Delta(\texttt{cat}, \texttt{cat}) = 0, \quad \Delta(\texttt{cat}, \texttt{dog}) = 1,$$

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



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

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

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

[L. Cai, T. Hofmann: "Hierarchical Document Categorization with Support Vector Machines", ACM CIKM, 2004]
[A. Binder, K.-R. Müller, M. Kawanabe: "On taxonomies for multi-class image categorization", IJCV, 2011]

Solving the Training Optimization Problem Numerically

We can solve SSVM training like CRF training:

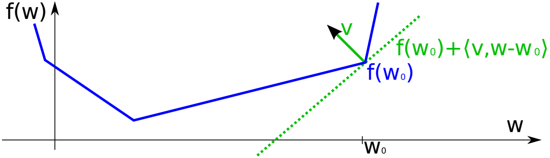
$$\min_{w} \; \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N}\Big[\max_{y\in\mathcal{Y}} \; \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle\Big]$$

- continuous ☺
- unconstrained ☺
- convex ☺
- non-differentiable ☹
  $\rightarrow$ we can't use gradient descent directly.
  $\rightarrow$ we'll have to use **subgradients**

Definition

Let $f : \mathbb{R}^D \to \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** of f at w_0, if

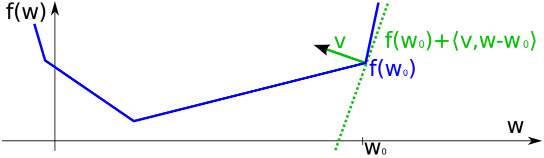
$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



Definition

Let $f : \mathbb{R}^D \to \mathbb{R}$ be a convex, not necessarily differentiable, function. A vector $v \in \mathbb{R}^D$ is called a **subgradient** of f at w_0, if

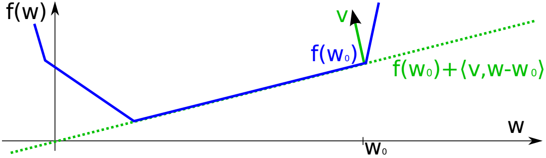
$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



Definition

Let $f : \mathbb{R}^D \to \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** of f at w_0, if

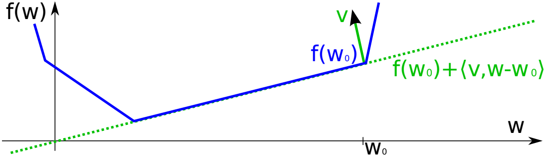
$$f(w) \geq f(w_0) + \langle v, w - w_0 \rangle \quad \text{for all } w.$$



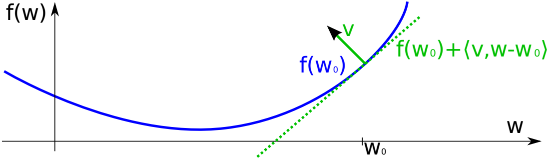
Definition

Let $f : \mathbb{R}^D \to \mathbb{R}$ be a convex, not necessarily differentiable, function.
A vector $v \in \mathbb{R}^D$ is called a **subgradient** 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.



Subgradient descent works basically like gradient descent:

Subgradient Descent Minimization – minimize $F(w)$

- **require:** tolerance $\epsilon > 0$, stepsizes η_t
- $w_{cur} \leftarrow 0$
- **repeat**
  - $v \in \nabla^{\text{sub}}{}_{w} F(w_{cur})$
  - $w_{cur} \leftarrow w_{cur} - \eta_t v$
- **until** F changed less than ϵ
- **return** w_{cur}

Converges to global minimum, but rather inefficient if F non-differentiable.

[Shor, "Minimization methods for non-differentiable functions", Springer, 1985.]

Computing a subgradient:

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

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

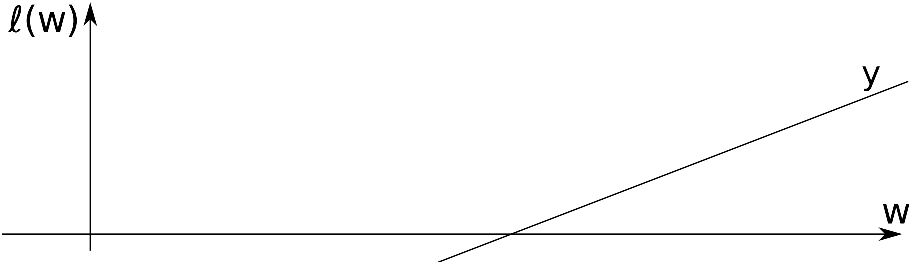
$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$

Computing a subgradient:

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

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



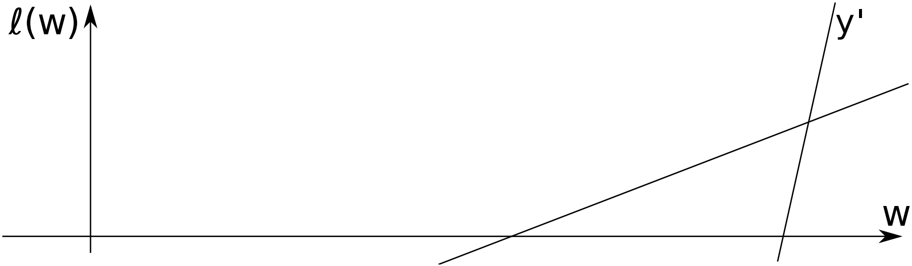
For each $y \in \mathcal{Y}$, $\ell_y(w)$ is a linear function.

Computing a subgradient:

$$\min_{w} \; \frac{1}{2}\|w\|^2 + \frac{C}{N}\sum_{n=1}^{N} \ell^n(w)$$

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



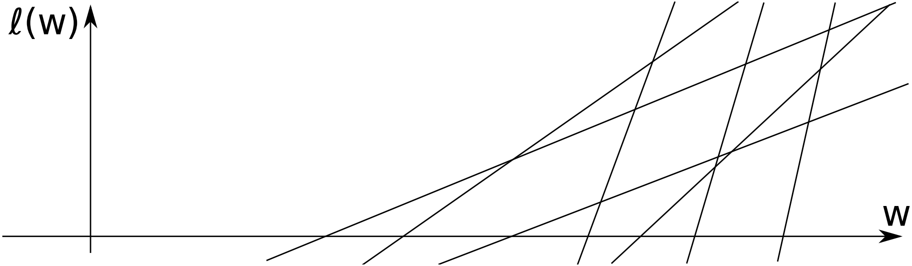
For each $y \in \mathcal{Y}$, $\ell_y(w)$ is a linear function.

Computing a subgradient:

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

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



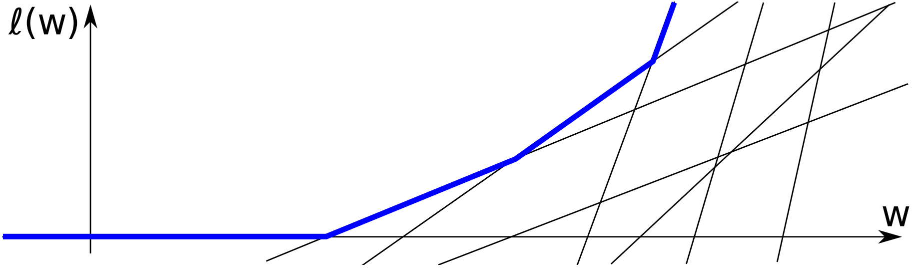
For each $y \in \mathcal{Y}$, $\ell_y(w)$ is a linear function.

Computing a subgradient:

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

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



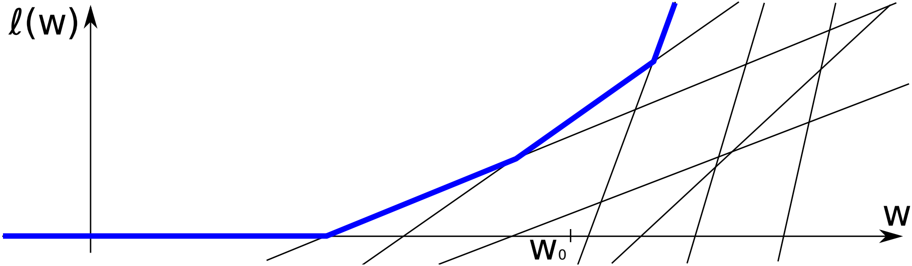
$\ell(w) = \max_y \ell_y(w)$: maximum over all $y \in \mathcal{Y}$.

Computing a subgradient:

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

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



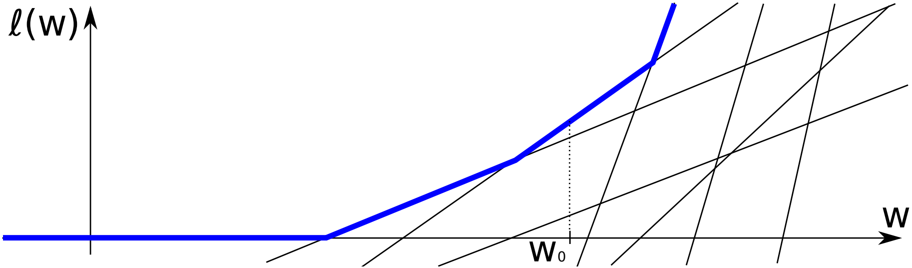
Subgradient of ℓ^n at w_0:

Computing a subgradient:

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

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



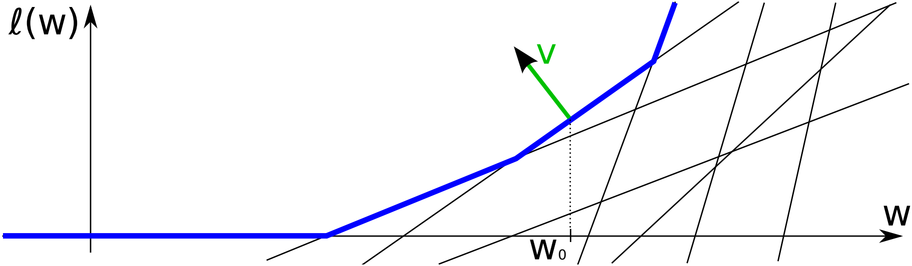
Subgradient of ℓ^n at w_0: find maximal (active) y.

Computing a subgradient:

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

with $\ell^n(w) = \max_y \ell^n_y(w)$, and

$$\ell^n_y(w) := \Delta(y^n, y) + \langle w, \phi(x^n, y)\rangle - \langle w, \phi(x^n, y^n)\rangle$$



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

Subgradient Descent S-SVM Training

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

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

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

Observation: each update of w needs 1 argmax-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 $\phi(x, y)$, 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: $\quad (x^n, y^n) \leftarrow$ randomly chosen training example pair
4: $\quad \hat{y} \leftarrow \operatorname{argmax}_{y \in \mathcal{Y}} \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle - \langle w, \phi(x^n, y^n) \rangle$
5: $\quad w \leftarrow w - \eta_t (w - \frac{C}{N}[\phi(x^n, \hat{y}) - \phi(x^n, y^n)])$
6: **end for**

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

Observation: each update of w needs only 1 argmax-prediction
(but we'll need many iterations until convergence)

Solving the Training Optimization Problem Numerically

We can solve an S-SVM like a linear SVM:

One of the equivalent formulations was:

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

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

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

Introduce feature vectors $\delta\phi(x^n, y^n, y) := \phi(x^n, y^n) - \phi(x^n, y)$.

Solve

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

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

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

This has the same structure as an ordinary SVM!

- quadratic objective ☺
- linear constraints ☺

Solve

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

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

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

This has the same structure as an ordinary SVM!

- quadratic objective ☺
- linear constraints ☺

Question: Can't we use a ordinary SVM/QP solver?

Solve

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

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

$$\langle w, \delta\phi(x^n, y^n, y)\rangle \geq \Delta(y^n, y) \ - \ \xi^n.$$

This has the same structure as an ordinary SVM!

- quadratic objective ☺
- linear constraints ☺

Question: Can't we use a ordinary SVM/QP solver?

Answer: Almost! We could, if there weren't $N|\mathcal{Y}|$ 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 $\leftarrow$ can of course be formalized.

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

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 $\leftarrow$ can of course be formalized.

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

Working Set Training

- 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*.

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:

Working Set Training

- 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 $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$,
input feature map $\phi(x, y)$, loss function $\Delta(y, y')$, regularizer C

```
S ← ∅
repeat
    (w, ξ) ← solution to QP only with constraints from S
    for i=1,...,n do
        ŷ ← argmax_{y∈Y} Δ(y^n, y) + ⟨w, φ(x^n, y)⟩
        if ŷ ≠ y^n then
            S ← S ∪ {(x^n, ŷ)}
        end if
    end for
until S doesn't change anymore.
```

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

Observation: each update of w needs 1 argmax-prediction per example.
(but we solve globally for next w, not by local steps)

We can solve an S-SVM like a non-linear SVM: compute Lagrangian dual

- $\min$ becomes $\max$,
- original (primal) variables w, ξ disappear,
- new (dual) variables α_{iy}: one per constraint of the original problem.

Dual S-SVM problem

$$\max_{\alpha \in \mathbb{R}_+^{n|\mathcal{Y}|}} \sum_{\substack{n=1,\dots,n \\ y \in \mathcal{Y}}} \alpha_{ny} \Delta(y^n, y) - \frac{1}{2} \sum_{\substack{y,\bar{y} \in \mathcal{Y} \\ n,\bar{n}=1,\dots,N}} \alpha_{ny} \alpha_{\bar{n}\bar{y}} \Big\langle \delta\phi(x^n, y^n, y), \delta\phi(x^{\bar{n}}, y^{\bar{n}}, \bar{y}) \Big\rangle$$

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

$$\sum_{y \in \mathcal{Y}} \alpha_{ny} \leq \frac{C}{N}.$$

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

We can **kernelize**:

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

$$k(\,(x, y)\,,\,(\bar{x}, \bar{y})\,) = \langle \phi(x, y), \phi(\bar{x}, \bar{y})\rangle.$$

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

- We can express the optimization in terms of k:

$$\begin{aligned}
\langle \delta\phi(x^n, y^n, y)\,, \delta\phi(x^{\bar{n}}, y^{\bar{n}}, \bar{y})\rangle \\
&= \big\langle\, \phi(x^n, y^n) - \phi(x^n, y)\ ,\ \phi(x^{\bar{n}}, y^{\bar{n}}) - \phi(x^{\bar{n}}, \bar{y})\,\big\rangle \\
&= \langle\, \phi(x^n, y^n), \phi(x^{\bar{n}}, y^{\bar{n}})\,\rangle - \langle\, \phi(x^n, y^n), \phi(x^{\bar{n}}, \bar{y})\,\rangle \\
&\quad - \langle\, \phi(x^n, y), \phi(x^{\bar{n}}, y^{\bar{n}})\rangle + \langle\, \phi(x^n, y), \phi(x^{\bar{n}}, \bar{y})\rangle \\
&= k(\,(x^n, y^n), (x^{\bar{n}}, y^{\bar{n}})\,) - k(\,(x^n, y^n), \phi(x^{\bar{n}}, \bar{y})\,) \\
&\quad - k(\,(x^n, y), (x^{\bar{n}}, y^{\bar{n}})\,) + k(\,(x^n, y), \phi(x^{\bar{n}}, \bar{y})\,) \\
&=: K_{i\bar{i}y\bar{y}}
\end{aligned}$$

Kernelized S-SVM problem:

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

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:

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

Summary – S-SVM Learning

Given:

- training set $\{(x^1, y^1), \dots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$
- loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.

Task: learn parameter w for $f(x) := \operatorname{argmax}_y \langle w, \phi(x, y) \rangle$ that minimizes expected loss on future data: $f(x^n) \approx y^n$.

Summary – S-SVM Learning

Given:

- training set $\{(x^1, y^1), \ldots, (x^n, y^n)\} \subset \mathcal{X} \times \mathcal{Y}$
- loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.

Task: learn parameter w for $f(x) := \operatorname{argmax}_y \langle w, \phi(x, y) \rangle$ that minimizes expected loss on future data: $f(x^n) \approx y^n$.

S-SVM solution derived by *maximum margin* framework:

- enforce correct output to be better than others by a margin:

$$\langle w, \phi(x^n, y^n) \rangle \geq \Delta(y^n, y) + \langle w, \phi(x^n, y) \rangle \quad \text{for all } y \in \mathcal{Y}.$$

- convex optimization problem, but non-differentiable
- many equivalent formulations → different training algorithms
- training needs repeated argmax prediction, no probabilistic inference

Structured Learning is full of Open Research Questions

- How to train faster?
  - CRFs need many runs of probablistic inference,
  - SSVMs need many runs of argmax-predictions.
- How to reduce the necessary amount of training data?
  - semi-supervised learning? transfer learning?
- How can we better understand different loss function?
  - when to use *probabilistic training*, when *maximum margin*?
  - CRFs are "consistent", SSVMs are not. Is this relevant?
- Can we understand structured learning with approximate inference?
  - often computing $\nabla\mathcal{L}(w)$ or $\operatorname{argmax}_y\langle w, \phi(x,y)\rangle$ *exactly* is infeasible.
  - can we guarantee good results even with approximate inference?
- More and new applications!