Introduction To Graphical Models

Peter V. Gehler
Max Planck Institute for Intelligent Systems,
Tübingen, Germany

ENS/INRIA Summer School, Paris, July 2013

MAX-PLANCK-GESELLSCHAFT

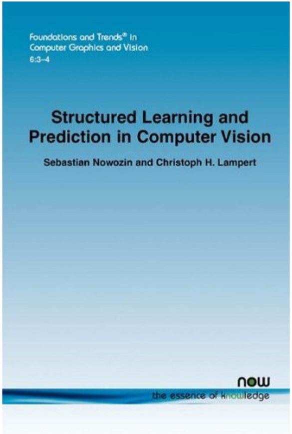
Extended version in book form

Sebastian Nowozin and Christoph Lampert

Structured Learning and Prediction in Computer Vision

ca 200 pages

Available free online
`http://pub.ist.ac.at/~chl/`



Slides mainly based on a tutorial version from Christoph – Thanks!

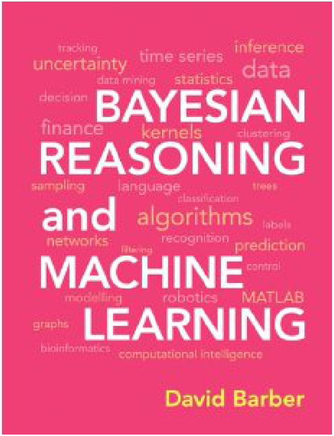
Literature Recommendation

David Barber

Bayesian Reasoning and Machine Learning

670 pages

Available free online
http://web4.cs.ucl.ac.uk/staff/D.Barber/pmwiki/pmwiki.php?n=Brml.Online



Standard Regression:

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

Structured Output Learning:

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

Standard Regression:

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

- 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 output prediction?

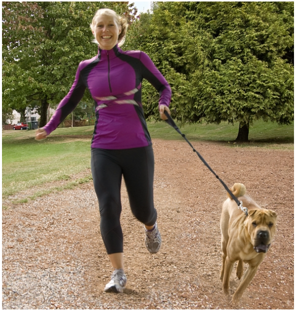
Ad hoc definition: predicting structured outputs from input data
(in contrast to predicting just a single number, like in classification or regression)

- 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)

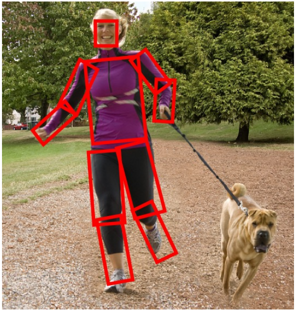
This tutorial: Applications and Examples from Computer Vision

Probabilistic Graphical Models

Example: Human Pose Estimation



$x \in \mathcal{X}$



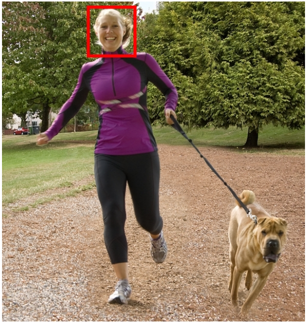
$y \in \mathcal{Y}$

- Given an image, where is a person and how is it articulated?

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

- Image x, but what is human pose $y \in \mathcal{Y}$ precisely?

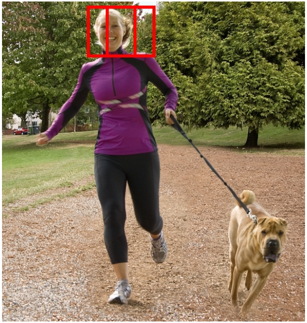
Human Pose $\mathcal{Y}$



Example y_{head}

- Body Part: $y_{head} = (u, v, \theta)$ where (u, v) center, θ rotation
  - $(u, v) \in \{1, \dots, M\} \times \{1, \dots, N\}, \theta \in \{0, 45°, 90°, \dots\}$

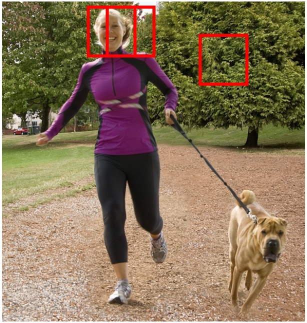
Human Pose $\mathcal{Y}$

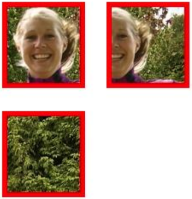


Example y_{head}

- Body Part: $y_{head} = (u, v, \theta)$ where (u, v) center, θ rotation
  - $(u, v) \in \{1, \dots, M\} \times \{1, \dots, N\}, \theta \in \{0, 45°, 90°, \dots\}$

Human Pose $\mathcal{Y}$

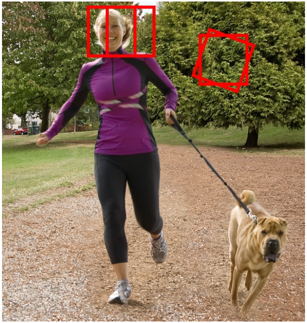




Example y_{head}

- Body Part: $y_{head} = (u, v, \theta)$ where (u, v) center, θ rotation
  - $(u, v) \in \{1, \dots, M\} \times \{1, \dots, N\}, \theta \in \{0, 45°, 90°, \dots\}$

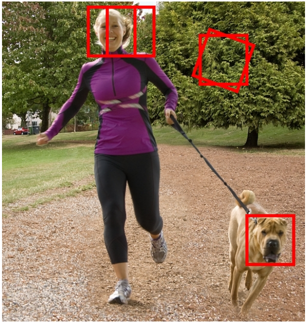
Human Pose $\mathcal{Y}$



Example y_{head}

- Body Part: $y_{head} = (u, v, \theta)$ where (u, v) center, θ rotation
  - $(u, v) \in \{1, \ldots, M\} \times \{1, \ldots, N\}, \theta \in \{0, 45°, 90°, \ldots\}$

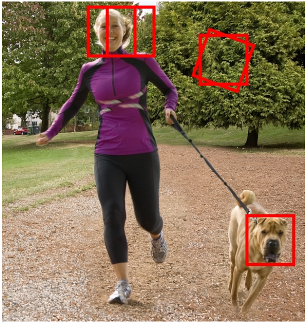
Human Pose $\mathcal{Y}$



Example y_{head}

- Body Part: $y_{head} = (u, v, \theta)$ where (u, v) center, θ rotation
  - $(u, v) \in \{1, \ldots, M\} \times \{1, \ldots, N\}, \theta \in \{0, 45°, 90°, \ldots\}$

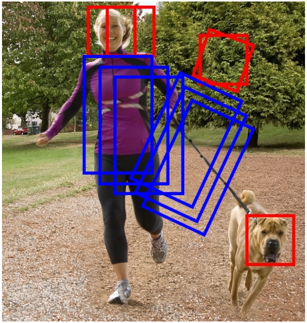
Human Pose $\mathcal{Y}$



Example y_{head}

- Body Part: $y_{head} = (u, v, \theta)$ where (u, v) center, θ rotation
  - $(u, v) \in \{1, \ldots, M\} \times \{1, \ldots, N\}, \theta \in \{0, 45°, 90°, \ldots\}$

Human Pose $\mathcal{Y}$



Example y_{head}

- Body Part: $y_{head} = (u, v, \theta)$ where (u, v) center, θ rotation
  - $(u, v) \in \{1, \ldots, M\} \times \{1, \ldots, N\}, \theta \in \{0, 45°, 90°, \ldots\}$
- Entire Body: $y = (y_{head}, y_{torso}, y_{left-lower-arm}, \ldots\} \in \mathcal{Y}$

Human Pose $\mathcal{Y}$

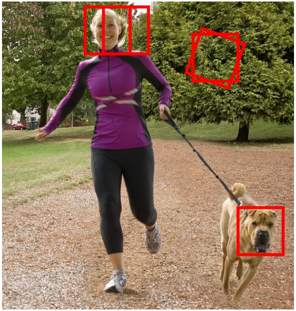
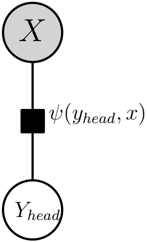


Image $x \in \mathcal{X}$

Example y_{head}



Head detector

- Idea: Have a head classifier (SVM, NN, ...)

$$\psi(y_{head}, x) \in \mathbb{R}_+$$

Human Pose $\mathcal{Y}$

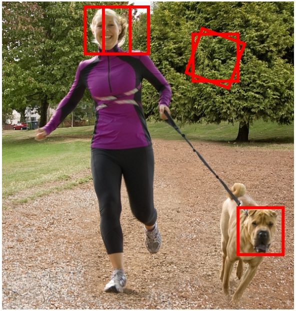
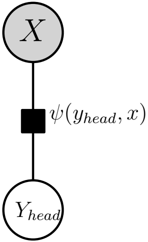


Image $x \in \mathcal{X}$

Example y_{head}



Head detector

- Idea: Have a head classifier (SVM, NN, ...)

$$\psi(y_{head}, x) \in \mathbb{R}_+$$

- Evaluate everywhere and record score

Human Pose $\mathcal{Y}$

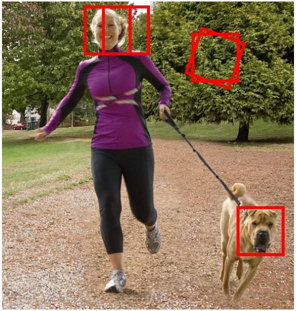
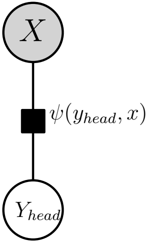


Image $x \in \mathcal{X}$

Example y_{head}



Head detector

- Idea: Have a head classifier (SVM, NN, ...)

$$\psi(y_{head}, x) \in \mathbb{R}_+$$

- Evaluate everywhere and record score
- Repeat for all body parts

Human Pose Estimation

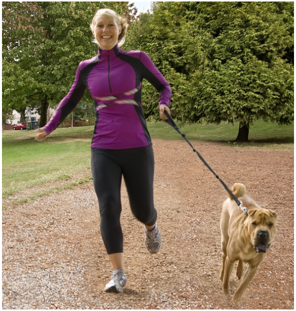
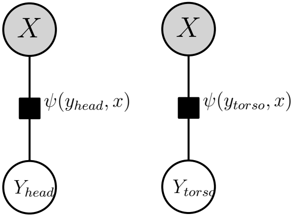


Image $x \in \mathcal{X}$



- Compute

$$y^* = (y^*_{head}, y^*_{torso}, \cdots) = \operatorname*{argmax}_{y_{head}, y_{torso}, \cdots} \psi(y_{head}, x)\psi(y_{torso}, x)\cdots$$

Human Pose Estimation

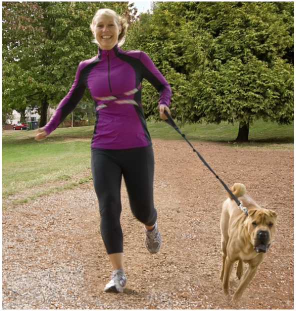
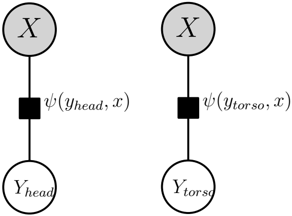


Image $x \in \mathcal{X}$



- Compute

$$
\begin{aligned}
y^* &= (y^*_{head}, y^*_{torso}, \cdots) = \operatorname*{argmax}_{y_{head}, y_{torso}, \cdots} \psi(y_{head}, x)\psi(y_{torso}, x)\cdots \\
&= (\operatorname*{argmax}_{y_{head}} \psi(y_{head}, x), \operatorname*{argmax}_{y_{torso}} \psi(y_{torso}, x), \cdots)
\end{aligned}
$$

Human Pose Estimation

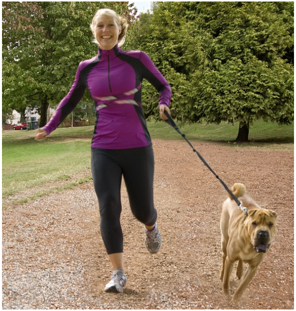
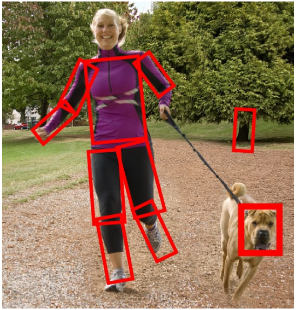


Image $x \in \mathcal{X}$



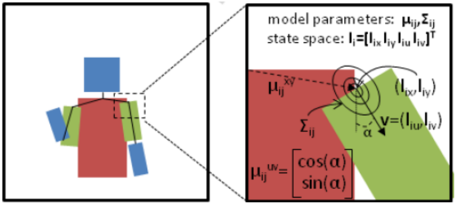
Prediction $y^* \in \mathcal{Y}$

- Compute

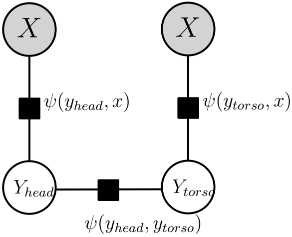
$$
\begin{aligned}
y^* &= (y^*_{head}, y^*_{torso}, \cdots) = \operatorname*{argmax}_{y_{head}, y_{torso}, \cdots} \psi(y_{head}, x)\psi(y_{torso}, x)\cdots \\
&= (\operatorname*{argmax}_{y_{head}} \psi(y_{head}, x), \operatorname*{argmax}_{y_{torso}} \psi(y_{torso}, x), \cdots)
\end{aligned}
$$

- Great! Problem solved!?

Idea: Connect up the body



$\psi(y_{torso}, y_{arm})$



Head-Torso Model

- Ensure *head* is on top of *torso*

$$\psi(y_{head}, y_{torso}) \in \mathbb{R}_+$$

- Compute

$$y^* = \operatorname*{argmax}_{y_{head}, y_{torso}, \cdots} \psi(y_{head}, x)\psi(y_{torso}, x)\psi(y_{head}, y_{torso}) \cdots$$

but this does not decompose anymore!

left image from Ben Sapp

The recipe

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

1) Define auxiliary function, $g : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$,

$$e.g. \qquad g(x, y) = \prod_i \psi_i(y_i, x) \prod_{i \sim j} \psi_{ij}(y_i, y_j, x)$$

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

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

A Probabilistic View

Computer Vision problems usually deal with *uncertain* information

- Incomplete information (observe static images, projections, etc)
- Annotation is "noisy" (wrong or ambiguous cases)
- ...

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 if 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:
The MAP estimate $y \in \mathcal{Y}$, is the most probable value (there can be multiple).

Probability Distributions

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

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

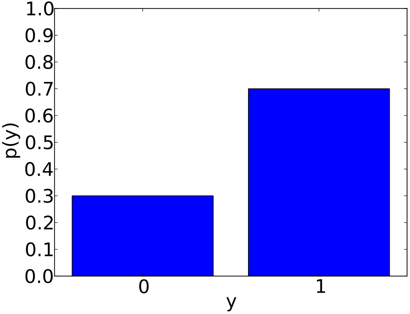
Probability Distributions

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

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

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

- 2 values,
- 1 degree of freedom



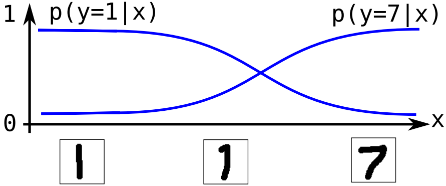
Conditional Probability Distributions

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

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

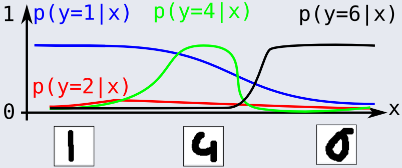
For example: **binary** prediction
$\mathcal{X} = \{\text{images}\}$, $y \in \mathcal{Y} = \{0, 1\}$

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



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

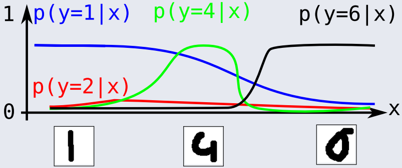
- each x: K values, $K-1$ d.o.f.
  $\rightarrow K-1$ *functions*
- or 1 vector-valued function with $K-1$ outputs



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

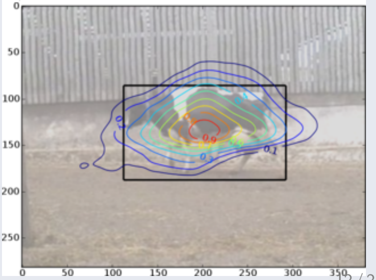


Typically: K functions, plus explicit normalization

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,

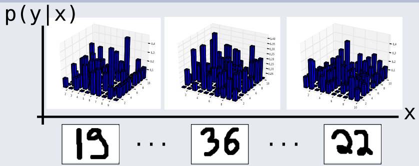


Structured objects: **predicting M variables jointly**

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

For each x:

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



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

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

$$y = (\textit{left}, \textit{top}, \textit{right}, \textit{bottom})$$

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*.

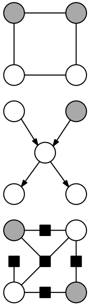
Probabilistic Graphical Models

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.



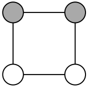
Probabilistic Graphical Models

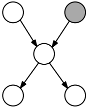
A **(probabilistic) graphical model** defines

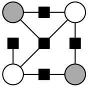
- a *family of probability distributions* over a set of random variables, by means of a *graph*.

Popular classes of graphical models,

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







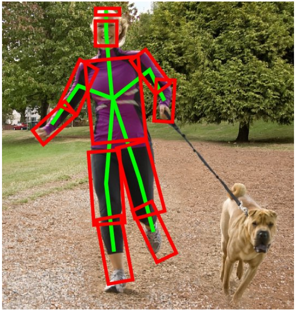
The graph encodes *conditional independence assumptions* between the variables:

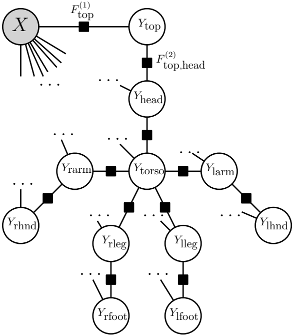
- for $N(i)$ are the neighbors of node i in the graph

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

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

Example: Pictorial Structures for Articulated Pose Estimation



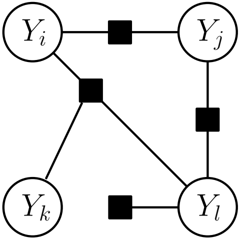


- In principle, all parts depend on each other.
  - Knowing where the head is puts constraints on where the feet can be.
- But **conditional independences** as specified by the graph:
  - If we *know* where the **left leg** is, the **left foot**'s position does not depend on the **torso** position anymore,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

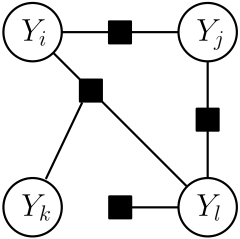
- Decomposable output $y = (y_1, \dots, y_{|V|})$
- Graph: $G = (V, \mathcal{F}, \mathcal{E})$, $\mathcal{E} \subseteq V \times \mathcal{F}$
  - variable nodes V (circles),
  - factor nodes $\mathcal{F}$ (boxes),
  - edges $\mathcal{E}$ between variable and factor nodes.
  - each factor $F \in \mathcal{F}$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



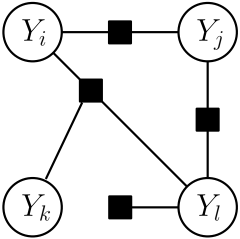
Factor graph

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

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

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 (circles),
  - factor nodes $\mathcal{F}$ (boxes),
  - 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 $y_F = (y_{v_1}, \dots, y_{v_{|F|}})$



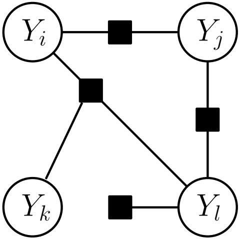
Factor graph

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

$$p(y) = \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(y_F) = \frac{1}{Z} \psi_1(Y_l) \psi_2(Y_j, Y_l) \psi_3(Y_i, Y_j) \psi_4(Y_i, Y_k, Y_l)$$

Factor Graphs

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



Factor graph

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

$$p(y) = \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(y_F) = \frac{1}{Z} \psi_1(Y_l) \psi_2(Y_j, Y_l) \psi_3(Y_i, Y_j) \psi_4(Y_i, Y_k, Y_l)$$

- 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 $p(y|x)$?

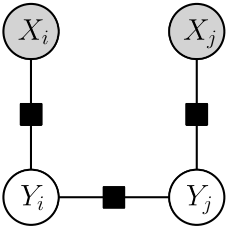
- Potentials become also functions of (part of) x: $\psi_F(y_F; x_F)$ instead of just $\psi_F(y_F)$

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

- 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

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

$$= \frac{1}{Z(x)} \exp(-\sum_{F \in \mathcal{F}} E_F(y_F; x_F)) \quad = \frac{1}{Z(x)} \exp(-E(y; x))$$

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

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

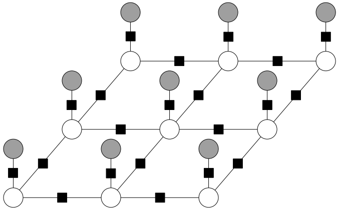
MAP prediction can be performed by *energy minimization*.

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

Example: An Energy Function for Image Segmentation

Foreground/background image segmentation

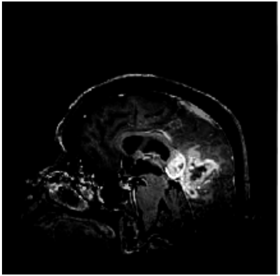
- $\mathcal{X} = [0, 255]^{WH}$, $\mathcal{Y} = \{0, 1\}^{WH}$
  foreground: $y_i = 1$, background: $y_i = 0$.
- graph: 4-connected grid
- Each output pixel depends on
  - local grayvalue (inputs)
  - neighboring outputs



Energy function components ("Ising" model):

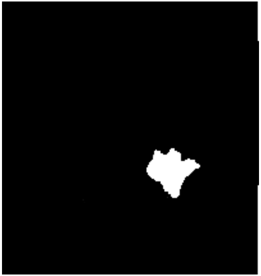
- $E_i(y_i = 1, x_i) = 1 - \frac{1}{255} x_i$ $\quad E_i(y_i = 0, x_i) = \frac{1}{255} x_i$
  x_i bright $\rightarrow$ y_i rather foreground, x_i dark $\rightarrow$ y_i rather background
- $E_{ij}(0, 0) = E_{ij}(1, 1) = 0$, $\quad E_{ij}(0, 1) = E_{ij}(1, 0) = \omega$ for $\omega > 0$
  prefer that neighbors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

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



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

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

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**

Parameter Learning

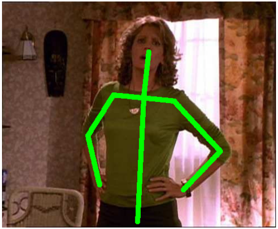
Assume fixed graph structure, **learn potentials/energies** (ψ_F)

Among other tasks (learn the graph structure, variables, etc.)

$\Rightarrow$ Topic of Wednesdays' lecture

Example: Pictorial Structures

input image x



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

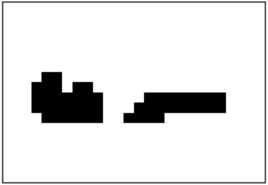
$p(y_i|x)$

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

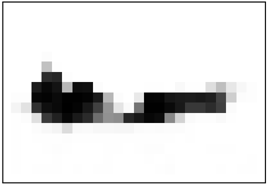
  of the individual body parts.

Example: Man-made structure detection

input image x



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



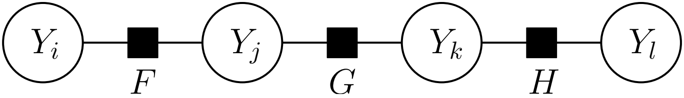
$p(y_i|x)$

- Task: Pixel depicts a man made structure or not? $y_i \in \{0, 1\}$
- Middle: MAP inference
- Right: variable marginals
- Attention: Max-Marginals $\neq$ MAP

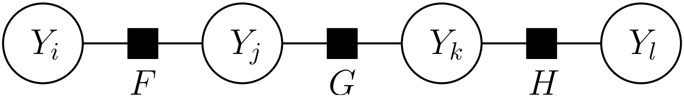
Probabilistic Inference

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

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:



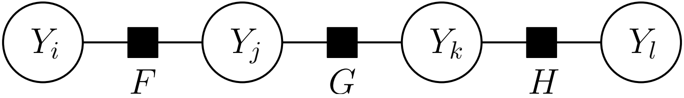
Assume $y = (y_i, y_j, y_k, y_l)$, $\mathcal{Y} = \mathcal{Y}_i \times \mathcal{Y}_j \times \mathcal{Y}_k \times \mathcal{Y}_l$, and an energy function $E(y; x)$ compatible with the following factor graph:



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

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

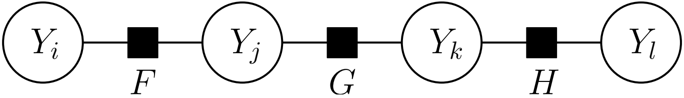
Problem: We don't know $Z(x)$, and computing it using

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

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** $Z(x)$.

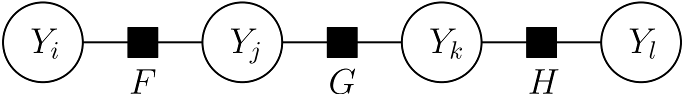
Probabilistic Inference – Belief Propagation / Message Passing



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

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

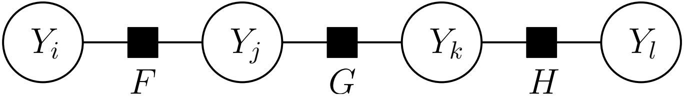
Probabilistic Inference – Belief Propagation / Message Passing



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

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

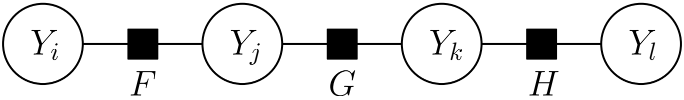
Probabilistic Inference – Belief Propagation / Message Passing



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

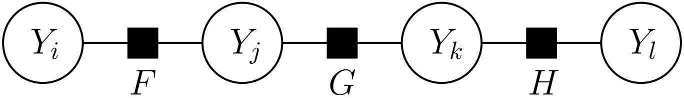
$$
\begin{aligned}
Z &= \sum_{y \in \mathcal{Y}} \exp(-E(y)) \\
&= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-E(y_i, y_j, y_k, y_l)) \\
&= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k}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

Probabilistic Inference – Belief Propagation / Message Passing



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

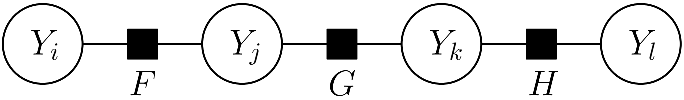
Probabilistic Inference – Belief Propagation / Message Passing



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

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

Probabilistic Inference – Belief Propagation / Message Passing

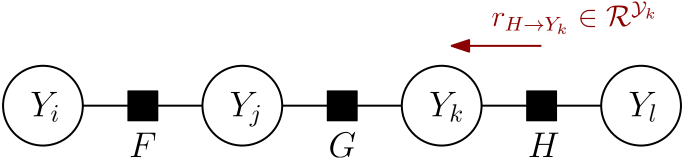


$$Z = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \exp(-(E_F(y_i, y_j) + E_G(y_j, y_k) + E_H(y_k, y_l)))$$

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

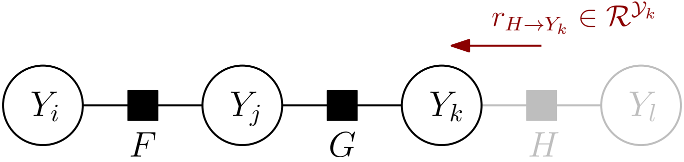
$$= \sum_{y_i} \sum_{y_j} \exp(-E_F(y_i, y_j)) \sum_{y_k} \exp(-E_G(y_j, y_k)) \sum_{y_l} \exp(-E_H(y_k, y_l))$$

Probabilistic Inference – Belief Propagation / Message Passing



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

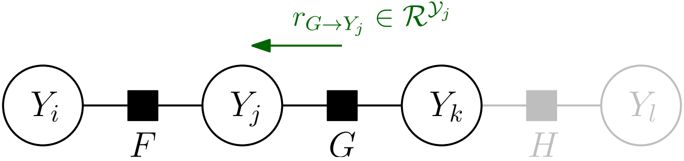
Probabilistic Inference – Belief Propagation / Message Passing



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

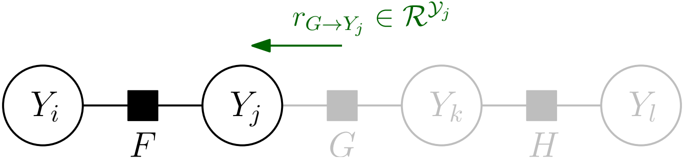
$$= \sum_{y_i} \sum_{y_j} \exp(-E_F(y_i, y_j)) \sum_{y_k} \exp(-E_G(y_j, y_k)) r_{H \to Y_k}(y_k)$$

Probabilistic Inference – Belief Propagation / Message Passing



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

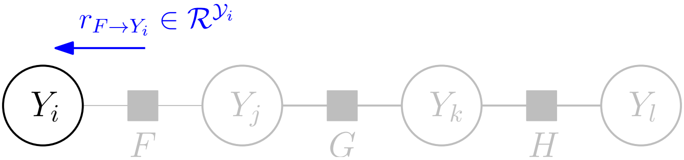
Probabilistic Inference – Belief Propagation / Message Passing



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

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

Probabilistic Inference – Belief Propagation / Message Passing

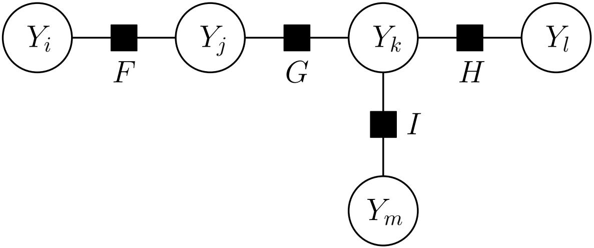


$$Z = \sum_{y_i} \sum_{y_j} \exp(-E_F(y_i, y_j)) \underbrace{\sum_{y_k} \exp(-E_G(y_j, y_k)) r_{H \to Y_k}(y_k)}_{r_{G \to Y_j}(y_j)}$$

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

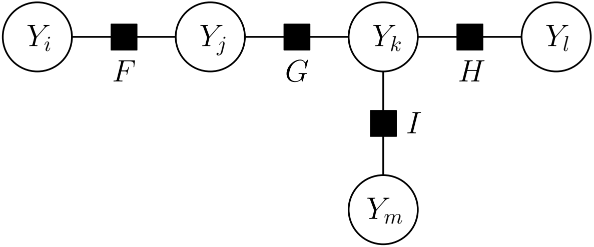
$$= \sum_{y_i} r_{F \to Y_i}(y_i)$$

Example: Inference on Trees



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

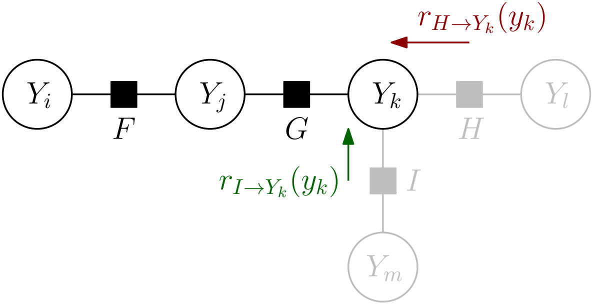
Example: Inference on Trees



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

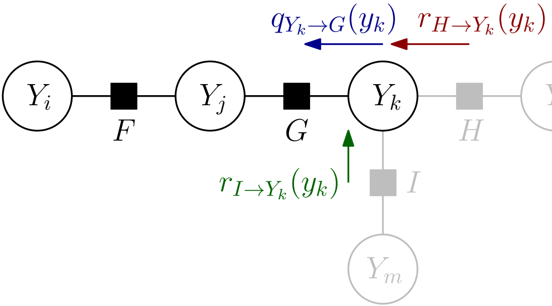
$$\left(\underbrace{\left(\sum_{y_l \in \mathcal{Y}_l} \exp(-E_H(y_k, y_l)) \right)}_{r_{H \to Y_k}(y_k)} \cdot \underbrace{\left(\sum_{y_m \in \mathcal{Y}_m} \exp(-E_I(y_k, y_m)) \right)}_{r_{I \to Y_k}(y_k)} \right)$$

Example: Inference on Trees



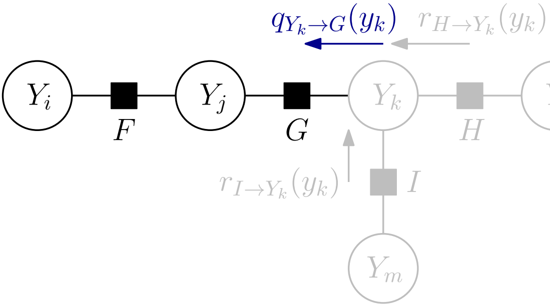
$$Z = \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \exp(-E_F(y_i, y_j)) \sum_{y_k \in \mathcal{Y}_k} \exp(-E_G(y_j, y_k)) \cdot \left(r_{H\to Y_k}(y_k) \cdot r_{I\to Y_k}(y_k)\right)$$

Example: Inference on Trees



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

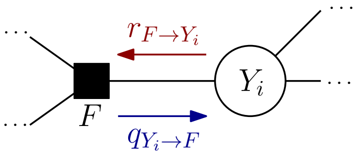
Example: Inference on Trees



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

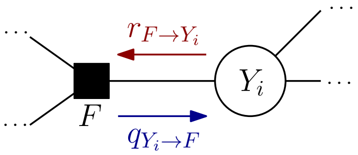
Factor Graph Sum-Product Algorithm

- "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

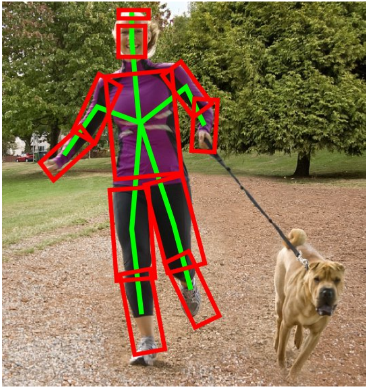
- "Message": pair of vectors at each factor graph edge $(i, F) \in \mathcal{E}$
  1. $r_{F \to Y_i} \in \mathbb{R}^{\mathcal{Y}_i}$: factor-to-variable message
  2. $q_{Y_i \to F} \in \mathbb{R}^{\mathcal{Y}_i}$: variable-to-factor message

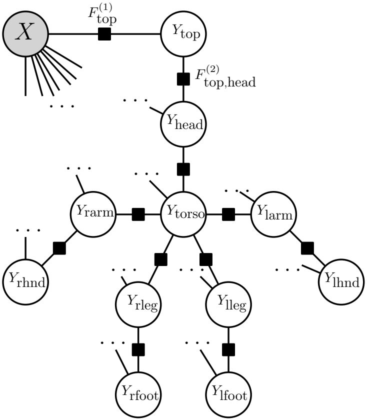


- Algorithm iteratively update messages
- After convergence: Z and $p(y_F)$ can be obtained from the messages.

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

input image x

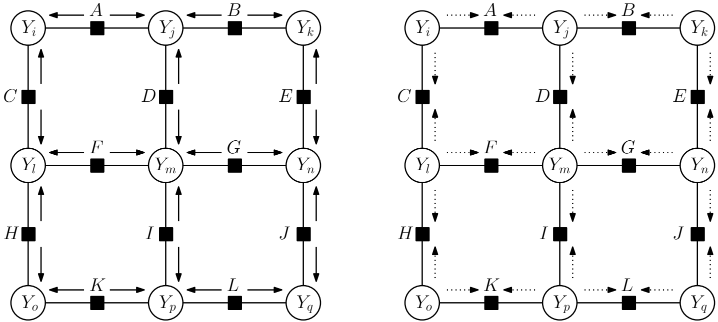
$p(y_i|x)$

- Exact marginals although state space is **huge** and thus partition function is a **huge** sum.

$$Z(x) = \sum_{\text{all bodies } y} \exp\left(-E(y;x)\right)$$

Belief Propagation in Loopy Graphs

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

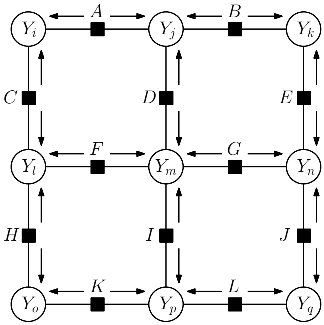


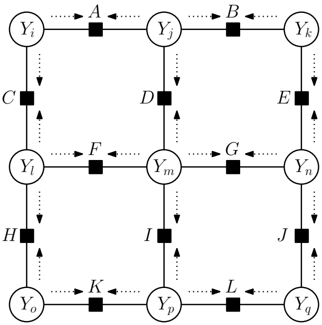
Problem: There is no well-defined *leaf–to–root* order.

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).

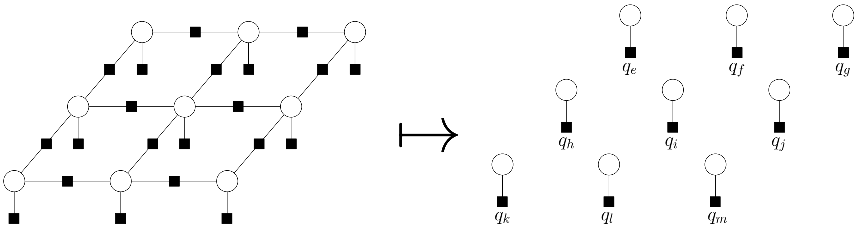
Probabilistic Inference – Variational Inference / Mean Field

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

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

Probabilistic Inference – Sampling / Markov-Chain Monte Carlo

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

- For fixed x, let $y^{(1)}, y^{(2)}, \ldots$ 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:

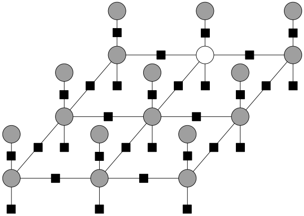
- We can get away with a sequence of *dependent* samples
  $\to$ Monte-Carlo Markov Chain (MCMC) sampling

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 $y^{(s)} = (y^{(s-1)}_1, \dots, y^{(s-1)}_{i-1},\ y_i,\ y^{(s-1)}_{i+1}, \dots, y^{(s-1)}_d)$

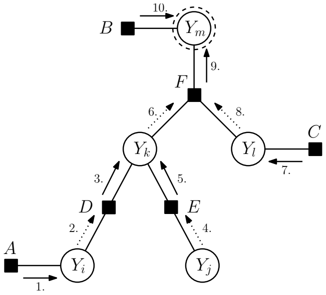
$$p(y_i | y^{(s)}_{V \setminus \{i\}}, x) = \frac{p(y_i, y^{(t)}_{V \setminus \{i\}} | x)}{\sum_{y_i \in \mathcal{Y}_i} p(y_i, y^{(t)}_{V \setminus \{i\}} | x)} = \frac{\exp(-E(y_i, y^{(t)}, x)}{\sum_{y_i \in \mathcal{Y}_i} \exp(-E(y_i, y^{(t)}, x)}$$

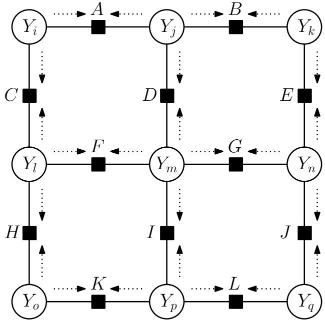


MAP Prediction

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 graphs, we can find the global optimum only in **special cases**:

- Binary output variables: $\mathcal{Y}_i = \{0, 1\}$ for $i = 1, \ldots, d$,
- 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 graphs, 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):

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

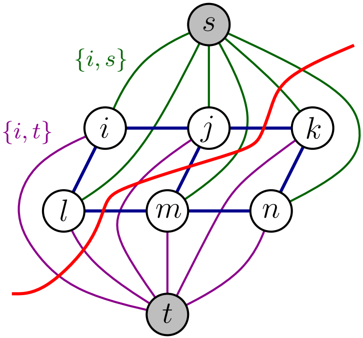
- Restriction 2 (regular/submodular/attractive pairwise potentials)

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

(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.

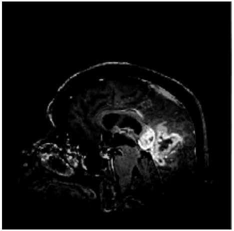
GraphCuts Example

Image segmentation energy:

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

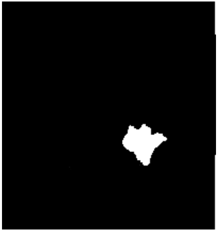
All conditions to apply GraphCuts are fulfilled.

- $E_i(y_i, x) \geq 0$,
- $E_{ij}(y_i, y_j) = 0 \quad$ 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, $\mathcal{Y}_i = \{1, \ldots, K\}$:

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

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 $\sum_k \mu_{i;k} = 1$.

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

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

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

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.

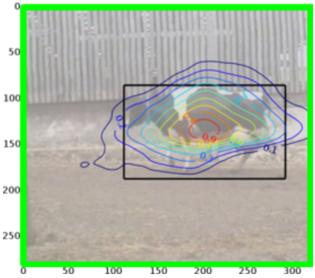
MAP Prediction – Custom solutions: E.g. branch-and-bound

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**:



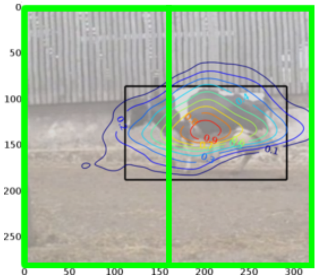
MAP Prediction – Custom solutions: E.g. branch-and-bound

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**:



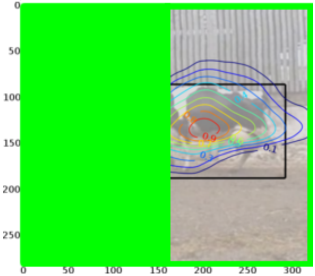
MAP Prediction – Custom solutions: E.g. branch-and-bound

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**:



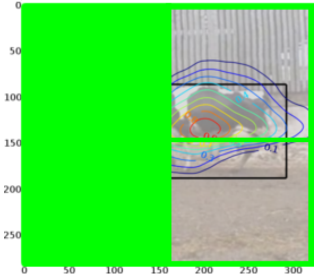
MAP Prediction – Custom solutions: E.g. branch-and-bound

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**:



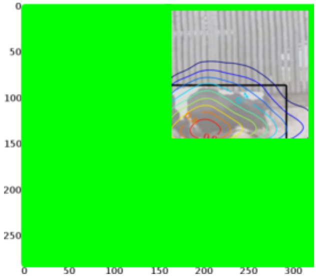
MAP Prediction – Custom solutions: E.g. branch-and-bound

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**:



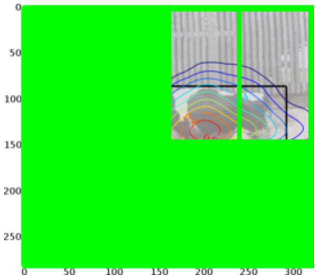
MAP Prediction – Custom solutions: E.g. branch-and-bound

Note: we just try to solve an optimization problem

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

We can use any optimization technique that fits the problem.

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



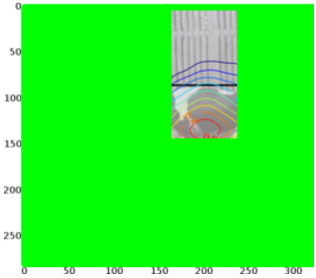
MAP Prediction – Custom solutions: E.g. branch-and-bound

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**:



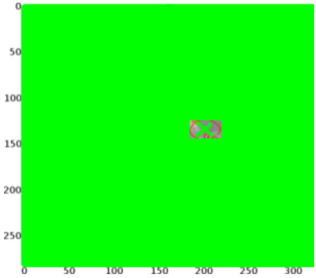
MAP Prediction – Custom solutions: E.g. branch-and-bound

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**:



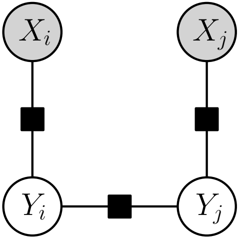
Optimal Prediction

Predict with loss function $\Delta(\bar{y}, y)$.

Optimal Prediction

- Optimal prediction is minimum expected risk – an expectation

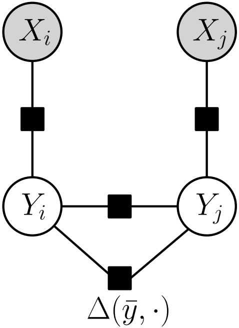
$$y^* = \operatorname*{argmin}_{\bar{y}\in\mathcal{Y}} \sum_{y\in\mathcal{Y}} \Delta(\bar{y}, y) p(y|x)$$



Optimal Prediction

- Optimal prediction is minimum expected risk – an expectation

$$y^* = \operatorname*{argmin}_{\bar{y}\in\mathcal{Y}} \sum_{y\in\mathcal{Y}} \Delta(\bar{y}, y) p(y|x)$$
$$= \operatorname*{argmin}_{\bar{y}\in\mathcal{Y}} \sum_{y\in\mathcal{Y}} \Delta(\bar{y}, y) \prod_{F} \psi_F(y_F; x)$$

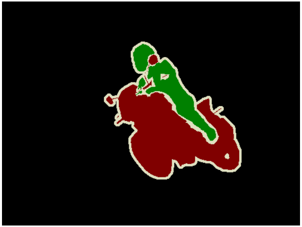


- Can think of Δ as another CRF factor
- Reuse inference techniques

Example: Hamming loss

Count the number of mislabeled variables:

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



- Makes more sense than 0/1 loss for image segmentation
- Optimal: predict maximum marginals (exercise)

$$y^* = (\operatorname*{argmax}_{y_1} p(y_1|x), \operatorname*{argmax}_{y_2} p(y_2|x), \dots)$$

Example: Pixel 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.

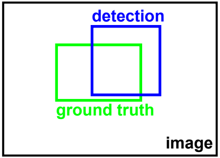
- Optimal: predict marginal mean (exercise)

$$y^* = (\mathbb{E}_{p(y|x)}[y_1], \mathbb{E}_{p(y|x)}[y_2], \ldots)$$

Example: Task specific losses

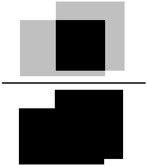
Object detection

- 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 Prediction

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

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

Structured prediction comes with structured loss functions, $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$.

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, ...

Max Planck Institute for Intelligent Systems

Other groups on Campus

- Empirical Inference (Machine Learning)
- **Perceiving Systems (Computer Vision)**
- Autonomous Motion (Robotics)

More information: http://ps.is.tue.mpg.de/