Statistical Machine Learning

Christoph Lampert

IST AUSTRIA

Institute of Science and Technology

Spring Semester 2015/2016 // Lecture 9

(lots of material courtesy of S. Nowozin, http://www.nowozin.net)

Standard Regression/Classification:

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

- inputs $\mathcal{X}$ can be any kind of objects
- output $y \in \mathcal{Y}$ is a number (real or integer)

Structured Prediction:

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

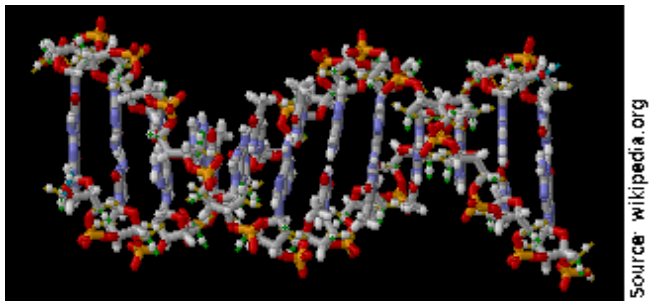
- inputs $\mathcal{X}$ can be any kind of objects
- outputs $y \in \mathcal{Y}$ are complex (structured) objects

What is structured data?

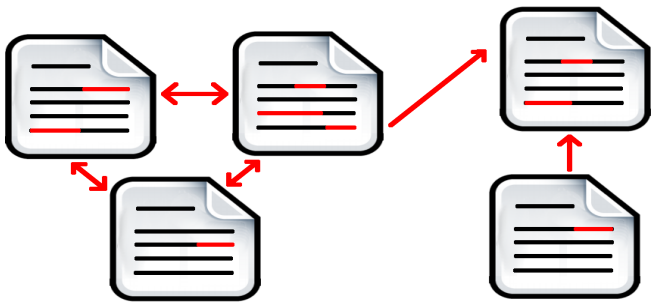
Ad hoc definition: data that consists of several parts, and not only the parts themselves contain information, but also the way in which the parts belong together.

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

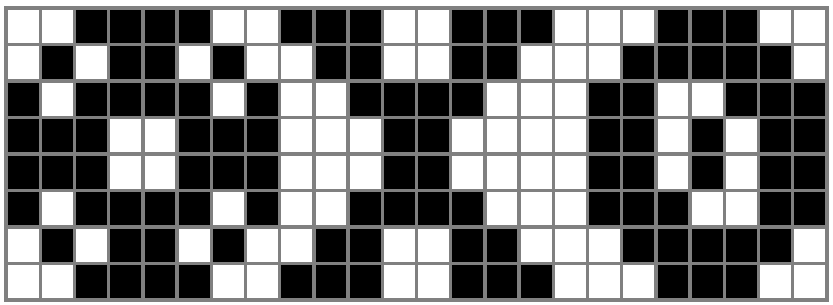
Text



Molecules / Chemical Structures



Documents/HyperText



Images

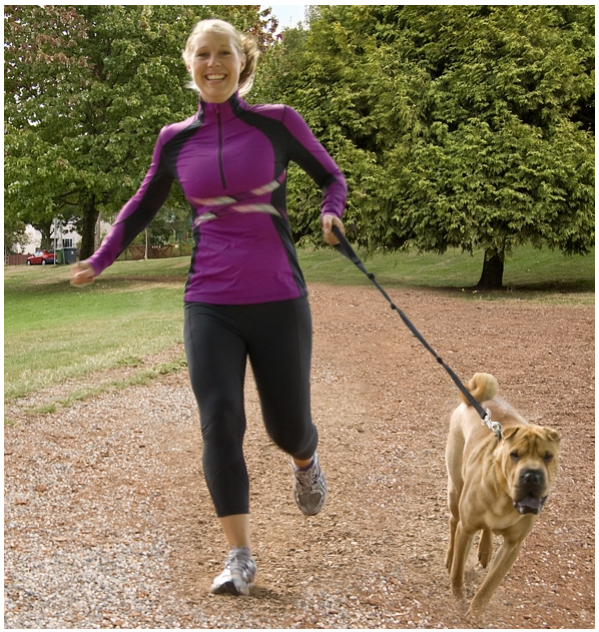
What is structured output prediction?

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

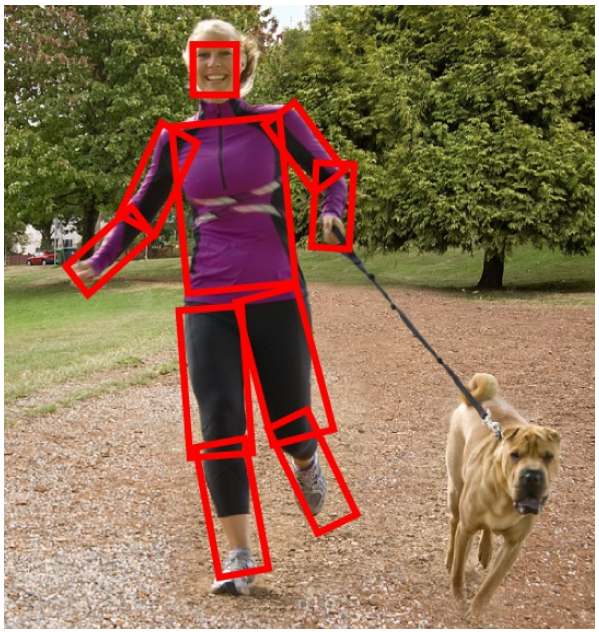
- Natural Language Processing:
  - Automatic Translation (output: sentences)
- Bioinformatics:
  - Secondary Structure Prediction (output: bipartite graphs)
- Speech Processing:
  - Text-to-Speech (output: audio signal)
- Robotics:
  - Planning (output: sequence of actions)
- Information Retrieval:
  - Ranking (output: ordered list of documents)

This lecture: mainly examples from Computer Vision

Example: Human Pose Estimation



$x \in \mathcal{X}$



$y \in \mathcal{Y}$

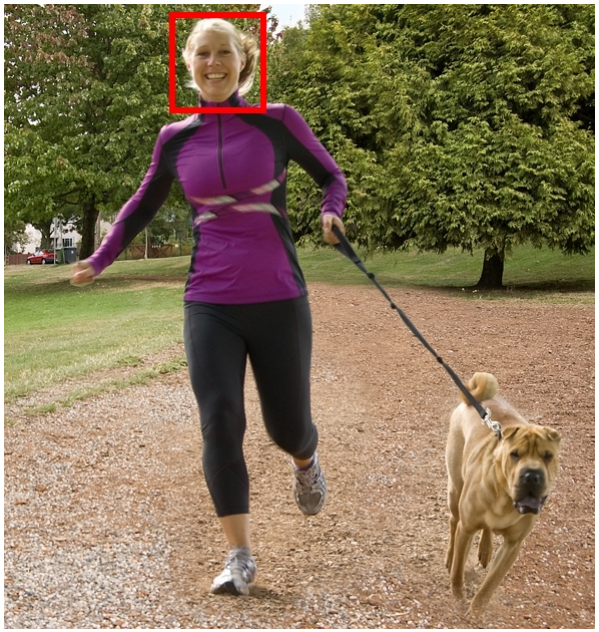
- Given an image, where is a person and how is it articulated?

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

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

Image: Flickr.com user lululemon athletica

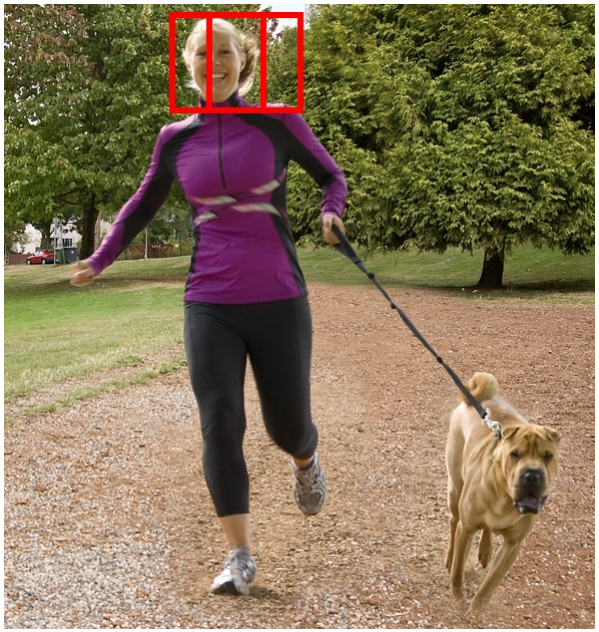
Example: Human Pose Estimation



Example y_{head}

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

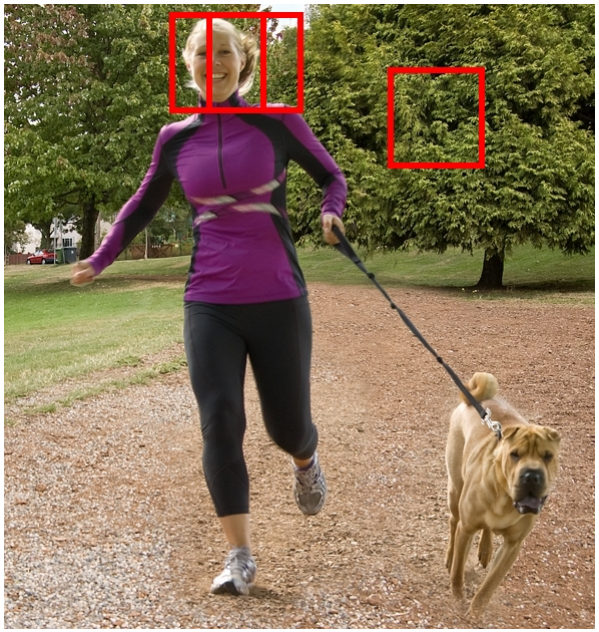
Example: Human Pose Estimation



Example y_{head}

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

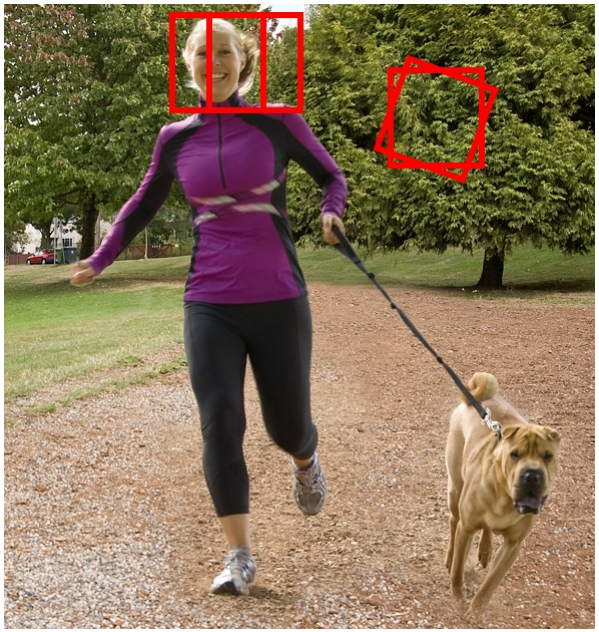
Example: Human Pose Estimation



Example y_{head}

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

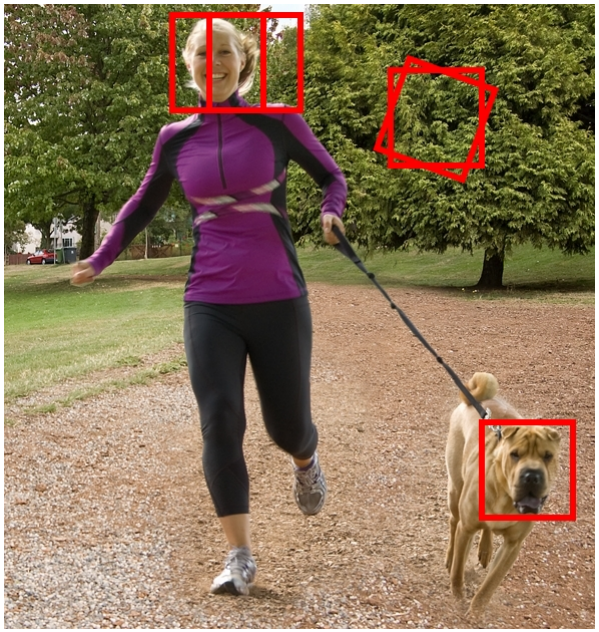
Example: Human Pose Estimation



Example y_{head}

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

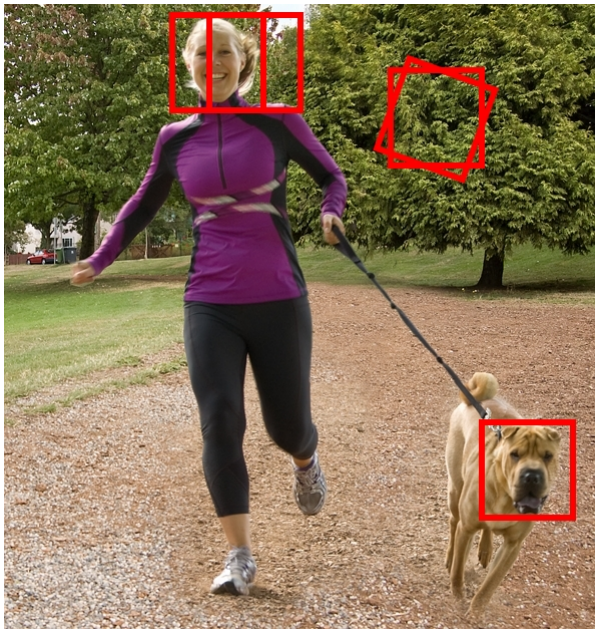
Example: Human Pose Estimation



Example y_{head}

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

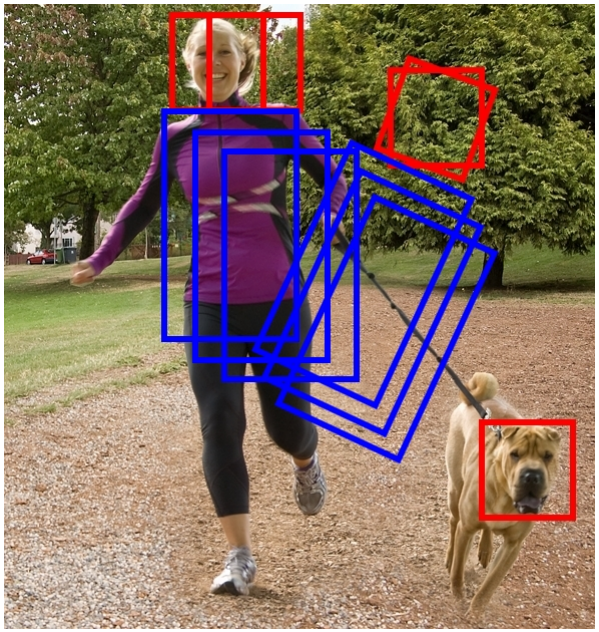
Example: Human Pose Estimation



Example y_{head}

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

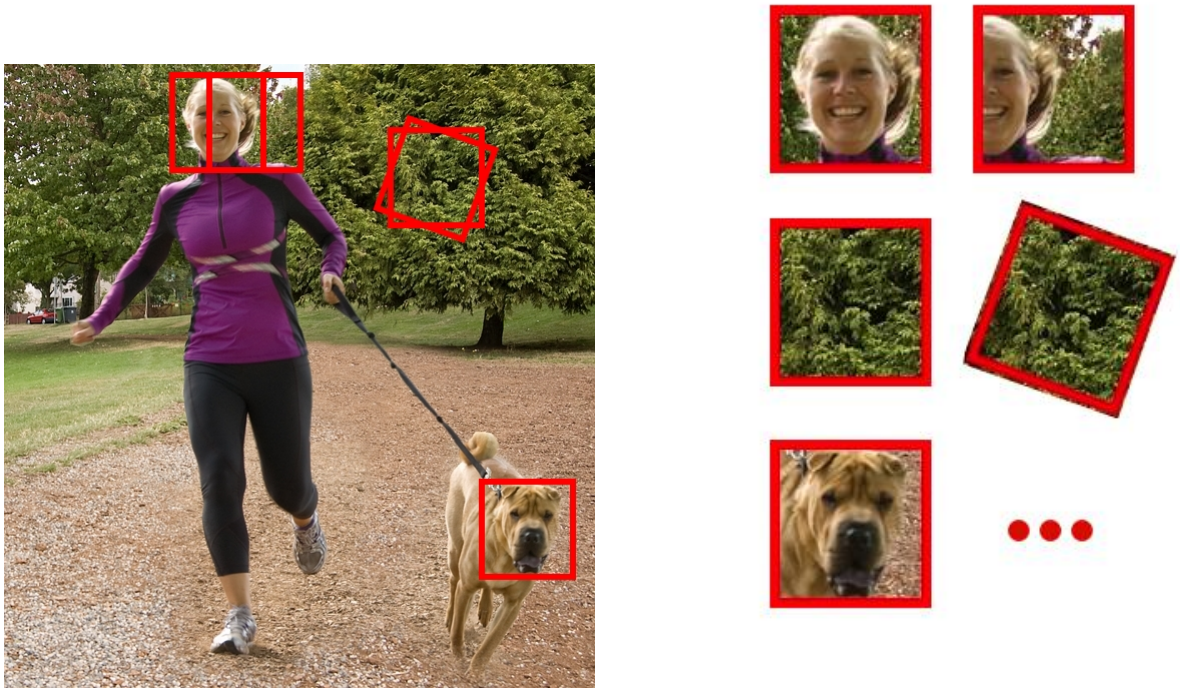
Example: Human Pose Estimation



Example y_{head}

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

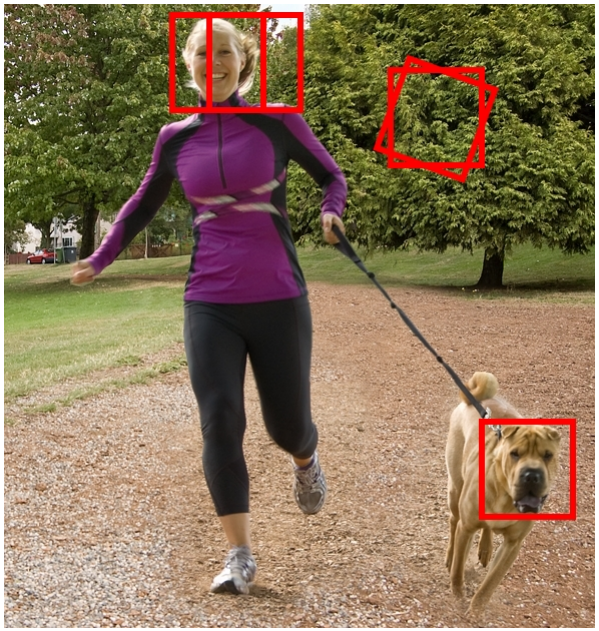
Example: Human Pose Estimation



- Idea: Have a head detector (CNN, SVM, RF, ...)

$$f_{\text{head}} : \mathcal{X} \to \mathbb{R}$$

Example: Human Pose Estimation



- Idea: Have a head detector (CNN, SVM, RF, ...)

$$f_{\text{head}} : \mathcal{X} \to \mathbb{R}$$

- Evaluate for every possible location and record score
- Same construction for all other body parts

Example: Human Pose Estimation

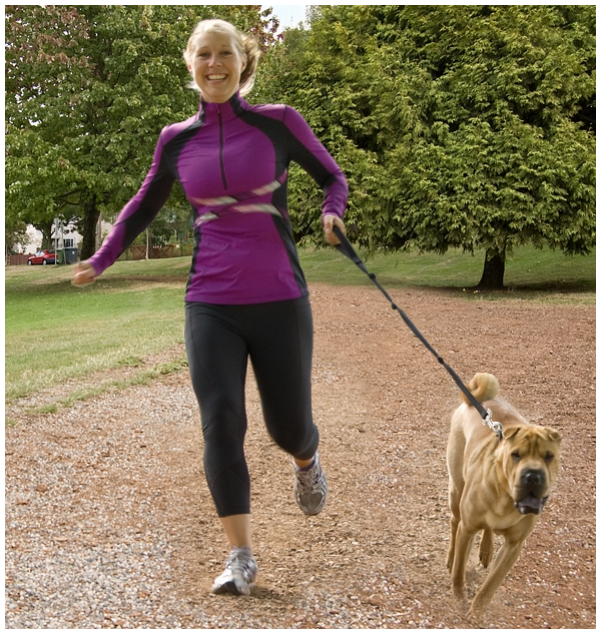


Image $x \in \mathcal{X}$

- Put together body from individual parts

$$y^{best} = (y^{best}_{\text{head}}, y^{best}_{\text{torso}}, \dots)$$

Example: Human Pose Estimation

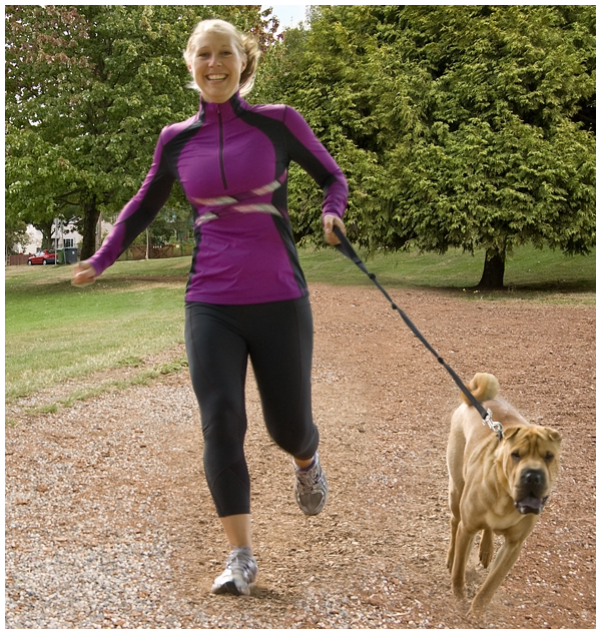
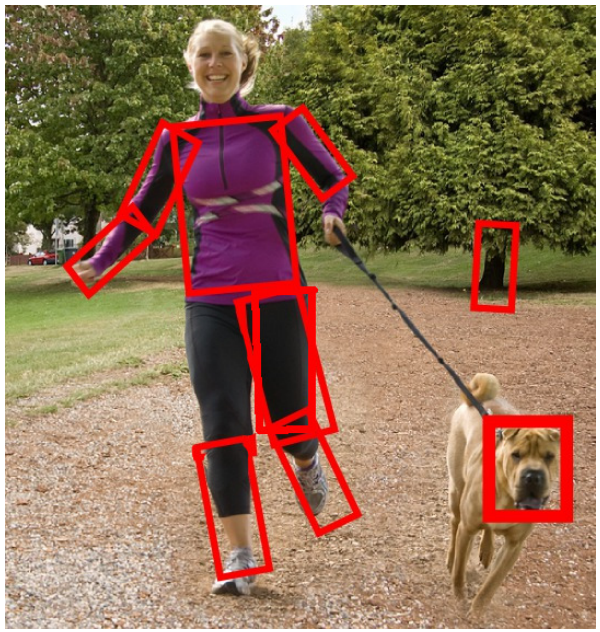


Image $x \in \mathcal{X}$



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

- Put together body from individual parts

$$y^{best} = (y^{best}_{\text{head}}, y^{best}_{\text{torso}}, \dots)$$

- Each part looks reasonable, but overall makes no sense

Example: Human Pose Estimation

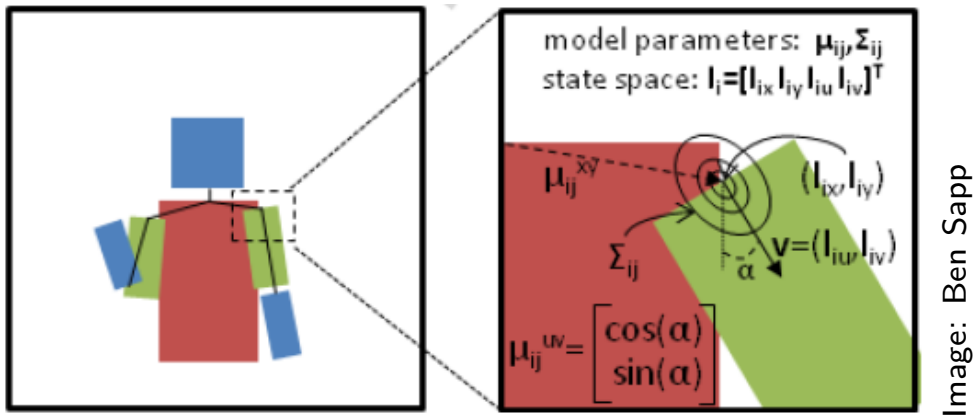


Image: Ben Sapp

Enforce **relations between parts**

- For example, *head* must be connected to *torso*
- Problem:

$$y^{best} \neq (y^{best}_{\text{head}}, y^{best}_{\text{torso}}, \dots)$$

independent decisions for each body part are not optimal anymore

Example: Human Pose Estimation

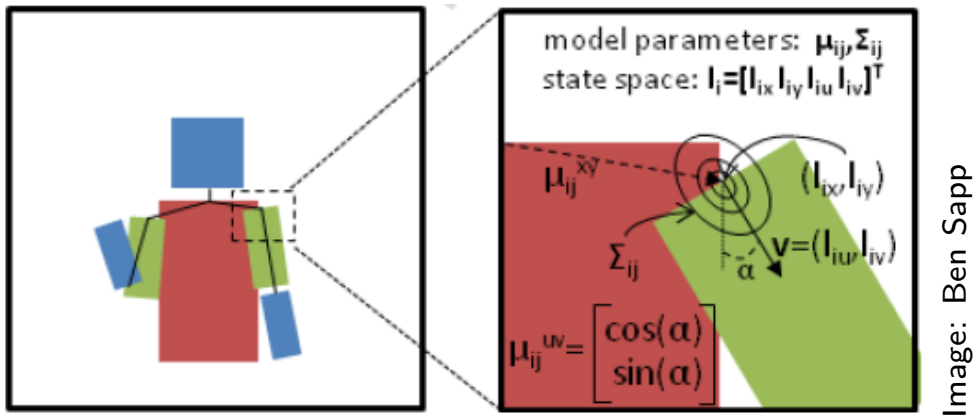


Image: Ben Sapp

Enforce **relations between parts**

- For example, *head* must be connected to *torso*
- Problem:

$$y^{best} \neq (y^{best}_{\text{head}}, y^{best}_{\text{torso}}, \dots)$$

  independent decisions for each body part are not optimal anymore

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

The general recipe

Normal prediction function, $\mathcal{X} =$ anything, $\mathcal{Y} = \mathbb{R}$

Extract feature vector from x and compute a number from it

$$\text{e.g.} \qquad f(x) = \langle w, \phi(x) \rangle + b$$

The general recipe

Normal prediction function, $\mathcal{X} =$ anything, $\mathcal{Y} = \mathbb{R}$

Extract feature vector from x and compute a number from it

$$\text{e.g.} \qquad f(x) = \langle w, \phi(x) \rangle + b$$

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

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

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

2) Construct $f : \mathcal{X} \to \mathcal{Y}$ from g, e.g., $f(x) = \mathbf{argmax}_{y \in \mathcal{Y}} g(x, y)$

Challenges:

- how to learn $g(x, y)$ from training data?
- how to compute $f(x)$ from $g(x, y)$?

Supervised Learning Problem

- Given training examples $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$
  $x \in \mathcal{X}$: input, e.g. image
  $y \in \mathcal{Y}$: structured output, e.g. human pose, sentence

Images: HumanEva dataset

- How to make predictions for new inputs, i.e. learn a function $f : \mathcal{X} \to \mathcal{Y}$?

Supervised Learning Problem

- Given training examples $(x_1, y_1), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$
  $x \in \mathcal{X}$: input, e.g. image
  $y \in \mathcal{Y}$: structured output, e.g. human pose, sentence
- How to make predictions for new inputs, i.e. learn $f : \mathcal{X} \to \mathcal{Y}$?

Approach 1) Discriminative Probabilistic Learning

1) Use training data to obtain an estimate $p(y|x)$.

2) Use $f(x) = \mathbf{argmin}_{\bar{y} \in \mathcal{Y}} \sum_y p(y|x) \Delta(y, \bar{y})$ to make predictions.

Approach 2) Loss-minimizing Parameter Estimation

1) Use training data to learn a compatibility function $g(x, y)$

2) Use $f(x) := \mathbf{argmax}_{y \in \mathcal{Y}} g(x, y)$ to make predictions.

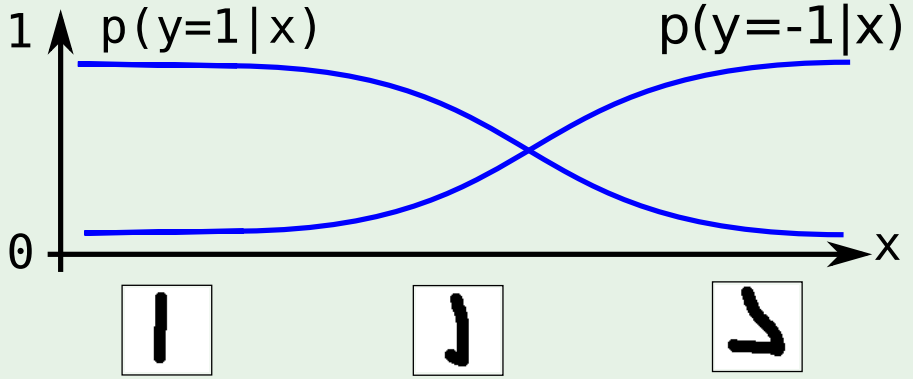
Probabilistic Graphical Models

Refresher: Conditional Probability Distributions

Binary Classification

$\mathcal{X} = \{\text{anything}\}$, $\mathcal{Y} = \{\pm 1\}$

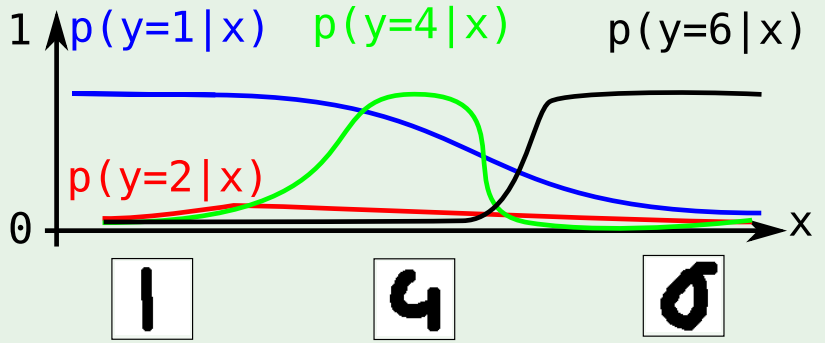
- $p(y|x)$: 2 values for each x, 1 degree of freedom
- learn *one function*: $\mathcal{X} \to \mathbb{R}$



Multi-class prediction

$y \in \mathcal{Y} = \{1, \ldots, K\}$

- $p(y|x)$: K values for each x,
- learn $K - 1$ functions, or K functions with normalization

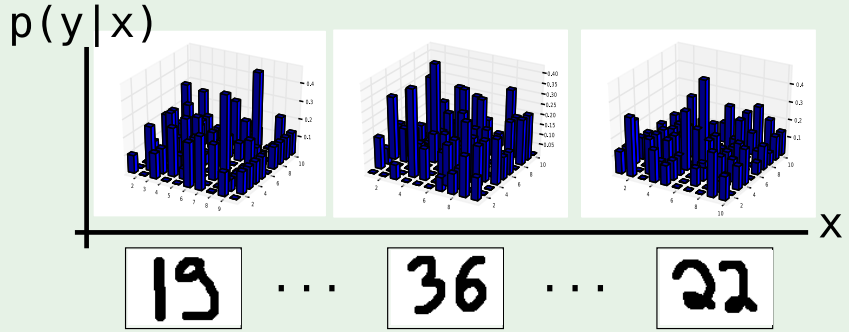


Structured objects: predicting M variables jointly

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

For each x:

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



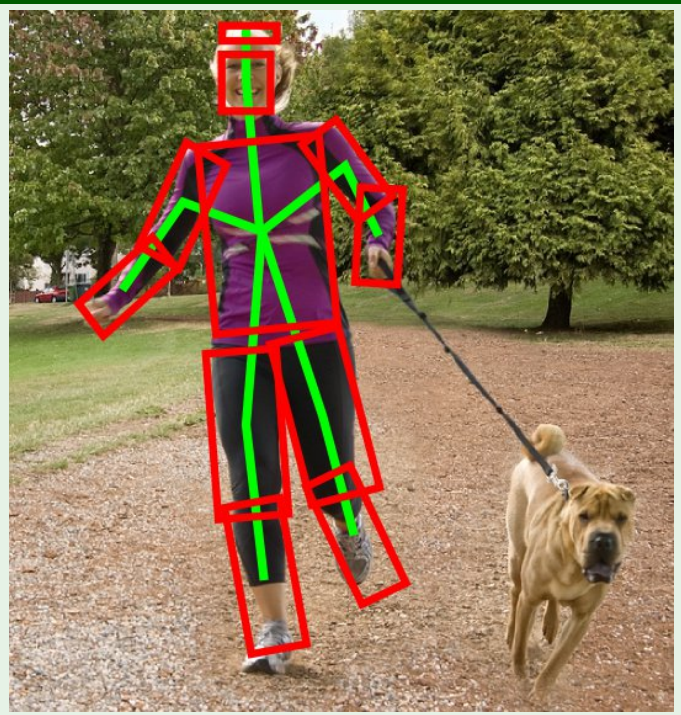
Example: pose estimation

$$\mathcal{Y}_{\text{part}} = \{1, \ldots, W\} \times \{1, \ldots, H\} \times \{1, \ldots, 360\}$$

$$\mathcal{Y} = \mathcal{Y}_{\text{head}} \times \mathcal{Y}_{\text{left-arm}} \times \cdots \times \mathcal{Y}_{\text{right-foot}}$$

For each x:

- $(360WH)^{\#\text{body parts}}$ values
  $\rightarrow$ many billions function



Example: image denoising

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

For each x:

- $(255^3)^{640 \cdot 480}$ values in $p(y|x)$,
  $\rightarrow$ over $10^{2,000,000}$ functions

too much!

Example: image denoising

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

For each x:

- $(255^3)^{640 \cdot 480}$ values in $p(y|x)$,
  $\rightarrow$ over $10^{2,000,000}$ 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.

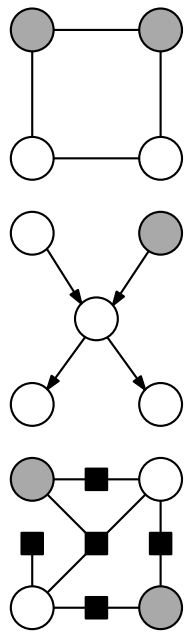
Probabilistic Graphical Models

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.

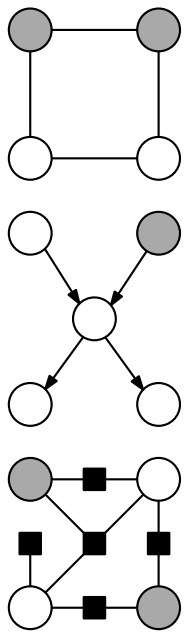


Probabilistic Graphical Models

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.

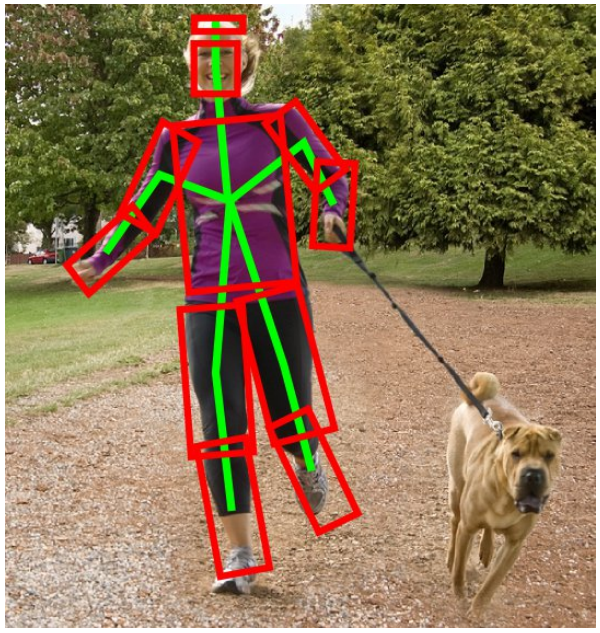
The graph encodes conditional independence assumptions between the variables:

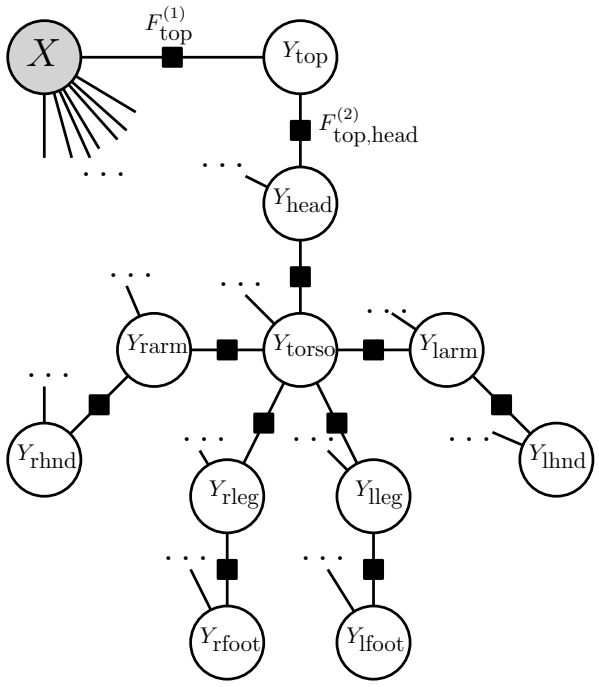
- Let $N(i)$ be the neighbors of node i in the graph $(V, \mathcal{E})$. Then

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

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

Example: Pictorial Structures for Articulated Pose Estimation



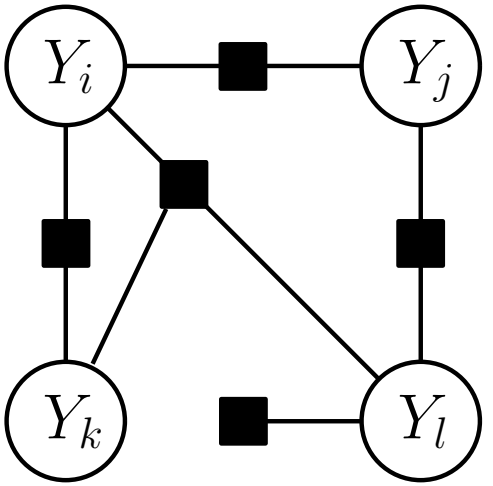


- All parts depend on each other.
  - Knowing where the head is puts constraints on where the feet can be.
- But **conditional independences** as specified by the graph:
  - If we *fix* where the **left leg** is, the **left foot**'s position does not depend on the **torso** or the **head** position anymore,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

$$p(y_{\text{left-foot}}|y_{\text{top}}, \ldots, y_{\text{torso}}, \ldots, y_{\text{right-foot}}, x) = p(y_{\text{left-foot}}|y_{\text{left-leg}}, x)$$

Factor Graphs

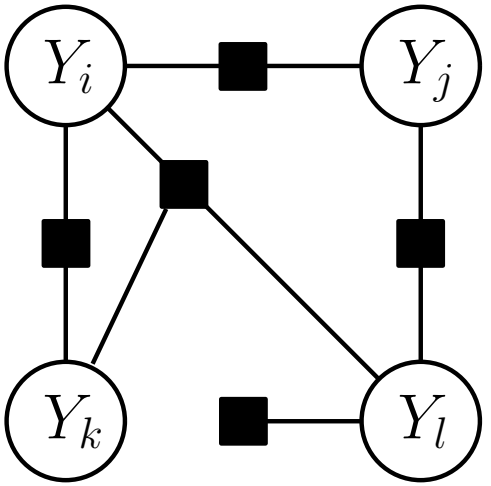
- Decomposable output $y = (y_1, \ldots, y_{|V|})$
- Graph: $G = (V, \mathcal{F})$,
  - variable nodes V,
  - factor nodes $\mathcal{F}$,
  - each factor $F \in \mathcal{F}$ connects a subset of nodes,
  - write $F = \{v_1, \ldots, v_{|F|}\}$ and $y_F = (y_{v_1}, \ldots, y_{v_{|F|}})$



Factor graph

Factor Graphs

- Decomposable output $y = (y_1, \ldots, y_{|V|})$
- Graph: $G = (V, \mathcal{F})$,
  - variable nodes V,
  - factor nodes $\mathcal{F}$,
  - 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

- Distribution factorizes 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**:

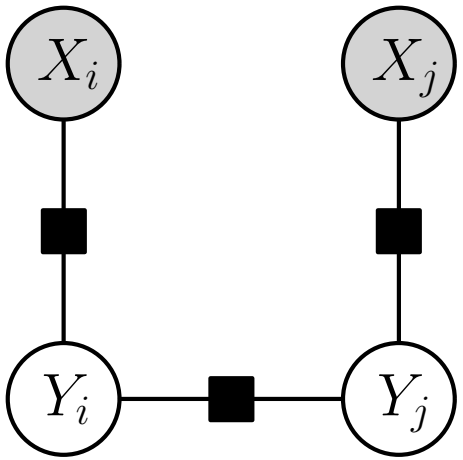
$$Z = \sum_{y \in \mathcal{Y}} \prod_{F \in \mathcal{F}} \psi_F(y_F).$$

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 $\psi_F(y_F)$

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



Factor graph

- 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.

Conditional random fields (CRFs)

Conventions: Potentials and Energy Functions

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

- instead of potentials, we can use **energies**:

$$E_F(y_F; x_F) = -\log(\psi_F(y_F; x_F)) \quad \text{for each factor } F.$$

$$E(y; x) = \sum_{F \in \mathcal{F}} E_F(y_F; x_F) \qquad \text{total energy}$$

Conventions: Potentials and Energy Functions

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

- instead of potentials, we can use **energies**:

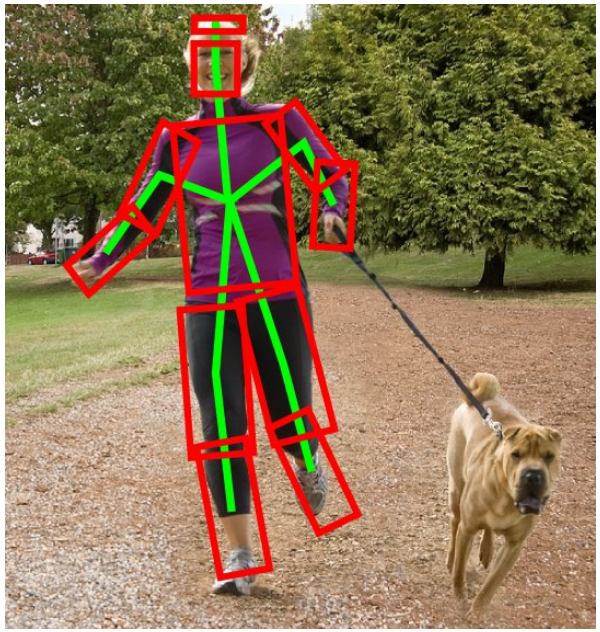
$$E_F(y_F; x_F) = -\log(\psi_F(y_F; x_F)) \quad \text{for each factor } F.$$
$$E(y; x) = \sum_{F \in \mathcal{F}} E_F(y_F; x_F) \qquad \text{total energy}$$

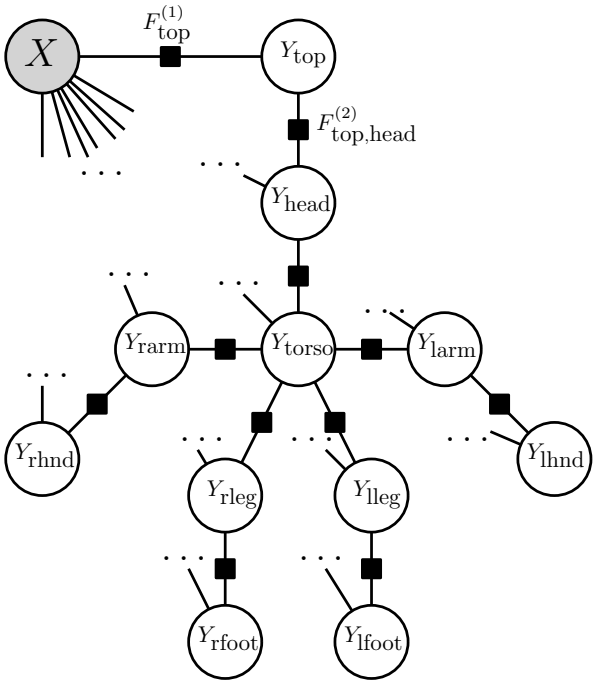
- $p(y|x)$ can be written as **Gibbs distribution**

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

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

Example: An Energy Function for Human Pose Estimation





$$E(y;x) = \sum_{i \in \{\text{head,torso,}\dots\}} E_i(y_i;x_i) + \sum_{(i,j)} E_{ij}(y_i,y_j)$$

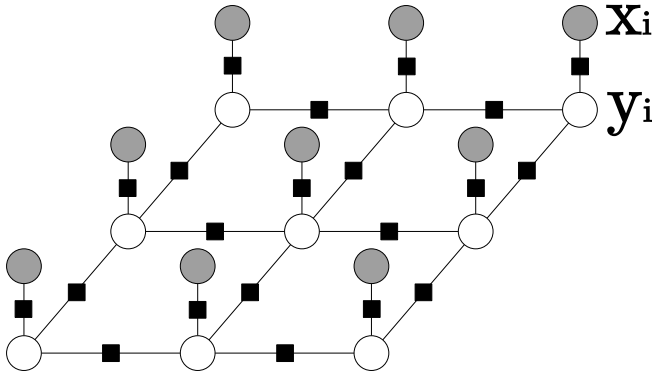
- unary factors (depend on one label): appearance
  - e.g. $E_{\text{head}}(y;x)$ *"Does location y in image x look like a head?"*
- pairwise factors (depend on two labels): geometry
  - e.g. $E_{\text{head-torso}}(y_{\text{head}}, y_{\text{torso}})$ *"Is location y_{head} above location y_{torso}?"*

Example: An Energy Function for Image Segmentation

Object segmentation: e.g. horse

$\mathcal{X}$: 

$\mathcal{Y}$: 



Energy function components ("Ising" model):

- $E_i(y_i = 1, x_i) = \begin{cases} \text{low} & \text{if } x_i \text{ is the right color, e.g. } \texttt{brown} \\ \text{high} & \text{otherwise} \end{cases}$
- $E_i(y_i = 0, x_i) = -E_i(y_i = 1, x_i)$
- $E_i(y_i, y_j) = \begin{cases} \text{low} & \text{if } y_i = y_j \\ \text{high} & \text{otherwise} \end{cases}$

  prefer that neighbors have the same label $\rightarrow$ smooth labelings

What to do with Structured Prediction Models?

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

MAP Prediction

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

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

Probabilistic Inference

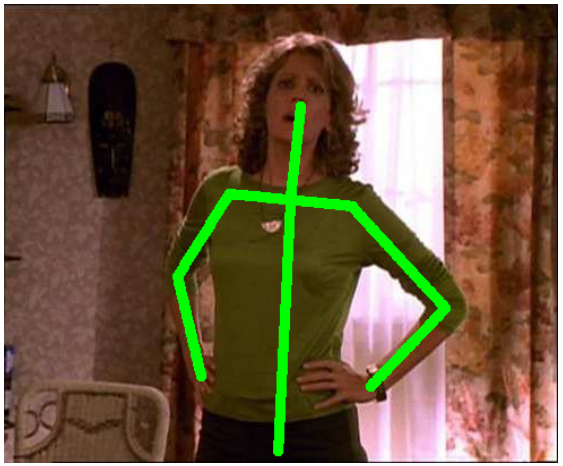
Compute marginal probabilities

$$p(y_F|x)$$

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

What to do with Structured Prediction Models?

input image



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

$p(y_i|x)$ for $i = 1, \ldots, 6$

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

  of the individual body parts.

Images: Buffy Stickmen dataset (Ferrari *et al.*)

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 a fixed factor graph, learn parameters of the energy.

Conditional Random Fields

$$\mathbf{max}_w \;\; p(y|x;w)$$

Conditional Random Field Learning

Goal: learn a conditional distribution

$$p(y|x) = \frac{1}{Z(x)} e^{-\sum_{F \in \mathcal{F}} E_F(y_F; x)\rangle}$$

with $\mathcal{F} = \{$ all factors $\}$: all unary, pairwise, potentially higher order, ...

- parameterize each $E_F(y_F; x) = \langle w_F, \phi_F(x, y_F)\rangle$.
- fixed feature functions $(\, \phi_1(x, y_1), \ldots, \phi_{|\mathcal{F}|}(x, y_{|\mathcal{F}|}) \,) \equiv: \phi(x, y)$
- weight vectors $(\, w_1, \ldots, w_{|\mathcal{F}|} \,) \equiv: w$

Result: log-linear model with parameter vector w

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

$$\text{with} \qquad Z(x; w) = \sum_{\bar{y} \in \mathcal{Y}} e^{-\langle w, \phi(\bar{y},x)\rangle} \quad \text{("partition function")}$$

New goal: find best parameter vector $w \in \mathbb{R}^D$.

Probabilistic Learning

Maximize conditional likelihood, $p(\mathcal{D}_y|\mathcal{D}_x; w)$, or maximum posterior, $p(w|\mathcal{D})$. Equivalently, minimize

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

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

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

- unconstrained
- smooth
- convex

Solving the Training Optimization Problem in Practice

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

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

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

Solving the Training Optimization Problem in Practice

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

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

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

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

- binary image segmentation: $|\mathcal{Y}| = 2^{640 \times 480} \approx 10^{92475}$
- ranking N images: $|\mathcal{Y}| = N!$, e.g. $N = 1000$: $|\mathcal{Y}| \approx 10^{2568}$.

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

Solving the Training Optimization Problem in Practice

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

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

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

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 variables $\approx$ 10s to 1,000,000s
- K: number of possible labels of each output variables $\approx$ 2 to 1000s

Solving the Training Optimization Problem in Practice

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

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

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

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

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

- are much smaller than complete joint distribution $p(y|x; w)$,
- compute/approximate them by probabilistic inference

Probabilistic Inference

$$p(y_F|x)$$

Probabilistic Inference – Overview

Goal: for fixed model and x, compute $Z(x)$ or $p(y_F|x; w)$

Exact Inference

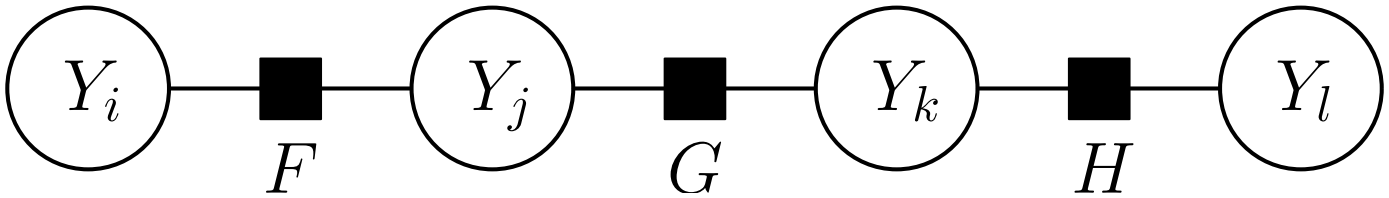
- Belief Propagation on chains
- Belief Propagation on trees
- Junction tree algorithm

Approximate Inference

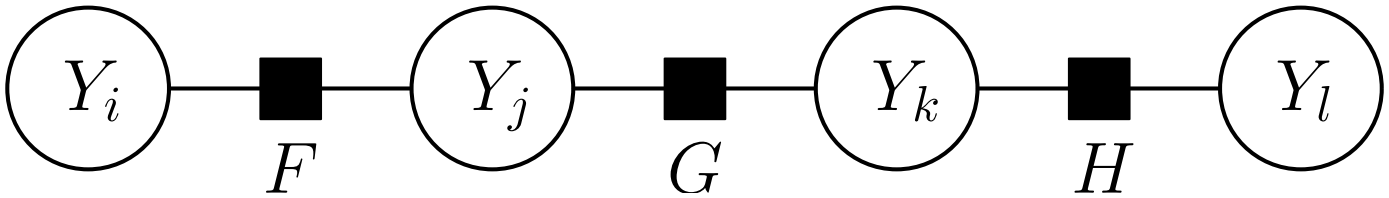
- Loopy Belief Propagation
- Markov Chain Monte Carlo (MCMC) Sampling
- Variational Inference / Mean Field

Probabilistic Inference – Belief Propagation

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:

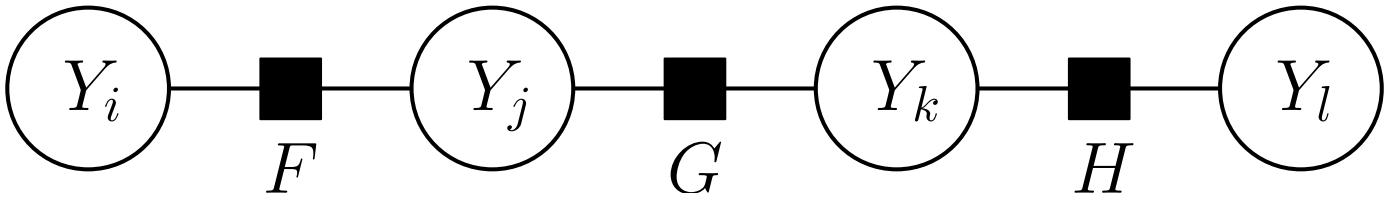


Task 1: for any $y \in \mathcal{Y}$, compute $p(y|x)$, using

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

Probabilistic Inference – Belief Propagation

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

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

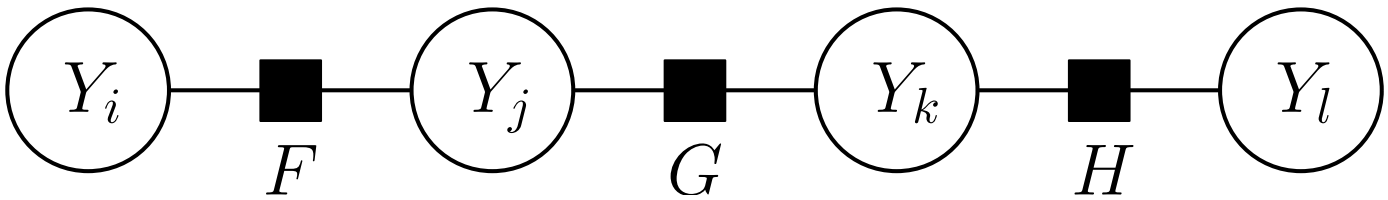
Problem: We don't know $Z(x)$, and computing it using

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

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

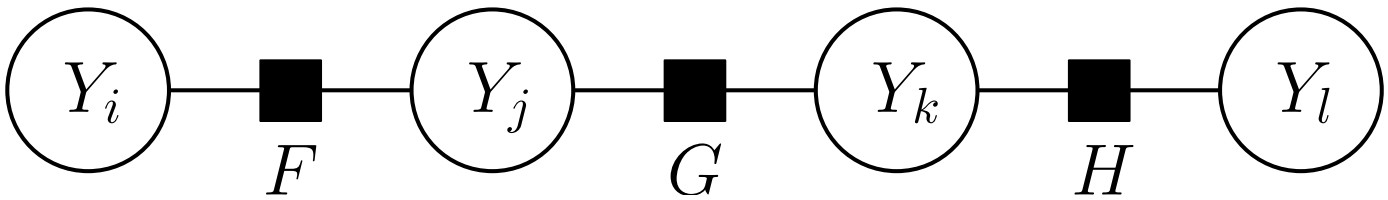
A lot research has been done on how to **efficiently compute** $Z(x)$.

Probabilistic Inference – Belief Propagation



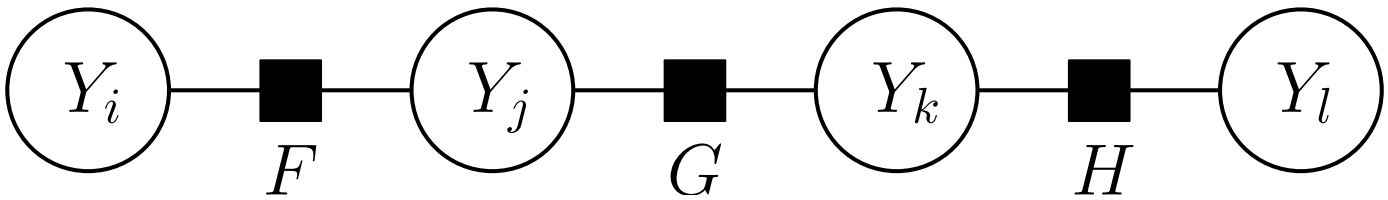
For notational simplicity, we drop the dependence on (fixed) x:

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



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

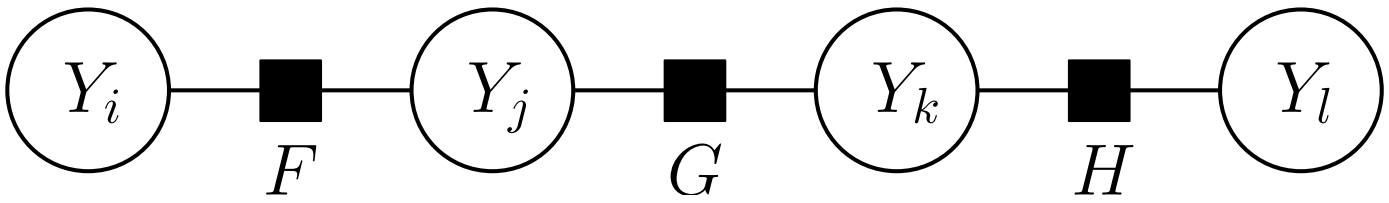
$$
\begin{aligned}
Z &= \sum_{y \in \mathcal{Y}} e^{-E(y)} \\
&= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} e^{-E(y_i, y_j, y_k, y_l)}
\end{aligned}
$$



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

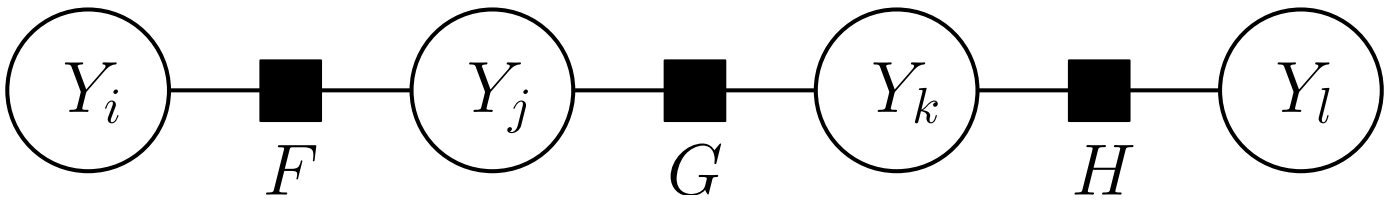
$$
\begin{aligned}
Z &= \sum_{y \in \mathcal{Y}} e^{-E(y)} \\
&= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} e^{-E(y_i, y_j, y_k, y_l)} \\
&= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k}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

Probabilistic Inference – Belief Propagation



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

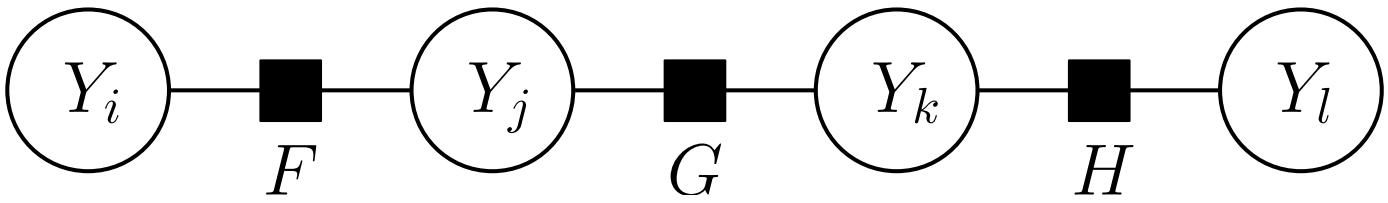
Probabilistic Inference – Belief Propagation



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

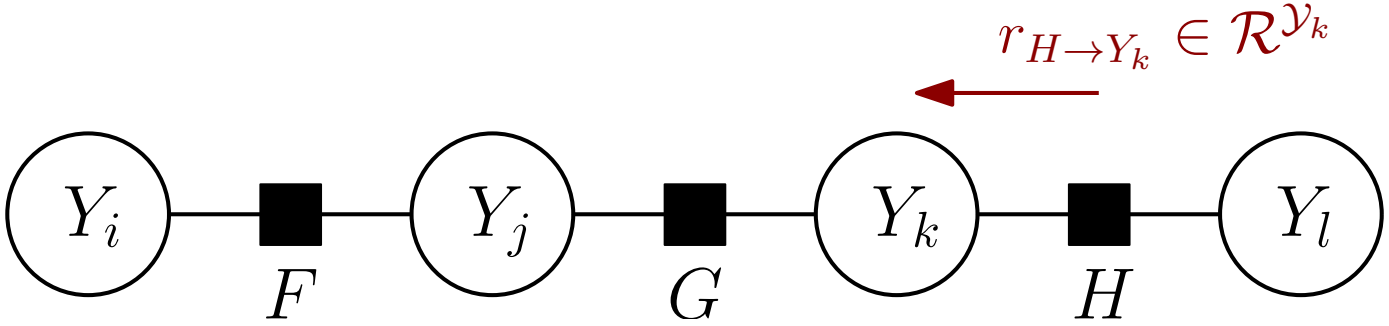
$$= \sum_{y_i} \sum_{y_j} \sum_{y_k} \sum_{y_l} e^{-E_F(y_i, y_j)} e^{-E_G(y_j, y_k)} e^{-E_H(y_k, y_l)}$$

Probabilistic Inference – Belief Propagation



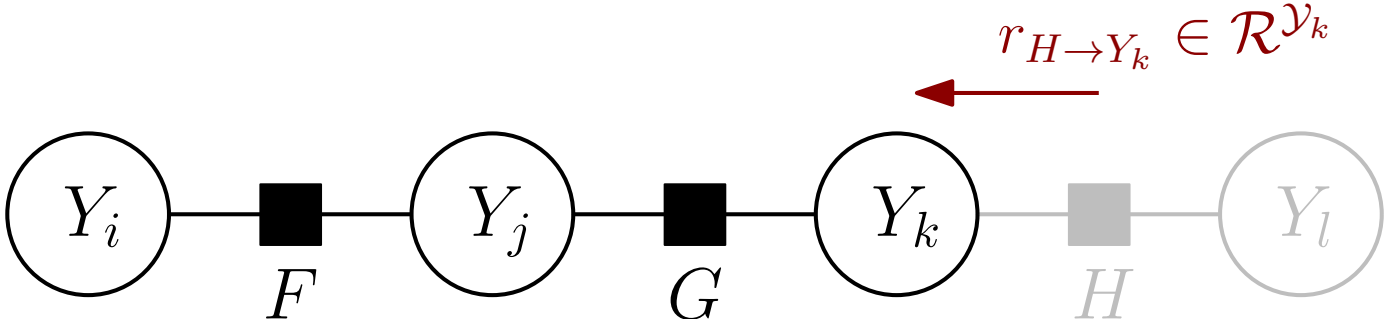
$$
\begin{aligned}
Z &= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} e^{-E_F(y_i,y_j)-E_G(y_j,y_k)-E_H(y_k,y_l)} \\
&= \sum_{y_i} \sum_{y_j} \sum_{y_k} \sum_{y_l} e^{-E_F(y_i,y_j)} e^{-E_G(y_j,y_k)} e^{-E_H(y_k,y_l)} \\
&= \sum_{y_i}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

Probabilistic Inference – Belief Propagation



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

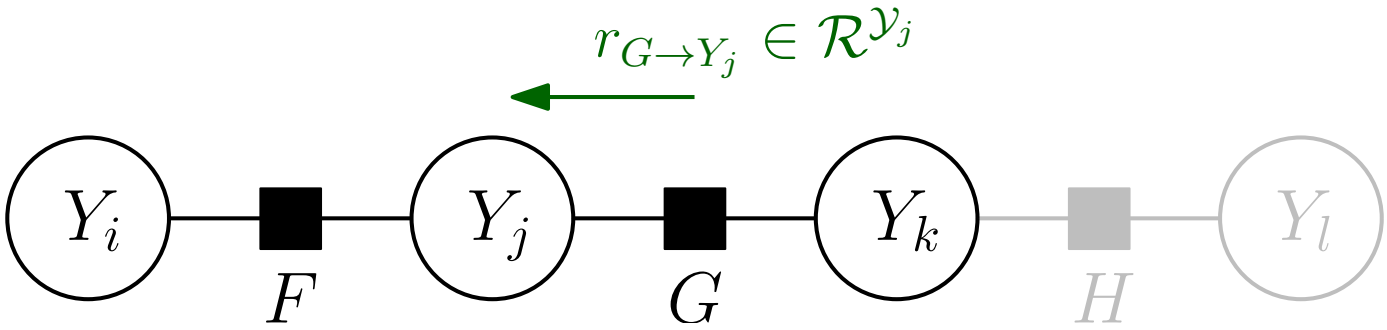
Probabilistic Inference – Belief Propagation



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

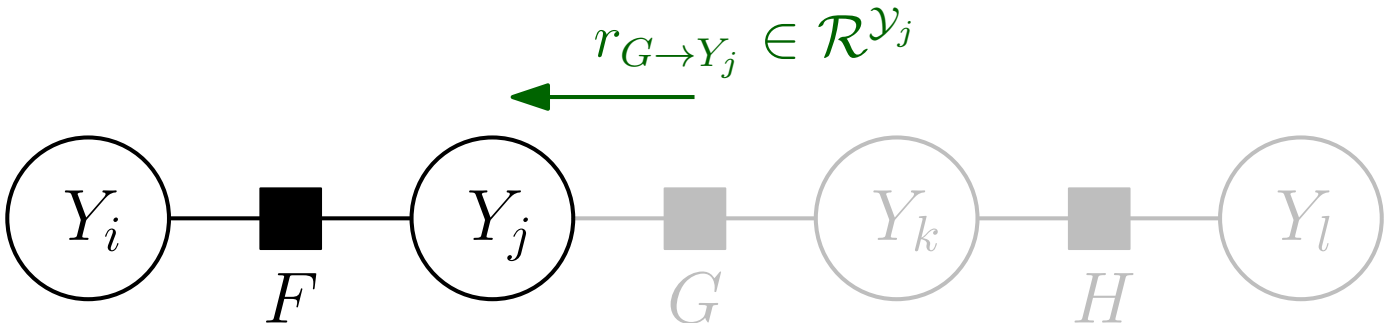
$$= \sum_{y_i} \sum_{y_j} e^{-E_F(y_i, y_j)} \sum_{y_k} e^{-E_G(y_j, y_k)} r_{H \to Y_k}(y_k)$$

Probabilistic Inference – Belief Propagation



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

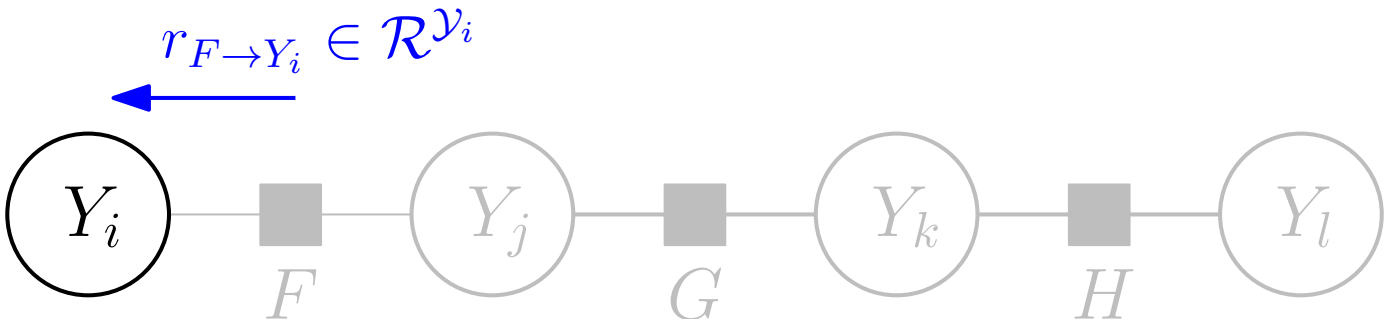
Probabilistic Inference – Belief Propagation



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

$$= \sum_{y_i} \underbrace{\sum_{y_j} e^{-E_F(y_i, y_j)} r_{G \to Y_j}(y_j)}_{r_{F \to Y_i}(y_i)}$$

Probabilistic Inference – Belief Propagation

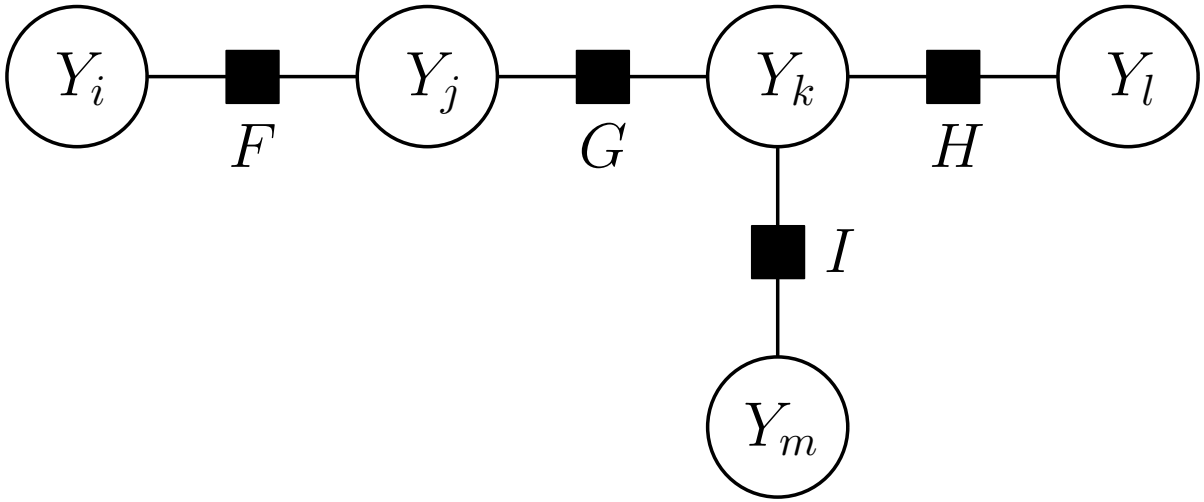


$$Z = \sum_{y_i} \sum_{y_j} e^{-E_F(y_i, y_j)} \underbrace{\sum_{y_k} e^{-E_G(y_j, y_k)} r_{H \to Y_k}(y_k)}_{r_{G \to Y_j}(y_j)}$$

$$= \sum_{y_i} \underbrace{\sum_{y_j} e^{-E_F(y_i, y_j)} r_{G \to Y_j}(y_j)}_{r_{F \to Y_i}(y_i)}$$

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

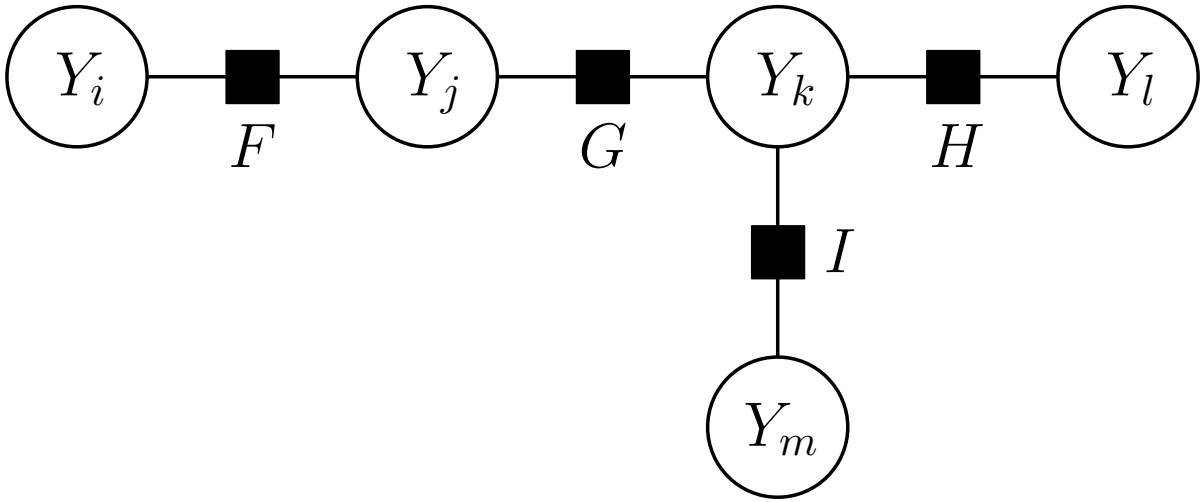
Example: Inference on Trees



1) pick a root (here: i)
2) and sort sums such that parents nodes are left of their children

$$
\begin{aligned}
Z &= \sum_{y \in \mathcal{Y}} e^{-E(y)} \\
&= \sum_{y_i \in \mathcal{Y}_i} \sum_{y_j \in \mathcal{Y}_j} \sum_{y_k \in \mathcal{Y}_k} \sum_{y_l \in \mathcal{Y}_l} \sum_{y_m \in \mathcal{Y}_m} e^{-E_F(y_i, y_j) - \ldots - E_I(y_k, y_m)}
\end{aligned}
$$

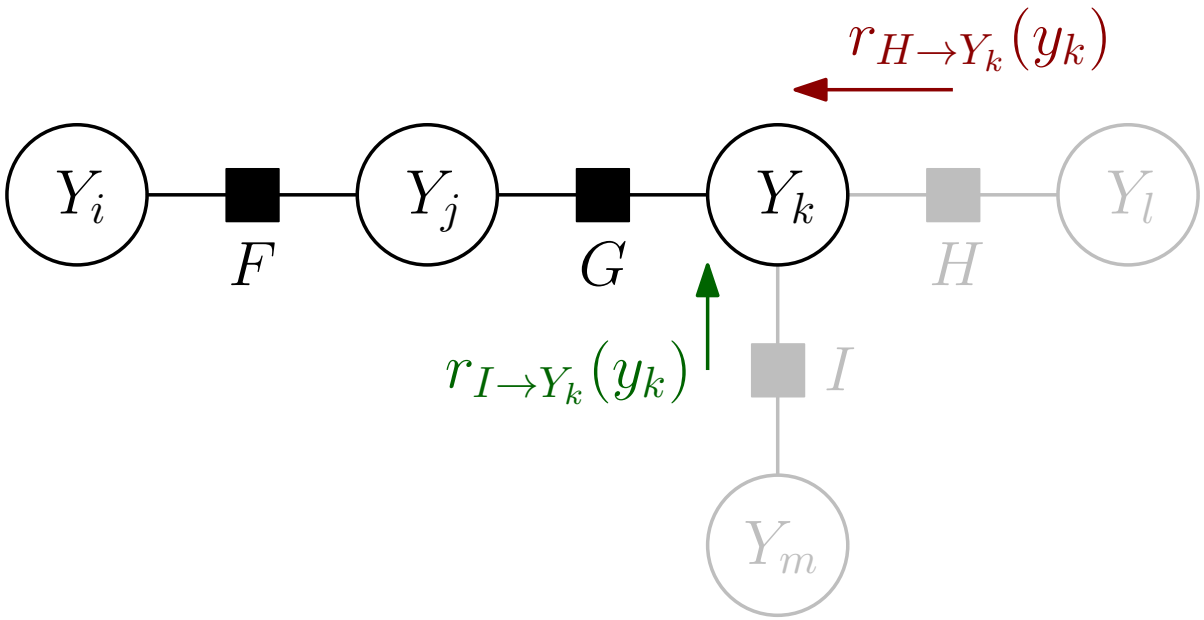
Example: Inference on Trees



1) pick a root (here: i)
2) and sort sums such that parents nodes are left of their children

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

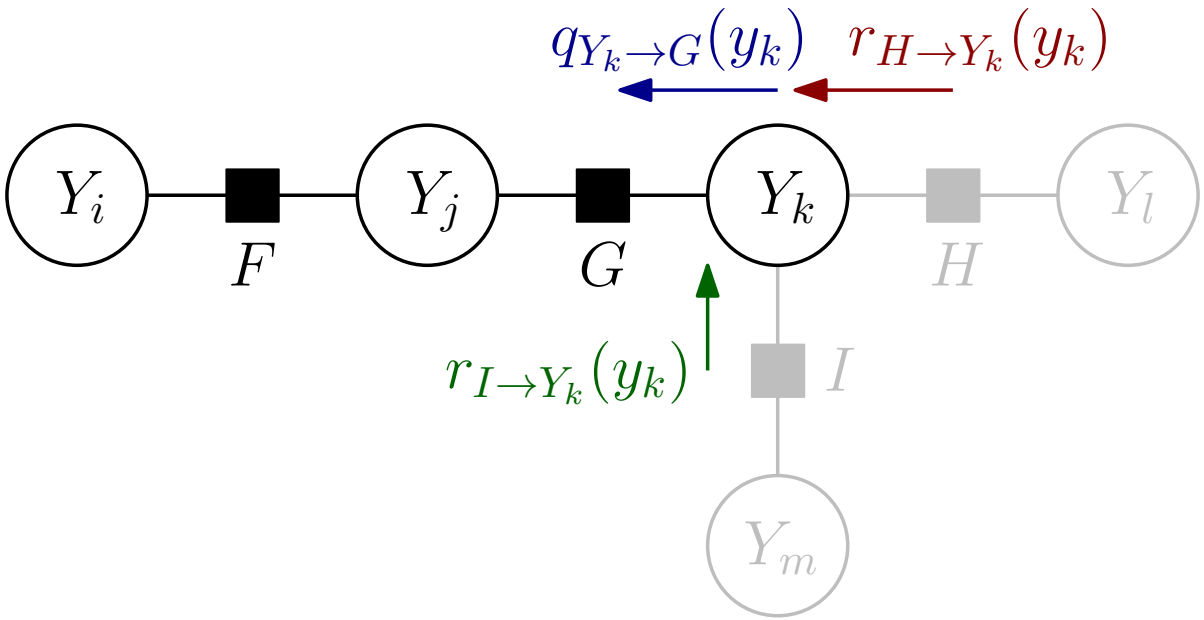
Example: Inference on Trees



1) pick a root (here: i)
2) and sort sums such that parents nodes are left of their children

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

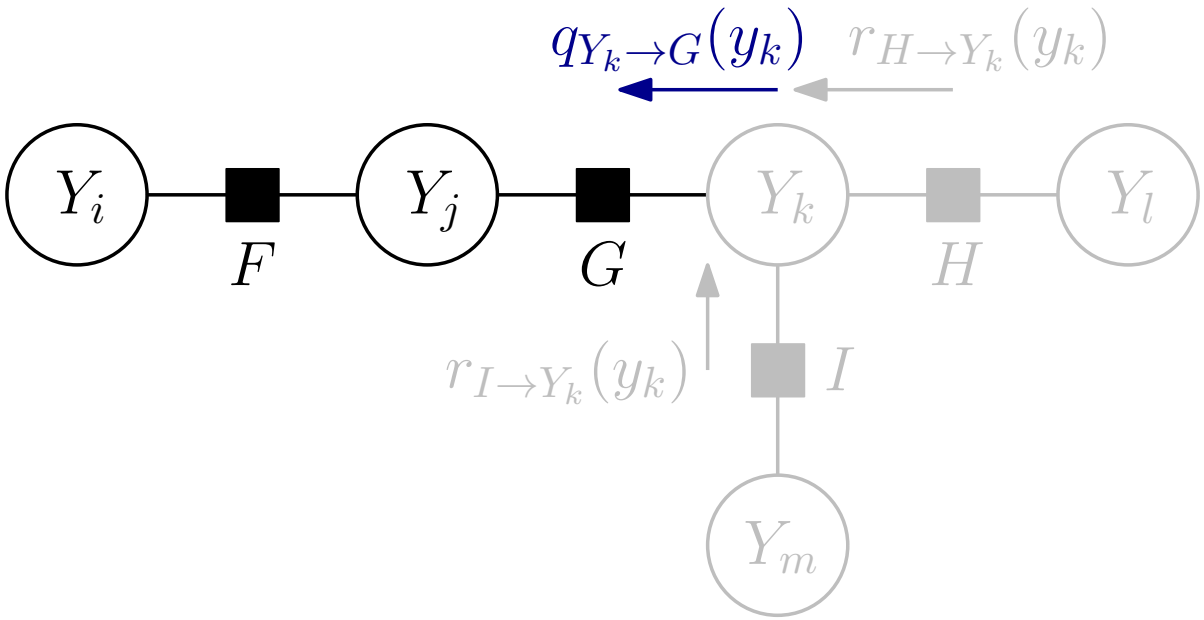
Example: Inference on Trees



1) pick a root (here: i)
2) and sort sums such that parents nodes are left of their children

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

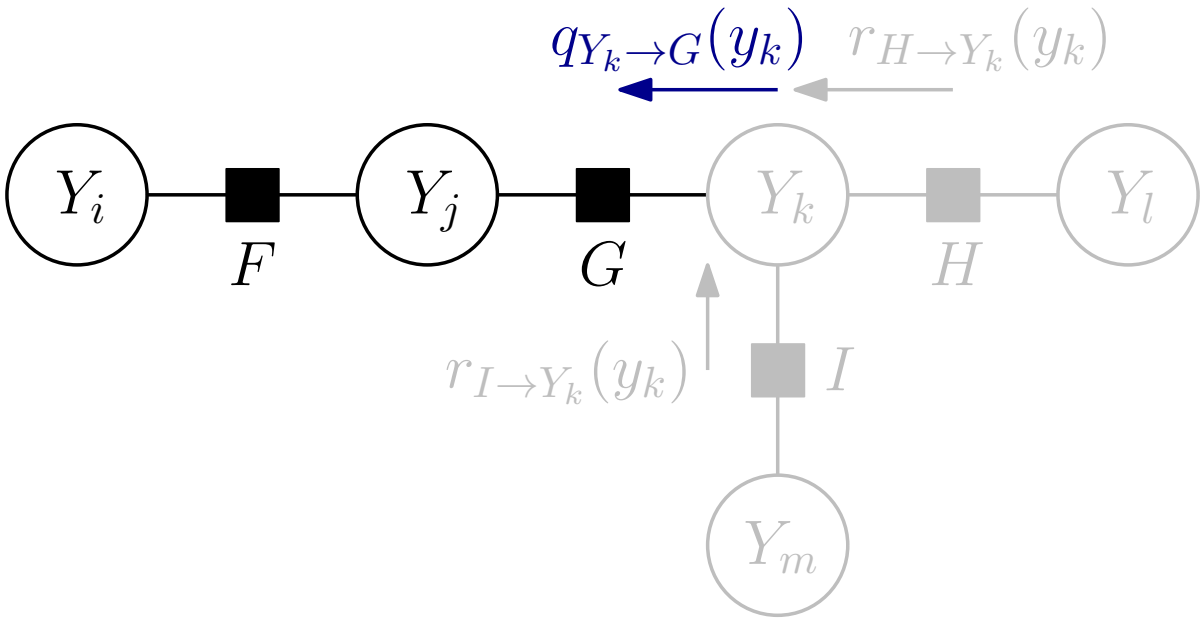
Example: Inference on Trees



1) pick a root (here: i)
2) and sort sums such that parents nodes are left of their children

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

Example: Inference on Trees



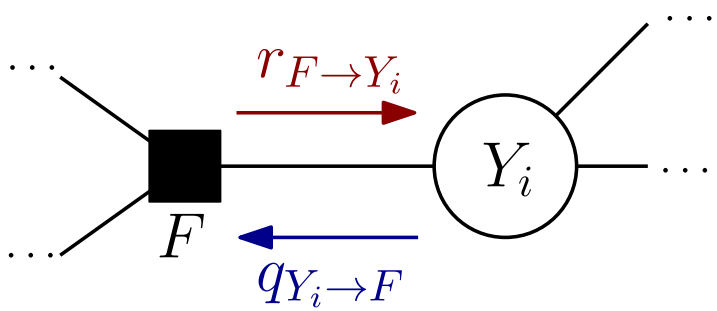
1) pick a root (here: i)
2) and sort sums such that parents nodes are left of their children

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

3) etc.

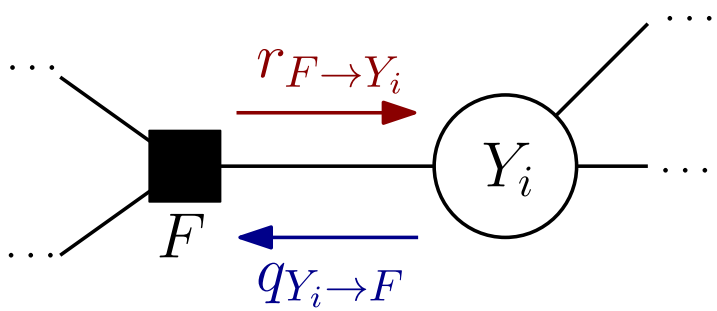
Factor Graph Sum-Product Algorithm

- "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



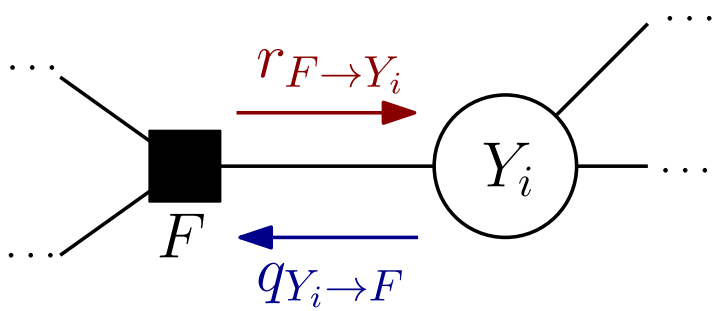
- Algorithm iteratively updates messages
- After convergence: Z and $p(y_F)$ can be obtained from the messages, e.g.

$$p(Y_i = y_i) \propto \prod_{F:(i,F)\in\mathcal{E}} r_{F \to Y_i}(y_i)$$

(Sum-Product) Belief Propagation

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



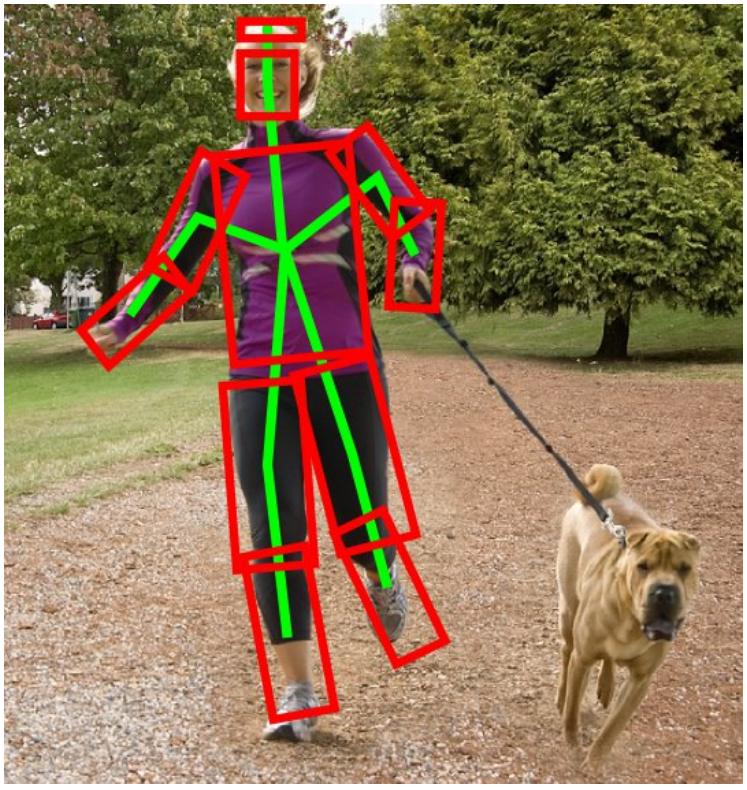
- Algorithm iteratively updates messages
- After convergence: Z and $p(y_F)$ can be obtained from the messages, e.g.

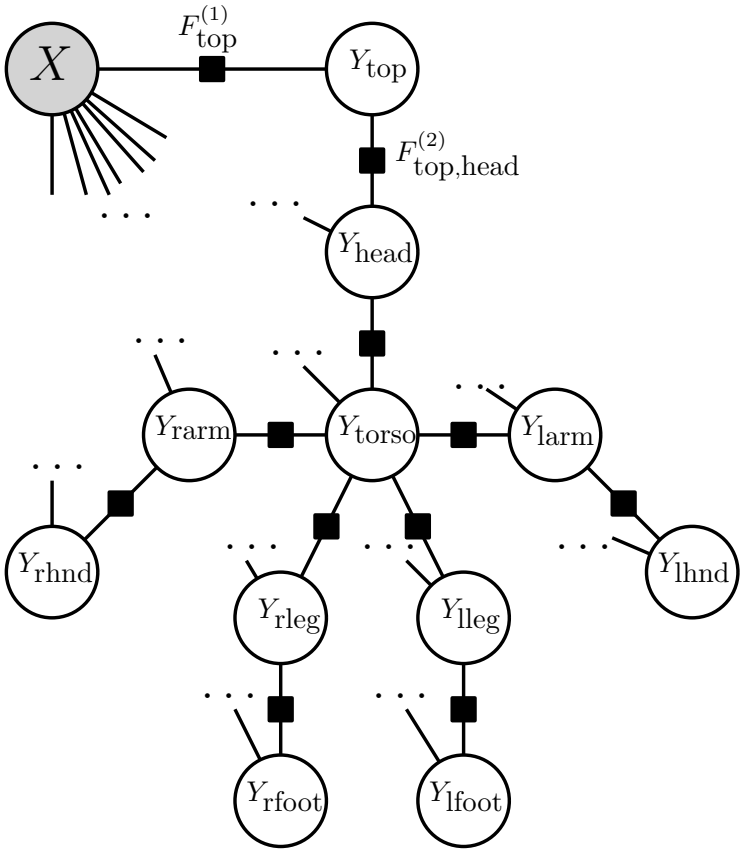
$$p(Y_i = y_i) \propto \prod_{F:(i,F)\in\mathcal{E}} r_{F \to Y_i}(y_i)$$

(Sum-Product) Belief Propagation

- Easier to implement than to explain...

Example: Pictorial Structures





- **Tree-structured model** for articulated pose
  (Felzenszwalb and Huttenlocher, 2000), (Fischler and Elschlager, 1973)
- Belief propagation is the state-of-the-art for prediction and inference

Example: Pictorial Structures

Probability of part states $\equiv$ body part locations:

estimated independently

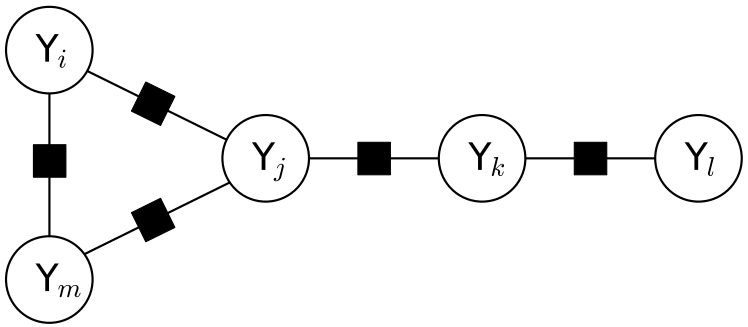
estimated from joint probability

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

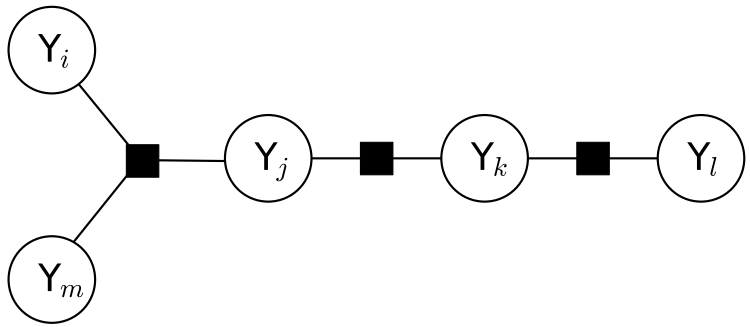
  of the body parts, taking into account also the other body parts.

Belief Propagation in Cyclic Graphs

Belief propagation does not work for



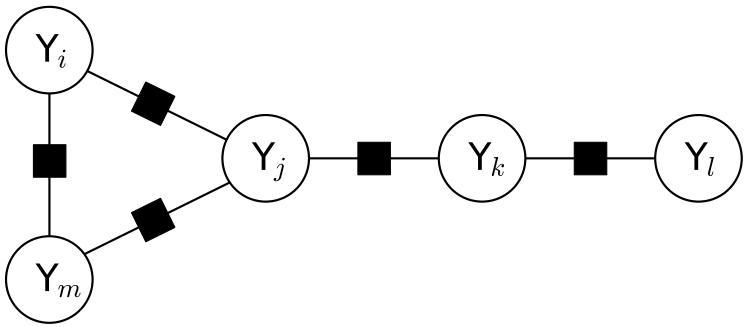
graph with cycles



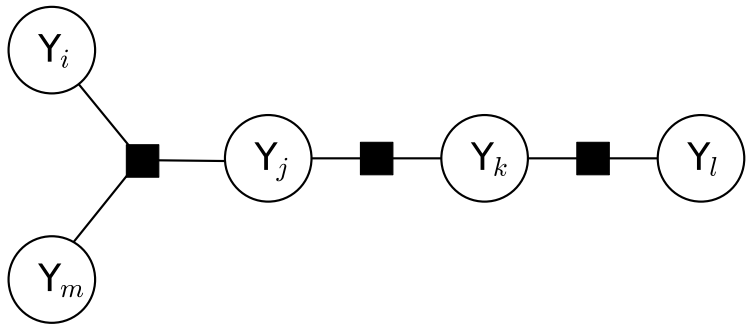
graph with factors of size more than 2

Belief Propagation in Cyclic Graphs

Belief propagation does not work for

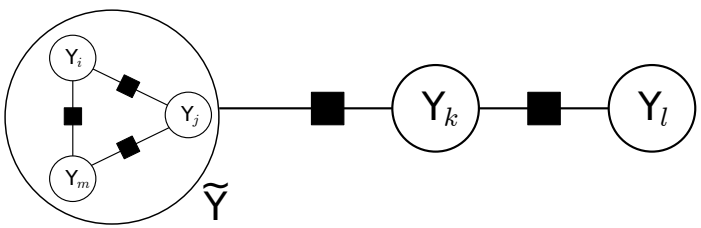


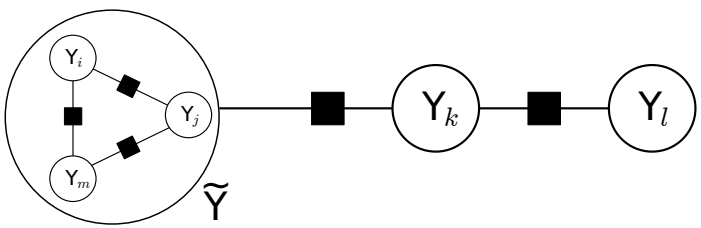
graph with cycles



graph with factors of size more than 2

We can construct equivalent chain/tree models:





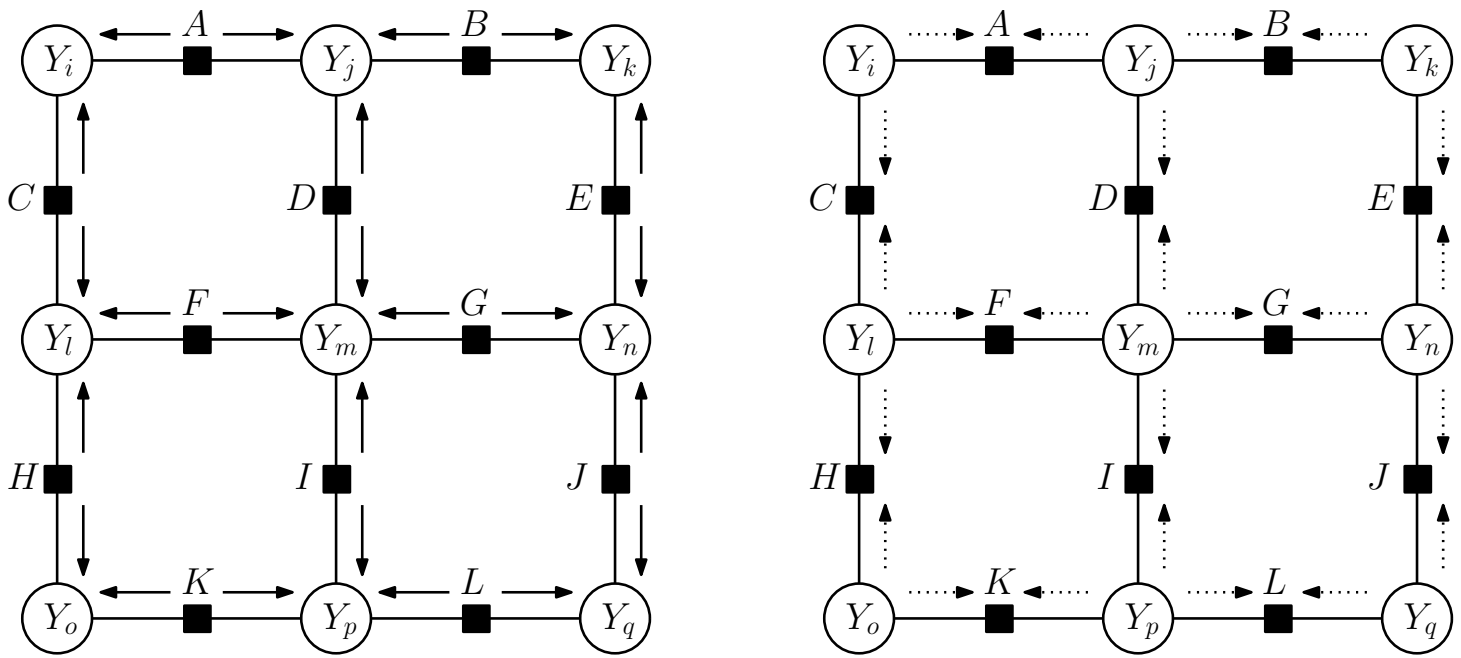
$$\tilde{Y} = (Y_i, Y_j, Y_m) \text{ with state space } \tilde{\mathcal{Y}} = \mathcal{Y}_i \times \mathcal{Y}_j \times \mathcal{Y}_m$$

General procedure: **junction tree algorithm**

Problem: exponentially growing state space $\rightarrow$ BP inefficient

Belief Propagation in Cyclic Graphs

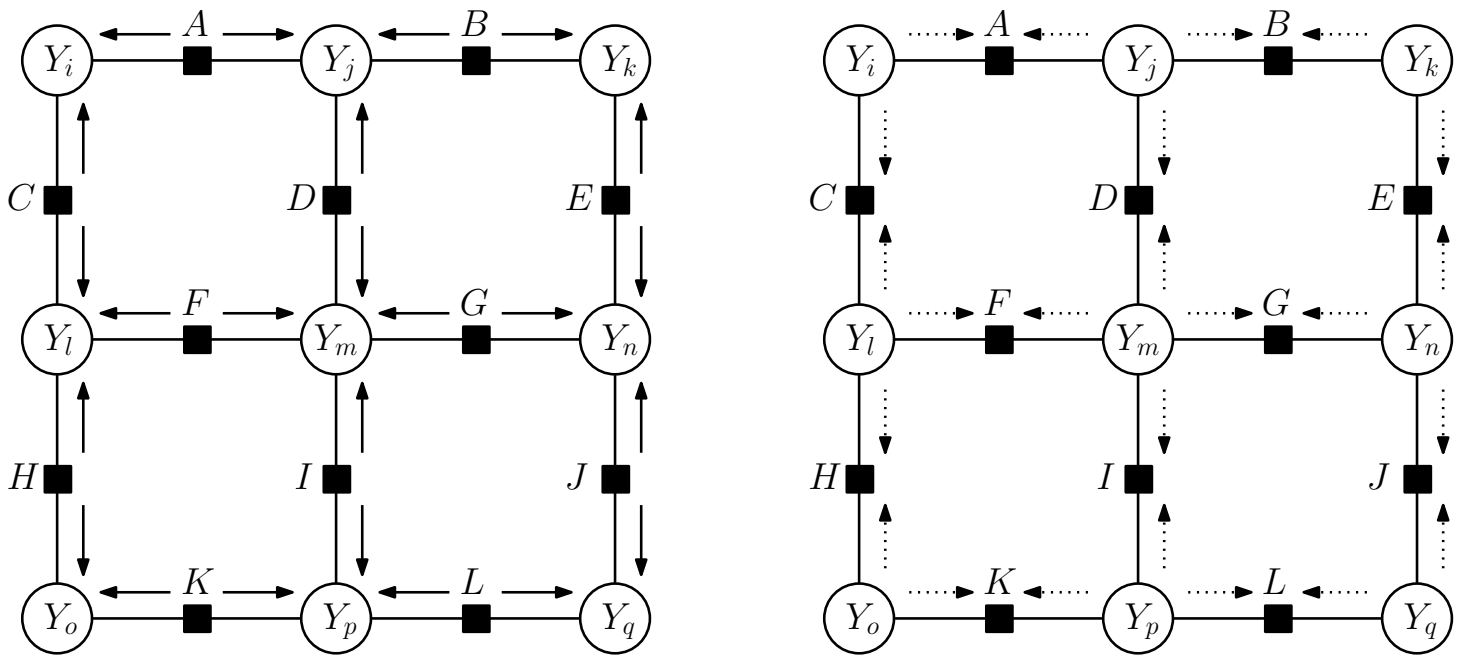
What if we do belief propagation even though the graphs has cycles?



Problem: There is no well-defined *leaf–to–root* order $\rightarrow$ where to start?

Belief Propagation in Cyclic Graphs

What if we do belief propagation even though the graphs has cycles?

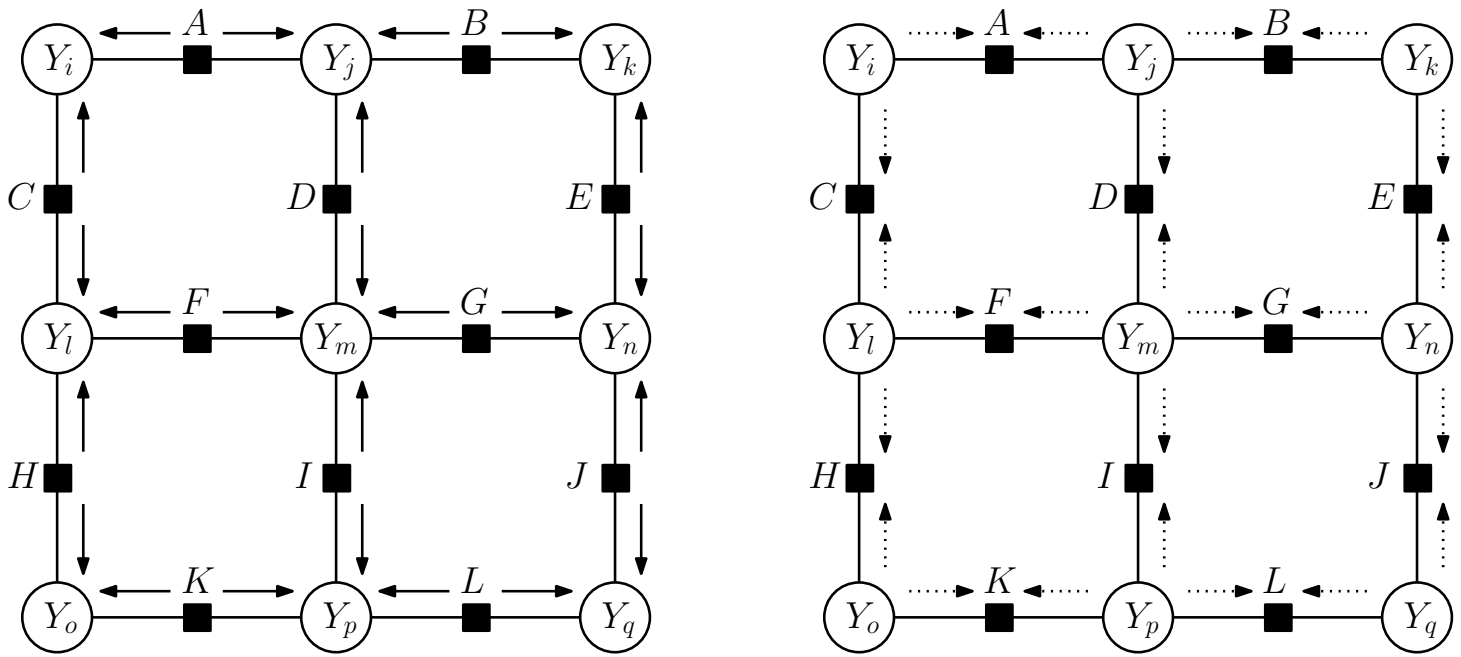


Problem: There is no well-defined *leaf–to–root* order $\rightarrow$ where to start?

Loopy Belief Propagation (LBP)

- initialize all messages as constant 1
- pass messages using rules of BP until a stop criterion

Belief Propagation in Cyclic Graphs



Problems:

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

Several improved schemes exist, some even convergent (but approximate)

Exact inference in general cyclic graph is **#P-hard**.

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

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

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

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

For probabilistic inference, this step is easy.

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

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 size of $\mathcal{Y}$.

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 size of $\mathcal{Y}$.

Problem:

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

Solution:

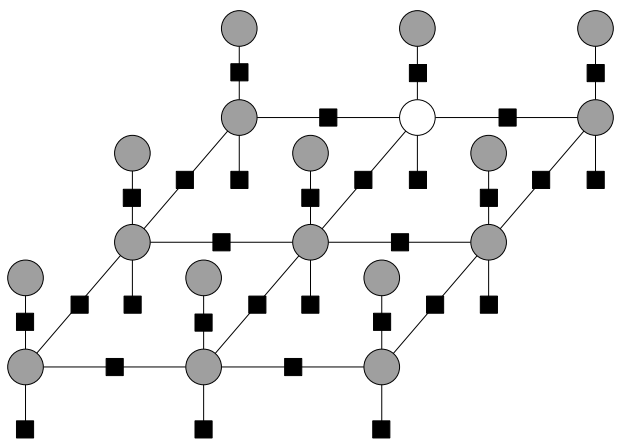
- We can get away with a sequence of dependent samples

Monte-Carlo Markov Chain (MCMC) sampling

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

- Initialize $y^{(1)} = (y_1, \ldots, y_d)$ arbitrarily
- For $s = 1, \ldots, S$:
  1. Select an index i,
  2. Re-sample $y_i \sim p(y_i | y^{(s)}_{V \setminus \{i\}}, x)$.
  3. Output sample $y^{(s+1)} = (y^{(s)}_1, \ldots, y^{(s)}_{i-1},\ y_i,\ y^{(s)}_{i+1}, \ldots, y^{(s)}_d)$

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



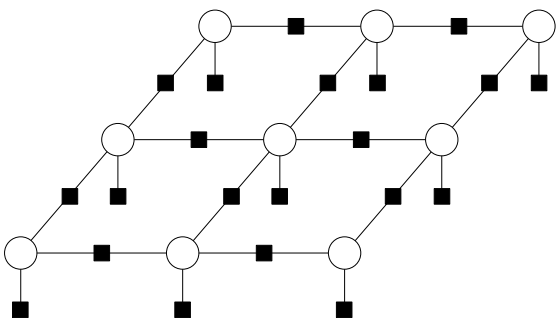
Task: Compute marginals $p(y_F|x)$ for general $p(y|x)$

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

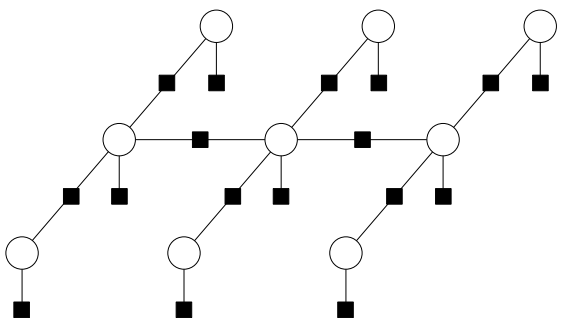
$$q^* = \underset{q \in \mathcal{Q}}{\mathbf{argmin}} \; D_{KL}(q(y) \| p(y|x))$$

$$p(y_F|x) \approx q^*(y_F)$$

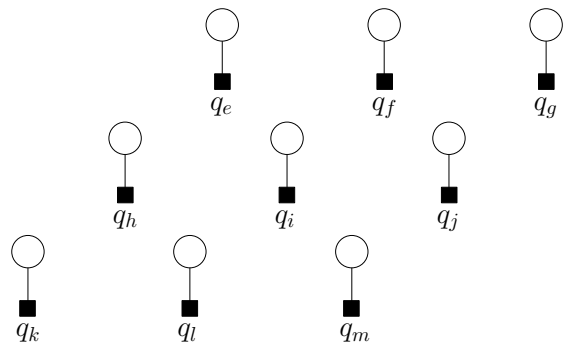
For example



original model



tree approximation



product of unary factors

Special case: **(Naive) Mean Field** for $p(y|x) = \frac{1}{Z(x)} e^{-E(y,x)}$

$$p(y|x) \approx q(y) = \prod_{i \in V} q_i(y_i)$$

No closed for expression for q^*, but optimality condition:

$$q_i^*(y_i) \propto e^{-\mathbb{E}_{y \setminus \{y_i\} \sim Q}\{E(y,x)\}}$$

$$\text{for} \quad Q(y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_n) = \prod_{j \neq i} q_j^*(y_j)$$

Iterative scheme:

- initialize q_i (e.g. uniform)
- repeat until convergence
  - for $i \in V$ in any order:
    - update q_i while keeping the others fixed

Probabilistic Inference – Summary

Task: compute (marginal) probabilities $p(y_F|x)$

Exact Probabilistic Inference

Only possible for certain models:

- trees/forests: sum-product belief propagation
- general graphs: junction chain algorithm (if tractable)

Approximate Probabilistic Inference

Many techniques with different properties and guarantees:

- loopy belief propagation
- MCMC sampling (e.g. Gibbs sampling)
- variational inference (e.g. mean field)
- ...

Best choice depends on model and requirements.

Training Conditional Random Fields in Practice

Gradient of the CRF training objective:

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

Feature function decompose over factors:

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

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

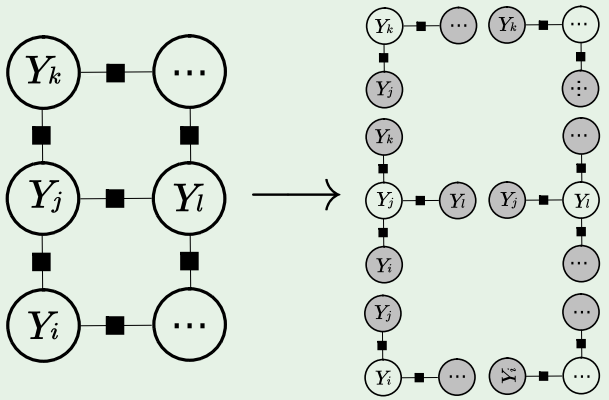
Problem: what if factor marginals $\mu_F = p(y_F|x;w)$ are intractable?

- approximate inference $\rightarrow$ approximate gradient
- convengence of gradient descent not guaranteed ☹

Alternative: optimize a simpler quantity instead of conditional likelihood

Pseudolikelihood [Besag, 1987]

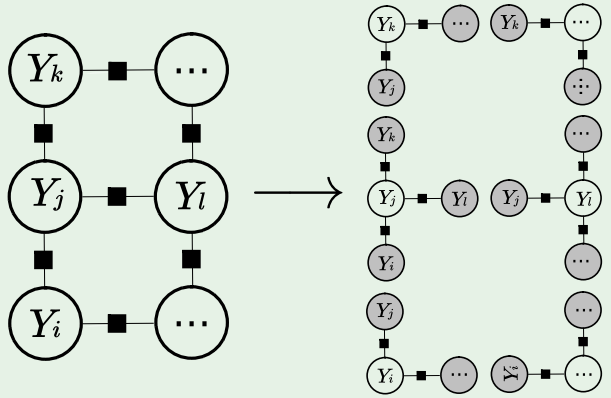
$$p(y|x) \approx \prod_{i \in V} p(y_i | y_{V \setminus \{i\}}, x)$$
$$= \prod_{i \in V} p(y_i | y_{N(i)}, x)$$



Alternative: optimize a simpler quantity instead of conditional likelihood

Pseudolikelihood [Besag, 1987]

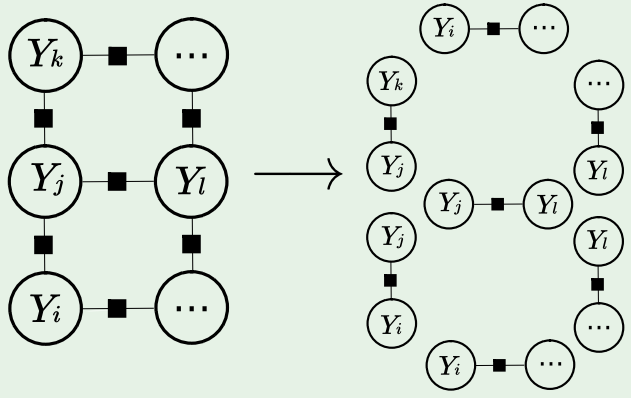
$$p(y|x) \approx \prod_{i \in V} p(y_i | y_{V \setminus \{i\}}, x)$$
$$= \prod_{i \in V} p(y_i | y_{N(i)}, x)$$



Piecewise Training [Sutton, McCallum, 2005]

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

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



Training with Approximate Likelihood – Pseudolikelihood (PL)

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

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

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

Training with Approximate Likelihood – Pseudolikelihood (PL)

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

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

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

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

Training with Approximate Likelihood – Piecewise Training (PW)

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

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

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

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

Training with Approximate Likelihood – Piecewise Training (PW)

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

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

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

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

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

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

Training with Approximate Likelihood – Piecewise Training (PW)

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

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

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

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

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

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

Structured Loss Functions

$$\Delta(\bar{y}, y)$$

Loss function

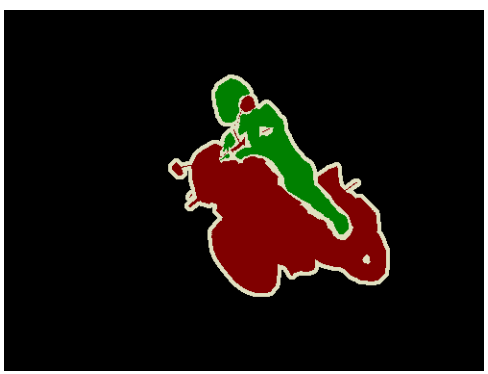
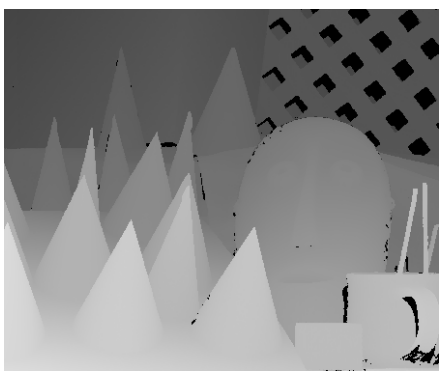
How to judge if a (structured) prediction is good?

- Define a *loss function*

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

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

- The *loss function* is application dependent


Example 1: 0/1 loss

Loss is 0 for perfect prediction, 1 otherwise:

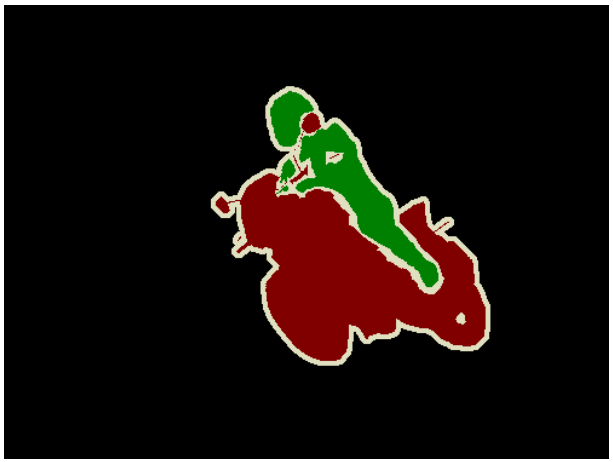
$$\Delta_{0/1}(\bar{y}, y) = [\![\bar{y} \neq y]\!] = \begin{cases} 0 & \text{if } \bar{y} = y \\ 1 & \text{otherwise} \end{cases}$$

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

Example 2: Hamming loss

Count the number of mislabeled variables:

$$\Delta_H(\bar{y}, y) = \frac{1}{|V|} \sum_{i \in V} [\![\bar{y}_i \neq y_i]\!]$$



Used, e.g., for graph labeling tasks

Example 3: Squared error

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

Sum of squared errors

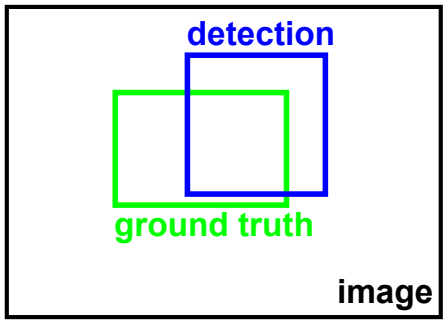
$$\Delta_Q(\bar{y}, y) = \frac{1}{|V|} \sum_{i \in V} \|\bar{y}_i - y_i\|^2.$$

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

Example 4: Task specific losses

Object detection

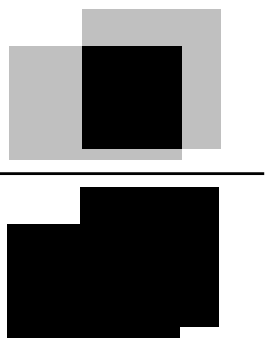
- bounding boxes, or
- arbitrarily shaped regions



Intersection-over-union loss:

$$\Delta_{\text{IoU}}(bary, y) = 1 - \frac{\text{area}(\bar{y} \cap y)}{\text{area}(\bar{y} \cup y)}$$

$$= 1 - \frac{\quad}{\quad}$$



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

Making Bayes-optimal Predictions

Given a distribution $p(y|x)$, what is the best way to predict $f : \mathcal{X} \to \mathcal{Y}$?

Bayesian decision theory: pick $f(x)$ that causes minimal expected loss:

$$f(x) = \operatorname*{\mathbf{argmin}}_{y \in \mathcal{Y}} \mathcal{R}_\Delta(y)$$

$$\text{for} \quad \mathcal{R}_\Delta(y) = \mathop{\mathbb{E}}_{\bar{y} \sim p(y|x)} \{\Delta(\bar{y}, y)\} = \sum_{\bar{y} \in \mathcal{Y}} \Delta(\bar{y}, y) p(\bar{y}|x)$$

For many loss functions not tractable, but some exceptions:

- $\mathcal{R}_{\Delta_{0/1}}(y) = 1 - p(y|x)$, so $f(x) = \mathbf{argmax}_y \, p(y|x)$
- $\mathcal{R}_{\Delta_H}(y) = 1 - \sum_{i \in V} p(y_i|x)$, so $f(x) = (y_1, \ldots, y_n)$ for $y_i = \mathbf{argmax}_{k \in \mathcal{Y}_i} \, p(y_i = k|x)$

Structured Support Vector Machines

$$\mathbf{min}_f \;\; \mathbb{E}_{(x,y)} \, \Delta(y, f(x))$$

Loss-Minimizing Parameter Learning

- $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ i.i.d. training set
- $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ be a feature function, like for CRF
- $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function.

- Find a weight vector w^* that minimizes the expected loss

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

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

Loss-Minimizing Parameter Learning

- $\mathcal{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ i.i.d. training set
- $\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ be a feature function, like for CRF
- $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function.

- Find a weight vector w^* that minimizes the expected loss

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

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

Advantage:

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

Disadvantage:

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

Inspiration: multi-class SVM

- $\mathcal{X}$ anything, $\quad \mathcal{Y} = \{1, 2, \ldots, K\}$,
- feature map $\phi : \mathcal{X} \to \mathcal{H}$ (explicit or implicit via kernel)
- training data $\{(x_1, y_1), \ldots, (x_n, y_n)\}$
- goal: learn functions $g_k(x) = \langle w_k, \phi(x) \rangle$ for $k = 1, \ldots, K$.

Prediction: $\quad f(x) = \underset{k=1,\ldots,K}{\mathbf{argmax}}\, g_k(x) = \underset{k=1,\ldots,K}{\mathbf{argmax}}\ \langle w_k, \phi(x) \rangle$

Enforce a margin between the correct and all incorrect labels:

$$\underset{w_1,\ldots,w_K,\boldsymbol{\xi}}{\mathbf{min}} \quad \frac{1}{2} \sum_{k=1}^{K} \|w_k\|^2 + \frac{C}{n} \sum_{i=1}^{n} \xi_i$$

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

$$\langle w_{y^i}, \phi(x^i) \rangle \geq 1 + \langle w_k, \phi(x^i) \rangle - \xi^i, \quad \text{for all } k \neq y_i.$$

Crammer-Singer Multiclass SVM

Equivalent parameterization:

- $\mathcal{X}$ anything, $\quad \mathcal{Y} = \{1, 2, \ldots, K\}$,
- feature map $\psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^D$ (explicit or implicit via kernel)
- $\psi(x, y) = \Big([\![y = 1]\!]\phi(x),\ [\![y = 2]\!]\phi(x), \ldots,\ [\![y = K]\!]$
- $w = (w_1, \ldots, w_K) \in \mathbb{R}^{KD}$
- goal: learn a function $g(x, y) = \langle w, \psi(x, y) \rangle$

Prediction: $\quad f(x) = \underset{k=1,\ldots,M}{\mathbf{argmax}} \ \langle w, \psi(x, y) \rangle$

Enforce a margin of 1 between the correct and any incorrect label:

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

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

$$\langle w, \psi(x_i, y_i) \rangle \geq 1 + \langle w, \psi(x_i, \bar{y}) \rangle - \xi_i, \quad \text{for all } \bar{y} \neq y_i.$$

Observation:

- for structure outputs, not all "incorrect" labels are equally bad
  $\rightarrow$ margin between y_i and $\bar{y}$ should depend on $\Delta(y_i, \bar{y})$

Structured (Output) Support Vector Machine

Goal: learn a function $g(x, y) = \langle w, \psi(x, y) \rangle$

Prediction: $f(x) = \underset{k=1,\ldots,M}{\mathbf{argmax}} \ \langle w, \psi(x, y) \rangle$

Enforce a margin $\Delta(y_i, y)$ between the correct and any incorrect label:

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

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

$$\langle w, \psi(x_i, y_i) \rangle \geq \Delta(y_i, \bar{y}) + \langle w, \psi(x_i, \bar{y}) \rangle - \xi_i, \quad \text{for all } \bar{y} \in \mathcal{Y}.$$

Structured Output Support Vector Machine

Equivalent unconstrained formulation (solve for optimal $\xi_1, \ldots, \xi_n$):

$$\min_w \ \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max_{\bar{y}\in\mathcal{Y}} \ \Big[\Delta(y_i, \bar{y}) + \langle w, \psi(x_i, \bar{y})\rangle - \langle w, \psi(x_i, y_i)\rangle\Big]$$

Conditional Random Field

Regularized conditional log-likelihood:

$$\mathbf{min}_w \ \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \log \sum_{\bar{y}\in\mathcal{Y}} \exp\left(\langle w, \psi(x_i, \bar{y})\rangle - \langle w, \phi(x_i, y_i)\rangle\right)$$

CRFs and SSVMs have more in common than usually assumed.

- $\log \sum_y \exp(\cdot)$ can be interpreted as a soft-max (differentiable)
- SSVM training takes loss function into account
- CRF is trained without specific loss, loss enters at prediction time

Structured Output Support Vector Machine

Equivalent unconstrained formulation (solve for optimal $\xi_1, \ldots, \xi_n$):

$$\min_w \ \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max_{\bar{y}\in\mathcal{Y}} \ \Big[\Delta(y_i, \bar{y}) + \langle w, \psi(x_i, \bar{y})\rangle - \langle w, \psi(x_i, y_i)\rangle\Big]$$

Conditional Random Field

Regularized conditional log-likelihood:

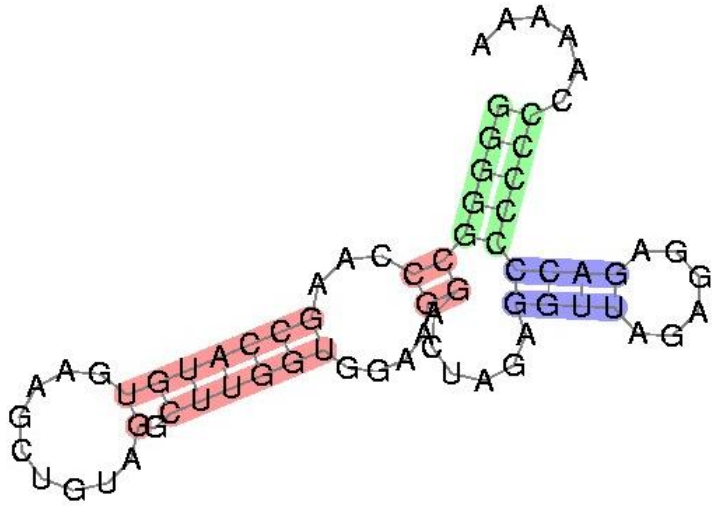
$$\mathbf{min}_w \ \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \log \sum_{\bar{y}\in\mathcal{Y}} \exp\left(\langle w, \psi(x_i, \bar{y})\rangle - \langle w, \phi(x_i, y_i)\rangle\right)$$

CRFs and SSVMs have more in common than usually assumed.

- $\log \sum_y \exp(\cdot)$ can be interpreted as a soft-max (differentiable)
- SSVM training takes loss function into account
- CRF is trained without specific loss, loss enters at prediction time

Example: RNA Secondary Structure Prediction De Bona *et al.*, 2007]

AAAAACCCCCCCCAGAGGAGAUUG
GAGAUCAAAGGUGGUUCGGAUGUC $\rightarrow$
GAAGUGUACCGAACCCGGGGG

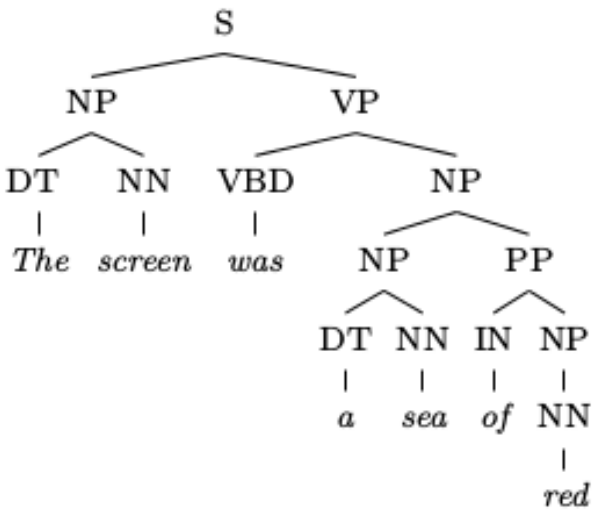


- $\mathcal{X} = \Sigma^*$ for $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{U}\}$ (nucleotide sequence)
- $\mathcal{Y} = \{(i, j) : i, j \in \mathbb{N}, i < j\}$ (i, j) mean "x_i binds with x_j"
- $\psi(x, y)$ domain-specific features: binding energy of $x_i \leftrightarrow x_j$, prefered patterns (motifs), loop properties, ...
- $\Delta(\bar{y}, y)$: number of wrong/missing bindings (Hamming loss)

$$\min_w \ \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max_{\bar{y}\in\mathcal{Y}} \Big[\, \Delta(y_i, \bar{y}) + \langle w, \psi(x_i, \bar{y})\rangle - \langle w, \psi(x_i, y_i)\rangle \Big]$$

Example: Sentence Parsing [Taskar *et al.*, 2004]

`The screen was a sea of red.` $\rightarrow$



- $\mathcal{X} = \{\text{English sentences}\}$
- $\mathcal{Y} = \{\text{parse tree}\}$
- $\psi(x, y)$ domain-specific features:
  - word properties, e.g. "· starts with capital letter", "· ends in `ing`"
  - grammatical rules: $NP \rightarrow DT + NN$
- $\Delta(\bar{y}, y)$: number of wrong assignments

Solving S-SVM Training in Practice

$$\min_w \ \frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max_{\bar{y}\in\mathcal{Y}} \Big[\Delta(y_i, \bar{y}) + \langle w, \psi(x_i, \bar{y})\rangle - \langle w, \psi(x_i, y_i)\rangle \Big]$$

- continuous
- unconstrained
- convex
- non-differentiable $\rightarrow$ use subgradients

Observation:

- Evaluating objective or computing subgradient require solving

  $$\max_{\bar{y}\in\mathcal{Y}} \Big[\Delta(y_i, \bar{y}) + \langle w, \psi(x_i, \bar{y})\rangle - \langle w, \psi(x_i, y_i)\rangle \Big] \qquad \text{(or } \mathbf{argmax}\text{)}$$

  ("loss-augmented energy minimization") for all $i = 1, \ldots, n$
- solving the optimization will require many evaluations

MAP Prediction / Energy Minimization

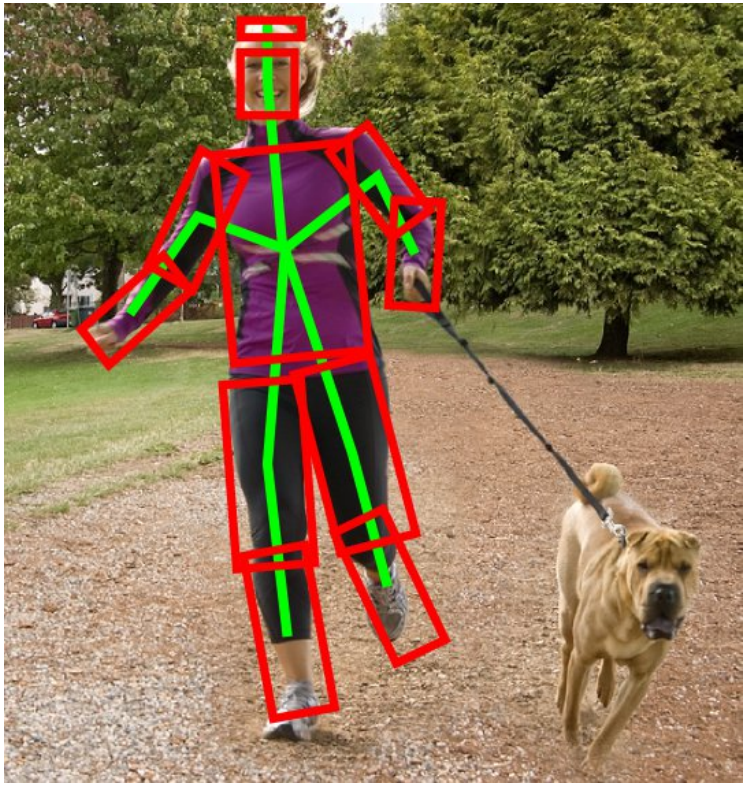
$$\mathbf{argmax}_y \, p(y|x) \;/\; \mathbf{argmin}_y \, E(x,y) \;/$$
$$\mathbf{argmax}_y \, \Delta(y_i, y) + \langle w, \psi(x,y) \rangle$$

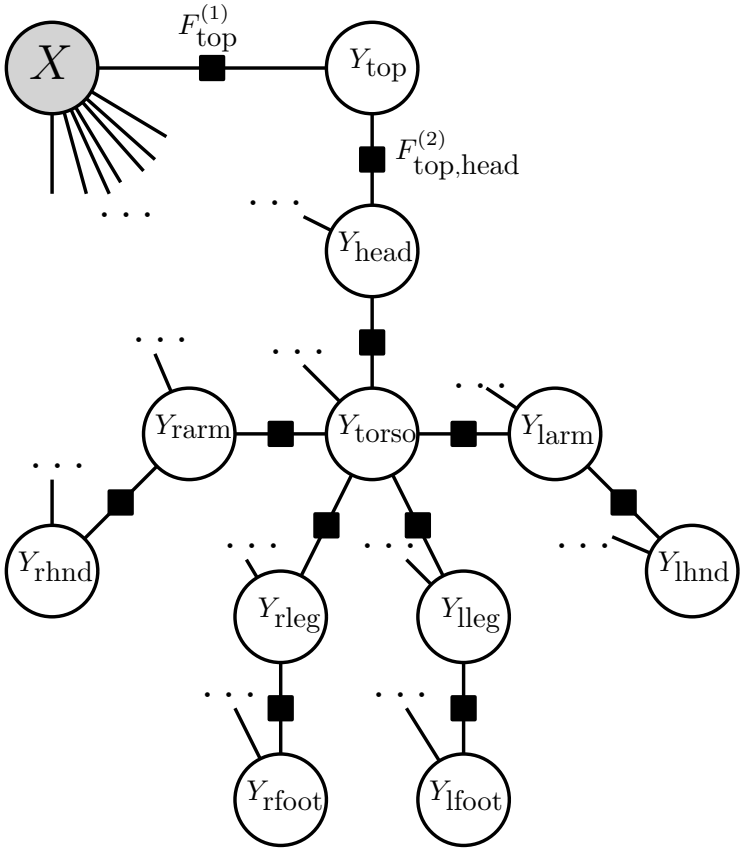
MAP Prediction / Energy Minimization

Task: Minimize $E(x, y)$ or $\Delta(y_i, y) + E(x, y)$ for $E(x, y) = \langle w, \psi(x, y) \rangle$

- Exact Energy Minimization
  - Belief Propagation on chains/trees
  - Graph-Cuts for submodular energies
  - Integer Linear Programming
- Approximate Energy Minimization
  - Linear Programming Relaxations
  - Local Search Methods
    - Iterative Conditional Modes
    - Multi-label Graph Cuts
  - Simulated Annealing

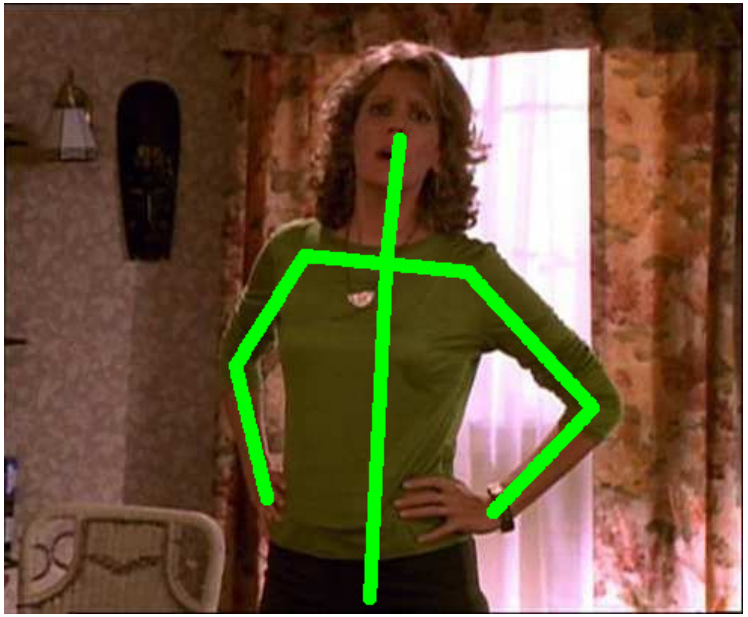
Example: Pictorial Structures / Deformable Parts Model





- **Tree-structured model** for articulated pose
  (Felzenszwalb and Huttenlocher, 2000), (Fischler and Elschlager, 1973), (Yang and Ramanan, 2013), (Pishchulin *et al.*, 2012)

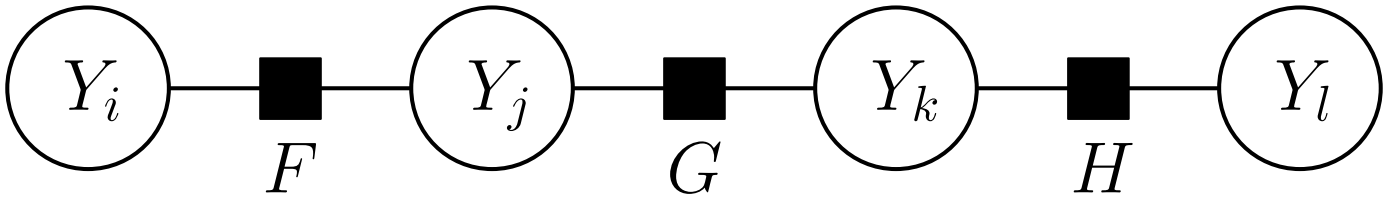
Example: Pictorial Structures / Deformable Parts Model



- most likely configuration $y^* = \underset{y \in \mathcal{Y}}{\mathbf{argmax}}\, p(y|x) = \underset{y}{\mathbf{argmin}}\, E(y, x)$

Energy Minimization – Belief Propagation

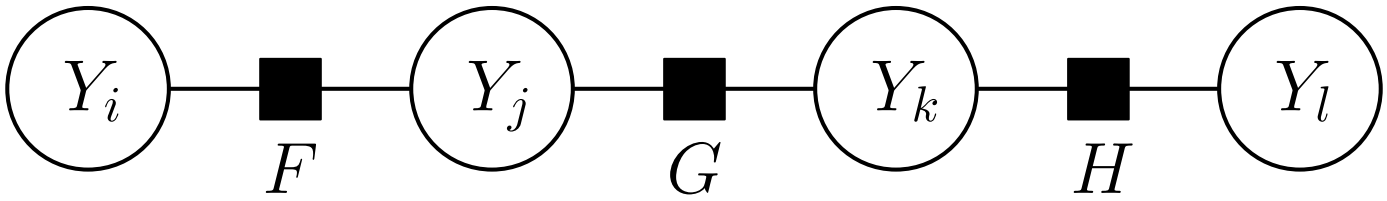
Chain model: same trick as for *inference*: **belief propagation**



$$\mathbf{min}_y\, E(y) = \mathbf{min}_{y_i,y_j,y_k,y_l} \quad E_F(y_i, y_j) + E_G(y_j, y_k) + E_H(y_k, y_l)$$

Energy Minimization – Belief Propagation

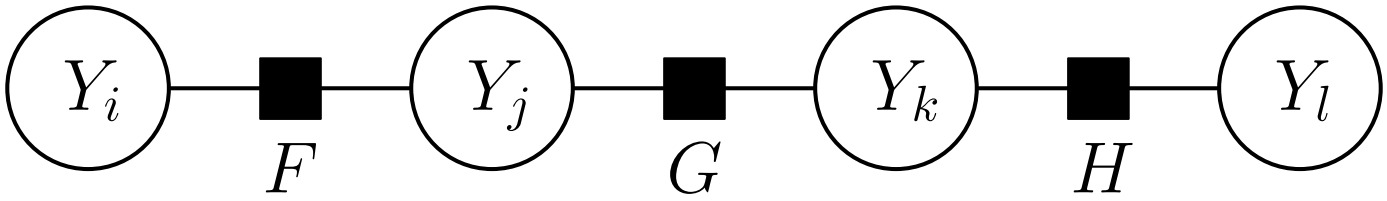
Chain model: same trick as for *inference*: **belief propagation**



$$\begin{aligned}\mathbf{min}_y\, E(y) &= \mathbf{min}_{y_i,y_j,y_k,y_l} \quad E_F(y_i,y_j) + E_G(y_j,y_k) + E_H(y_k,y_l) \\ &= \mathbf{min}_{y_i,y_j}\left[E_F(y_i,y_j) + \mathbf{min}_{y_k}\left[E_G(y_j,y_k) + \mathbf{min}_{y_l} E_H(y_k,y_l)\right.\right.\end{aligned}$$

Energy Minimization – Belief Propagation

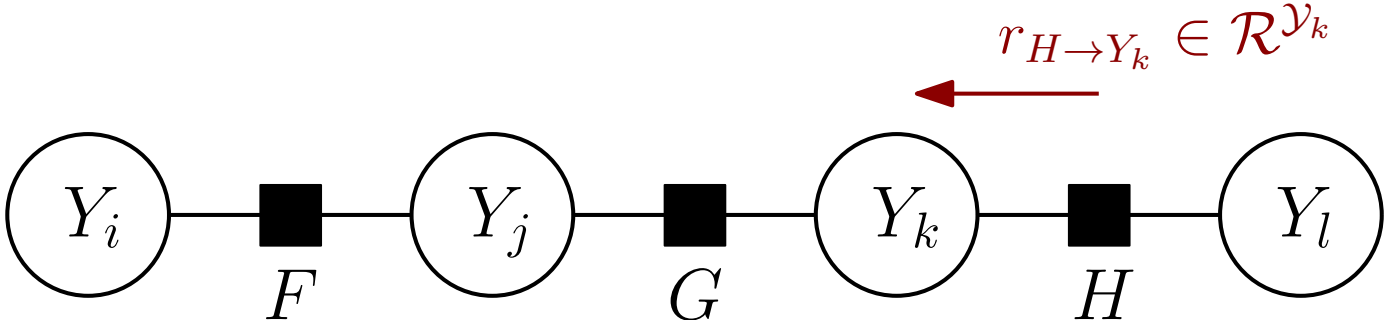
Chain model: same trick as for *inference*: **belief propagation**



$$\begin{aligned}\mathbf{min}_y\, E(y) &= \mathbf{min}_{y_i,y_j,y_k,y_l} \quad E_F(y_i,y_j) + E_G(y_j,y_k) + E_H(y_k,y_l) \\ &= \mathbf{min}_{y_i,y_j}\,[E_F(y_i,y_j) + \mathbf{min}_{y_k}\,[E_G(y_j,y_k) + \underbrace{\mathbf{min}_{y_l}\, E_H(y_k,y_l)}_{r_{H\to Y_k}(y_k)}\end{aligned}$$

Energy Minimization – Belief Propagation

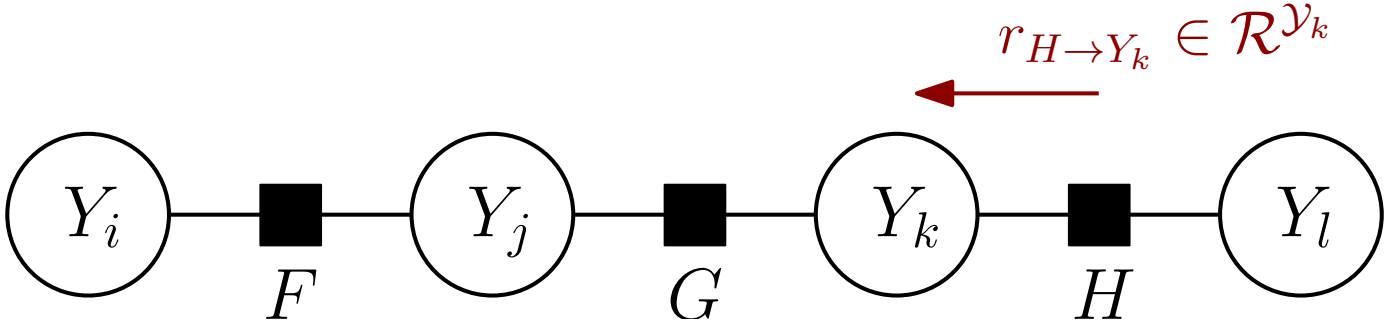
Chain model: same trick as for *inference*: **belief propagation**



$$\begin{aligned}
\mathbf{min}_y E(y) &= \mathbf{min}_{y_i,y_j,y_k,y_l} \quad E_F(y_i,y_j) + E_G(y_j,y_k) + E_H(y_k,y_l) \\
&= \mathbf{min}_{y_i,y_j} \left[E_F(y_i,y_j) + \mathbf{min}_{y_k} \left[E_G(y_j,y_k) + \underbrace{\mathbf{min}_{y_l} E_H(y_k,y_l)}_{r_{H \to Y_k}(y_k)} \right] \right] \\
&= \mathbf{min}_{y_i,y_j} \left[E_F(y_i,y_j) + \mathbf{min}_{y_k} E_G(y_j,y_k) + r_{H \to Y_k}(y_k) \right]
\end{aligned}$$

Energy Minimization – Belief Propagation

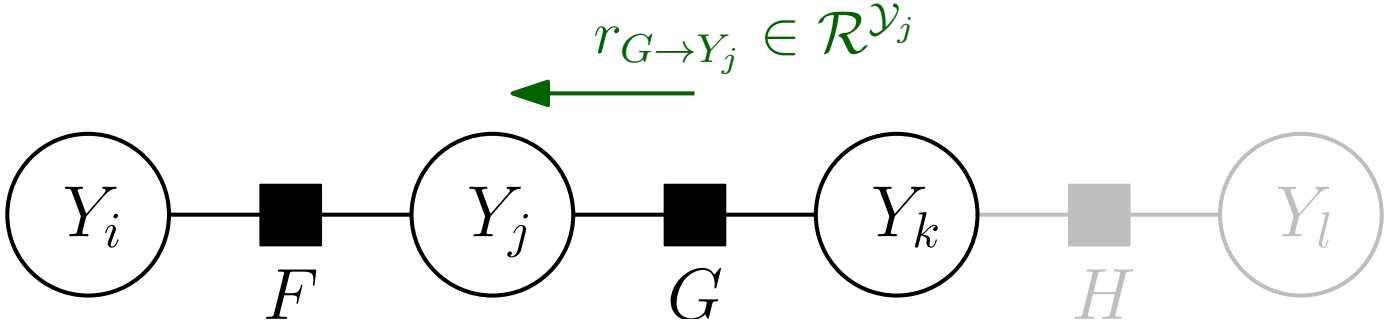
Chain model: same trick as for *inference*: **belief propagation**



$$\begin{aligned}
\mathbf{min}_y E(y) &= \mathbf{min}_{y_i,y_j,y_k,y_l} \quad E_F(y_i,y_j) + E_G(y_j,y_k) + E_H(y_k,y_l) \\
&= \mathbf{min}_{y_i,y_j} \left[E_F(y_i,y_j) + \mathbf{min}_{y_k} \left[E_G(y_j,y_k) + \underbrace{\mathbf{min}_{y_l} E_H(y_k,y_l)}_{r_{H \to Y_k}(y_k)} \right] \right] \\
&= \mathbf{min}_{y_i,y_j} \left[E_F(y_i,y_j) + \underbrace{\mathbf{min}_{y_k} E_G(y_j,y_k) + r_{H \to Y_k}(y_k)}_{r_{G \to Y_j}(y_j)} \right]
\end{aligned}$$

Energy Minimization – Belief Propagation

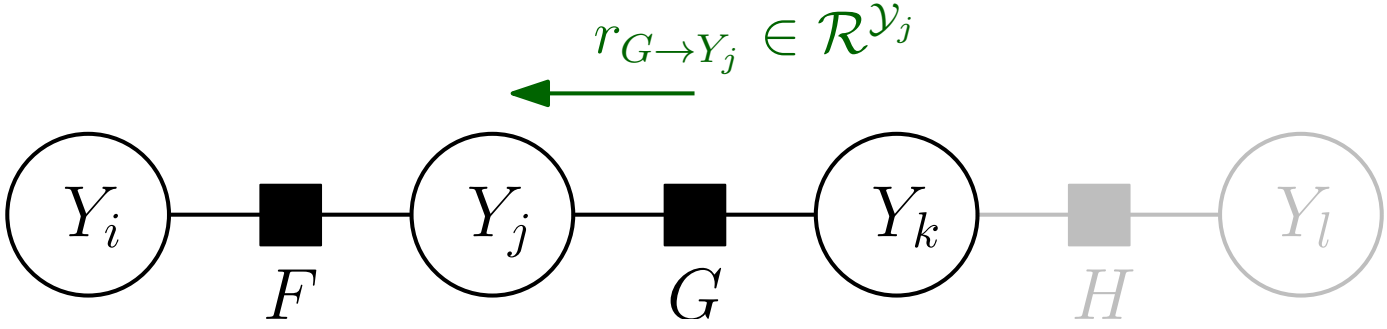
Chain model: same trick as for *inference*: **belief propagation**



$$\begin{aligned}
\mathbf{min}_y E(y) &= \mathbf{min}_{y_i,y_j,y_k,y_l} \quad E_F(y_i,y_j) + E_G(y_j,y_k) + E_H(y_k,y_l) \\
&= \mathbf{min}_{y_i,y_j} \left[E_F(y_i,y_j) + \mathbf{min}_{y_k} \left[E_G(y_j,y_k) + \underbrace{\mathbf{min}_{y_l} E_H(y_k,y_l)}_{r_{H \to Y_k}(y_k)} \right] \right] \\
&= \mathbf{min}_{y_i,y_j} \left[E_F(y_i,y_j) + \underbrace{\mathbf{min}_{y_k} E_G(y_j,y_k) + r_{H \to Y_k}(y_k)}_{r_{G \to Y_j}(y_j)} \right] \\
&= \mathbf{min}_{y_i,y_j} \left[E_F(y_i,y_j) + r_{G \to Y_j}(y_j) \right] \quad \ldots
\end{aligned}$$

Energy Minimization – Belief Propagation

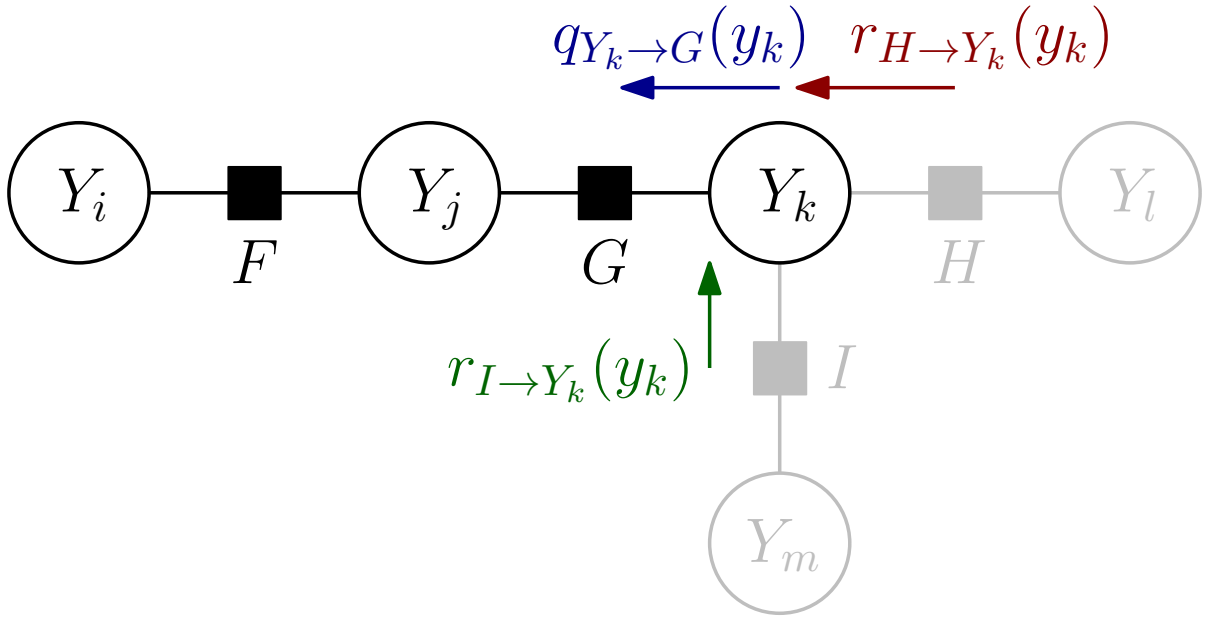
Chain model: same trick as for *inference*: **belief propagation**



$$\begin{aligned}
\mathbf{min}_y E(y) &= \mathbf{min}_{y_i,y_j,y_k,y_l} \quad E_F(y_i,y_j) + E_G(y_j,y_k) + E_H(y_k,y_l) \\
&= \mathbf{min}_{y_i,y_j} \left[E_F(y_i,y_j) + \mathbf{min}_{y_k} \left[E_G(y_j,y_k) + \underbrace{\mathbf{min}_{y_l} E_H(y_k,y_l)}_{r_{H \to Y_k}(y_k)} \right] \right] \\
&= \mathbf{min}_{y_i,y_j} \left[E_F(y_i,y_j) + \underbrace{\mathbf{min}_{y_k} E_G(y_j,y_k) + r_{H \to Y_k}(y_k)}_{r_{G \to Y_j}(y_j)} \right] \\
&= \mathbf{min}_{y_i,y_j} \left[E_F(y_i,y_j) + r_{G \to Y_j}(y_j) \right] \quad \ldots
\end{aligned}$$

- actual **argmax** by backtracking which choices were maximal

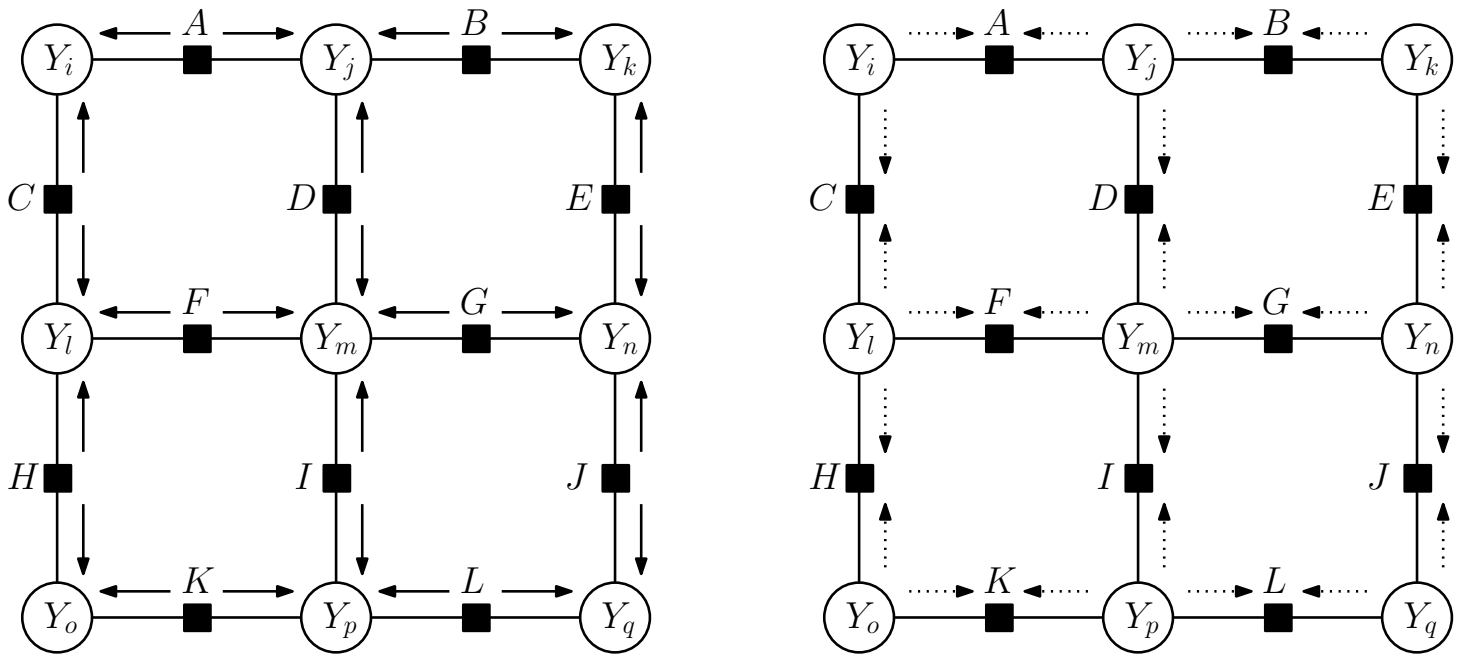
Tree models:



- $q_{H \to Y_k}(y_k) = \mathbf{min}_{y_l} E_H(y_k, y_l)$
- $q_{I \to Y_k}(y_k) = \mathbf{min}_{y_m} E_I(y_k, y_m)$
- $q_{Y_k \to G}(y_k) = q_{H \to Y_k}(y_k) + q_{I \to Y_k}(y_k)$

min-sum (more common **max-sum**) **belief propagation**

Belief Propagation in Cyclic Graphs



Loopy Max-Sum Belief Propagation

Same problem as in probabilistic inference:

- no guarantee of convergence
- no guarantee of optimality

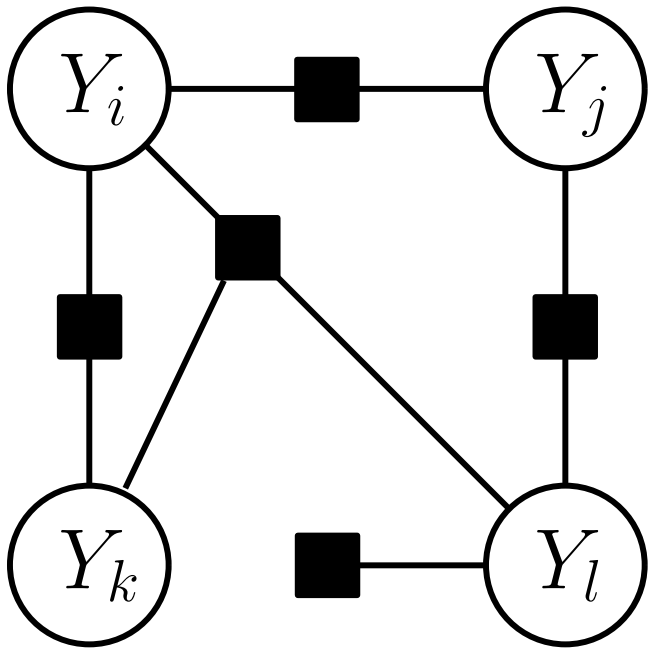
Convergent variants, e.g. TRW-S [Kolmogorov, PAMI 2006] still approximate

Cyclic Graphs

In general, MAP prediction/energy minimization in models with cycles or higher-order terms is **intractable** (NP-hard).

Some important exceptions:

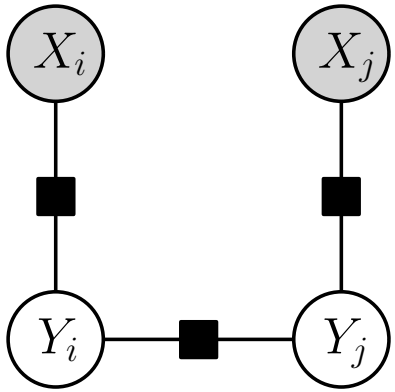
- low tree-width [Lauritzen, Spiegelhalter, 1988]
- **binary states, pairwise submodular interactions [Boykov, Jolly, 2001]**
- binary states, only pairwise interactions, planar graph [Globerson, Jaakkola, 2006]
- special (Potts $\mathcal{P}^n$) higher order factors [Kohli, Kumar, 2007]
- perfect graph structure [Jebara, 2009]



Submodular Energy Functions

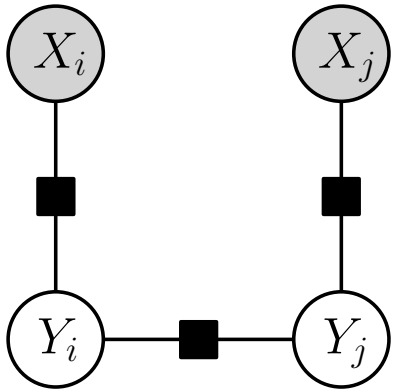
- Binary variables: $\mathcal{Y}_i = \{0, 1\}$ for all $i \in \mathcal{V}$
- Energy function: unary and pairwise factors

$$E(y; x, w) = \sum_{i \in \mathcal{V}} E_i(y_i) + \sum_{(i,j) \in \mathcal{E}} E_{ij}(y_i, y_j)$$



Submodular Energy Functions

- Binary variables: $\mathcal{Y}_i = \{0, 1\}$ for all $i \in \mathcal{V}$
- Energy function: unary and pairwise factors



$$E(y; x, w) = \sum_{i \in \mathcal{V}} E_i(y_i) + \sum_{(i,j) \in \mathcal{E}} E_{ij}(y_i, y_j)$$

- Restriction 1 (without loss of generality):

$$E_i(y_i) \geq 0$$

(always achievable by adding a constant to E)

- Restriction 2 (**submodularity**):

$$\begin{aligned} E_{ij}(y_i, y_j) &= 0, && \text{if } y_i = y_j, \\ E_{ij}(y_i, y_j) &= E_{ij}(y_j, y_i) \geq 0, && \text{otherwise.} \end{aligned}$$

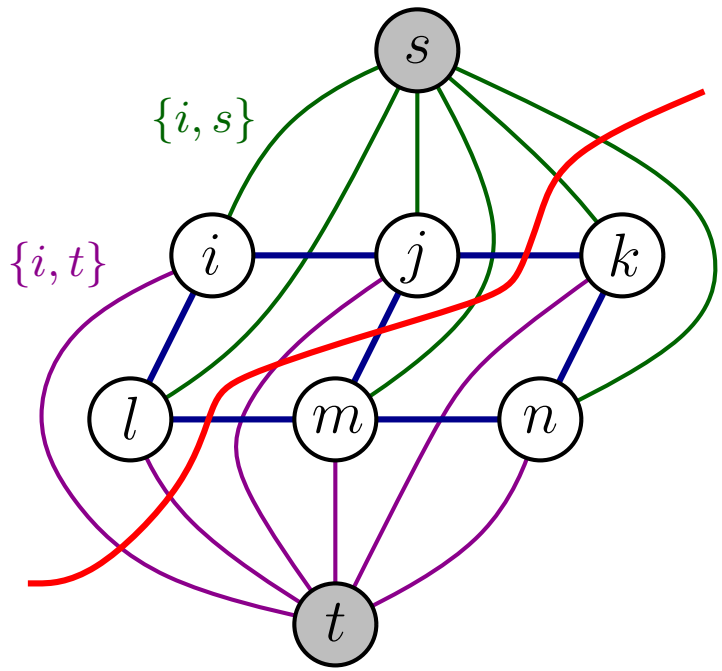
"neighbors prefer to have the same labels"

Graph-Cuts Algorithm for Submodular Energy Minimization [Greig *et al.*, 1989]

If conditions are fulfilled, energy minimization can be performed by a solving an s-t-**mincut** problem:

- construct auxiliary undirected graph
- one node $\{i\}_{i \in V}$ per variable
- two extra nodes: source s, sink t
- weighted edges

Edge	weight
$\{i, j\}$	$E_{ij}(y_i = 0, y_j = 1)$
$\{i, s\}$	$E_i(y_i = 1)$
$\{i, t\}$	$E_i(y_i = 0)$



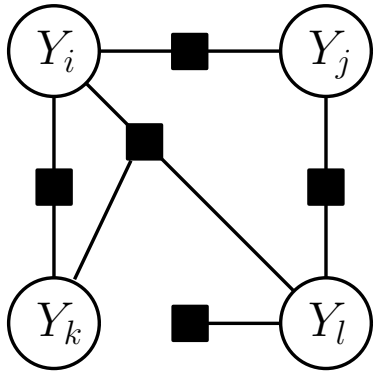
- find s-t-cut of minimal weight (polynomial time using max-flow theorem)

From minimal weight cut we recover labeling of minimal energy:

Integer Linear Programming (ILP)

General energy $E(y) = \sum_F E_F(y_F)$

- variables with more than 2 states
- higher-order factors (more than 2 variables)
- non-submodular factors



Formulate as integer linear program (ILP)

- linear objective function
- linear constraints
- variables to optimize over are integer-valued

ILPs are in general NP-hard, but some individual instances can be solved

- standard toolboxes: e.g. CPLEX, Gurobi, COIN-OR, ...

Integer Linear Programming (ILP)

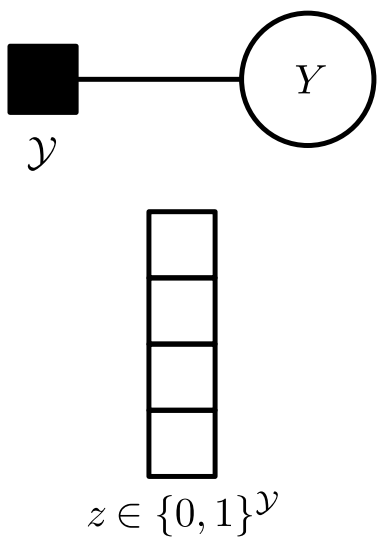
We can write any discrete optimization as an Integer Linear Program:

$$\min_{y \in \mathcal{Y}} \quad E(y) \quad \text{for} \quad \mathcal{Y} = \{1, \ldots, K\}$$

$$\min_{z \in \{0,1\}^K} \sum_{k=1}^{K} \theta_k z_k \text{ subject to } \sum_{k=1}^{K} z_k = 1$$

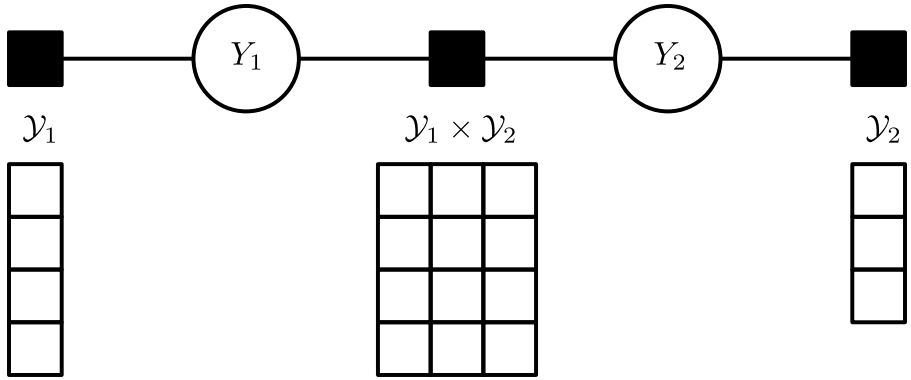
Encode assignment in indicator variables:

- $z \in \{0,1\}^K \qquad z_k = 1 \;\Leftrightarrow\; [\![y = k]\!]$
- coefficient vector: $\theta_k = E(k)$
- constraint: $\sum_k z_k = 1 \;\rightarrow$ exactly one 1



Integer Linear Programming (ILP)

Example:

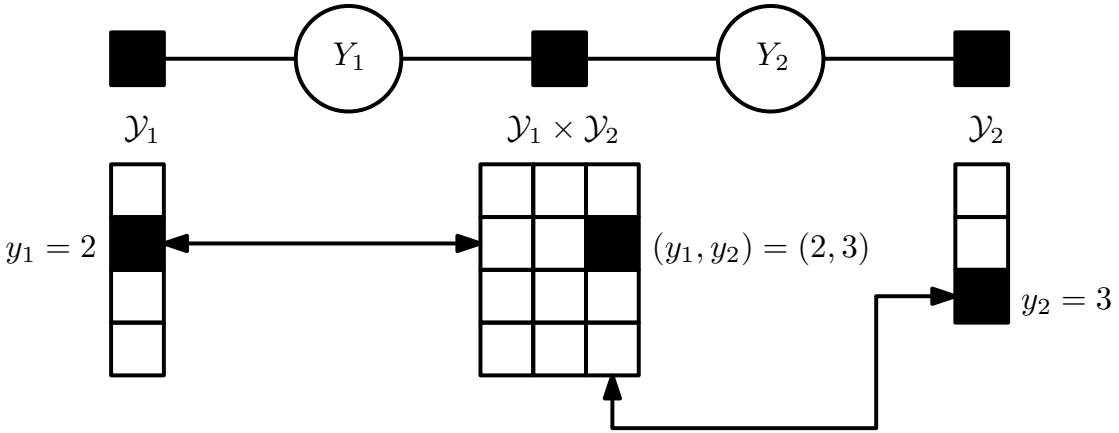


Encode assignment in indicator variables:

- $z_1 \in \{0,1\}^{\mathcal{Y}_1}$ $\quad z_{1;k} = 1 \Leftrightarrow [\![y_1 = k]\!]$
- $z_2 \in \{0,1\}^{\mathcal{Y}_2}$ $\quad z_{2;l} = 1 \Leftrightarrow [\![y_2 = l]\!]$
- $z_{12} \in \{0,1\}^{\mathcal{Y}_1 \times \mathcal{Y}_2}$ $\quad z_{12;kl} = 1 \Leftrightarrow [\![y_1 = k \wedge y_2 = l]\!]$

Integer Linear Programming (ILP)

Example:



Encode assignment in indicator variables:

- $z_1 \in \{0,1\}^{\mathcal{Y}_1}$ $\quad z_{1;k} = 1 \;\Leftrightarrow\; [\![y_1 = k]\!]$
- $z_2 \in \{0,1\}^{\mathcal{Y}_2}$ $\quad z_{2;l} = 1 \;\Leftrightarrow\; [\![y_2 = l]\!]$
- $z_{12} \in \{0,1\}^{\mathcal{Y}_1 \times \mathcal{Y}_2}$ $\quad z_{12;kl} = 1 \;\Leftrightarrow\; [\![y_1 = k \wedge y_2 = l]\!]$

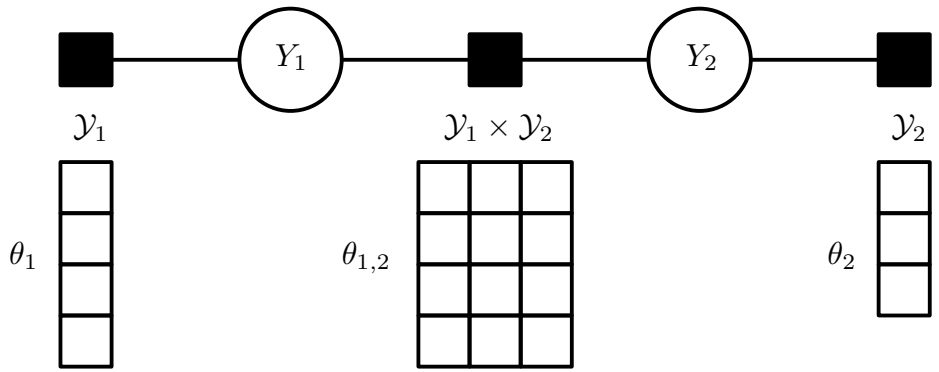
Constraints:

$$\sum_{k \in \mathcal{Y}_1} z_{1;k} = 1, \quad \sum_{l \in \mathcal{Y}_2} z_{2;l} = 1, \quad \sum_{k,l \in \mathcal{Y}_1 \times \mathcal{Y}_2} z_{12;kl} = 1 \quad \text{(indicator property)}$$

$$\sum_{k \in \mathcal{Y}_1} z_{12;kl} = z_{2;l} \qquad \sum_{l \in \mathcal{Y}_2} z_{12;kl} = z_{1;k} \qquad \text{(consistency)}$$

Integer Linear Programming (ILP)

Example: $E(y_1, y_2) = E_1(y_1) + E_{12}(y_1, y_2) + E_2(y_2)$



Define coefficient vectors:

- $\theta_1 \in \mathbb{R}^{\mathcal{Y}_1}$ $\quad \theta_{1;k} = E_1(k)$
- $\theta_2 \in \mathbb{R}^{\mathcal{Y}_2}$ $\quad \theta_{2;l} = E_2(l)$
- $\theta_{12} \in \mathbb{R}^{\mathcal{Y}_1 \times \mathcal{Y}_2}$ $\quad \theta_{12;kl} = E_{1,2}(k, l)$

Energy is a linear function of unknown z:

$$E(y_1, y_2) = \sum_{i \in V} \sum_{k \in \mathcal{Y}_i} \theta_{i;k} [\![y_i = k]\!] + \sum_{i,j \in \mathcal{E}} \sum_{k,l \in \mathcal{Y}_i \times \mathcal{Y}_j} \theta_{ij;kl} [\![y_i = k \wedge y_j = l]\!]$$

$$= \sum_{i \in V} \sum_{k \in \mathcal{Y}_i} \theta_{i;k} z_{i;k} + \sum_{(i,j) \in \mathcal{E}} \sum_{(k,l) \in \mathcal{Y}_i \times \mathcal{Y}_j} \theta_{ij;kl} z_{ij;kl}$$

Integer Linear Programming (ILP)

$$\mathbf{min}_{\boldsymbol{z}} \quad \sum_{i \in V} \sum_{k \in \mathcal{Y}_i} \theta_{i;k} z_{i;k} + \sum_{(i,j) \in \mathcal{E}} \sum_{(k,l) \in \mathcal{Y}_i \times \mathcal{Y}_j} \theta_{ij;kl} z_{ij;kl}$$

subject to

$$z_{i;k} \in \{0,1\} \qquad \text{for all } i \in V,\ \forall k \in \mathcal{Y}_i,$$

$$z_{ij;kl} \in \{0,1\} \qquad \text{for all } (i,j) \in \mathcal{E}, (k,l) \in \mathcal{Y}_i \times \mathcal{Y}_j,$$

$$\sum_{k \in \mathcal{Y}_i} z_{i;k} = 1, \quad \text{for all } i \in V,$$

$$\sum_{k,l \in \mathcal{Y}_i \times \mathcal{Y}_j} z_{ij;kl} = 1, \quad \text{for all } (i,j) \in \mathcal{E},$$

$$\sum_{k \in \mathcal{Y}_i} z_{ij;kl} = z_{j;l} \quad \text{for all } (i,j) \in \mathcal{E},\ l \in \mathcal{Y}_j,$$

$$\sum_{l \in \mathcal{Y}_j} z_{ij;kl} = z_{i;k} \quad \text{for all } (i,j) \in \mathcal{E},\ k \in \mathcal{Y}_i,$$

Integer Linear Programming (ILP)

$$\mathbf{min}_{\boldsymbol{z}} \quad \sum_{i \in V} \sum_{k \in \mathcal{Y}_i} \theta_{i;k} z_{i;k} + \sum_{(i,j) \in \mathcal{E}} \sum_{(k,l) \in \mathcal{Y}_i \times \mathcal{Y}_j} \theta_{ij;kl} z_{ij;kl}$$

subject to

$$z_{i;k} \in \{0,1\} \qquad \text{for all } i \in V,\ \forall k \in \mathcal{Y}_i,$$

$$z_{ij;kl} \in \{0,1\} \qquad \text{for all } (i,j) \in \mathcal{E}, (k,l) \in \mathcal{Y}_i \times \mathcal{Y}_j,$$

$$\sum_{k \in \mathcal{Y}_i} z_{i;k} = 1, \quad \text{for all } i \in V,$$

$$\sum_{k,l \in \mathcal{Y}_i \times \mathcal{Y}_j} z_{ij;kl} = 1, \quad \text{for all } (i,j) \in \mathcal{E},$$

$$\sum_{k \in \mathcal{Y}_i} z_{ij;kl} = z_{j;l} \quad \text{for all } (i,j) \in \mathcal{E},\ l \in \mathcal{Y}_j,$$

$$\sum_{l \in \mathcal{Y}_j} z_{ij;kl} = z_{i;k} \quad \text{for all } (i,j) \in \mathcal{E},\ k \in \mathcal{Y}_i,$$

NP-hard to solve because of integrality constraints.

Linear Programming (LP) Relaxation

$$\mathbf{min}_{\boldsymbol{z}} \quad \sum_{i \in V} \sum_{k \in \mathcal{Y}_i} \theta_{i;k} z_{i;k} + \sum_{(i,j) \in \mathcal{E}} \sum_{(k,l) \in \mathcal{Y}_i \times \mathcal{Y}_j} \theta_{ij;kl} z_{ij;kl}$$

subject to

$$\cancel{z_{i;k} \in \{0,1\}} \quad z_{i;k} \in [0,1] \qquad \text{for all } i \in V,\ \forall k \in \mathcal{Y}_i,$$

$$\cancel{z_{ij;kl} \in \{0,1\}} \quad z_{ij;kl} \in [0,1] \qquad \text{for all } (i,j) \in \mathcal{E}, (k,l) \in \mathcal{Y}_i \times \mathcal{Y}_j,$$

$$\sum_{k \in \mathcal{Y}_i} z_{i;k} = 1, \quad \text{for all } i \in V,$$

$$\sum_{k,l \in \mathcal{Y}_i \times \mathcal{Y}_j} z_{ij;kl} = 1, \quad \text{for all } (i,j) \in \mathcal{E},$$

$$\sum_{k \in \mathcal{Y}_i} z_{ij;kl} = z_{j;l} \quad \text{for all } (i,j) \in \mathcal{E},\ l \in \mathcal{Y}_j,$$

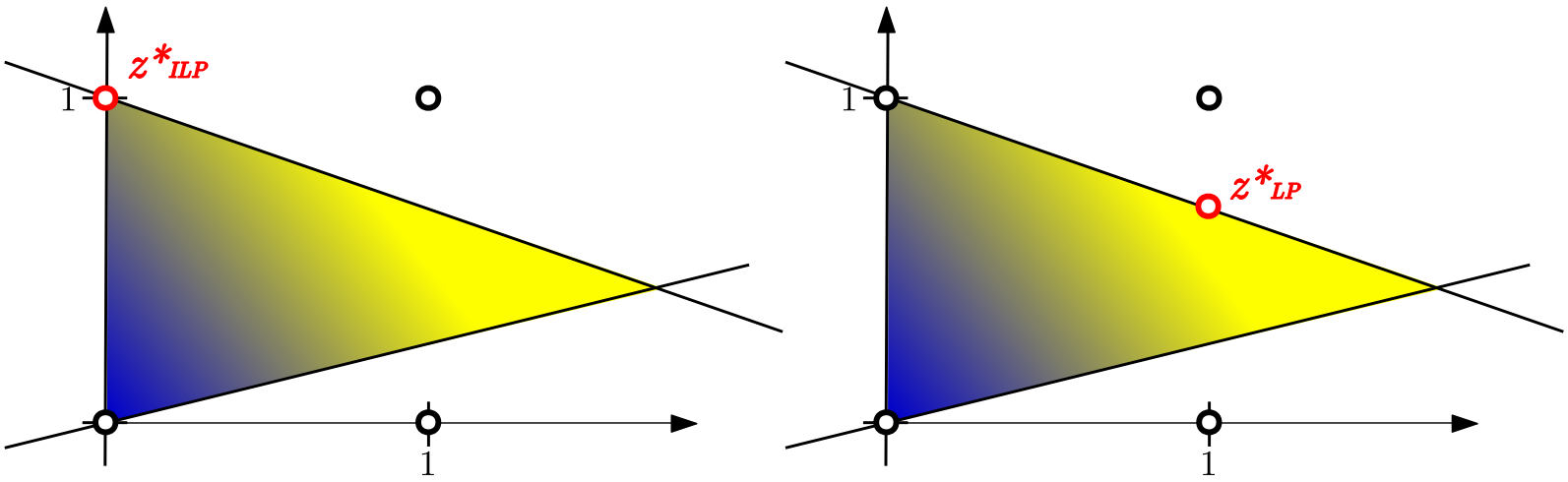
$$\sum_{l \in \mathcal{Y}_j} z_{ij;kl} = z_{i;k} \quad \text{for all } (i,j) \in \mathcal{E},\ k \in \mathcal{Y}_i,$$

Relax constraints $\rightarrow$ tractable optimization problem

Linear Programming (LP) Relaxation

Solution z^*_{LP} might have fractional values

- $\rightarrow$ no corresponding labeling $y \in \mathcal{Y}$
- $\rightarrow$ round LP solution to $\{0, 1\}$ values



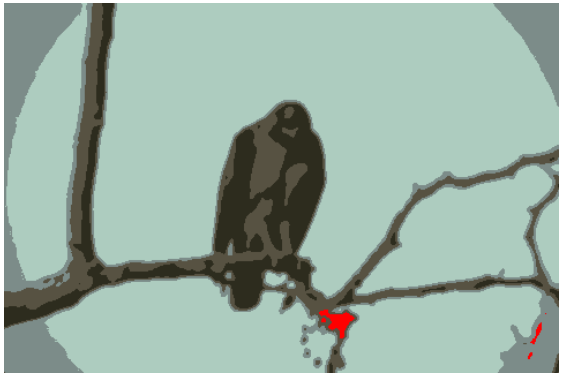
Problem:

- rounded solution usually not optimal, i.e. not identical to ILP solution

LP relaxations perform approximate energy minimization

Linear Programming (LP) Relaxation

Example: color quantization



Example: stereo reconstruction

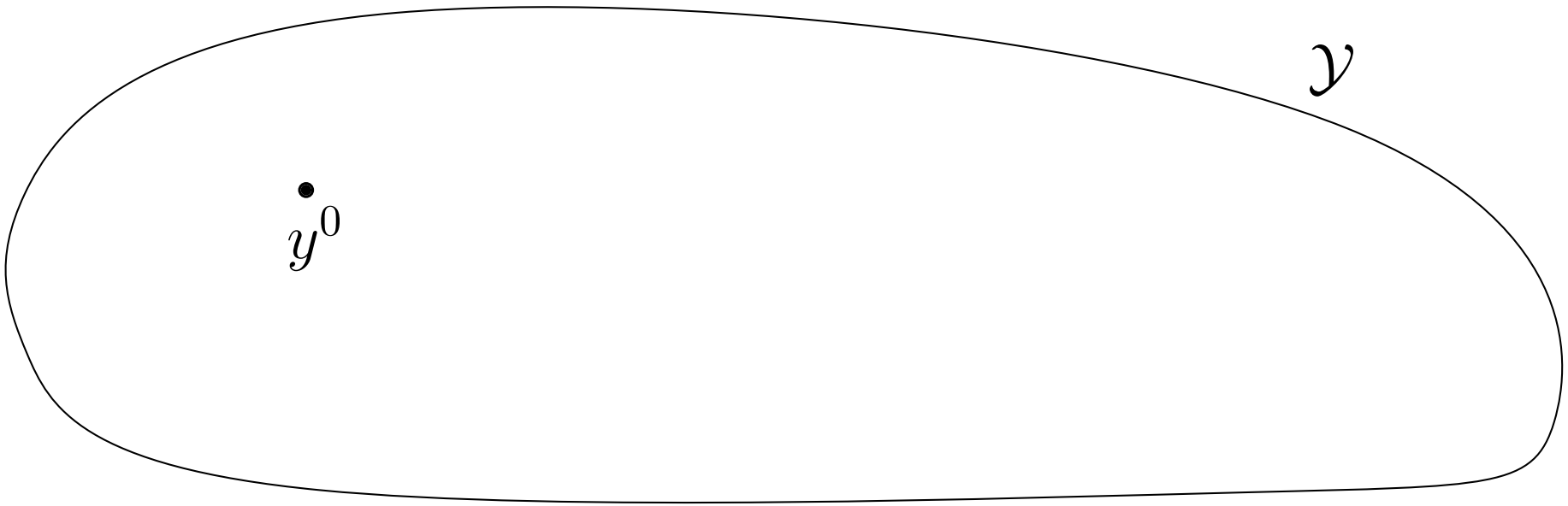
Images: Berkeley Segmentation Dataset

Local Search

Avoid getting fractional solutions: energy minimization by **local search**

Local Search

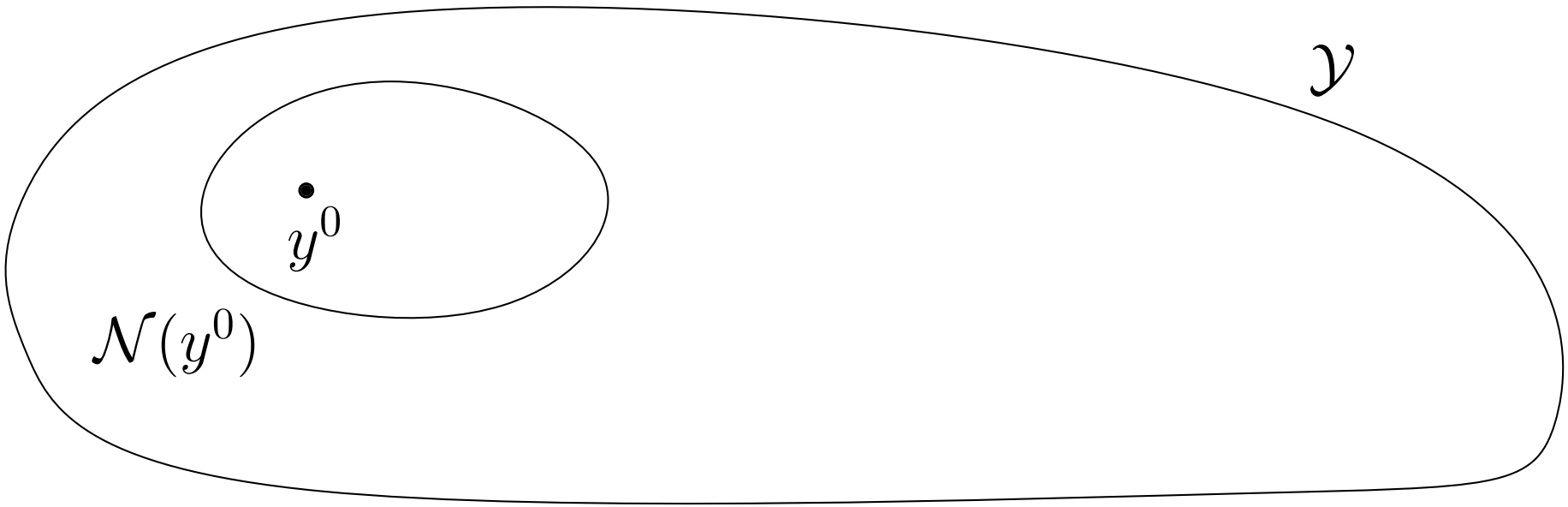
Avoid getting fractional solutions: energy minimization by **local search**



- choose starting labeling y^0

Local Search

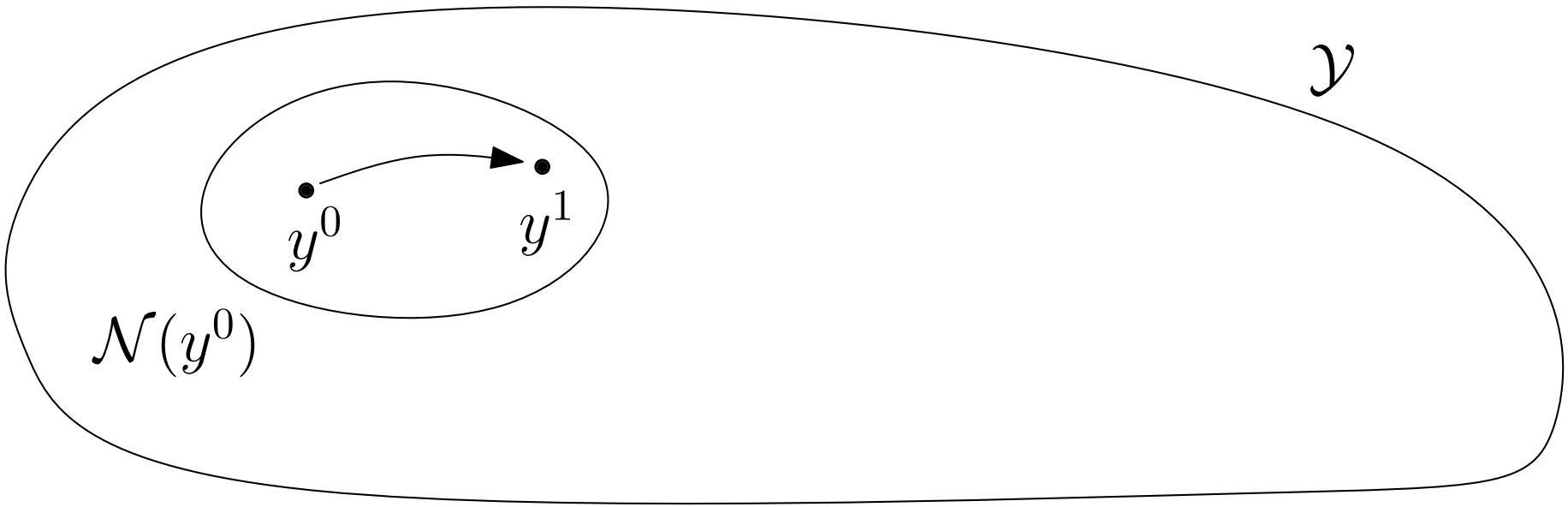
Avoid getting fractional solutions: energy minimization by **local search**



- choose starting labeling y^0
- construct neighborhood $\mathcal{N}(y^0) \subset \mathcal{Y}$ of labelings

Local Search

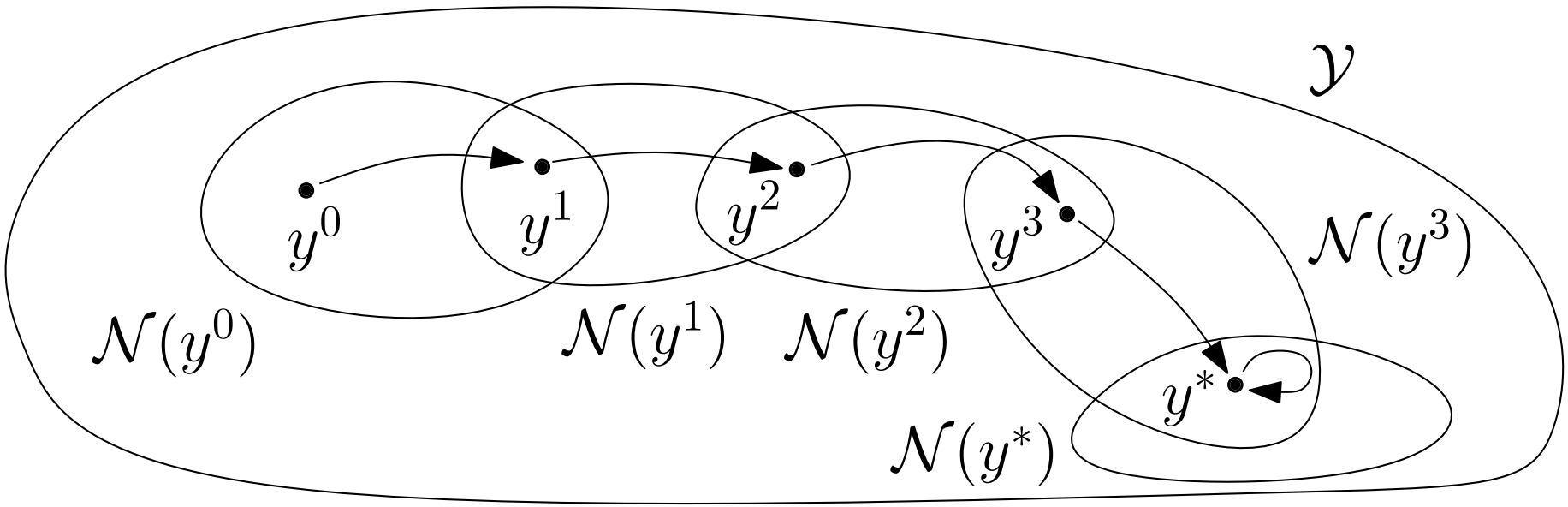
Avoid getting fractional solutions: energy minimization by **local search**



- choose starting labeling y^0
- construct neighborhood $\mathcal{N}(y^0) \subset \mathcal{Y}$ of labelings
- find minimizer within neighborhood, $y^1 = \mathbf{argmin}_{y \in \mathcal{N}(y^0)} E(y)$

Local Search

Avoid getting fractional solutions: energy minimization by **local search**



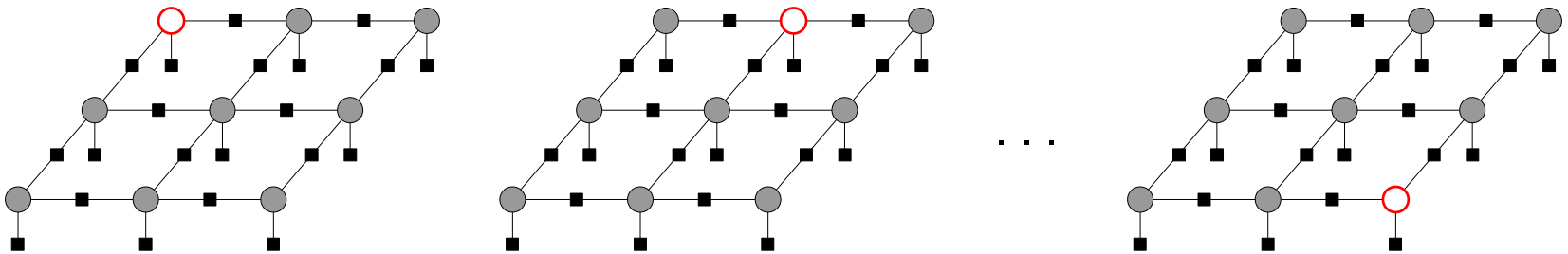
- choose starting labeling y^0
- construct neighborhood $\mathcal{N}(y^0) \subset \mathcal{Y}$ of labelings
- find minimizer within neighborhood, $y^1 = \mathbf{argmin}_{y \in \mathcal{N}(y^0)} E(y)$
- iterate until no more changes

Iterated Conditional Modes (ICM) [Besag, 1986]

Define local neighborhoods:

- $\mathcal{N}_i(y) = \{(y_1, \ldots, y_{i-1}, \bar{y}, y_{i+1}, \ldots, y_n) | \bar{y} \in \mathcal{Y}_i\}$ for $i \in V$.

  all labeling reachable from y by changing value of y_i



Iterated Conditional Modes (ICM) [Besag, 1986]

Define local neighborhoods:

- $\mathcal{N}_i(y) = \{(y_1, \ldots, y_{i-1}, \bar{y}, y_{i+1}, \ldots, y_n) | \bar{y} \in \mathcal{Y}_i\}$ for $i \in V$.

  all labeling reachable from y by changing value of y_i

ICM procedure:

- neighborhood $\mathcal{N}(y) = \bigcup_{i \in V} \mathcal{N}_i(y)$

  all states reachable from y by changing a single variable

- $y^{t+1} = \underset{y \in \mathcal{N}(y^t)}{\mathbf{argmin}} E(y)$ by exhaustive search $\quad (\sum_i |\mathcal{Y}_i|$ evaluations)

Iterated Conditional Modes (ICM) [Besag, 1986]

Define local neighborhoods:

- $\mathcal{N}_i(y) = \{(y_1, \ldots, y_{i-1}, \bar{y}, y_{i+1}, \ldots, y_n) | \bar{y} \in \mathcal{Y}_i\}$ for $i \in V$.

  all labeling reachable from y by changing value of y_i

ICM procedure:

- neighborhood $\mathcal{N}(y) = \bigcup_{i \in V} \mathcal{N}_i(y)$

  all states reachable from y by changing a single variable

- $y^{t+1} = \underset{y \in \mathcal{N}(y^t)}{\mathbf{argmin}} E(y)$ by exhaustive search $\quad (\sum_i |\mathcal{Y}_i|$ evaluations)

Observation: larger neighborhood sizes are better

- ICM: $|\mathcal{N}(y)|$ linear in $|V|$
  $\rightarrow$ many iterations to explore exponentially large $\mathcal{Y}$
- ideal: $|\mathcal{N}(y)|$ exponential in $|V|$,
  $\rightarrow$ but: we must ensure that $\mathbf{argmin}_{y \in \mathcal{N}(y)} E(y)$ remains tractable

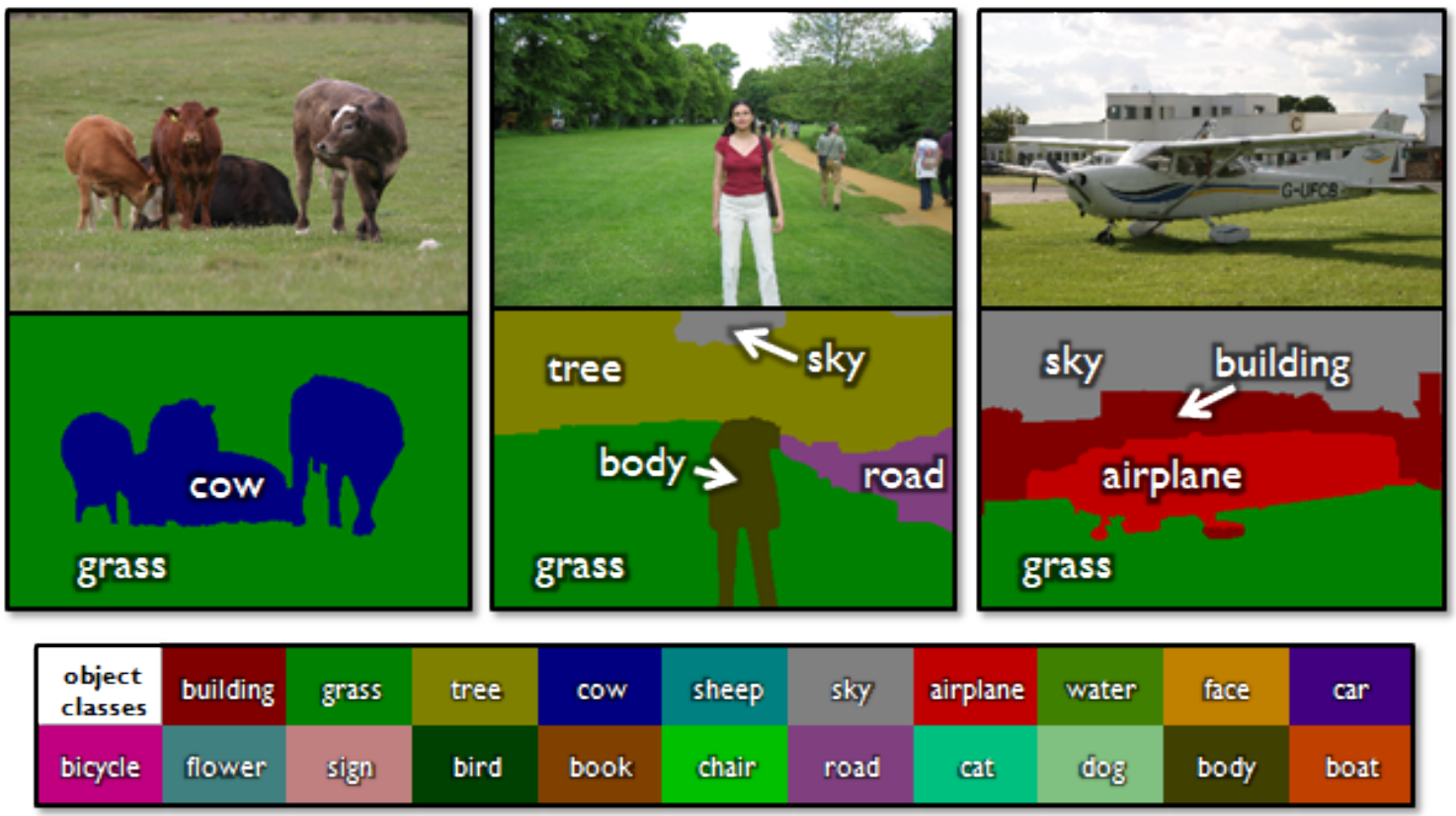
Multilabel Graph-Cut: α-expansion

- $E(y)$ with unary and pairwise terms
- $\mathcal{Y}_i = \mathcal{L} = \{1, \ldots, K\}$ for $i \in V$ (multi-class)

Multilabel Graph-Cut: α-expansion

- $E(y)$ with unary and pairwise terms
- $\mathcal{Y}_i = \mathcal{L} = \{1, \ldots, K\}$ for $i \in V$ (multi-class)

Example: semantic segmentation



Multilabel Graph-Cut: α-expansion

- $E(y)$ with unary and pairwise terms
- $\mathcal{Y}_i = \mathcal{L} = \{1, \ldots, K\}$ for $i \in V$ (multi-class)

Algorithm

- initialize y^0 arbitrarily (e.g. everything label 0)
- repeat
  - for any $\alpha \in \mathcal{L}$
    - construct neighborhood:

$$\mathcal{N}(y) = \{(\bar{y}_1, \ldots, \bar{y}_{|V|}) \ : \ \bar{y}_i \in \{y_i, \alpha\}\}$$

"each variable can keep its value or switch to α"

    - solve $y \leftarrow \mathbf{argmin}_{y \in \mathcal{N}(y)} E(y)$
- until y has not changed for a whole iteration

Multilabel Graph-Cut: α-expansion

Theorem [Boykov *et al*. 2001]
If all pairwise terms are *metric*, i.e. for all $(i,j) \in \mathcal{E}$

$$E_{ij}(k,l) \geq 0 \quad \text{with} \quad E_{ij}(k,l) = 0 \Leftrightarrow k = l$$

$$E_{ij}(k,l) = E_{ij}(l,k)$$

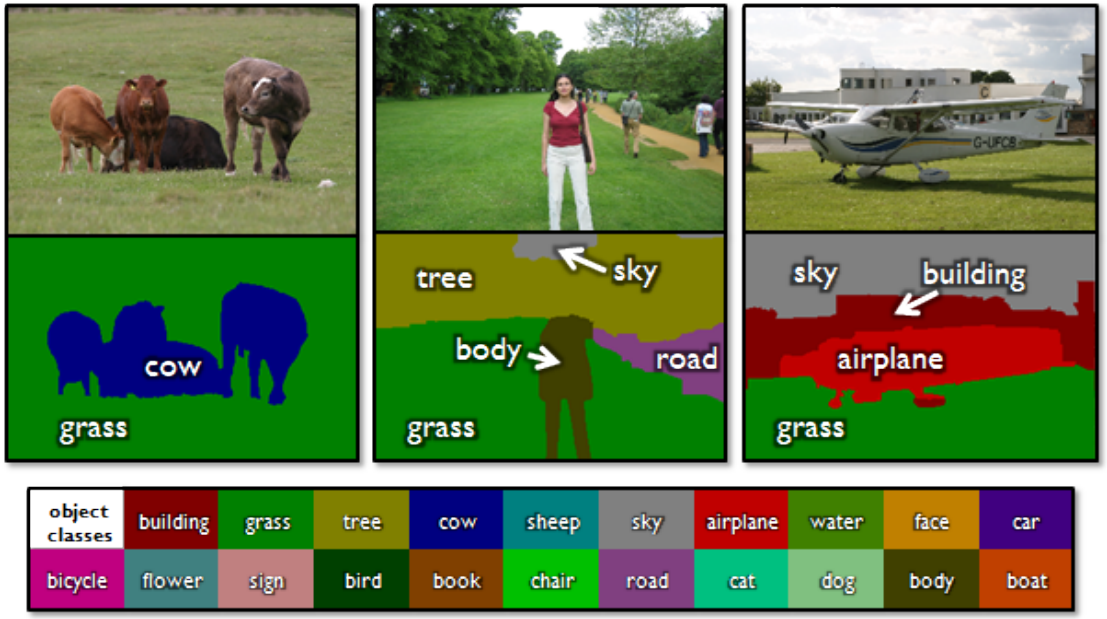
$$E_{ij}(k,l) \leq E_{ij}(k,m) + E_{ij}(m,l) \quad \text{for all } k,l,m$$

Then $\mathbf{argmin}_{y \in \mathcal{N}(y)} E(y)$ can be solved optimally using GraphCut.

Theorem [Veksler 2001]**.** The solution, y_α, returned by α-expansion fulfills

$$E(y_\alpha) \;\leq\; 2c \cdot \mathbf{min}_{y \in \mathcal{Y}} E(y) \quad \text{for } c = \mathbf{max}_{(i,j) \in \mathcal{E}} \frac{\mathbf{max}_{k \neq l} E_{ij}(k,l)}{\mathbf{min}_{k \neq l} E_{ij}(k,l)}$$

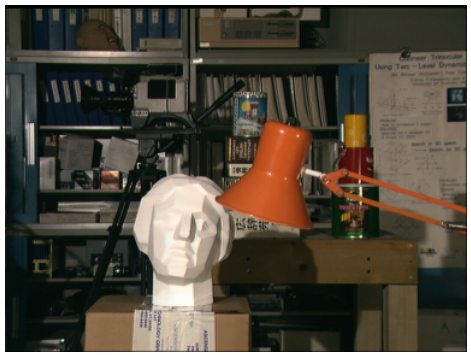
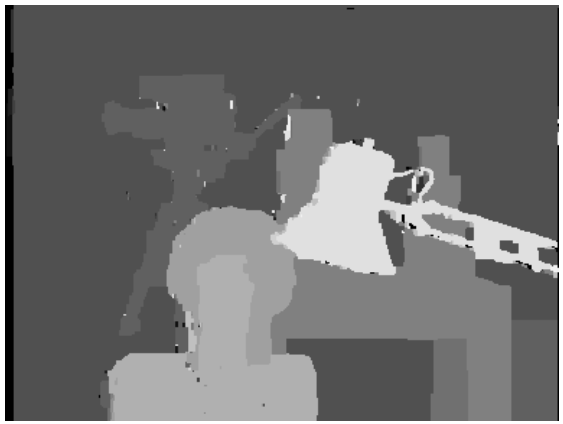
Example: Semantic Segmentation



$$E(y) = \sum_{i \in V} E_i(y_i) + \lambda \sum_{(i,j) \in \mathcal{E}} [\![y_i \neq y_j]\!] \qquad \text{"Potts model"}$$

- $E_{ij}(k,l) \geq 0 \qquad E_{ij}(k,l) = 0 \Leftrightarrow k = l \qquad E_{ij}(k,l) = E_{ij}(l,k)$ ✓
- $E_{ij}(k,l) \leq E_{ij}(k,m) + E_{ij}(m,l)$ ✓
- $c = \mathbf{max}_{(i,j) \in \mathcal{E}} \frac{\mathbf{max}_{k \neq l} E_{ij}(k,l)}{\mathbf{min}_{k \neq l} E_{ij}(k,l)} = 1$
- factor-2 approximation guarantee: $E(y_\alpha) \leq 2\, \mathbf{min}_{y \in \mathcal{Y}} E(y)$

Example: Stereo Estimation

$$E(y) = \sum_{i \in V} E_i(y_i) + \lambda \sum_{(i,j) \in \mathcal{E}} |y_i - y_j|$$

- $|y_i - y_j|$ is metric ✓
- $c = \mathbf{max}_{(i,j) \in \mathcal{E}} \frac{\mathbf{max}_{k \neq l} E_{ij}(k,l)}{\mathbf{min}_{k \neq l} E_{ij}(k,l)} = |\mathcal{L} - 1|$
- weak guarantees, but often close to optimal labelings in practice

Images: Middlebury stereo vision dataset

MAP Prediction / Energy Minimization – Summary

Task: compute $\mathbf{argmin}_{y \in \mathcal{Y}} E(x, y)$

Exact Energy Minimization

Only possible for certain models:

- trees/forests: max-sum belief propagation
- general graphs: junction chain algorithm (if tractable)
- submodular energies: GraphCut
- general graphs: integer linear programming (if tractable)

Approximate Energy Minimization

Many techniques with different properties and guarantees:

- linear programs relaxations
- ICM
- α-expansion

Best choice depends on model and requirements.

Summary

Graphical Models

Model probability distributions with explicit independencies

Conditional Random Fields

Log-linear models of conditional probability $p(y|x;w)$, ML training

Structured Support Vector Machine

Non-probabilistic structured prediction models, maximum margin training

Probabilistic Inference

Compute $p(y_F|x)$ for a subset F of variables: exactly or approximately

MAP Prediction / Energy Minimization

Compute $\mathbf{argmax}_y\, p(y|x)$: exactly or approximately

Structured Loss Function

Measure difference between two structured outputs (task-specific)