CSC 2515: Structured Prediction

Raquel Urtasun & Rich Zemel

University of Toronto

March 16, 2015
Discriminant Functions for $K > 2$ classes

- Use $K - 1$ classifiers, each solving a two class problem of separating point in a class $C_k$ from points not in the class.
- Known as 1 vs all or 1 vs the rest classifier

PROBLEM: More than one good answer!
Discriminant Functions for $K > 2$ classes

- Introduce $K(K - 1)/2$ two-way classifiers, one for each possible pair of classes
- Each point is classified according to majority vote amongst the disc. func.
- Known as the 1 vs 1 classifier

PROBLEM: Two-way preferences need not be transitive
We can avoid these problems by considering a single K-class discriminant comprising $K$ functions of the form

$$y_k(x) = \mathbf{w}_k^T \mathbf{x} + w_{k,0}$$
We can avoid these problems by considering a single $K$-class discriminant comprising $K$ functions of the form

$$y_k(x) = \mathbf{w}_k^T \mathbf{x} + w_{k,0}$$

and then assigning a point $\mathbf{x}$ to class $C_k$ if

$$\forall j \neq k \quad y_k(\mathbf{x}) > y_j(\mathbf{x})$$
We can avoid these problems by considering a single K-class discriminant comprising $K$ functions of the form

$$y_k(x) = w_k^T x + w_{k,0}$$

and then assigning a point $x$ to class $C_k$ if

$$\forall j \neq k \quad y_k(x) > y_j(x)$$

Note that $w_k^T$ is now a vector, not the $k$-th coordinate.
We can avoid these problems by considering a single $K$-class discriminant comprising $K$ functions of the form

$$y_k(x) = w_k^T x + w_{k,0}$$

and then assigning a point $x$ to class $C_k$ if

$$\forall j \neq k \quad y_k(x) > y_j(x)$$

Note that $w_k^T$ is now a vector, not the $k$-th coordinate.

In this lecture we will look at generalizations of this idea.
 Contents

- Introduction to Structured prediction
- Inference
- Learning
What is structured prediction?
In "typical" machine learning

\[ f : \mathcal{X} \rightarrow \mathbb{R} \]

the input \( \mathcal{X} \) can be anything, and the output is a real number (e.g., classification, regression)
Structured Prediction

- In "typical" machine learning
  \[ f : \mathcal{X} \rightarrow \mathbb{R} \]
  the input \( \mathcal{X} \) can be anything, and the output is a real number (e.g., classification, regression)

- In **Structured Prediction**
  \[ f : \mathcal{X} \rightarrow \mathcal{Y} \]
  the input \( \mathcal{X} \) can be anything, and the output is a **complex** object (e.g., image segmentation, parse tree)
Structured Prediction

- In "typical" machine learning
  \[ f : \mathcal{X} \to \mathbb{R} \]
  
  the input \( \mathcal{X} \) can be anything, and the output is a real number (e.g., classification, regression)

- In **Structured Prediction**
  \[ f : \mathcal{X} \to \mathcal{Y} \]
  
  the input \( \mathcal{X} \) can be anything, and the output is a **complex** object (e.g., image segmentation, parse tree)

- In this lecture \( \mathcal{Y} \) is a discrete space, ask me later if you are interested in continuous variables.
We want to predict multiple random variables which are related

Computer Vision:

- Semantic Segmentation (output: pixel-wise labeling)
- Object detection (output: 2D or 3D bounding boxes)
- Stereo Reconstruction (output: 3D map)
- Scene Understanding (output: 3D bounding box reprinting the layout)
Structured Prediction and its Applications

We want to predict multiple random variables which are related

- Natural Language processing
  - Machine Translation (output: sentence in another language)
  - Parsing (output: parse tree)

- Computational Biology
  - Protein Folding (output: 3D protein)
Why structured?

- Independent prediction is good but...
Why structured?

- Independent prediction is good but...

- Neighboring pixels should have same labels (if they look similar).
A graphical model defines

- A family of probability distributions over a set of random variables
- This is expressed via a graph, which encodes the conditional independences of the distribution

Two types of graphical models: Directed and undirected
Bayesian Networks (Directed Graphical Models)

- The graph $G = (V, E)$ is acyclic and directed
- Factorization over distributions by conditioning on parent nodes

$$p(y) = \prod_{i \in V} p(y_i | y_{pa}(i))$$

- Example

$$p(y) = p(y_l | y_k) p(y_k | y_i, y_j) p(y_i) p(y_j)$$
Undirected Graphical Model

- Also called Markov Random Field, or Markov Network
- Graph $G = (V, E)$ is undirected and has no self-edges
- Factorization over cliques

$$p(y) = \frac{1}{Z} \prod_{r \in R} \psi_r(y_r)$$

with $Z = \sum_{y \in \mathcal{Y}} \prod_{r \in R} \psi_r(y_r)$ the partition function

- Example

$$p(y) = \frac{1}{Z} \psi(y_i, y_j) \psi(y_j, y_k) \psi(y_i) \psi(y_j) \psi(y_k)$$

- **Difficulty**: Exponentially many configurations
- Undirected models will be the focus of this lecture
Factor Graph Representation

- Graph $G = (V, F, E)$, with variable nodes $V$, factor nodes $F$ and edges $E$
- **Scope** of a factor $N(F) = \{i \in V : (i, F) \in E\}$
- Factorization over factors

$$p(y) = \frac{1}{Z} \prod_{F \in F} \psi_F(y_{N(F)})$$
Factor graphs are explicit about the factorization

Figure: from [Nowozin et al]
They define the family of distributions and thus the capacity.

Figure: from [Nowozin et al]
Markov Random Fields vs Conditional Random Fields

- Markov Random Fields (MRFs) define

  \[ p(y) = \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(y_{N(F)}) \]

- Conditional Random Fields (CRFs) define

  \[ p(y|x) = \frac{1}{Z(x)} \prod_{F \in \mathcal{F}} \psi_F(y_{N(F)}; x) \]

- \( x \) is not a random variable (i.e., not part of the probability distribution)
The probability is completely determined by the energy

\[ p(y) = \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(y_{N(F)}) \]

\[ = \frac{1}{Z} \exp \left( \log(\psi_F(y_{N(F)})) \right) \]

\[ = \frac{1}{Z} \exp \left( - \sum_{F \in \mathcal{F}} E_F(y_F) \right) \]

where \( E_F(y_F) = - \log(\psi_F(y_{N(F)})) \)
Parameterization: log linear model

- Factor graphs define a family of distributions
- We are interested in identifying individual members by parameters

\[ E_F(y_F) = -w^T \phi_F(y_F) \]

Figure: from [Nowozin et al]
Learning Tasks

- Estimation of the parameters $w$
  \[ E_F(y_F) = -w^T \phi_F(y_F) \]
- Learn the structure of the model
- Learn with hidden variables
Inference Tasks

Given an input $x \in \mathcal{X}$ we want to compute

- **MAP estimate** or minimum energy configuration

$$\arg\max_{y \in \mathcal{Y}} p(y|x) = \arg\max_{y \in \mathcal{Y}} \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_{F}(y_{N(F)}; x, w)$$

$$= \arg\max_{y \in \mathcal{Y}} \exp\left(- \sum_{F \in \mathcal{F}} E_{F}(y_{F}, x, w)\right)$$

$$= \arg\min_{y \in \mathcal{Y}} \sum_{F \in \mathcal{F}} E_{F}(y_{F}, x, w)$$

Marginals $p(y_{i})$ or max marginals $\max_{y_{i} \in \mathcal{Y}_{i}} p(y_{i})$, which requires computing the partition function $Z$, i.e.,

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

$\mu_{F}(y_{F}) = p(y_{F} | x, w)$
Inference Tasks

Given an input $x \in \mathcal{X}$ we want to compute

- **MAP estimate** or minimum energy configuration

$$\text{argmax}_{y \in \mathcal{Y}} p(y|x) = \text{argmax}_{y \in \mathcal{Y}} \frac{1}{Z} \prod_{F \in \mathcal{F}} \psi_F(y_{N(F)}; x, w)$$

$$= \text{argmax}_{y \in \mathcal{Y}} \exp(- \sum_{F \in \mathcal{F}} E_F(y_F, x, w))$$

$$= \text{argmin}_{y \in \mathcal{Y}} \sum_{F \in \mathcal{F}} E_F(y_F, x, w)$$

- **Marginals** $p(y_i)$ or max marginals $\max_{y_i \in \mathcal{Y}_i} p(y_i)$, which requires computing the partition function $Z$, i.e.,

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

$$\mu_F(y_F) = p(y_F|x, w)$$
Inference in Markov Random Fields
Compute the MAP estimate is typically NP-hard

\[
\max_{y \in \mathcal{Y}} p(y | x) = \max_{y \in \mathcal{Y}} \sum_{r \in \mathcal{R}} w^T \phi_r(y_r)
\]
Compute the MAP estimate is typically NP-hard

\[
\max_{y \in \mathcal{Y}} p(y|x) = \max_{y \in \mathcal{Y}} \sum_{r \in \mathcal{R}} w^T \phi_r(y_r)
\]

Notable exceptions are:

- Belief propagation for tree-structure models
Compute the MAP estimate is typically NP-hard

$$\max_{y \in \mathcal{Y}} p(y|x) = \max_{y \in \mathcal{Y}} \sum_{r \in \mathcal{R}} \mathbf{w}^T \phi_r(y_r)$$

Notable exceptions are:

- Belief propagation for tree-structure models
- Graph cuts for binary energies with sub modular potentials
Compute the MAP estimate is typically NP-hard

\[
\max_{y \in \mathcal{Y}} p(y|x) = \max_{y \in \mathcal{Y}} \sum_{r \in \mathcal{R}} w^T \phi_r(y_r)
\]

Notable exceptions are:

- Belief propagation for tree-structure models
- Graph cuts for binary energies with sub modular potentials
- Branch and bound: exponential in worst case, but works much faster in practice
Compute the MAP estimate is typically NP-hard

\[
\max_{\mathbf{y} \in \mathcal{Y}} p(\mathbf{y} | \mathbf{x}) = \max_{\mathbf{y} \in \mathcal{Y}} \sum_{r \in \mathcal{R}} \mathbf{w}^T \phi_r(\mathbf{y}_r)
\]

Notable exceptions are:

- Belief propagation for tree-structure models
- Graph cuts for binary energies with sub modular potentials
- Branch and bound: exponential in worst case, but works much faster in practice

Difficulties

- Deal with the exponentially many states in \( \mathbf{y} \)
Belief Propagation

- **Compact notation**
  \[ \theta_r(y_r) = w^T \phi_r(y_r) \]

- **Inference can be written as**
  \[ \max_{y \in \mathcal{Y}} \sum_{r \in \mathcal{R}} \theta_r(y_r) \]

- **For the example**
  \[ \max_{y_i, y_j, y_k, y_l} \{ \theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_G(y_k, y_l) \} \]
\[
\theta^*(y) = \max_{y_i, y_j, y_k, y_l} \left\{ \theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_H(y_k, y_l) \right\}
\]
Belief Propagation

\[
\theta^*(y) = \max_{y_i, y_j, y_k, y_l} \{ \theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_H(y_k, y_l) \}
\]

\[
= \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + \max_{y_l} \theta_H(y_k, y_l)
\]
Belief Propagation

\[
\theta^*(\mathbf{y}) = \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + \max_{y_l} \theta_H(y_k, y_l) + r_{H \rightarrow Y_k} \in \mathcal{R}^{Y_k}
\]
Belief Propagation

\[ \theta^*(\mathbf{y}) = \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + \max_{y_l} \theta_H(y_k, y_l) \]

\[ = \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + r_{H \rightarrow y_k}(y_k) \]
\[ \theta^*(y) = \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + r_{H \rightarrow y_k}(y_k) \]

\[ = \max_{y_i, y_j} \theta_F(y_i, y_j) + \]

\[ r_{G \rightarrow y_j}(y_j) \in \mathcal{R}^y_j \]
Belief Propagation

\[
\theta^*(y) = \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + r_{H \rightarrow y_k}(y_k)
\]

\[
= \max_{y_i, y_j} \theta_F(y_i, y_j) + r_{G \rightarrow y_j}(y_j)
\]
\[ \theta^*(y) = \max_{y_i, y_k, y_l, y_m} \theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_I(y_m, y_k) + \theta_H(y_l, y_k) \]
\[
\theta^*(y) = \max_{y_i,y_j,y_k,y_l,y_m} \theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_I(y_m, y_k) + \theta_H(y_l, y_k)
\]

\[
= \max_{y_i,y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + \max_{y_m} \theta_I(y_m, y_k) + \max_{y_l} \theta_H(y_l, y_k)
\]
\[ \theta^*(y) = \max_{y_i, y_k, y_j, y_m} \theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_I(y_m, y_k) + \theta_H(y_l, y_k) \]
\[ = \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + \max_{y_m} \theta_I(y_m, y_k) + \max_{y_l} \theta_H(y_l, y_k) \]
\[ = \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + r_{H \rightarrow Y_k}(y_k) + r_{I \rightarrow Y_k}(y_k) \]
\[
\theta^*(\mathbf{y}) = \max_{y_i, y_k, y_k, y_l, y_m} \theta_F(y_i, y_j) + \theta_G(y_j, y_k) + \theta_I(y_m, y_k) + \theta_H(y_l, y_k)
\]

\[
= \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + \max_{y_m} \theta_I(y_m, y_k) + \max_{y_l} \theta_H(y_l, y_k)
\]

\[
= \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + \max_{y_k} (r_{H \rightarrow y_k}(y_k) + r_{I \rightarrow y_k}(y_k)) \rightarrow y_k
\]

\[
= \max_{y_i, y_j} \theta_F(y_i, y_j) + \max_{y_k} \theta_G(y_j, y_k) + q_{y_k \rightarrow G}(y_k)
\]
Iteratively updates and passes messages:

- $r_{F \rightarrow Y_i} \in \Re^{Y_i}$: factor to variable message
- $q_{Y_i \rightarrow F} \in \Re^{Y_i}$: variable to factor message

Figure: from [Nowozin et al]
Variable to factor

- Let $M(i)$ be the factors adjacent to variable i, $M(i) = \{F \in \mathcal{F} : (i, F) \in \mathcal{E}\}$
- Variable-to-factor message

$$q_{y_i \rightarrow F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow y_i}(y_i)$$

Figure: from [Nowozin et al]
Factor to variable

- Factor-to-variable message

\[ r_{F \rightarrow y_i}(y_i) = \max_{y'_F \in Y_F, y'_i = y_i} \left( \theta(y'_F) + \sum_{j \in N(F) \setminus \{i\}} q_{y_j \rightarrow F}(y'_j) \right) \]

**Figure**: from [Nowozin et al]
Message Scheduling

1. Select one variable as tree root
2. Compute leaf-to-root messages
3. Compute root-to-leaf messages

Figure: from [Nowozin et al]
Max Product vs Sum Product

Max sum version of max-product

1. Compute leaf-to-root messages

\[ q_{y_i \rightarrow F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \rightarrow y_i}(y_i) \]

2. Compute root-to-leaf messages

\[ r_{F \rightarrow y_i}(y_i) = \max_{y'_F \in \mathcal{Y}_F, y'_i = y_i} \left( \theta(y'_F) + \sum_{j \in N(F) \setminus \{i\}} q_{y_j \rightarrow F}(y'_j) \right) \]
Max Product v Sum Product

Max sum version of max-product

1. Compute leaf-to-root messages

\[ q_{y_i \to F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \to y_i}(y_i) \]

2. Compute root-to-leaf messages

\[ r_{F \to y_i}(y_i) = \max_{y'_F \in Y_F, y'_i = y_i} \left( \theta(y'_F) + \sum_{j \in N(F) \setminus \{i\}} q_{y_j \to F}(y'_j) \right) \]

Sum-product

1. Compute leaf-to-root messages

\[ q_{y_i \to F}(y_i) = \sum_{F' \in M(i) \setminus \{F\}} r_{F' \to y_i}(y_i) \]

2. Compute root-to-leaf messages

\[ r_{F \to y_i}(y_i) = \log \sum_{y'_F \in Y_F, y'_i = y_i} \exp \left( \theta(y'_F) + \sum_{j \in N(F) \setminus \{i\}} q_{y'_j \to F}(y'_j) \right) \]
Computing marginals

- Partition function can be evaluated at the root

\[
\log Z = \log \sum_{y_r} \exp \left( \sum_{F \in M(r)} r_{F \rightarrow y_r}(y_r) \right)
\]

- Marginal distributions, for each factor

\[
\mu_F(y_F) = p(y_F) = \frac{1}{Z} \exp \left( \theta_F(y_F) + \sum_{i \in N(F)} q_{y_i \rightarrow F}(y_i) \right)
\]
Computing marginals

- Partition function can be evaluated at the root

\[ \log Z = \log \sum_{y_r} \exp \left( \sum_{F \in M(r)} r_{F \rightarrow y_r}(y_r) \right) \]

- Marginal distributions, for each factor

\[ \mu_F(y_F) = p(y_F) = \frac{1}{Z} \exp \left( \theta_F(y_F) + \sum_{i \in N(F)} q_{y_i \rightarrow F}(y_i) \right) \]

- Marginals at every node

\[ \mu_{y_i}(y_i) = p(y_i) = \frac{1}{Z} \exp \left( \sum_{F \in M(i)} r_{F \rightarrow y_i}(y_i) \right) \]
Generalizations to loops

- It is call **loopy belief propagation** (Perl, 1988)
- No schedule that removes dependencies
- Different messaging schedules (synchronous/asynchronous, static/dynamic)
- Slight changes in the algorithm
Integer Linear Program (LP) equivalence [Werner 2007]:

- Inference task:
  \[ \hat{y} = \arg \max_y \sum_r \theta_r(y_r) \]

- Variables \( b_r(y_r) \):
  \[
  \begin{bmatrix}
    b_1(0) \\
    b_1(1) \\
    b_2(0) \\
    b_2(1) \\
    b_{12}(0, 0) \\
    b_{12}(1, 0) \\
    b_{12}(0, 1) \\
    b_{12}(1, 1)
  \end{bmatrix}^T
  \begin{bmatrix}
    \theta_1(0) \\
    \theta_1(1) \\
    \theta_2(0) \\
    \theta_2(1) \\
    \theta_{12}(0, 0) \\
    \theta_{12}(1, 0) \\
    \theta_{12}(0, 1) \\
    \theta_{12}(1, 1)
  \end{bmatrix}
  \]

\[ \max_{b_1, b_2, b_{12}} \]
MAP LP Relaxation Task

Integer Linear Program (LP) equivalence [Werner 2007]:

- Inference task:
  \[ \hat{y} = \arg \max_y \sum_r \theta_r(y_r) \]

- Variables \( b_r(y_r) \):

\[
\begin{bmatrix}
    b_1(0) \\
    b_1(1) \\
    b_2(0) \\
    b_2(1) \\
    b_{12}(0, 0) \\
    b_{12}(1, 0) \\
    b_{12}(0, 1) \\
    b_{12}(1, 1)
\end{bmatrix}^T \begin{bmatrix}
    \theta_1(0) \\
    \theta_1(1) \\
    \theta_2(0) \\
    \theta_2(1) \\
    \theta_{12}(0, 0) \\
    \theta_{12}(1, 0) \\
    \theta_{12}(0, 1) \\
    \theta_{12}(1, 1)
\end{bmatrix}
\]

\[ b_r(y_r) \in \{0, 1\} \]

s.t.
MAP LP Relaxation Task

Integer Linear Program (LP) equivalence [Werner 2007]:

- Inference task:
  \[
  \hat{y} = \arg \max_y \sum_r \theta_r(y_r)
  \]

- Variables \(b_r(y_r)\):

\[
\begin{bmatrix}
  b_1(0) \\
  b_1(1) \\
  b_2(0) \\
  b_2(1) \\
  b_{12}(0, 0) \\
  b_{12}(1, 0) \\
  b_{12}(0, 1) \\
  b_{12}(1, 1)
\end{bmatrix}^{\top}
\begin{bmatrix}
  \theta_1(0) \\
  \theta_1(1) \\
  \theta_2(0) \\
  \theta_2(1) \\
  \theta_{12}(0, 0) \\
  \theta_{12}(1, 0) \\
  \theta_{12}(0, 1) \\
  \theta_{12}(1, 1)
\end{bmatrix}
\]

\[
\max_{b_1, b_2, b_{12}} \quad \sum_{y_r} b_r(y_r) = 1
\]

\[
\begin{cases}
  b_r(y_r) \in \{0, 1\} \\
  \sum_{y_r} b_r(y_r) = 1
\end{cases}
\]
MAP LP Relaxation Task

Integer Linear Program (LP) equivalence [Werner 2007]:

- Inference task:
  \[
  \hat{y} = \arg \max_y \sum_r \theta_r(y_r)
  \]

- Variables \( b_r(y_r) \):

\[
\begin{bmatrix}
  b_1(0) \\
  b_1(1) \\
  b_2(0) \\
  b_2(1) \\
  b_{12}(0, 0) \\
  b_{12}(1, 0) \\
  b_{12}(0, 1) \\
  b_{12}(1, 1)
\end{bmatrix}^T
\begin{bmatrix}
  \theta_1(0) \\
  \theta_1(1) \\
  \theta_2(0) \\
  \theta_2(1) \\
  \theta_{12}(0, 0) \\
  \theta_{12}(1, 0) \\
  \theta_{12}(0, 1) \\
  \theta_{12}(1, 1)
\end{bmatrix}
\]

\[\max_{b_1, b_2, b_{12}} \]

\[b_r(y_r) \in \{0, 1\}\]

\[\sum_{y_r} b_r(y_r) = 1\]

\[\sum_{y_p \setminus y_r} b_p(y_p) = b_r(y_r)\]
MAP LP Relaxation Task

\[
\begin{bmatrix}
    b_1(1) \\
    b_1(2) \\
    b_2(1) \\
    b_2(2) \\
    b_{12}(1, 1) \\
    b_{12}(2, 1) \\
    b_{12}(1, 2) \\
    b_{12}(2, 2)
\end{bmatrix}^T \begin{bmatrix}
    \theta_1(1) \\
    \theta_1(2) \\
    \theta_2(1) \\
    \theta_2(2) \\
    \theta_{12}(1, 1) \\
    \theta_{12}(2, 1) \\
    \theta_{12}(1, 2) \\
    \theta_{12}(2, 2)
\end{bmatrix}
\]

s.t. \( b_r(\mathbf{y}_r) \in \{0, 1\} \)

\[
\sum_{\mathbf{y}_r} b_r(\mathbf{y}_r) = 1
\]

\[
\sum_{\mathbf{y}_p \setminus \mathbf{y}_r} b_p(\mathbf{y}_p) = b_r(\mathbf{y}_r)
\]
MAP LP Relaxation Task

\[
\max_{b_r} \sum_{r, y_r} b_r(y_r) \theta_r(y_r)
\]

s.t.

\[
\begin{align*}
& b_r(y_r) \in \{0, 1\} \\
& \sum_{y_r} b_r(y_r) = 1 \\
& \sum_{y_p \setminus y_r} b_p(y_p) = b_r(y_r)
\end{align*}
\]
MAP LP Relaxation Task

\[
\max_{b_r} \sum_{r, y_r} b_r(y_r) \theta_r(y_r) \quad \text{s.t.} \quad \sum_{y_r} b_r(y_r) = 1
\]

Marginalization

\[b_r(y_r) \in \{0, 1\}\]
MAP LP Relaxation Task

\[
\max_{b_r} \sum_{r,y_r} b_r(y_r) \theta_r(y_r)
\]

s.t.

\[b_r(y_r) \in \{0, 1\}\]

Local probability \(b_r\)

Marginalization
LP relaxation:

$$\max_{b_r} \sum_{r,y_r} b_r(y_r) \theta_r(y_r)$$

s.t. 

$$b_r(y_r) \in \{0, 1\}$$

Local probability $b_r$

Marginalization
LP relaxation:

\[
\max_{b_r} \sum_{r, y_r} b_r(y_r) \theta_r(y_r)
\]

s.t.

\[b_r(y_r) \in \{0, 1\}\]

Local probability \(b_r\)

Marginalization

Can be solved by any standard LP solver but **slow** because of typically many variables and constraints. Can we do better?
**Observation:** Graph structure in marginalization constraints.

Use dual to take advantage of structure in constraint set

- Set of parents of region \( r \): \( P(r) \)
- Set of children of region \( r \): \( C(r) \)

\[
\forall r, y_r, p \in P(r) \quad \sum_{y_p \setminus y_r} b_p(y_p) = b_r(y_r)
\]

- Lagrange multipliers for every constraint:

\[
\forall r, y_r, p \in P(r) \quad \lambda_{r \rightarrow p}(y_r)
\]
Re-parameterization of score $\theta_r(y_r)$:

$$\hat{\theta}_r(y_r) = \theta_r(y_r) + \sum_{p \in P(r)} \lambda_{r \rightarrow p}(y_r) - \sum_{c \in C(r)} \lambda_{c \rightarrow r}(y_c)$$

Properties of dual program:

$$\min_{\lambda} q(\lambda) = \min_{\lambda} \sum_r \max_{y_r} \hat{\theta}_r(y_r)$$

- **Dual upper-bounds primal** $\forall \lambda$
- Convex problem
- Unconstrained task
- Doing block coordinate descent in the dual results on message passing (Lagrange multipliers are your messages)
Block-coordinate descent solvers iterate the following steps:

- Take a block of Lagrange multipliers
- Optimize sub-problem of dual function w.r.t. this block while keeping all other variables fixed

**Advantage:** fast due to analytically computable sub-problems

Same type of algorithms also exist to compute approximate marginals
**Theorem [Kolmogorov and Zabih, 2004]:** If the energy function is a function of binary variables containing only unary and pairwise factors, the discrete energy minimization problem

$$\min_y \sum_{r \in R} E(y_r, x)$$

can be formulated as a graph cut problem if all pairwise energies are submodular

$$E_{i,j}(0,0) + E_{i,j}(1,1) \leq E_{i,j}(0,1) + E_{i,j}(1,0)$$
The ST-mincut problem

- The st-mincut is the st-cut with the minimum cost

[Source: P. Kohli]
Back to our energy minimization

Construct a graph such that

1. Any st-cut corresponds to an assignment of $x$
2. The cost of the cut is equal to the energy of $x$: $E(x)$

[Source: P. Kohli]
St-mincut and Energy Minimization

\[ E(x) = \sum_i \theta_i(x_i) + \sum_{i,j} \theta_{ij}(x_i, x_j) \]

For all \( ij \)

\[ \theta_{ij}(0,1) + \theta_{ij}(1,0) \geq \theta_{ij}(0,0) + \theta_{ij}(1,1) \]

Equivalent (transformable)

\[ E(x) = \sum_i c_i x_i + \sum_{i,j} c_{ij} x_i(1-x_j) \]

\[ c_{ij} \geq 0 \]

[Source: P. Kohli]
How are they equivalent?

\[
A = \theta_{ij}(0,0) \quad B = \theta_{ij}(0,1) \quad C = \theta_{ij}(1,0) \quad D = \theta_{ij}(1,1)
\]

\[
\begin{array}{c|c|c|c|c|c|c}
0 & x_j & 1 & \\
\hline
x_i & A & B & C & D & \\
\hline
0 & & & & & \\
1 & & & & & \\
\end{array}
\]

\[
= A + (\theta_{ij}(1,0) - \theta_{ij}(0,0)) x_i + (\theta_{ij}(1,0) - \theta_{ij}(0,0)) x_j + (\theta_{ij}(1,0) + \theta_{ij}(0,1) - \theta_{ij}(0,0) - \theta_{ij}(1,1)) (1-x_i) x_j
\]

If \(x_1 = 1\) add \(C-A\)  
If \(x_2 = 1\) add \(D-C\)

\[
\begin{array}{c|c|c|c|c|c|c|c}
0 & 1 & & & & & & \\
\hline
0 & 0 & & & & & & \\
1 & C-A & C-A & & & & & \\
\hline
0 & 0 & & & & & & \\
1 & 0 & & & & & & \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c}
0 & 1 & & & & & & \\
\hline
0 & D-C & & & & & & \\
1 & 0 & & & & & & \\
\hline
0 & 0 & & & & & & \\
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c}
0 & 1 & & & & & & \\
\hline
0 & B & +C-A-D & & & & & \\
1 & 0 & & & & & & \\
\end{array}
\]

[B+C-A-D \geq 0 is true from the submodularity of \(\theta_{ij}\)]

[Source: P. Kohli]
Graph Construction

\[ E(a_1, a_2) \]

Source (0)

\( a_1 \)

\( a_2 \)

Sink (1)

[Source: P. Kohli]
Graph Construction

\[ E(a_1, a_2) = 2a_1 \]

[Source: P. Kohli]
Graph Construction

\[ E(a_1, a_2) = 2a_1 + 5\bar{a}_1 \]

[Source: P. Kohli]
Graph Construction

\[ E(a_1, a_2) = 2a_1 + 5\bar{a}_1 + 9a_2 + 4\bar{a}_2 \]

[Source: P. Kohli]
Graph Construction

\[ E(a_1, a_2) = 2a_1 + 5a_1 + 9a_2 + 4a_2 + 2a_1a_2 \]

[Source: P. Kohli]
Graph Construction

\[ E(a_1, a_2) = 2a_1 + 5a_1 + 9a_2 + 4a_2 + 2a_1a_2 + \bar{a}_1a_2 \]
Graph Construction

\[ E(a_1, a_2) = 2a_1 + 5\bar{a}_1 + 9a_2 + 4\bar{a}_2 + 2a_1\bar{a}_2 + \bar{a}_1a_2 \]

[Source: P. Kohli]
Graph Construction

\[ E(a_1, a_2) = 2a_1 + 5\bar{a}_1 + 9a_2 + 4\bar{a}_2 + 2a_1\bar{a}_2 + \bar{a}_1a_2 \]

[Source: P. Kohli]
Graph Construction

\[ E(a_1, a_2) = 2a_1 + 5\bar{a}_1 + 9a_2 + 4\bar{a}_2 + 2a_1\bar{a}_2 + \bar{a}_1a_2 \]

[Source: P. Kohli]
How to compute the St-mincut?

Solve the dual maximum flow problem:

Compute the maximum flow between Source and Sink s.t.

- Edges: Flow < Capacity
- Nodes: Flow in = Flow out

**Min-cut/Max-flow Theorem**

In every network, the maximum flow equals the cost of the st-mincut.

**Assuming non-negative capacity**

[Source: P. Kohli]
How does the code look like

```c
Graph *g;

For all pixels p

    /* Add a node to the graph */
    nodeID(p) = g->add_node();

    /* Set cost of terminal edges */
    set_weights(nodeID(p), fgCost(p), bgCost(p));

end

for all adjacent pixels p,q

    add_weights(nodeID(p), nodeID(q), cost(p,q));
end

g->compute_maxflow();

label_p = g->is_connected_to_source(nodeID(p));
// is the label of pixel p (0 or 1)
```

[Source: P. Kohli]
How does the code look like

```c
Graph *g;

For all pixels p
    /* Add a node to the graph */
    nodeID(p) = g->add_node();
    /* Set cost of terminal edges */
    set_weights(nodeID(p), fgCost(p), bgCost(p));
end

for all adjacent pixels p,q
    add_weights(nodeID(p), nodeID(q), cost(p,q));
end

g->compute_maxflow();

label_p = g->is_connected_to_source(nodeID(p));
// is the label of pixel p (0 or 1)
```

[Source: P. Kohli]
How does the code look like

```c
Graph *g;

For all pixels p
    /* Add a node to the graph */
    nodeID(p) = g->add_node();
    /* Set cost of terminal edges */
    set_weights(nodeID(p), fgCost(p), bgCost(p));
end

for all adjacent pixels p,q
    add_weights(nodeID(p), nodeID(q), cost(p,q));
end

g->compute_maxflow();

label_p = g->is_connected_to_source(nodeID(p));
// is the label of pixel p (0 or 1)
```

[Source: P. Kohli]
How does the code look like

Graph *g;

For all pixels p

    /* Add a node to the graph */
    nodeId(p) = g->add_node();

    /* Set cost of terminal edges */
    set_weights(nodeId(p), fgCost(p), bgCost(p));

end

for all adjacent pixels p,q

    add_weights(nodeId(p), nodeId(q), cost(p,q));

end

    g->compute_maxflow();

label_p = g->is_connected_to_source(nodeId(p));
    // is the label of pixel p (0 or 1)

[Source: P. Kohli]
Example: Figure-Ground Segmentation

Binary labeling problem

(Original) (Color model) (Indep. Prediction)

*Figure*: from [Nowozin et al]
Example: Figure-Ground Segmentation

Markov Random Field

\[ E(y, x, w) = \sum_i \log p(y_i | x_i) + w \sum_{(i,j) \in E} C(x_i, x_j) I(y_i \neq y_j) \]

with \( C(x_i, x_j) = \exp(\gamma \| x_i - x_j \|^2) \), and \( w \geq 0 \).

**Figure:** from [Nowozin et al]

- (w=0)
- (w small)
- (w medium)
- (large w)

Why do we need the condition \( w \geq 0 \)?
Generalization to Multi-label Problems

- Optimal solution is not possible anymore
- Solve to optimality subproblems that include current iterate
- This guarantees decrease in the objective

Figure: from [Nowozin et al]
Generalization to Multi-label Problems

- Optimal solution is not possible anymore
- Solve to optimality subproblems that include current iterate
- This guarantees decrease in the objective

Figure: from [Nowozin et al]
Generalization to Multi-label Problems

- Optimal solution is not possible anymore
- Solve to optimality subproblems that include current iterate
- This guarantees decrease in the objective

Figure: from [Nowozin et al]
Generalization to Multi-label Problems

- Optimal solution is not possible anymore
- Solve to optimality subproblems that include current iterate
- This guarantees decrease in the objective

Figure: from [Nowozin et al]
Metric vs Semimetric

Two general classes of pairwise interactions

- **Metric** if it satisfies for any set of labels $\alpha, \beta, \gamma$

  \[
  V(\alpha, \beta) = 0 \iff \alpha = \beta
  \]

  \[
  V(\alpha, \beta) = V(\beta, \alpha) \geq 0
  \]

  \[
  V(\alpha, \beta) \leq V(\alpha, \gamma) + V(\gamma, \beta)
  \]
Metric vs Semimetric

Two general classes of pairwise interactions

- **Metric** if it satisfies for any set of labels $\alpha, \beta, \gamma$

  \[
  V(\alpha, \beta) = 0 \iff \alpha = \beta \\
  V(\alpha, \beta) = V(\beta, \alpha) \geq 0 \\
  V(\alpha, \beta) \leq V(\alpha, \gamma) + V(\gamma, \beta)
  \]

- **Semi-metric** if it satisfies for any set of labels $\alpha, \beta, \gamma$

  \[
  V(\alpha, \beta) = 0 \iff \alpha = \beta \\
  V(\alpha, \beta) = V(\beta, \alpha) \geq 0
  \]
Examples for 1D label set

- Truncated quadratic is a semi-metric

\[ V(\alpha, \beta) = \min(K, |\alpha - \beta|^2) \]

with \( K \) a constant.
Examples for 1D label set

- Truncated quadratic is a semi-metric

\[ V(\alpha, \beta) = \min(K, |\alpha - \beta|^2) \]

with \( K \) a constant.

- Truncated absolute distance is a metric

\[ V(\alpha, \beta) = \min(K, |\alpha - \beta|) \]

with \( K \) a constant.
Examples for 1D label set

- Truncated quadratic is a semi-metric

\[ V(\alpha, \beta) = \min(K, |\alpha - \beta|^2) \]

with \( K \) a constant.

- Truncated absolute distance is a metric

\[ V(\alpha, \beta) = \min(K, |\alpha - \beta|) \]

with \( K \) a constant.

- Potts model is a metric

\[ V(\alpha, \beta) = K \cdot T(\alpha \neq \beta) \]

with \( T(\cdot) = 1 \) if the argument is true and 0 otherwise.
**Move Making Algorithms**

- **Alpha Expansion**: Checks if current nodes want to switch to label $\alpha$.
- **Alpha - Beta Swaps**: Checks if a node with class $\alpha$ wants to switch to $\beta$.
- Binary problems that can be solved exactly for certain type of potentials.

*Figure*: Alpha-beta Swaps. Figure from [Nowozin et al].
Move Making Algorithms

- **Alpha Expansion**: Checks if current nodes want to switch to label $\alpha$.
- **Alpha - Beta Swaps**: Checks if a node with class $\alpha$ wants to switch to $\beta$.
- Binary problems that can be solve exactly for certain type of potentials.

**Figure**: Alpha-beta Swaps. Figure from [Nowozin et al]
Move Making Algorithms

- **Alpha Expansion**: Checks if current nodes want to switch to label $\alpha$
- **Alpha - Beta Swaps**: Checks if a node with class $\alpha$ wants to switch to $\beta$.
- Binary problems that can be solved exactly for certain types of potentials

**Figure**: Alpha-beta Swaps. Figure from [Nowozin et al]
Move Making Algorithms

- **Alpha Expansion**: Checks if current nodes want to switch to label $\alpha$
- **Alpha - Beta Swaps**: Checks if a node with class $\alpha$ wants to switch to $\beta$.
- Binary problems that can be solved exactly for certain type of potentials

**Figure**: Alpha-beta Swaps. Figure from [Nowozin et al]
Move Making Algorithms

- **Alpha Expansion**: Checks if current nodes want to switch to label $\alpha$
- **Alpha - Beta Swaps**: Checks if a node with class $\alpha$ wants to switch to $\beta$.
- Binary problems that can be solved exactly for certain type of potentials

Figure: Alpha-beta Swaps. Figure from [Nowozin et al]
Binary Moves

- $\alpha - \beta$ moves works for semi-metrics
- $\alpha$ expansion works for $V$ being a metric

\[ x = t x^1 + (1 - t) x^2 \]

Minimize over move variables $t$

Figure: from P. Kohli tutorial on graph-cuts

- For certain $x^1$ and $x^2$, the move energy is sub-modular
Graph Construction

- The set of vertices includes the two terminals $\alpha$ and $\beta$, as well as image pixels $p$ in the sets $P_\alpha$ and $P_\beta$ (i.e., $f_p \in \{\alpha, \beta\}$).
- Each pixel $p \in P_{\alpha\beta}$ is connected to the terminals $\alpha$ and $\beta$, called $t$-links.
- Each set of pixels $p, q \in P_{\alpha\beta}$ which are neighbors is connected by an edge $e_{p,q}$.