Deep Structured Models

Deep Learning + Graphical Model

Chen Liang, Ruoqi Shen, Yu Su, Yongliang Zhang

California Institute of Technology

May 11, 2017

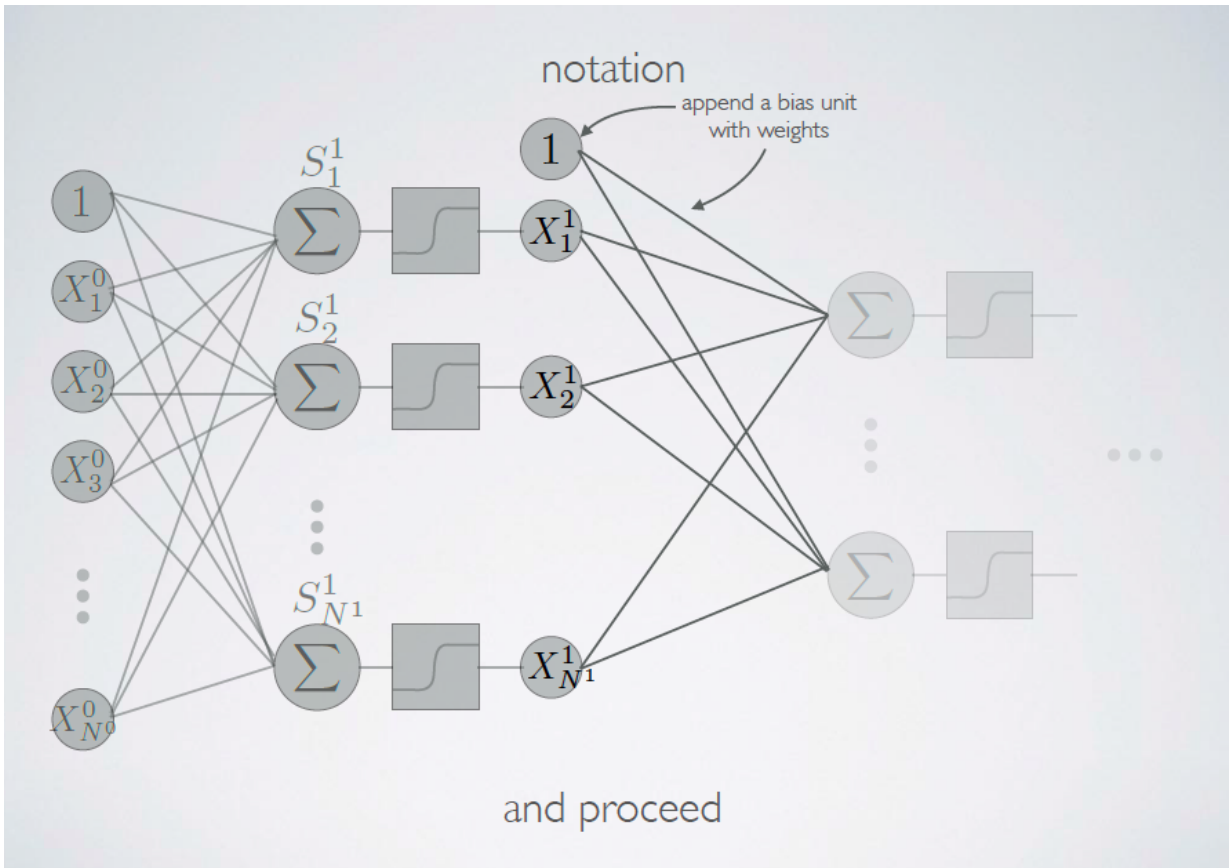
Outline

Outline

Review of Deep Neural Network

Deep neural network:

- Match the data's distribution $f(data) \sim P(label|data)$.
- multiple hidden layers of units: neurons, activations
- deep structures $\rightarrow$ non-linear classifier



Review of Deep Neural Network

output

categorical labels: $Y \in \{1, \ldots, K\}$
(multi-class classification)

represent with K output units,
multi-class logistic regression at the last layer

one-hot vector encoding

example $K = 5$

$Y = 1$	$Y = 2$	$Y = 3$	$Y = 4$	$Y = 5$
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

Review of Deep Neural Network

backpropagation

recall

$$X^\ell = \sigma(S^\ell) \qquad S^\ell = W^\ell X^{\ell-1}$$

at the output layer

$$\frac{\partial \mathcal{L}}{\partial W^L} = \frac{\partial \mathcal{L}}{\partial X^L}\frac{\partial X^L}{\partial S^L}\frac{\partial S^L}{\partial W^L}$$

the first two terms are the same $\frac{\partial \mathcal{L}}{\partial S^L}$

at the layer before

$$\frac{\partial \mathcal{L}}{\partial W^{L-1}} = \frac{\partial \mathcal{L}}{\partial X^L}\frac{\partial X^L}{\partial S^L}\frac{\partial S^L}{\partial X^{L-1}}\frac{\partial X^{L-1}}{\partial S^{L-1}}\frac{\partial S^{L-1}}{\partial W^{L-1}}$$

Review of Deep Neural Network

backpropagation

general overview - *dynamic programming*

compute gradient of loss w.r.t. W^L

store $\frac{\partial \mathcal{L}}{\partial S^L}$

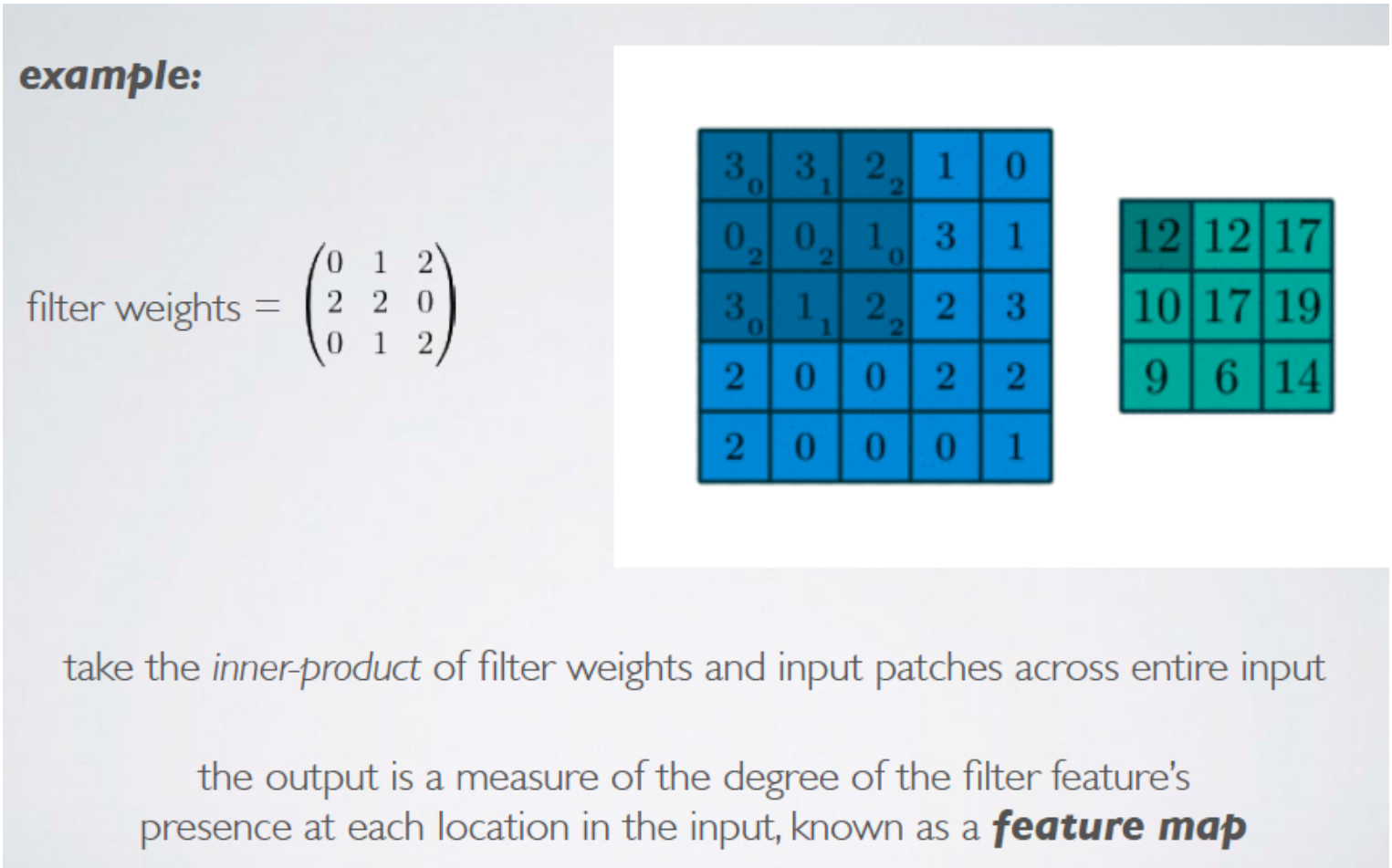
for $\ell = L-1, \ldots, 1$

use $\frac{\partial \mathcal{L}}{\partial S^{\ell+1}}$ to compute gradient of loss w.r.t. W^ℓ

store $\frac{\partial \mathcal{L}}{\partial S^\ell}$

Review of Convolutional Neural Network

Convolutional neural networks are inspired by biological processes. The response of an individual neuron to stimuli within its receptive field can be approximated by the convolution operation.



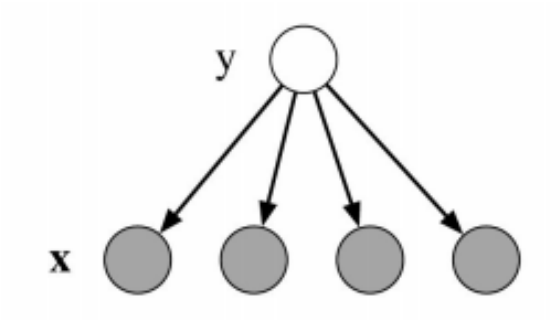
Review of Graphical Models

- Dataset $D = \{(\mathbf{x}, \mathbf{y})\}$ drawn from the probability distribution $p(\mathbf{x}, \mathbf{y})$.
- Generative models: learn the joint probability $p(\mathbf{x}, \mathbf{y})$
- Hard: diff y_i have complicated dependencies
- Assumptions for simplification: which variables affect other variables.
- A graphical model:
  - vertices: output random variables
  - edges: conditional-dependence structures

Review of Bayes Network

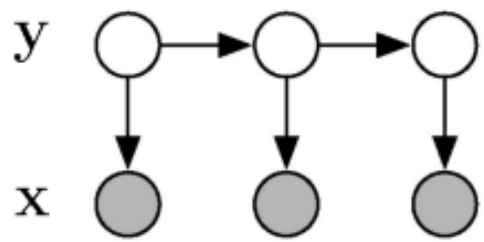
Bayes network: directed a-cyclic probabilistic graphical model.
Example: Naive Bayes Network and Hidden Markov Model.

Naive Bayes



$$p(\mathbf{x}, y) = p(y) \prod_i p(\mathbf{x}_i|y)$$

HMM



$$p(\mathbf{x}, \mathbf{y}) = \prod_i p(\mathbf{x}_i|\mathbf{y}_i)p(\mathbf{y}_i|\mathbf{y}_{i-1})$$

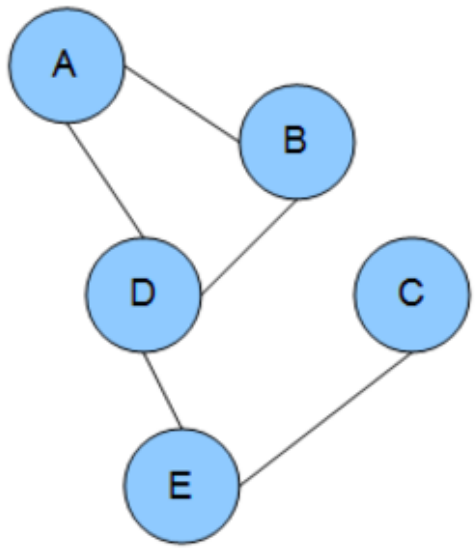
Roughly speaking, an edge from A to B means B depends on A. Absence of edge means conditional independence.

Review of Markov Random Field Graphical Model

Markov Random Field (MRF): undirected graphical probabilistic model. Each edge represents dependency. The joint probability can be decomposed to

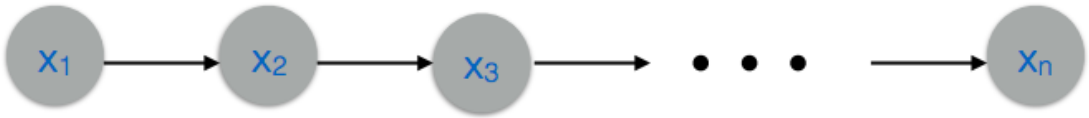
$$p(x_1, x_2, \ldots, x_N) = \frac{1}{Z} \prod_{c \in C} \phi_c(x_c) \tag{1}$$

where C is the set of cliques and Z is the total partition function.
Example: $p(A, B, C, D, E) = \phi_1(A, B, D)\phi_2(D, E)\phi_3(C, E)$.



Review of Message passing

Exact inference of graphical model: message passing



$$p(x_1, ..., x_n) = p(x_1) \prod_{i=2}^{n} p(x_i \mid x_{i-1}).$$

$$p(x_n) = \sum_{x_1} \cdots \sum_{x_{n-1}} p(x_1) \prod_{i=2}^{n} p(x_i \mid x_{i-1})$$

$$= \sum_{x_{n-1}} p(x_n \mid x_{n-1}) \sum_{x_{n-2}} p(x_{n-1} \mid x_{n-2}) \cdots \underbrace{\sum_{x_1} p(x_2 \mid x_1) p(x_1)}_{m_{12}(x_2)}$$

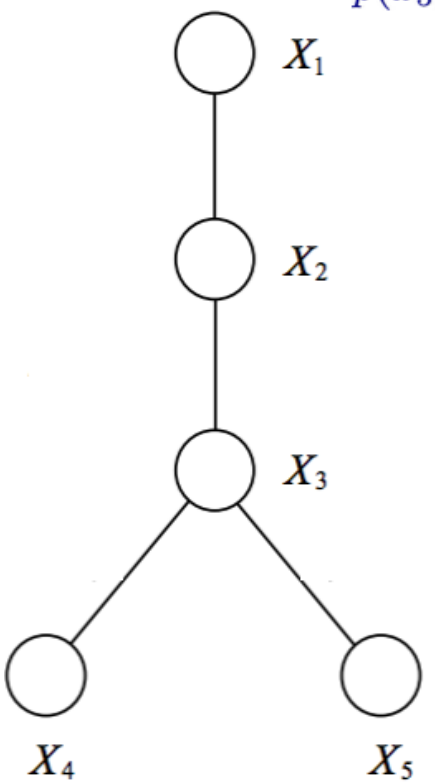
$$p(x_n) = \sum_{x_{n-1}} p(x_n \mid x_{n-1}) \sum_{x_{n-2}} p(x_{n-1} \mid x_{n-2}) \cdots \underbrace{\sum_{x_2} p(x_3 \mid x_2) m_{12}(x_2)}_{m_{23}(x_3)}$$

$$= \dots$$

$$= m_{(n-1)n}(x_n)$$

Review of Message passing

$p(\mathbf{x}) = \frac{1}{Z}\phi_{12}(x_1, x_2)\phi_{23}(x_2, x_3)\phi_{34}(x_3, x_4)\phi_{35}(x_3, x_5)$.
Elimination order of message passing to calculate $p(x_5)$: (1,2,4,3)



$$p(x_5) = \frac{1}{Z}\sum_{x_3}\phi_{35}(x_3, x_5)\sum_{x_4}\phi_{34}(x_3, x_4)\sum_{x_2}\phi_{23}(x_2, x_3)\underbrace{\sum_{x_1}\phi_{12}(x_1, x_2)}_{m_{12}(x_2)}$$

$$= \frac{1}{Z}\sum_{x_3}\phi_{35}(x_3, x_5)\sum_{x_4}\phi_{34}(x_3, x_4)\underbrace{\sum_{x_2}\phi_{23}(x_2, x_3)m_{12}(x_2)}_{m_{23}(x_3)}$$

$$= \frac{1}{Z}\sum_{x_3}\phi_{35}(x_3, x_5)\sum_{x_4}\phi_{34}(x_3, x_4)m_{23}(x_3)$$

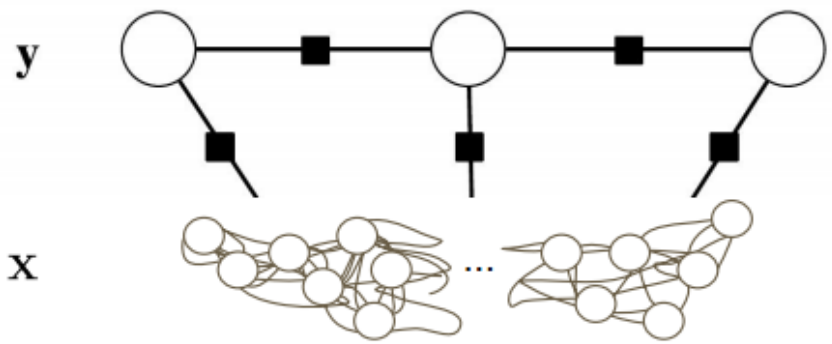
$$= \frac{1}{Z}\sum_{x_3}\phi_{35}(x_3, x_5)m_{23}(x_3)\underbrace{\sum_{x_4}\phi_{34}(x_3, x_4)}_{m_{43}(x_3)}$$

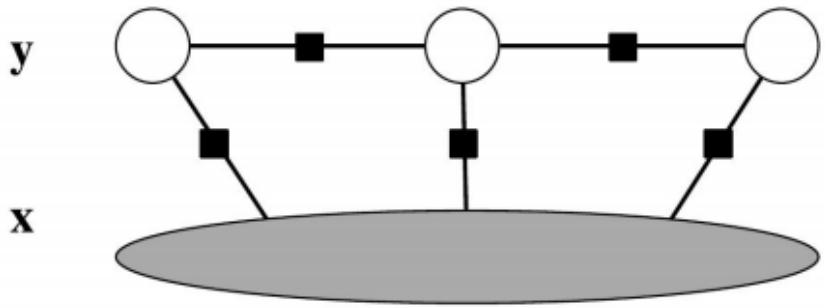
$$= \frac{1}{Z}\sum_{x_3}\phi_{35}(x_3, x_5)m_{23}(x_3)m_{43}(x_3)$$

$$= \frac{1}{Z}m_{35}(x_5)$$

Review of Conditional Random Field Graphical Model

Conditional Random Fields are a type of discriminative undirected probabilistic graphical model. "Discriminative" means that CRF tries to learn the conditional distribution $p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z}\prod_z e^{\Psi_a(\mathbf{x}_a,\mathbf{y}_a;\theta_a)}$.
While the Markov Random Field graphical model tries to learn the joint distribution $p(\mathbf{x},\mathbf{y}) = \frac{1}{Z}\prod_z e^{\Psi_a(\mathbf{x}_a,\mathbf{y}_a;\theta_a)}$.





Log-linear model: $\Psi_a(\mathbf{x}_a,\mathbf{y}_a;\theta_a) = \theta_a^T\phi_a(\mathbf{x}_a,\mathbf{y}_a)$

Motivation of Deep Learning + Graphical Model

- $\mathbf{y} = (y_1, y_2, \ldots, y_N)$: vector of output random variables
- Deep and Convolutional neural network: no dependency between diff output random variables.

  example: one-hot vector
- Deep structured model:
  - combine deep learning and graphical model (MRF or CRF)
  - capture the statistical dependencies
- Learning deep structured models develop an more efficient algorithm that blends learning and inference.

Outline

Learning Deep Structured Models

Chen et al, Learning Deep Structured Models, ICML 2015

Combine MRFs with deep learning

- Learn structured models jointly with deep feature
- Blend learning and inference

Learning Deep Structured Models

Data samples: $(x, y) \in D$
Scoring function: $F(x, y; w)$ with parameter w
Inference:
Given input $x \in X$, find the highest scoring configuration:

$$y* = \arg\max_y F(x, y; w)$$

Learning Deep Structured Models

Inference: $y* = \arg\max_y F(x, y; w)$
Scoring function: $F(x, y; w)$

Log-linear models (CRF, structured SVMs):

$$F(x, y; w) = w^T \psi(x, y)$$

Non-linear models (CNNs):

$$F(x, y; w)$$

Learning Deep Structured Models

This work considers the **general setting** where $F(x, y; w)$ is an arbitrary scalar-valued function of w and (x, y).
F is a function composition of non-linear base mappings such as convolutions, rectifications and pooling.

Learning Deep Structured Models

We let **the probability of an arbitrary configuration** $\hat{y}$ be given by the annealed soft-max

$$p_{x,y}(\hat{y}|w,\epsilon) = \frac{1}{Z_\epsilon(x,w)} exp(F(x,\hat{y};w))^{1/\epsilon}$$

$Z_\epsilon(x,w)$ - **partition function** (normalize the distribution $p(x,y)$ to lie within the probability simplex)

$$Z_\epsilon(x,w) = \sum_{\hat{y}\in Y} exp(F(x,\hat{y};w))^{1/\epsilon}$$

$\epsilon \geq 0$ - **annealing parameter** (adjust the uniformity of the distribution)

Learning Deep Structured Models

Given a training set D of input-output pairs $(x, y) \in D$, **find the parameters** w of the model.
Find by **maximizing the data likelihood**

$$\min_{w} -\ln \prod_{(x,y)\in D} p_{(x,y)}(y|w, \epsilon)$$

Learning Deep Structured Models

Given a training set D of input-output pairs $(x, y) \in D$, **find the parameters** w of the model.
Find by **maximizing the data likelihood**

$$\min_{w} -\ln \prod_{(x,y)\in D} p_{(x,y)}(y|w, \epsilon)$$

$$p_{x,y}(\hat{y}|w, \epsilon) = \frac{1}{Z_\epsilon(x, w)} exp(F(x, \hat{y}; w))^{1/\epsilon}$$

Learning Deep Structured Models

Given a training set D of input-output pairs $(x, y) \in D$, **find the parameters** w of the model.
Find by **maximizing the data likelihood**

$$\min_w -\ln \prod_{(x,y)\in D} p_{(x,y)}(y|w, \epsilon)$$

$$p_{x,y}(\hat{y}|w, \epsilon) = \frac{1}{Z_\epsilon(x, w)} exp(F(x, \hat{y}; w))^{1/\epsilon}$$

Equivalently,

$$\min_w \sum_{(x,y)\in D} (\epsilon \ln Z_\epsilon(x, w) - F(x, y; w))$$

Learning Deep Structured Models

$$\min_{w} \sum_{(x,y)\in D} (\epsilon \ln Z_\epsilon(x, w) - F(x, y; w))$$

optimize via **gradient descent**

Learning Deep Structured Models

$$\min_{w} \sum_{(x,y)\in D} (\epsilon \ln Z_\epsilon(x, w) - F(x, y; w))$$

optimize via **gradient descent**
Plug in $Z_\epsilon(x, w)$,

$$\frac{\partial}{\partial w} \sum_{(x,y)\in D} (\epsilon \ln Z_\epsilon(x, w) - F(x, y; w))$$

$$= \sum_{(x,y)\in D} \sum_{\hat{y}\in Y} \frac{\partial}{\partial w} F(x, \hat{y}; w)(p_{(x,y)}(\hat{y}|w, \epsilon) - \delta(y = \hat{y}))$$

Learning Deep Structured Models

A gradient descent algorithm for minimizing

$$\min_{w} -\ln \prod_{(x,y)\in D} p_{(x,y)}(y|w,\epsilon)$$

Learning Deep Structured Models

A gradient descent algorithm for minimizing

$$\min_{w} -\ln \prod_{(x,y)\in D} p_{(x,y)}(y|w,\epsilon)$$

Algorithm: Deep Structured Learning
Repeat until stopping criteria

1. Forward pass to compute $F(x, \hat{y}; w)$
2. Obtain $p_{(x,y)}(\hat{y}|w,\epsilon)$ via a soft-max
3. Backward pass via chain rule to obtain gradient
4. Update parameters w

Learning Deep Structured Models

The exact computation of $p_{(x,y)}(\hat{y}|w;\epsilon)$ is **not possible** since the state-space size

$$|Y| = \prod_{i=1}^{N} |Y_i|$$

is exponential in the number of variables N.

Learning Deep Structured Models

For most application, $F(x, y; w)$ **decomposes into a sum of functions, each depending on a local subset of variables** y_r, i.e.,

$$F(x, y; w) = \sum_{r \in R} f_r(x, y_r; w)$$

r is a restriction of the variable tuple $y = (y_1, ..., y_N)$ to the subset $r \subseteq \{1, ..., N\}$, i.e., $y_r = (y_i)_{i \in r}$.
All subsets r required to compute the model function F are summarized in the set R.

Learning Deep Structured Models

We make use of the following identity (Wainwright & Jordan, 2008; Koller & Friedman, 2009):

$$\epsilon \ln Z_\epsilon = \max_{p_{(x,y)} \in \Delta} \mathbb{E}[F(x, \hat{y}; w)] + \epsilon H(p_{(x,y)})$$

Learning Deep Structured Models

We make use of the following identity (Wainwright & Jordan, 2008; Koller & Friedman, 2009):

$$\epsilon \ln Z_\epsilon = \max_{p_{(x,y)} \in \Delta} \mathbb{E}[F(x, \hat{y}; w)] + \epsilon H(p_{(x,y)})$$

Pluggin in

$$F(x, y; w) = \sum_{r \in R} f_r(x, y_r; w)$$

We can get

$$\epsilon \ln Z_\epsilon = \max_{p_{(x,y)} \in \Delta} \sum_{r, \hat{y}_r} p_{(x,y),r}(\hat{y}_r) f_r(x, \hat{y}_r; w) + \epsilon H(p_{(x,y)})$$

where marginals $p_{(x,y),r}(\hat{y}_r) = \sum_{y \setminus y_r} p_{(x,y)}(y)$.

Learning Deep Structured Models

$$\epsilon \ln Z_\epsilon = \max_{p_{(x,y)} \in \Delta} \sum_{r, \hat{y}_r} p_{(x,y),r}(\hat{y}_r) f_r(x, \hat{y}_r; w) + \epsilon H(p_{(x,y)})$$

There are two challenges.
First, the entropy $H(p(x; y))$ can only be computed exactly for a very small set of models.

Second, the marginalization constraints $p_{(x,y),r}(\hat{y}_r) = \sum_{y \setminus y_r} p_{(x,y)}(y)$ are exponential in size.

Learning Deep Structured Models

To deal with both issues a common solution in log-linear models is to **approximate the true marginals** $p_{(x,y);r}$ **with local beliefs** $b_{(x,y);r}$ that are not required to fulfill marginalization constraints globally, but only **locally** (Wainwright & Jordan, 2008).

Learning Deep Structured Models

To deal with both issues a common solution in log-linear models is to **approximate the true marginals** $p_{(x,y);r}$ **with local beliefs** $b_{(x,y);r}$ that are not required to fulfill marginalization constraints globally, but only **locally** (Wainwright & Jordan, 2008).

Two types of constraints on local belief $b_{(x,y),r}$:
First, $\forall r, b_{(x,y),r} \in \Delta$.
Second, $\forall r, \hat{y}_r, p \in P(r)$,

$$\sum_{\hat{y}_p / \hat{y}_r} b_{(x,y),p}(\hat{y}_p) = b_{(x,y),r}(\hat{y}_r) \tag{2}$$

where $P(r)$ the set of parents of region r, i.e., $P(r) \subseteq \{p \in R : r \subset p\}$.

Learning Deep Structured Models

$$\epsilon \ln Z_\epsilon = \max_{p_{(x,y)} \in \Delta} \sum_{r, \hat{y}_r} p_{(x,y),r}(\hat{y}_r) f_r(x, \hat{y}_r; w) + \epsilon H(p_{(x,y)})$$

- approximate $p_{(x,y);r}$ with $b_{(x,y);r}$

Learning Deep Structured Models

$$\epsilon \ln Z_\epsilon = \max_{p_{(x,y)} \in \Delta} \sum_{r, \hat{y}_r} p_{(x,y),r}(\hat{y}_r) f_r(x, \hat{y}_r; w) + \epsilon H(p_{(x,y)})$$

- approximate $p_{(x,y);r}$ with $b_{(x,y);r}$
- approximate entropy by fractional entropy (Wiegerinck & Heskes, 2003)

$$H(p_{(x,y)}) \approx \sum_r c_r H(b_{(x,y),r})$$

Learning Deep Structured Models

$$\epsilon \ln Z_\epsilon = \max_{p_{(x,y)} \in \Delta} \sum_{r, \hat{y}_r} p_{(x,y),r}(\hat{y}_r) f_r(x, \hat{y}_r; w) + \epsilon H(p_{(x,y)})$$

- approximate $p_{(x,y);r}$ with $b_{(x,y);r}$
- approximate entropy by fractional entropy (Wiegerinck & Heskes, 2003)

$$H(p_{(x,y)}) \approx \sum_r c_r H(b_{(x,y),r})$$

Approximation for $\epsilon \ln Z_\epsilon(x, w)$:

$$\max_{b_(x,y) \in C_(x,y)} \sum_{r, \hat{y}_r} b_{(x,y),r}(\hat{y}_r) f_r(x, \hat{y}_r; w) + \sum_r \epsilon c_r H(b_{(x,y),r})$$

Learning Deep Structured Models

Approximation for $\epsilon \ln Z_\epsilon(x, w)$:

$$\max_{b_(x,y) \in C_(x,y)} \sum_{r, \hat{y}_r} b_{(x,y),r}(\hat{y}_r) f_r(x, \hat{y}_r; w) + \sum_r \epsilon c_r H(b_{(x,y),r})$$

Learning Deep Structured Models

Approximation for $\epsilon \ln Z_\epsilon(x, w)$:

$$\max_{b_(x,y) \in C_(x,y)} \sum_{r, \hat{y}_r} b_{(x,y),r}(\hat{y}_r) f_r(x, \hat{y}_r; w) + \sum_r \epsilon c_r H(b_{(x,y),r})$$

Maximizing data likelihood

$$\min_w \sum_{(x,y) \in D} (\epsilon \ln Z_\epsilon(x, w) - F(x, y; w))$$

Plug in approximation for $\epsilon \ln Z_\epsilon(x, w)$,

$$\min_w \sum_{(x,y) \in \mathcal{D}} \left(\max_{b_{(x,y)} \in \mathcal{C}_{(x,y)}} \left\{ \sum_{r, \hat{y}_r} b_{(x,y),r}(\hat{y}_r) f_r(x, \hat{y}_r; w) + \sum_r \epsilon c_r H(b_{(x,y),r}) \right\} - F(x, y; w) \right)$$

Learning Deep Structured Models

$$\min_{w} \sum_{(x,y)\in\mathcal{D}} \left(\max_{b_{(x,y)}\in\mathcal{C}_{(x,y)}} \left\{ \sum_{r,\hat{y}_r} b_{(x,y),r}(\hat{y}_r) f_r(x,\hat{y}_r;w) + \sum_{r} \epsilon c_r H(b_{(x,y),r}) \right\} - F(x,y;w) \right)$$

Challenge: double-loop algorithm which would be slow

Goal: blend learning (i.e., parameter updates) and inference
derive an algorithm that is able to interleave minimization w.r.t. w and maximization of the beliefs b

Solution: convert maximization of the beliefs into a minimization by **employing the dual program**

Learning Deep Structured Models

Claim 1 *Assume* $\epsilon c_r \geq 0 \ \forall r$, *and let* $\overline{F}(w) = \sum_{(x,y)\in\mathcal{D}} F(x, y; w)$ *denote the sum of empirical function observations. Let* $\lambda_{(x,y),r\to p}(\hat{y}_r)$ *be the Lagrange multipliers for each marginalization constraint* $\sum_{\hat{y}_p \setminus \hat{y}_r} b_{(x,y),p}(\hat{y}_p) = b_{(x,y),r}(\hat{y}_r)$ *within the polytope* $\mathcal{C}_{(x,y)}$. *Then the approximated general structured prediction task shown in Fig. 2 is equivalent to*

$$\min_{w,\lambda} \sum_{(x,y),r} \epsilon c_r \ln \sum_{\hat{y}_r} \exp \frac{\hat{f}_r(x, \hat{y}_r; w, \lambda)}{\epsilon c_r} - \overline{F}(w), \quad (5)$$

where we employed the re-parameterization score $\hat{f}_r(x, \hat{y}_r; w, \lambda) = f_r(x, \hat{y}_r; w) + \sum_{c\in C(r)} \lambda_{(x,y),c\to r}(\hat{y}_c) - \sum_{p\in P(r)} \lambda_{(x,y),r\to p}(\hat{y}_r)$.

Learning Deep Structured Models

$$\min_{w} \sum_{(x,y)\in\mathcal{D}} \left(\max_{b_{(x,y)}\in\mathcal{C}_{(x,y)}} \left\{ \sum_{r,\hat{y}_r} b_{(x,y),r}(\hat{y}_r) f_r(x,\hat{y}_r;w) + \sum_r \epsilon c_r H(b_{(x,y),r}) \right\} - F(x,y;w) \right)$$

is equivalent to

$$\min_{w,\lambda} \sum_{(x,y),r} \epsilon c_r \ln \sum_{\hat{y}_r} \exp \frac{\hat{f}_r(x,\hat{y}_r;w,\lambda)}{\epsilon c_r} - \overline{F}(w)$$

where

$$\hat{f}_r(x,\hat{y}_r;w,\lambda) = f_r(x,\hat{y}_r;w) + \sum_{c\in C(r)} \lambda_{(x,y),c\to r}(\hat{y}_c) - \sum_{p\in P(r)} \lambda_{(x,y),r\to p}(\hat{y}_r)$$

and

$$\bar{F}(w) = \sum_{(x,y)\in D} F(x,y;w)$$

Learning Deep Structured Models

iterate through all regions r and use block-coordinate descent to find the globally optimal value of

$$\min_{w,\lambda} \sum_{(x,y),r} \epsilon c_r \ln \sum_{\hat{y}_r} \exp \frac{\hat{f}_r(x, \hat{y}_r; w, \lambda)}{\epsilon c_r} - \overline{F}(w)$$

w.r.t. $\lambda_{(x,y),r \to p}(\hat{y}_c)$, which can be done in closed form and therefore is computed very efficiently (Globerson & Jaakkola, 2007; Sontag et al., 2008; Hazan & Shashua, 2010; Schwing, 2013).

Learning Deep Structured Models

Algorithm: Efficient Deep Structured Learning

Repeat until stopping criteria

1. Forward pass to compute $f_r(x, \hat{y}_r; w) \forall (x, y), r, y_r$
2. Compute approximate belief $b_{(x,y);w}$ by iterating for a fixed number of times over r
3. Backward pass via chain-rule to obtain gradient g w.r.t w
4. Update parameter w using stepsize η via $w \leftarrow \eta g$

Learning Deep Structured Models

Technique

- Decompose scoring function $F(x, y; w)$ into local functions $f_r(x, y_r; w)$.
- Approximate the true marginals $p_{(x;y);r}$ with local beliefs $b_{(x;y);r}$.
- Interleave minimization w.r.t. w and maximization of the beliefs b by converting maximization of the beliefs into a minimization

Experiments

Two tasks:

- Word recognition: Word50

banal

julep

resty

- Image tagging: Flicker

female/indoor/portrait
female/indoor/portrait

sky/plant life/tree
sky/plant life/tree

water/animals/sea
water/animals/sky

animals/dog/indoor
animals/dog

indoor/flower/plant life
∅

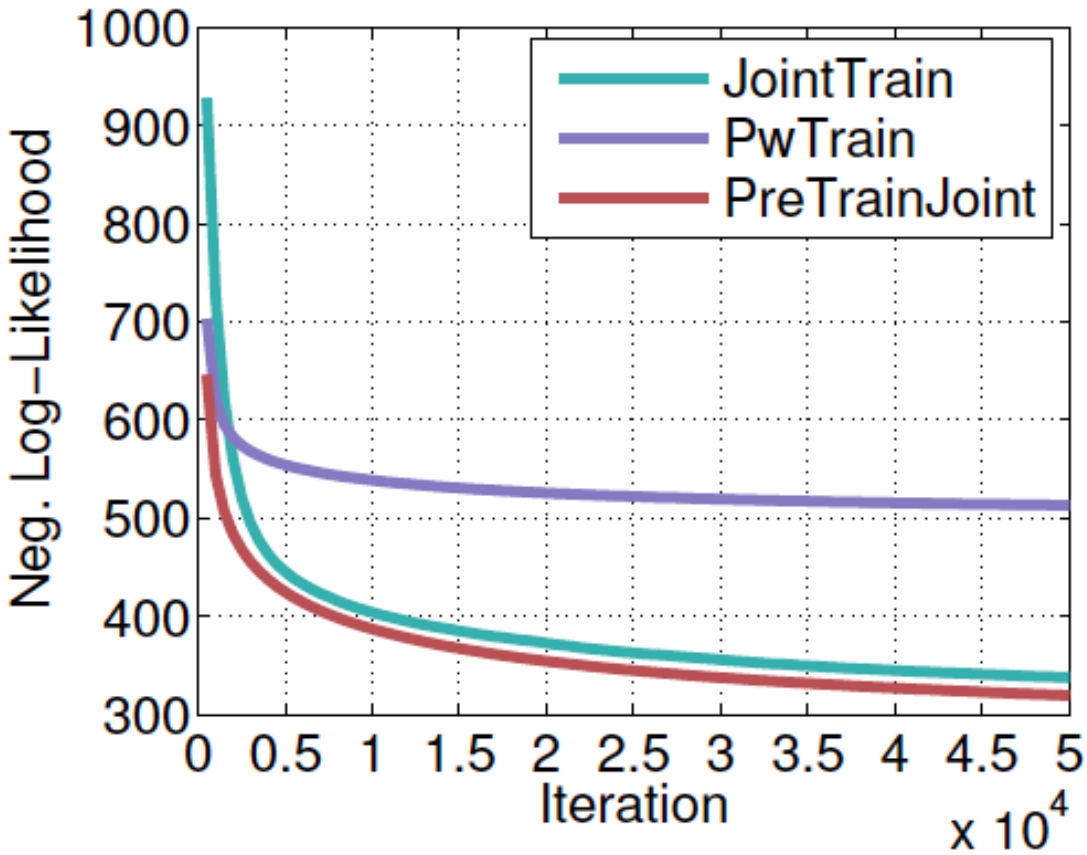
Experiments

Four strategies to learn the model parameters:

- **Unary Only**: only consider unary classifiers while ignoring the structure of the graphical model
- **JointTrain**: randomly initialize all weights and train them jointly
- **PwTrain**: first train the unary potentials, then keep them fixed when training the pairwise potentials
- **PreTrainJoint**: pre-train the unaries but jointly optimize unary and pairwise weights together in a second step

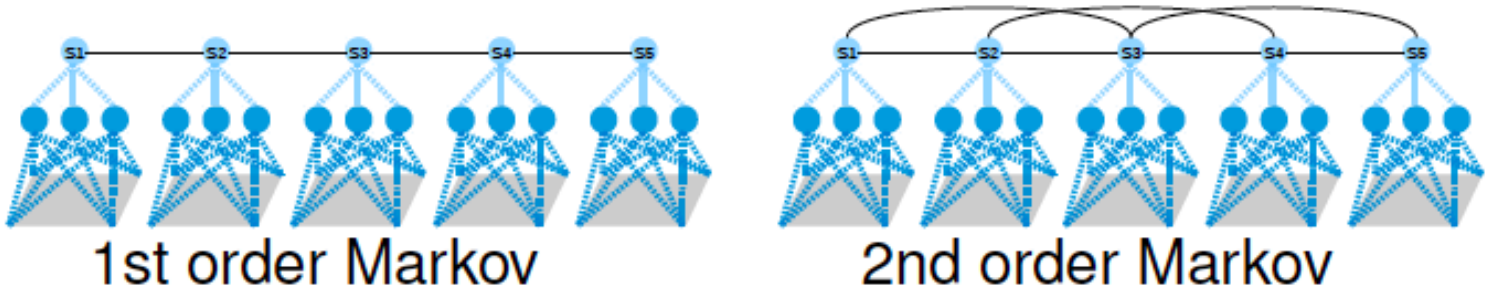
Word Recognition

- PreTrainJoint: lower neg. log-likehood, better performance



Word Recognition

- Dataset Word50: 10000 training samples, 2000 test samples
- Graphical models composed of five random variables: 1st and 2nd order of Markov Random Field



- Encode unary potentials $f_r(x, y_i; w_u)$ using multi-layer perceptrons with rectified linear units
- Encode pairwise potentials $f_r(x, y_i, y_j; w_p)$ using linear functions or deep structured hidden units

Word Recognition

- y axis: percentage of accuracy;
- x axis: hidden units in the MLP model
- Linear: 1st oder MRF; PairH32: hidden units in 2nd order MRF

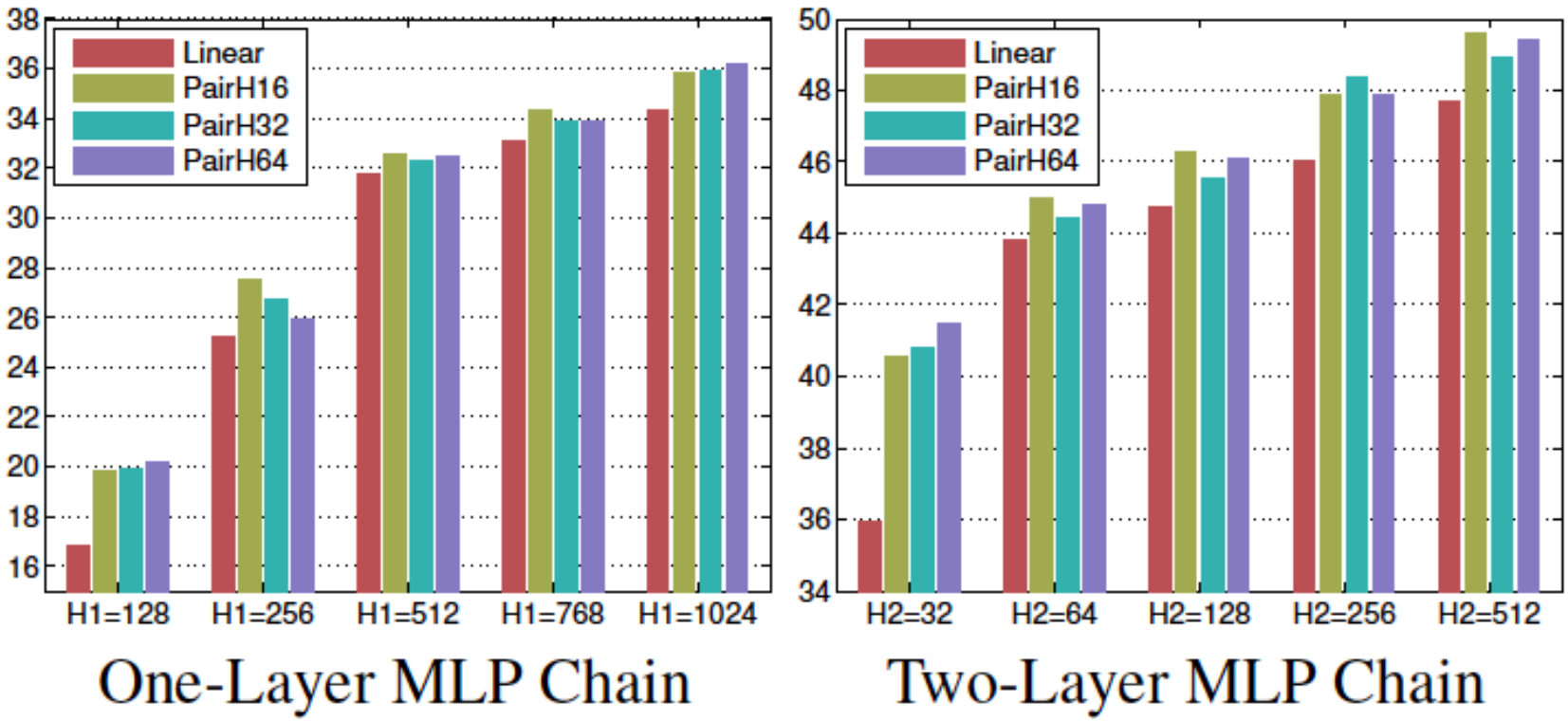


Image tagging

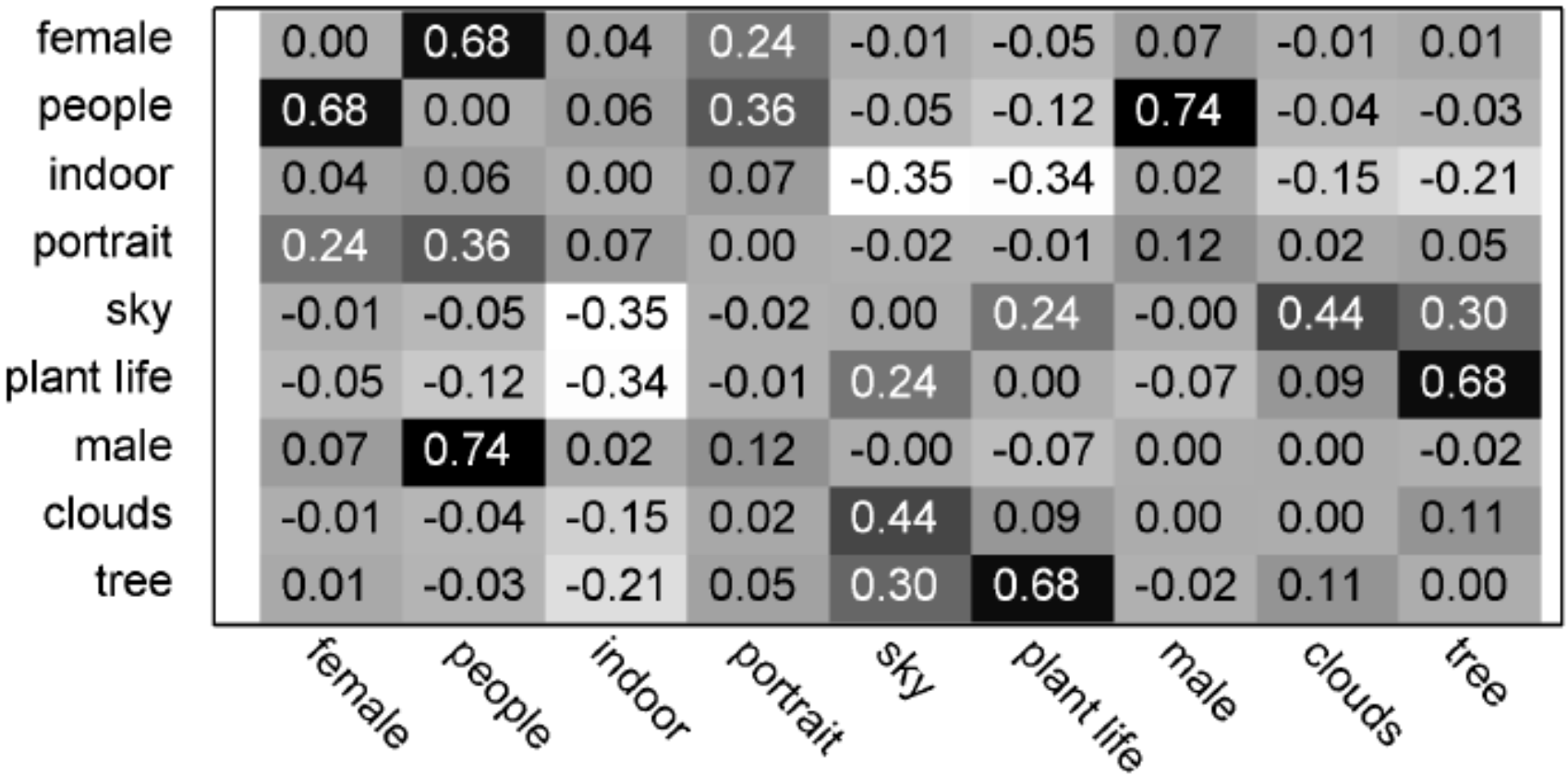
Given an image, assign a combination of tags that describe the content

- Flickr dataset: 10000 training samples, 10000 test samples
- Graphical models have 38 binary random variables (tags)
- Encode the non-linear unary potentials $f_r(x, y_i; w_u)$ by using deep neural networks
- Pairwise potentials: linear functions
- PreTrainJoint helps

Method	Mean error
Unary only	9.36
PwTrain	7.70
PreTrainJoint	**7.25**

Image tagging

Different tags have correlations with each other.



Outline

Multi-class image segmentation

Assign a class label to each pixel in the image.

CRF models for multi-class image segmentation

- MAP inference in conditional random field
- Characterized by a Gibbs distribution

$$P(\mathbf{x}|\mathbf{I}) = \frac{1}{Z(\mathbf{I})} \exp(-E(\mathbf{x}|\mathbf{I}))$$

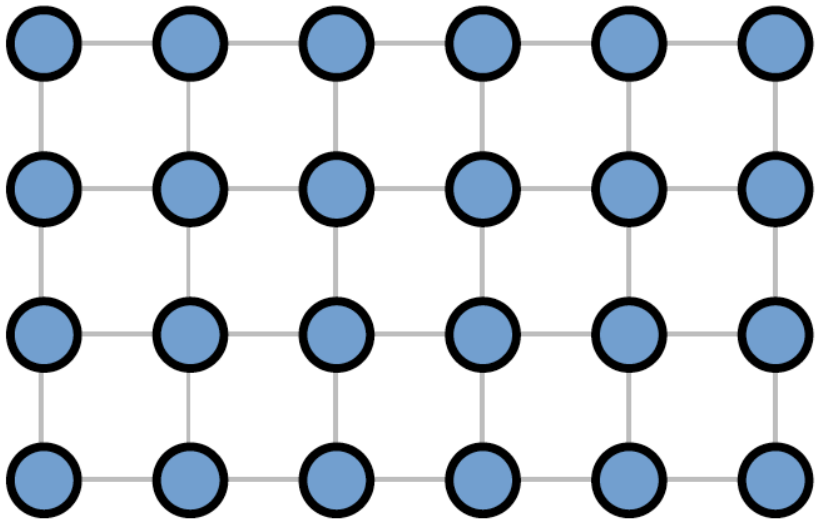
- Gibbs energy $E(\mathbf{x}|\mathbf{I})$ (Denoted $E(\mathbf{x})$)

$$E(\mathbf{x}) = \sum_i \underbrace{\psi_u(x_i)}_{\text{Unary}} + \sum_i \sum_{j \in N_i} \underbrace{\psi_p(x_i, x_j)}_{\text{Pairwise}}$$

Adjacency CRF model

$$E(\mathbf{x}) = \sum_i \underbrace{\psi_u(x_i)}_{\text{Unary}} + \sum_i \sum_{j \in N_i} \underbrace{\psi_p(x_i, x_j)}_{\text{Pairwise}}$$

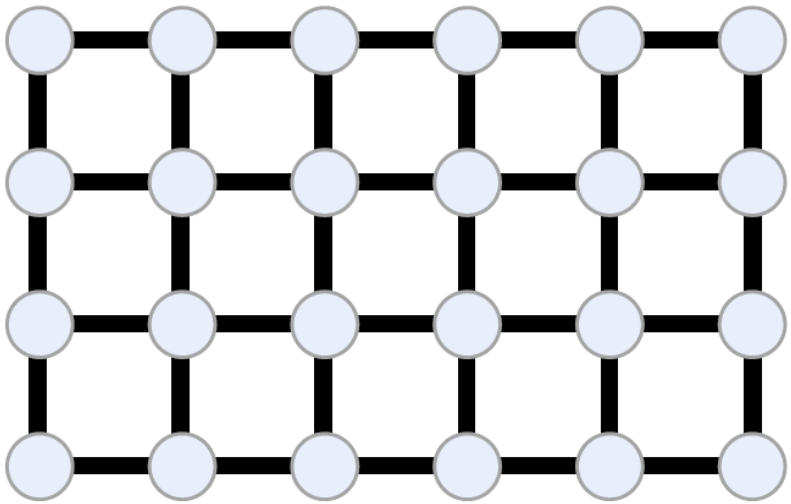
- Unary Term
  - From classifier
  - TextonBoost [Shotton et al. 09]
- Pairwise Term: Consistent Labelling



Adjacency CRF model

$$E(\mathbf{x}) = \sum_i \underbrace{\psi_u(x_i)}_{\text{Unary}} + \sum_i \sum_{j \in N_i} \underbrace{\psi_p(x_i, x_j)}_{\text{Pairwise}}$$

- Pairwise Term
  - Neighboring pixels
  - Color-sensitive Potts model

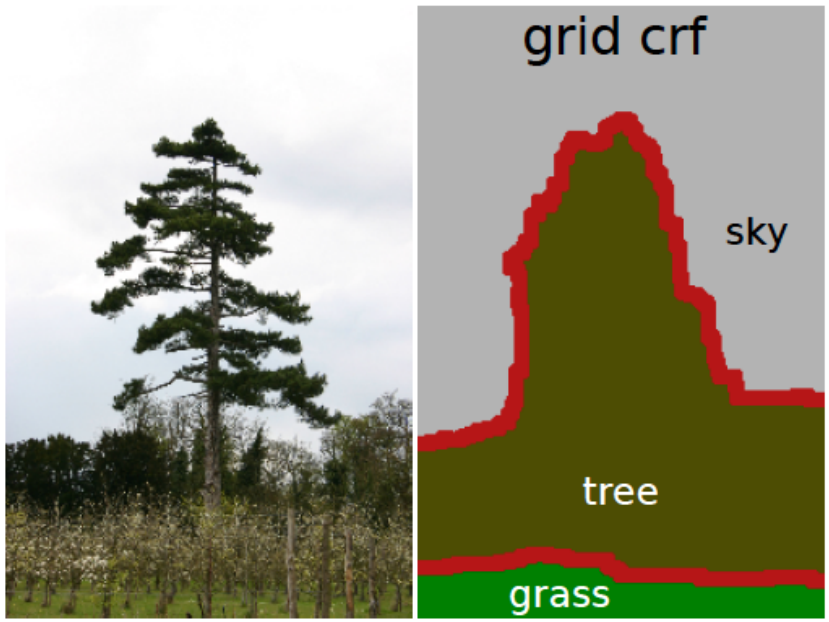


$$\psi_p(x_i, x_j) = \mathbb{1}_{x_i \neq x_j} \left(w^{(1)} exp\left(-\frac{|\mathbf{I}_i - \mathbf{I}_j|^2}{2\theta_\beta^2} \right) + w^{(2)} \right)$$

Adjacency CRF model

$$E(\mathbf{x}) = \sum_i \underbrace{\psi_u(x_i)}_{\text{Unary}} + \sum_i \sum_{j \in N_i} \underbrace{\psi_p(x_i, x_j)}_{\text{Pairwise}}$$

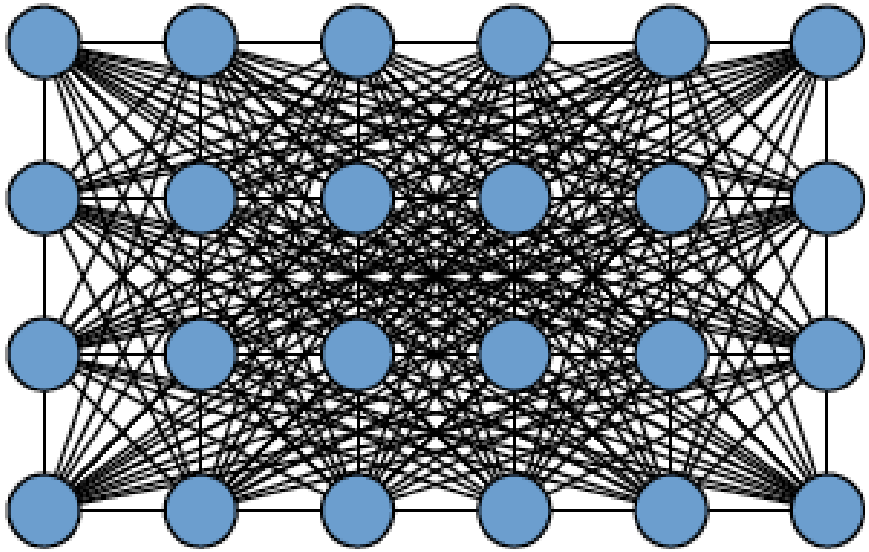
- Efficient inference
  - 1 sec for 50,000 variables
- Limited expressive power
- Only local interactions
- Excessive smoothing of object boundaries
  - Shrinking bias



Fully connected CRF

$$E(\mathbf{x}) = \sum_i \underbrace{\psi_u(x_i)}_{\text{Unary}} + \sum_i \sum_{j>i} \underbrace{\psi_p(x_i, x_j)}_{\text{Pairwise}}$$

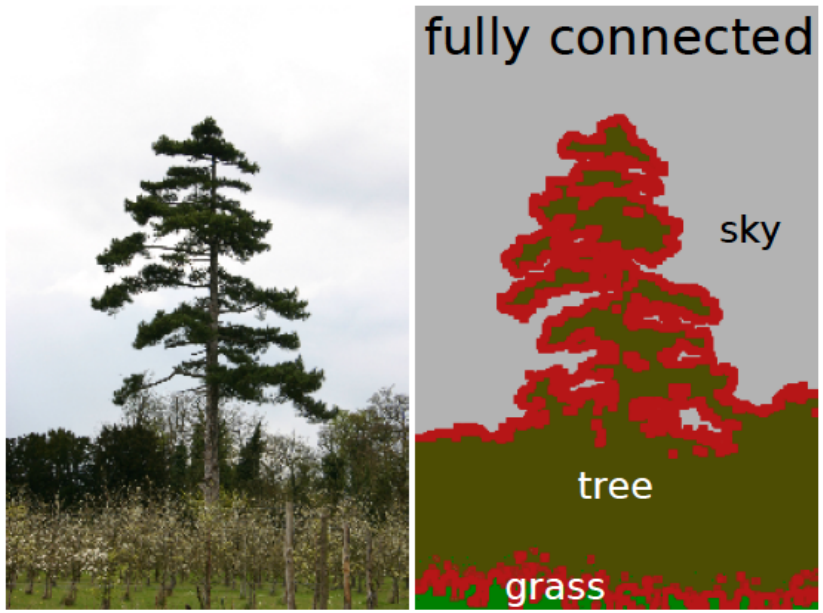
- Every node is connected to every other node
  - Connections weighted differently



Fully connected CRF

$$E(\mathbf{x}) = \sum_i \underbrace{\psi_u(x_i)}_{\text{Unary}} + \sum_i \sum_{j>i} \underbrace{\psi_p(x_i, x_j)}_{\text{Pairwise}}$$

- Long-range interactions
- No more shrinking bias



Fully connected CRF

$$E(\mathbf{x}) = \sum_i \underbrace{\psi_u(x_i)}_{\text{Unary}} + \sum_i \sum_{j>i} \underbrace{\psi_p(x_i, x_j)}_{\text{Pairwise}}$$

- Region-based [Rabinovich et al. 07, Galleguillos et al. 08, Toyoda & Hasegawa 08, Payet & Todorovic 10]
  - Tractable up to hundreds of variables
- Pixel-based
  - Tens of thousands of variables
    - Billions of edges
  - Computationally expensive

Efficient Inference in Fully connected CRFs with Gaussian Edge Potentials

- Inference in 0.2 sec
  - 50,000 variables
  - MCMC inference: 36 hrs
- Pairwise potentials: linear combinations of Gaussian

Model definition

$$E(\mathbf{x}) = \sum_i \underbrace{\psi_u(x_i)}_{\text{Unary}} + \sum_i \sum_{j>i} \underbrace{\psi_p(x_i, x_j)}_{\text{Pairwise}}$$

Gaussian edge potentials

$$\psi_p(x_i, x_j) = \mu(x_i, x_j) \sum_{m=1}^{K} w^{(m)} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j)$$

- Label compatibility function μ
- Linear combinations of Gaussian kernels

$$k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) = exp\left(-\frac{1}{2}(\mathbf{f}_i - \mathbf{f}_j)\sum^{(m)}(\mathbf{f}_i - \mathbf{f}_j)\right)$$

- Arbitrary feature space $\mathbf{f}_i$

Detailed model definition

- Label compatibility
  - Potts model: $\mu(x_i, x_j) = \mathbb{1}_{x_i \neq x_j}$
  - Semi-metric model: $\mu(x_i, x_j)$ learned from data
- Appearance kernel
  - Color-sensitive model
- Local smoothness
  - Discourages pixel level noise

$$\psi_p(x_i, x_j) = \mu(x_i, x_j)[w^{(1)} \underbrace{exp(-\frac{|\mathbf{p}_i - \mathbf{p}_j|^2}{2\theta_\alpha^2} - \frac{|\mathbf{I}_i - \mathbf{I}_j|^2}{2\theta_\beta^2})}_{\text{appearance kernel}} +$$

$$w^{(2)} \underbrace{exp(-\frac{|\mathbf{p}_i - \mathbf{p}_j|^2}{2\theta_\gamma^2})}_{\text{smoothness kernel}}]$$

Inference

Find the most likely assignment (MAP)

$$\hat{\mathbf{x}} = \underset{\mathbf{x}}{\text{argmax}}\, P(\mathbf{x}) \text{ where } P(\mathbf{x}) = exp(-E(\mathbf{x}))$$

Mean field approximation

- Find $Q(\mathbf{x}) = \prod_i Q_i(x_i)$ close to $P(\mathbf{x})$ in terms of KL-divergence $D(Q||P)$
- $\hat{\mathbf{x}} \approx \text{argmax}_{x_i}\, Q_i(x_i)$

Mean field approximation

$$
\begin{aligned}
D(Q||P) &= \sum_{\mathbf{x}} Q(\mathbf{x}) \log \frac{Q(\mathbf{x})}{P(\mathbf{x})} \\
&= -\sum_{\mathbf{x}} Q(\mathbf{x}) \log P(\mathbf{x}) + \sum_{\mathbf{x}} Q(\mathbf{x}) \log Q(\mathbf{x}) \\
&= -\mathbb{E}_{U\sim Q}[\log P(U)] + \mathbb{E}_{U\sim Q}[\log Q(U)] \\
&= -\mathbb{E}_{U\sim Q}[\log \widetilde{P}(U)] + \mathbb{E}_{U\sim Q}[\log Z] + \sum_i \mathbb{E}_{U_i\sim Q}[\log Q_i(U_i)] \\
&= -\mathbb{E}_{U\sim Q}[E(U)] + + \sum_i \mathbb{E}_{U_i\sim Q}[\log Q_i(U_i)] + \log Z
\end{aligned}
$$

The marginal $Q_i(x_i)$ that minimizes the KL-divergence is found by analytically minimizing a Lagrangian that consists $D(Q||P)$ plus Lagrange multipliers assuring the marginals $Q_i(x_i)$ are probability distributions

Mean field approximation

$$Q_i(x_i = l) = \frac{1}{Z_i} \exp\{-\psi_u(x_i) - \sum_{l' \in \mathcal{L}} \mu(l, l') \sum_{m=1}^{K} w^{(m)} \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l')\}$$

- Initialize $Q_i(x_i) \leftarrow \frac{1}{Z_i} \exp\{-\psi_u(x_i)\}$
- While not converged
  - Message passing: $\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$
  - Compatibility transform: $\hat{Q}_i(x_i) \leftarrow \sum_{l \in L} \mu^{(m)}(x_i, l) \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$
  - Local update: $Q_i(x_i) = \exp\{-\psi_u(x_i) - \hat{Q}_i(x_i)\}$
  - Normalize: $Q_i(x_i)$

Mean field approximation

$$Q_i(x_i = l) = \frac{1}{Z_i} \exp\{-\psi_u(x_i) - \sum_{l' \in \mathcal{L}} \mu(l, l') \sum_{m=1}^{K} w^{(m)} \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l')\}$$

- Initialize $Q_i(x_i) \leftarrow \frac{1}{Z_i} \exp\{-\psi_u(x_i)\}$
- While not converged
  - Message passing: $\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$
  - Compatibility transform: $\hat{Q}_i(x_i) \leftarrow \sum_{l \in L} \mu^{(m)}(x_i, l) \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$
  - Local update: $Q_i(x_i) = \exp\{-\psi_u(x_i) - \hat{Q}_i(x_i)\}$
  - Normalize: $Q_i(x_i)$

Mean field approximation

$$Q_i(x_i = l) = \frac{1}{Z_i} \exp\{-\psi_u(x_i) - \sum_{l' \in \mathcal{L}} \mu(l, l') \sum_{m=1}^{K} w^{(m)} \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l')\}$$

- Initialize $Q_i(x_i) \leftarrow \frac{1}{Z_i} \exp\{-\psi_u(x_i)\}$
- While not converged
  - Message passing: $\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$
  - Compatibility transform: $\hat{Q}_i(x_i) \leftarrow \sum_{l \in L} \mu^{(m)}(x_i, l) \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$
  - Local update: $Q_i(x_i) = \exp\{-\psi_u(x_i) - \hat{Q}_i(x_i)\}$
  - Normalize: $Q_i(x_i)$

Mean field approximation

$$Q_i(x_i = l) = \frac{1}{Z_i} \exp\{-\psi_u(x_i) - \sum_{l' \in \mathcal{L}} \mu(l, l') \sum_{m=1}^{K} w^{(m)} \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l')\}$$

- Initialize $Q_i(x_i) \leftarrow \frac{1}{Z_i} \exp\{-\psi_u(x_i)\}$
- While not converged
  - Message passing: $\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$
  - Compatibility transform: $\hat{Q}_i(x_i) \leftarrow \sum_{l \in L} \mu^{(m)}(x_i, l) \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$
  - Local update: $Q_i(x_i) = \exp\{-\psi_u(x_i) - \hat{Q}_i(x_i)\}$
  - Normalize: $Q_i(x_i)$

Mean field approximation

$$Q_i(x_i = l) = \frac{1}{Z_i} \exp\{-\psi_u(x_i) - \sum_{l' \in \mathcal{L}} \mu(l, l') \sum_{m=1}^{K} w^{(m)} \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l')\}$$

Runtime analysis for N variables

- Initialize $Q_i(x_i) \leftarrow \frac{1}{Z_i} \exp\{-\psi_u(x_i)\}$ $O(N)$
- While not converged ~ 10
  - Message passing: $\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$ $\sim O(N^2)$
  - Compatibility transform: $\hat{Q}_i(x_i) \leftarrow \sum_{l \in L} \mu^{(m)}(x_i, l) \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$ $\sim O(N)$
  - Local update: $Q_i(x_i) = \exp\{-\psi_u(x_i) - \hat{Q}_i(x_i)\}$ $\sim O(N)$
  - Normalize: $Q_i(x_i)$ $\sim O(N)$

Efficient message passing using high-dimensional filtering

- Update all $\tilde{Q}_i^{(m)}(l)$ simultaneously

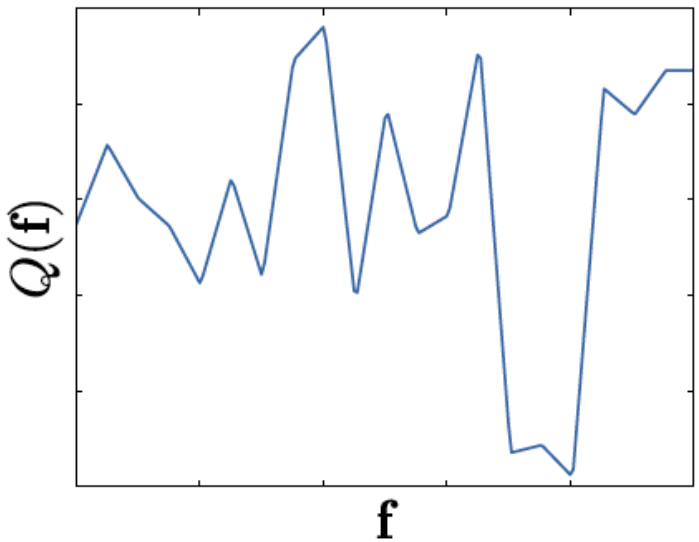
$$\tilde{Q}_i^{(m)}(l) = \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l) = \underbrace{\sum_{j \in \mathcal{V}} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)}_{\overline{Q}_i^{(m)}(l)} - Q_i(l)$$

- Efficiently computed using a cross-bilateral filter [Paris & Durand 09, Adams et al. 09, Adams et al. 10]

High-dimensional filtering [Paris & Durand 09]

$$\overline{Q}_i^{(m)}(I) = \sum_{j \in \mathcal{V}} \exp\left(\frac{1}{2}(\mathbf{f}_i^{(m)} - \mathbf{f}_j^{(m)})^2\right) Q_j(I)$$

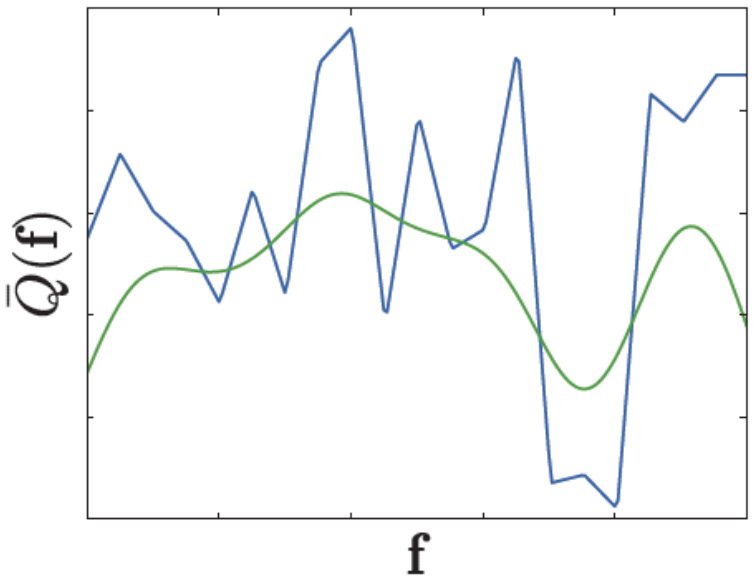
- High-dimensional input signal $Q_j(I)$



High-dimensional filtering [Paris & Durand 09]

$$\overline{Q}_i^{(m)}(I) = \sum_{j \in \mathcal{V}} \exp\left(\frac{1}{2}(\mathbf{f}_i^{(m)} - \mathbf{f}_j^{(m)})^2\right) Q_j(I)$$

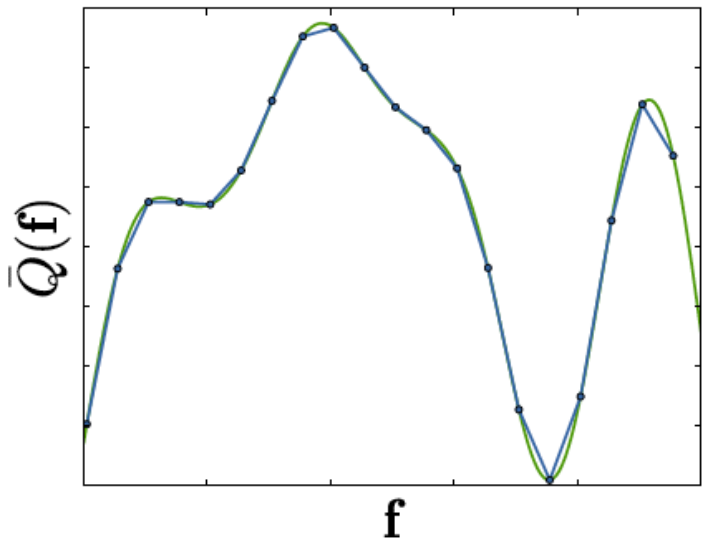
- High-dimensional input signal $Q_j(I)$
- Gaussian convolution $\overline{Q}_i^{(m)}(I) = \mathcal{G} \otimes Q_j(I)$
  - Band-limited, smooth function



High-dimensional filtering [Paris & Durand 09]

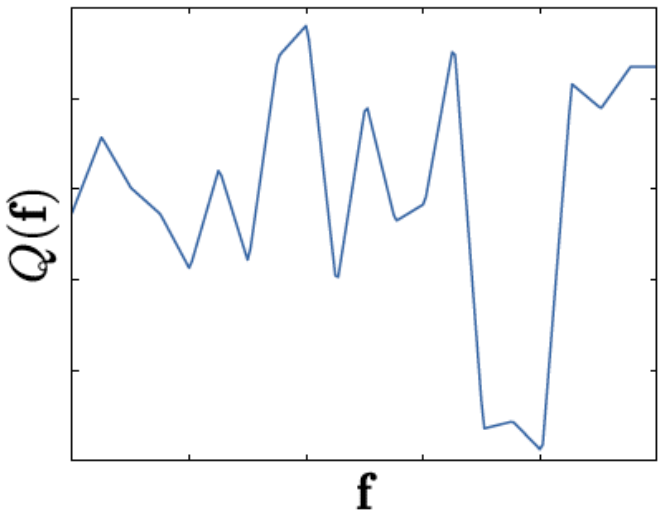
$$\overline{Q}_i^{(m)}(I) = \sum_{j \in \mathcal{V}} \exp\left(\frac{1}{2}(\mathbf{f}_i^{(m)} - \mathbf{f}_j^{(m)})^2\right) Q_j(I)$$

- High-dimensional input signal $Q_j(I)$
- Gaussian convolution $\overline{Q}_i^{(m)}(I) = \mathcal{G} \otimes Q_j(I)$
  - Band-limited, smooth function
- Can be contructed from a number of samples
  - Nyquist Theorem



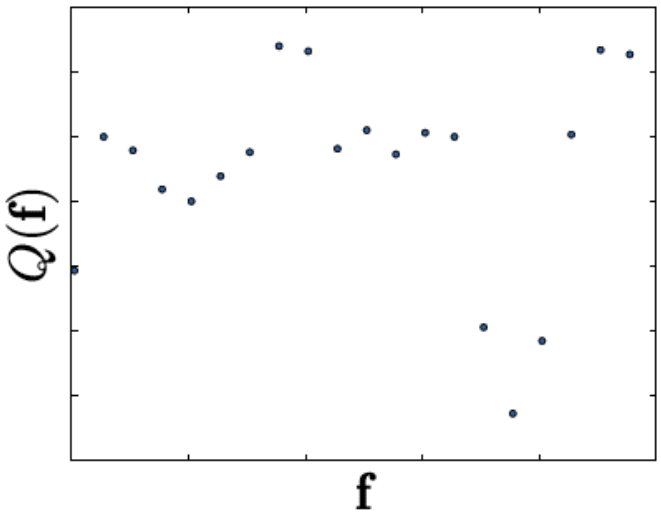
High-dimensional filtering [Paris & Durand 09]

- Given a high-dimensional input signal



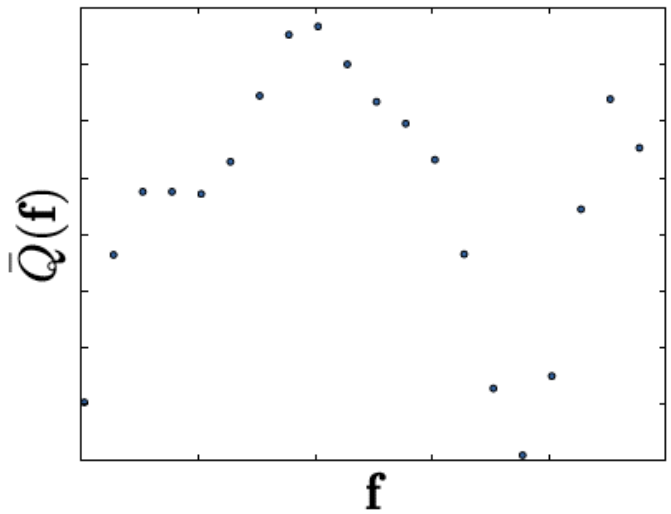
High-dimensional filtering [Paris & Durand 09]

- Given a high-dimensional input signal
- Downsample input signal $Q_j(I)$



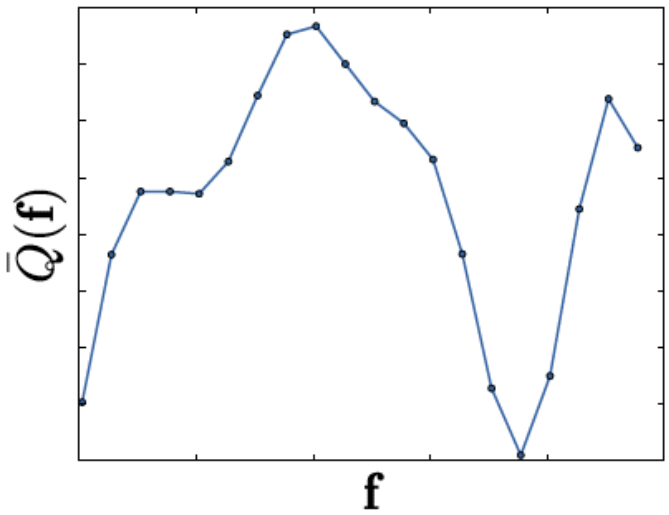
High-dimensional filtering [Paris & Durand 09]

- Given a high-dimensional input signal
- Downsample input signal $Q_j(I)$
- Blur the sampled signal



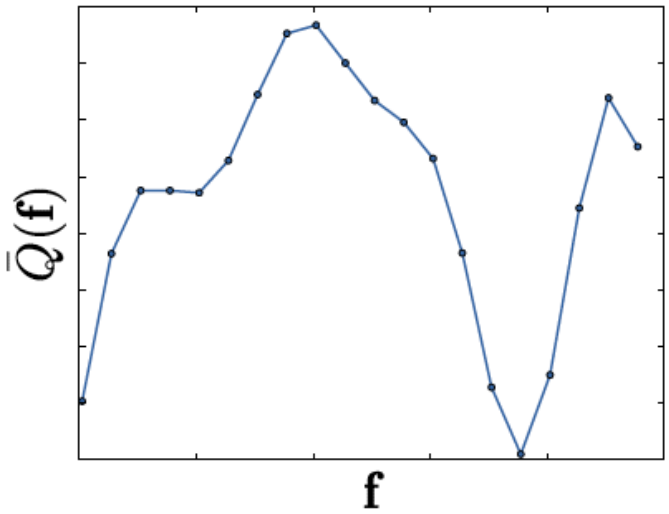
High-dimensional filtering [Paris & Durand 09]

- Given a high-dimensional input signal
- Downsample input signal $Q_j(I)$
- Blur the sampled signal
- Upsample to reconstruct the filtered signal $\overline{Q}_j(I)$



High-dimensional filtering [Paris & Durand 09]

- Given a high-dimensional input signal
- Downsample input signal $Q_j(I)$
- Blur the sampled signal
- Upsample to reconstruct the filtered signal $\overline{Q}_j(I)$



However, high-dimensional filtering algorithm still has computational complexity exponential in the dimension d.

High-dimensional filtering [Adams et al. 10]

Permutohedral lattice

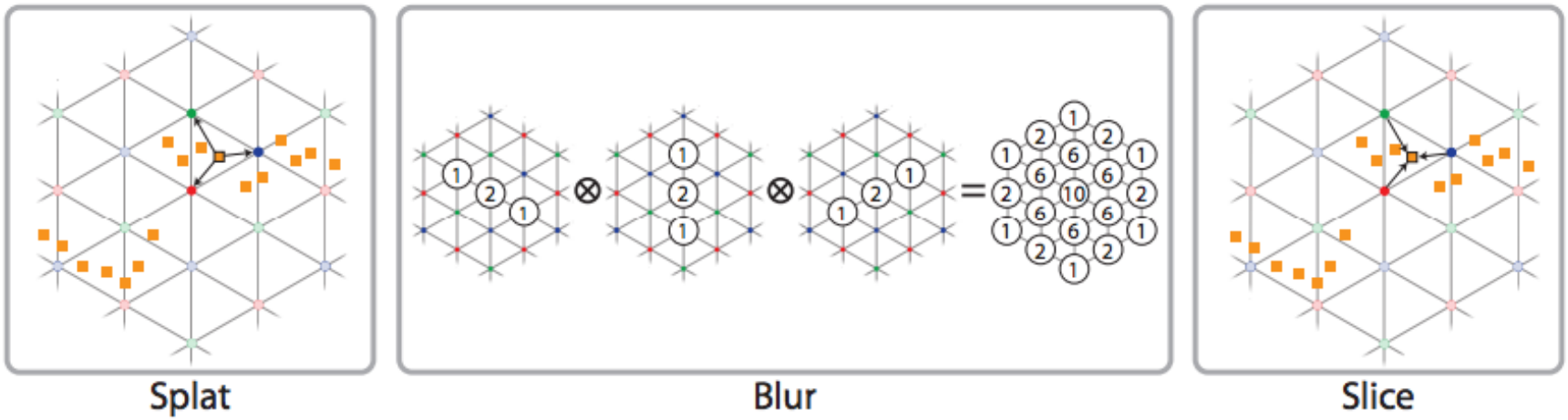


Figure 4: *To perform a high-dimensional Gaussian filter using the permutohedral lattice, first the position vectors* $\vec{p}_i \in \mathbb{R}^d$ *are embedded in the hyperplane* H_d *using an orthogonal basis for* H_d *(not pictured). Then, each input value* ***splats*** *onto the vertices of its enclosing simplex using barycentric weights. Next, lattice points* ***blur*** *their values with nearby lattice points using a separable filter. Finally, the space is* ***sliced*** *at each input position using the same barycentric weights to interpolate output values.*

This filtering scheme can reduce the complexity of the convolution operation to $O(Nd)$

Mean field approximation

$$Q_i(x_i = l) = \frac{1}{Z_i} \exp\{-\psi_u(x_i) - \sum_{l' \in \mathcal{L}} \mu(l, l') \sum_{m=1}^{K} w^{(m)} \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l')\}$$

Runtime analysis for N variables

- Initialize $Q_i(x_i) \leftarrow \frac{1}{Z_i} \exp\{-\phi_u(x_i)\}$ $O(N)$
- While not converged ~ 10
  - Message passing: High-dimensional filtering $\sim O(N)$
  - Compatibility transform: $\hat{Q}_i(x_i) \leftarrow \sum_{l \in L} \mu^{(m)}(x_i, l) \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$ $\sim O(N)$
  - Local update: $Q_i(x_i) = \exp\{-\psi_u(x_i) - \hat{Q}_i(x_i)\}$ $\sim O(N)$
  - Normalize: $Q_i(x_i)$ $\sim O(N)$

So far

Takeaway points

- Pixel-based fully connected CRF
- Pairwise Term of weighted Gaussian kernels: appearance + smoothness
- Mean field approximation to the CRF distribution
- Iterative message passing algorithm for approximate inference
- Efficient message passing using high-dimensional filtering: Permutohedral lattice

Learning

$$E(\mathbf{x}) = \sum_i \underbrace{\psi_u(x_i)}_{\text{Unary}} + \sum_i \sum_{j>i} \underbrace{\psi_p(x_i, x_j)}_{\text{Pairwise}}$$

The unary potentials are derived from TextonBoost, where they use a 17-dimensional filter bank and add color, histogram of oriented gradients, and pixel location features.

Learning

$$\psi_p(x_i, x_j) = \mu(x_i, x_j)[w^{(1)} exp(-\frac{|\mathbf{p}_i - \mathbf{p}_j|^2}{2\theta_\alpha^2} - \frac{|\mathbf{I}_i - \mathbf{I}_j|^2}{2\theta_\beta^2}) + w^{(2)} exp(-\frac{|\mathbf{p}_i - \mathbf{p}_j|^2}{2\theta_\gamma^2})]$$

- The appearance kernel parameters $w^{(1)}$, θ_α, and θ_β can be learned with grid search on a holdout validation set.
- The smoothness kernel parameters $w^{(2)}$ and θ_γ do not significantly affect classification accuracy.
- The compatibility parameters $\mu(a, b)$ are learned to maximize the log-likelihood $\ell(\mu : \mathcal{I}, \mathcal{T})$ for a validation set of images $\mathcal{I}$ with ground truth labelings $\mathcal{T}$.

Learning

$$\ell(\mu : \mathcal{I}^{(n)}, \mathcal{T}^{(n)}) = \log P(\mathbf{x} = \mathcal{T}^{(n)} | \mathcal{I}^{(n)}, \mu)$$
$$= -E(\mathcal{T}^{(n)} | \mathcal{I}^{(n)}, \mu) - \log Z(\mathcal{I}^{(n)}, \mu)$$

It can be shown that the partition function Z is convex and hence the the log-liklihood function is concave (Koller & Friedman, 2009). Thus we can use gradient-based optimization techniques.

$$\frac{\partial}{\partial \mu(a,b)} \ell(\mu : \mathcal{I}^{(n)}, \mathcal{T}^{(n)}) \approx -\sum_i \mathcal{T}_i^{(n)}(a) \sum_{j \neq i} k(\mathbf{f}_i, \mathbf{f}_j) \mathcal{T}_j^{(n)}(b) + \sum_i Q_i(a) \sum_{j \neq i} k(\mathbf{f}_i, \mathbf{f}_j) Q_i(b),$$

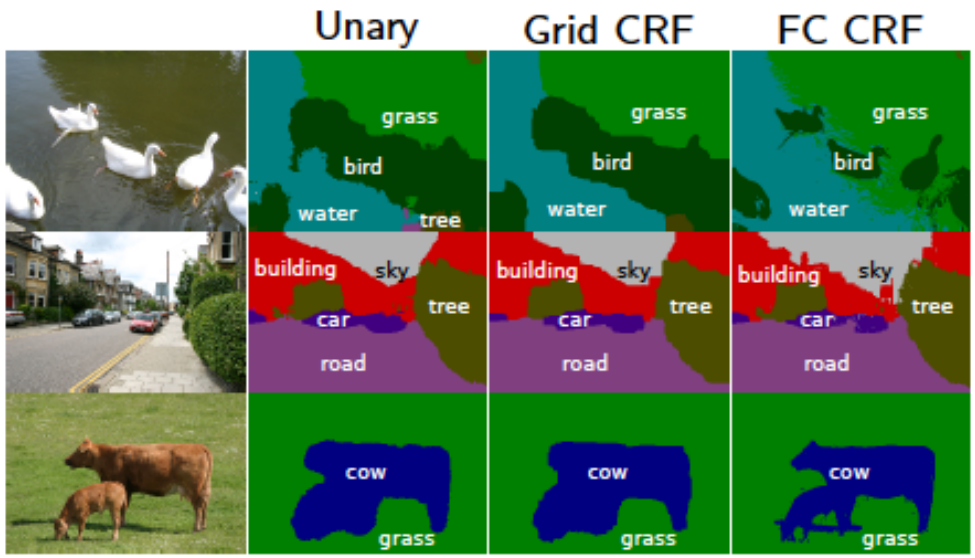
The sums can be computed efficiently using high-dimensional filtering.

Results: MSRC

MSRC dataset

- 591 images
- 21 classes

	Time	Global	Avg
Unary	-	84.0	76.6
Grid CRF	1s	84.6	77.2
FC CRF	**0.2s**	**86.0**	**78.3**



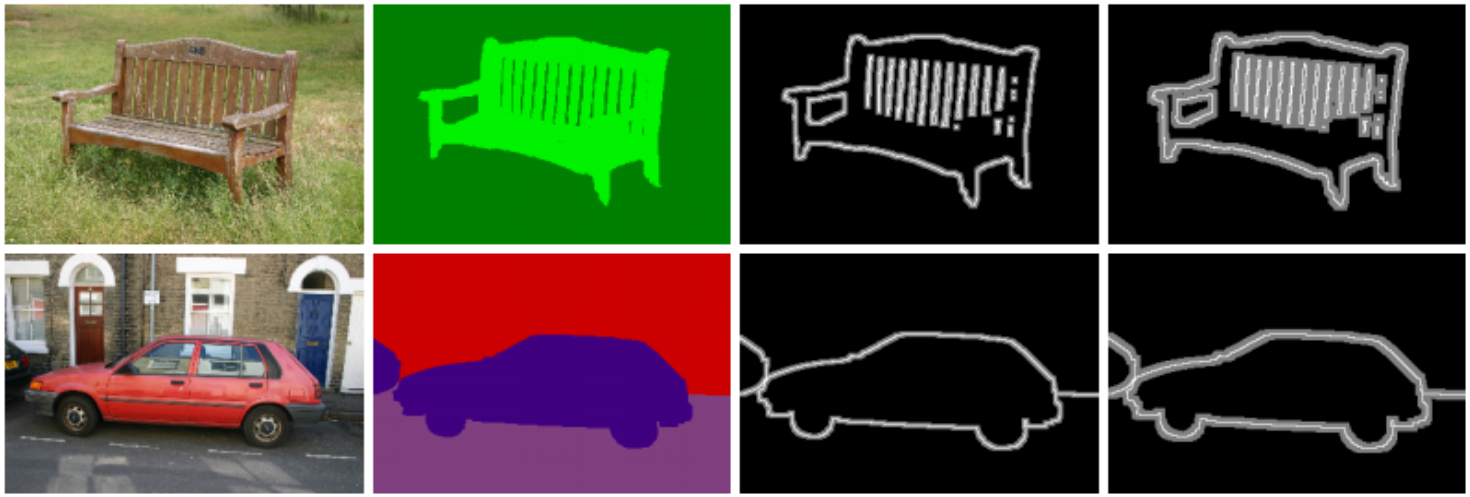
Global: the overall percentage of correctly classified pixels

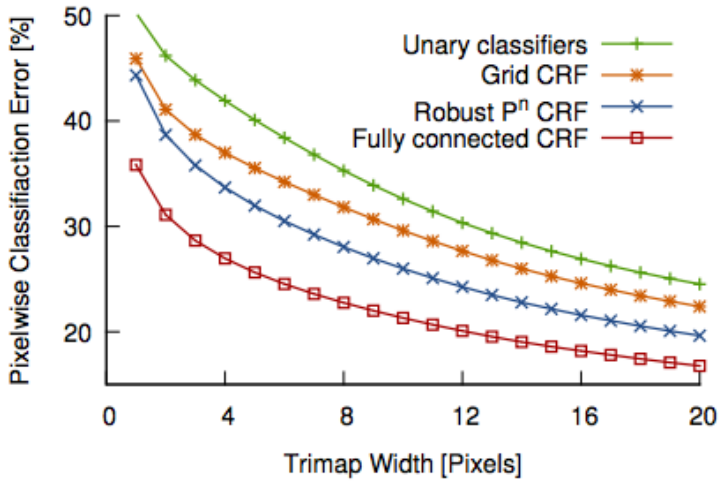
Average: the unweighted average of per-category classification accuracy.

Results: MSRC – Trimap

- 94 images
- hand annotated pixel accurately
- Trimap [Kohli et al. 2009]: Percentage of misclassified pixel around object boundaries

Results: MSRC – Trimap



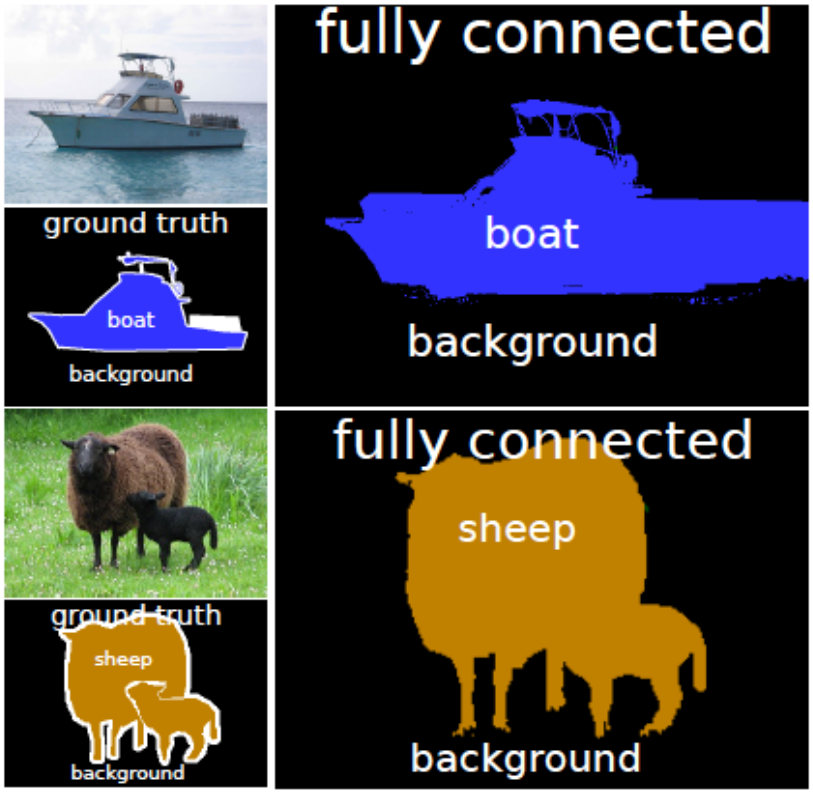


Results: PASCAL VOC 2010

PASCAL VOC 2010 dataset

- 1928 images
- 20 classes + background

	Time	Acc
Unary	-	27.6
Grid CRF	2.5s	28.3
FC Potts	0.5s	29.1
FC label comp	**0.5s**	**30.2**



Deep Convolutional Nets and Fully Connected CRF

Chen et al, "Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs", ICLR 2015

Chen et al, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs", arXiv preprint arXiv:1606.00915 2016

Deep Convolutional Nets and Fully Connected CRF

- DCNNs outperform other systems relying on carefully engineered representation on a broad array of high-level tasks.
- The built-in invariance of DCNNs to local image transformation is hampers low-level tasks.

Deep Convolutional Nets and Fully Connected CRF

Challenges in application of DCNNs to semantic image segmentation:

- Reduced feature resolution: repeated combination of max-pooling and downsampling.
- Multiple-scale objects
- Reduced localization accuracy: DCNN invariance to spatial transformations.

Deep Convolutional Nets and Fully Connected CRF

Approaches to overcome the challenges:

- Reduced feature resolution – Remove the downsampling operator from the last few max-pooling layers and instead upsample the filters with efficient atrous convolution method.
- Multiple-scale objects – Multiple parallel atrous convolution layers with different sampling rates, atrous spatial pyramid pooling (ASPP).
- Reduced localization accuracy – Fully connected CRF!

Deep Convolutional Nets and Fully Connected CRF

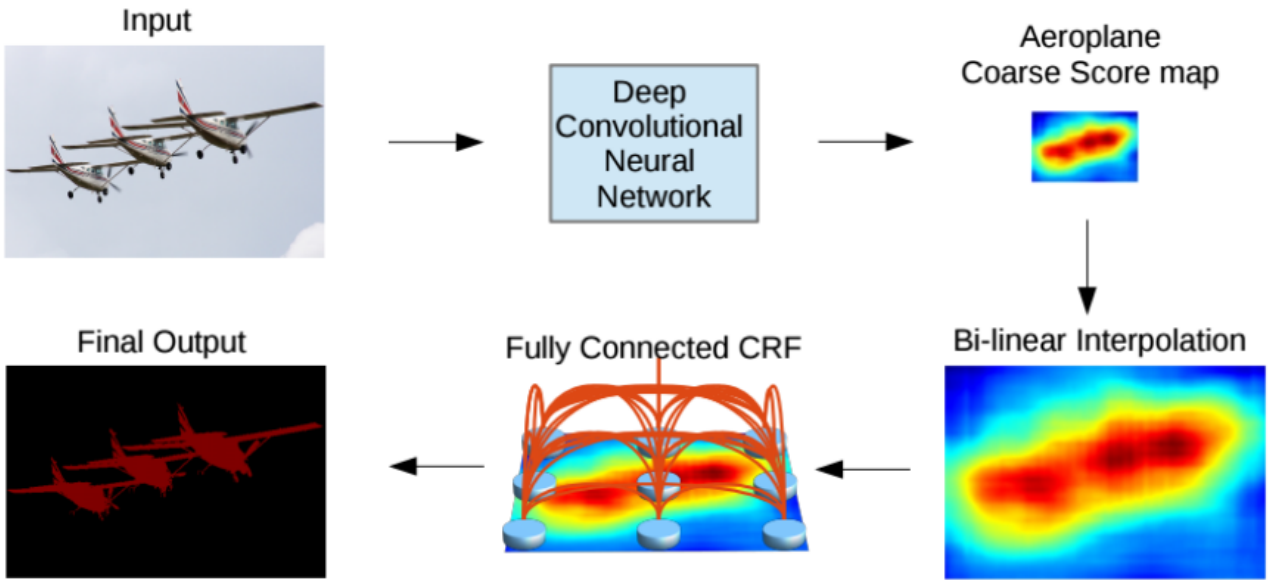


Fig. 1: Model Illustration. A Deep Convolutional Neural Network such as VGG-16 or ResNet-101 is employed in a fully convolutional fashion, using atrous convolution to reduce the degree of signal downsampling (from 32x down 8x). A bilinear interpolation stage enlarges the feature maps to the original image resolution. A fully connected CRF is then applied to refine the segmentation result and better capture the object boundaries.

Deep Convolutional Nets and Fully Connected CRF

PASCAL VOC 2012

PASCAL VOC 2012

Method	before CRF	after CRF
LargeFOV	65.76	69.84
ASPP-S	66.98	69.73
ASPP-L	68.96	71.57

TABLE 3: Effect of ASPP on PASCAL VOC 2012 *val* set performance (mean IOU) for VGG-16 based DeepLab model. **LargeFOV**: single branch, $r = 12$. **ASPP-S**: four branches, $r = \{2, 4, 8, 12\}$. **ASPP-L**: four branches, $r = \{6, 12, 18, 24\}$.

MSC	COCO	Aug	LargeFOV	ASPP	CRF	mIOU
						68.72
✓						71.27
✓	✓					73.28
✓	✓	✓				74.87
✓	✓	✓	✓			75.54
✓	✓	✓		✓		76.35
✓	✓	✓		✓	✓	77.69

TABLE 4: Employing ResNet-101 for DeepLab on PASCAL VOC 2012 *val* set. **MSC**: Employing mutli-scale inputs with max fusion. **COCO**: Models pretrained on MS-COCO. **Aug**: Data augmentation by randomly rescaling inputs.

IOU performance: intersection over union.

CRF helps.

Outline

Summary

- Deep Learning + Graphical Model
  - Deep Learning: predict one random variable.
  - Graphical Model: encode dependencies between random variables.
  - Together, able to estimate complex representations with complex dependencies.
- Combine the MRFs with deep learning to learn structured models jointly with deep features.
  - An algorithm to efficiently blend learning and inference.
- Combine the DCNNs with fully connected CRF for semantic image segmentation.
  - An efficient algorithm using high-dimensional filtering.

Reference I

Chen, Liang-Chieh, et al. "Learning Deep Structured Models." ICML. 2015.

Krhenbhl, Philipp, and Vladlen Koltun. "Efficient inference in fully connected crfs with gaussian edge potentials." Advances in neural information processing systems. 2011.

Chen, Liang-Chieh, et al. "Semantic image segmentation with deep convolutional nets and fully connected crfs." arXiv preprint arXiv:1412.7062 (2014).

Chen, Liang-Chieh, et al. "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs." arXiv preprint arXiv:1606.00915 (2016).