Machine Learning in Graphics and Vision

Prof. Dr.-Ing. Andreas Geiger

Autonomous Vision Group
MPI-IS / University of Tübingen

July 26, 2018

University of Tübingen
MPI for Intelligent Systems
Autonomous Vision Group

Lecture: Self-Driving Cars (WS 18/19)

COMPUTER SCIENCE

- Computational Systems Biology of Infections
- Algorithmen der Bioinformatik
- Kommunikationsnetze
- **Lernbasierte Computer Vision**
  - Home
  - People
  - Research
  - Publications
  - Teaching
    - Lecture: Self-Driving Cars
  - Theses
- Data Science & Analytics
- Neuronale Informationsverarbeitung
- Kognitive Systeme
- Cognitive Modeling
- Computergrafik
- Symbolisches Rechnen
- Human-Computer Interaction

Lecture: Self-Driving Cars

Content

This new course will give an introduction to self-driving cars. The course covers topics in perception, planning, control and end-to-end driving, amongst others.

Overview

→ SWS: 2 V + 2 Ü
→ 6 ECTS
→ Veranstaltungsnummer: INF

News

→ Please enroll in ILIAS at the beginning of the semester.

Exercises

By continuous and active participation in the weekly exercises, students may obtain a 0.3 bonus on the final grade, when passing the exam. To qualify for this bonus, the student must successfully solve 60% of the assigned homework problems which will be determined by grading the submitted homework solutions.

Lecturer
Prof. Dr. Andreas Geiger

TAs
tbd

Lecture Dates
Wintersemester 2018

Exam Dates
→ tbd

Overview

Structured Prediction I

- Graphical Models: Factor Graphs
- Inference: Belief Propagation

Structured Prediction II

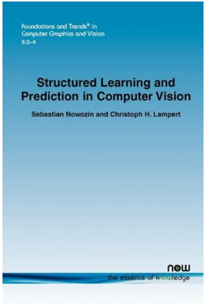
- Stereo & Optical Flow
- Multi-view Reconstruction

Structured Prediction III

- Parameter Estimation
- Deep Structured Models

Materials

- Nowozin, Lampert: Structured Learning and Prediction in Computer Vision
  Foundations and Trends in Computer Graphics and Vision, Volume 6, Number 3-4
- `http://www.nowozin.net/sebastian/cvpr2012tutorial/`



Recap

Factor Graph

Factor Graph

Given a function

$$f(x_1, \ldots, x_n) = \prod_i f_i(\mathcal{X}_i)$$

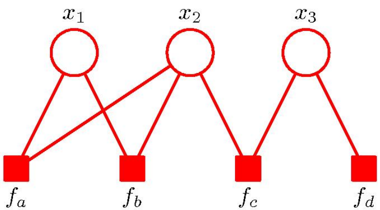
the **factor graph (FG)** has a **square node** for each factor $f_i(\mathcal{X}_i)$ and a **circle node** for each variable x_j. We typically specify this factorization up to a normalization constant

$$p(x_1, \ldots, x_n) = \frac{1}{Z} \prod_i f_i(\mathcal{X}_i)$$

when representing a distribution $p(\cdot)$.

Example

- Factor Graph:

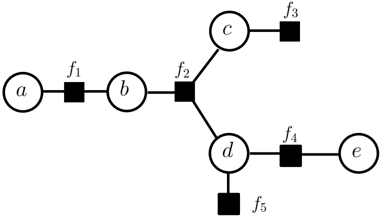


- Distribution:

$$p(x) = \frac{1}{Z} f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)$$

Marginal Inference

- Consider a branching graph:



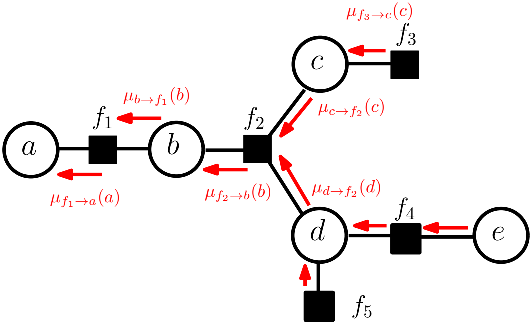
with factors

$$f_1(a, b) f_2(b, c, d) f_3(c) f_4(d, e) f_5(d)$$

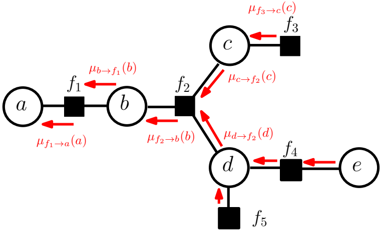
- How to find marginal $p(a, b)$?

Marginal Inference

- Idea: compute messages



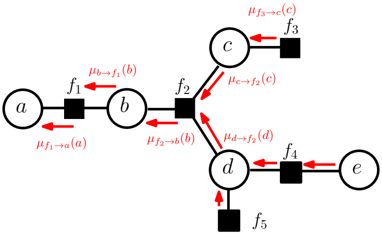
Marginal Inference



$$p(a,b) = \frac{1}{Z} f_1(a,b) \underbrace{\sum_{c,d,e} f_2(b,c,d) f_3(c) f_5(d) f_4(d,e)}_{\mu_{f_2 \to b}(b)}$$

$$\mu_{f_2 \to b}(b) = \sum_{c,d} f_2(b,c,d) f_3(c) f_5(d) \sum_{e} f_4(d,e)$$

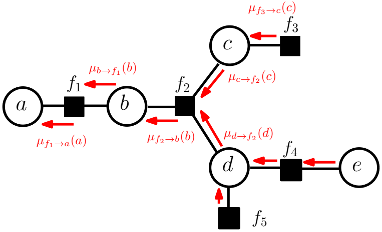
Marginal Inference



$$p(a,b) = \frac{1}{Z} f_1(a,b) \underbrace{\sum_{c,d,e} f_2(b,c,d) f_3(c) f_5(d) f_4(d,e)}_{\mu_{f_2 \to b}(b)}$$

$$\mu_{f_2 \to b}(b) = \sum_{c,d} f_2(b,c,d) \underbrace{f_3(c)}_{\mu_{c \to f_2}(c)} \underbrace{f_5(d) \sum_e f_4(d,e)}_{\mu_{d \to f_2}(d)}$$

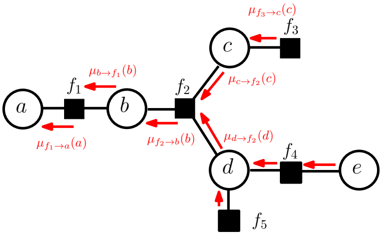
Marginal Inference



$$p(a,b) = \frac{1}{Z} f_1(a,b) \underbrace{\sum_{c,d,e} f_2(b,c,d) f_3(c) f_5(d) f_4(d,e)}_{\mu_{f_2 \to b}(b)}$$

$$\mu_{f_2 \to b}(b) = \sum_{c,d} f_2(b,c,d) \mu_{c \to f_2}(c) \mu_{d \to f_2}(d)$$

Marginal Inference



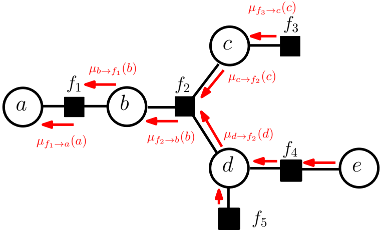
- Here (repeated from last slide):

$$\mu_{f_2\to b}(b) = \sum_{c,d} f_2(b, c, d)\mu_{c\to f_2}(c)\mu_{d\to f_2}(d)$$

- More general:

$$\mu_{f\to x}(x) = \sum_{\mathcal{X}_f \setminus x} f(\mathcal{X}_f) \prod_{y\in\{ne(f)\setminus x\}} \mu_{y\to f}(y)$$

Marginal Inference



- Here (repeated from last slide):

$$\mu_{d \to f_2}(d) = \mu_{f_5 \to d}(d)\mu_{f_4 \to d}(d)$$

- General:

$$\mu_{x \to f}(x) = \prod_{g \in \{ne(x) \setminus f\}} \mu_{g \to x}(x)$$

Log Representation

- Work with log-messages instead $\lambda = \log \mu$
- Factor-to-variable messages

$$\mu_{f\to x}(x) = \sum_{\mathcal{X}_f \setminus x} f(\mathcal{X}_f) \prod_{y \in \{ne(f) \setminus x\}} \mu_{y \to f}(y)$$

then become

$$\lambda_{f\to x}(x) = \log \left(\sum_{\mathcal{X}_f \setminus x} f(\mathcal{X}_f) \exp \left[\sum_{y \in \{ne(f) \setminus x\}} \lambda_{y \to f}(y) \right] \right)$$

Sum-Product Belief Propagation

- **Goal:** Compute marginals of distribution
- **Factor-to-variable messages:**

$$\lambda_{f\to x}(x) = \log\left(\sum_{\mathcal{X}_f\setminus x} f(\mathcal{X}_f)\exp\left\{\sum_{y\in\{ne(f)\setminus x\}}\lambda_{y\to f}(y)\right\}\right) \quad (1)$$

- **Variable-to-factor messages:**

$$\lambda_{x\to f}(x) = \sum_{g\in\{ne(x)\setminus f\}}\lambda_{g\to x}(x) \quad (2)$$

- $\sum_{\mathcal{X}_f\setminus x}$: Summation over all states of $\mathcal{X}_f \setminus x$ (Eq. 1)
- $\sum_{y\in\{ne(f)\setminus x\}}$ / $\sum_{g\in\{ne(x)\setminus f\}}$: Summation over all incoming messages / factors
- To avoid large values, subtract mean from $\lambda_{x\to f}(x)$ after message update (Eq. 2)

Max-Product Belief Propagationn

- **Goal:** Find most likely state (MAP state)
- **Factor-to-variable messages:**

$$\lambda_{f \to x}(x) = \max_{\mathcal{X}_f \setminus x} \left[\log f(\mathcal{X}_f) + \sum_{y \in \{ne(f) \setminus x\}} \lambda_{y \to f}(y) \right] \qquad (3)$$

- **Variable-to-factor messages:**

$$\lambda_{x \to f}(x) = \sum_{g \in \{ne(x) \setminus f\}} \lambda_{g \to x}(x) \qquad (2)$$

- $\max_{\mathcal{X}_f \setminus x}$: Maximization over all states of $\mathcal{X}_f \setminus x$ (Eq. 3)
- $\sum_{y \in \{ne(f) \setminus x\}}$ / $\sum_{g \in \{ne(x) \setminus f\}}$: Summation over all incoming messages / factors
- To avoid large values, subtract mean from $\lambda_{x \to f}(x)$ after message update (Eq. 2)

Readout

Read off marginal or MAP state at each variable:

- Similar to variable-to-factor messages
- However: summing over **all** incoming messages

$$p(x) = \exp\{\lambda(x)\} / \sum_x \exp\{\lambda(x)\} \quad (4)$$

$$x^* = \operatorname*{argmax}_x \sum_{g \in \{ne(x)\}} \lambda_{g \to x}(x) \quad (5)$$

with $\lambda(x) = \sum_{g \in \{ne(x)\}} \lambda_{g \to x}(x)$

Inference Algorithm Overview

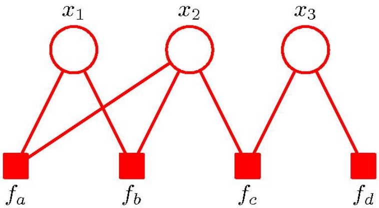
Belief Propagation Algorithm

- Input: `variables` and `factors`
- Allocate all `messages`
- Initialize the `message` log values to 0 (=uniform distribution)
- For N iterations do
  - Update all `factor-to-variable messages` (Eq. 1 or Eq. 3)
  - Update all `variable-to-factor messages` (Eq. 2)
  - Normalize all `variable-to-factor messages`:

    $\mu_{x \to f}(x) \leftarrow \mu_{x \to f}(x) - \text{mean}\left(\mu_{x \to f}(x)\right)$
- Read off marginal or MAP state at each variable (Eq. 4 or Eq. 5)

Log-Linear Models

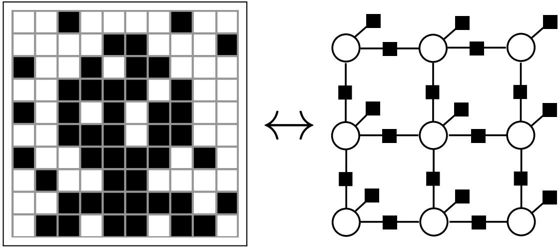
Example 1



$$
\begin{aligned}
p(x) &= \frac{1}{Z} f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3) \\
&= \frac{1}{Z} \exp\left\{\log\left(f_a(x_1, x_2) f_b(x_1, x_2) f_c(x_2, x_3) f_d(x_3)\right)\right\} \\
&= \frac{1}{Z} \exp\left\{\log f_a(x_1, x_2) + \log f_b(x_1, x_2) + \log f_c(x_2, x_3) + \log f_d(x_3)\right\} \\
&= \frac{1}{Z} \exp\left\{\psi_a(x_1, x_2) + \psi_b(x_1, x_2) + \psi_c(x_2, x_3) + \psi_d(x_3)\right\}
\end{aligned}
$$

- Log factors: $\psi_i(x_i) = \log f_i(x_i)$

Example 2: Image Denoising

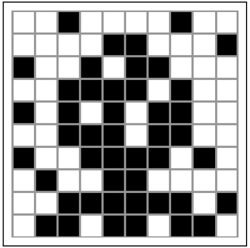


- Factor representation:

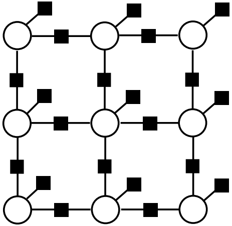
$$p(x) \propto \prod_{i=1}^{100} f_i(x_i) \prod_{i \sim j} f_{ij}(x_i, x_j)$$

- Variables: $x_1, \ldots, x_{100} \in \{0, 1\}$
- Unary factors: $f_i(x_i)$
- Pairwise factors: $f_{ij}(x_i, x_j)$

Example 2: Image Denoising



$\leftrightarrow$

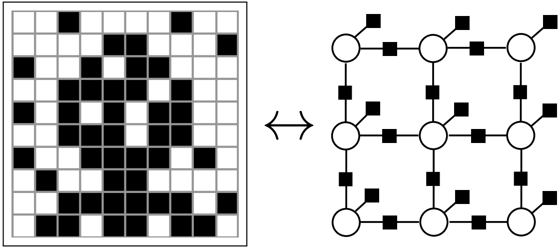


- Factor representation:

$$p(x) \propto \exp\left\{\sum_{i=1}^{100} \log f_i(x_i) \sum_{i\sim j} \log f_{ij}(x_i, x_j)\right\}$$

- Variables: $x_1, \ldots, x_{100} \in \{0, 1\}$
- Unary factors: $f_i(x_i)$
- Pairwise factors: $f_{ij}(x_i, x_j)$

Example 2: Image Denoising



- Log-linear representation:

$$p(x) \propto \exp\left\{\sum_{i=1}^{100}\psi_i(x_i) + \alpha\sum_{i\sim j}\psi_{ij}(x_i,x_j)\right\}$$

- Factors $f(\cdot)$ become potentials $\psi(\cdot)$ in log-representation
- Unary potentials: $\psi_i(x_i) = [x_i = o_i]$ with observation $o_i \in \{0,1\}$
- Pairwise potentials: $\psi_{ij}(x_i,x_j) = [x_i = x_j]$
- Parameter α controls strength of prior – how to choose α? Learn from data!

Parameter Estimation

Factor Graph: Inference vs. Learning

$$p(x_1, \ldots, x_{100}) = \frac{1}{Z} \exp \left\{ \sum_{i=1}^{100} \psi_i(x_i) + \alpha \sum_{i \sim j} \psi_{ij}(x_i, x_j) \right\}$$

- So far: Inference
  - Marginal distributions: $p(x_i) = \sum_{x \setminus x_i} p(x_1, \ldots x_{100})$
  - MAP solution: $x_1^*, \ldots, x_{100}^* = \operatorname{argmax}_{x_1, \ldots, x_{100}} p(x_1, \ldots x_{100})$
- Now: Learning
  - Estimate parameters (here: α) from dataset

Conditional Random Fields

Markov Random Field:

$$p(x) = \frac{1}{Z} \exp\left\{ \sum_{i=1}^{100} \psi_i(x_i) + \alpha \sum_{i \sim j} \psi_{ij}(x_i, x_j) \right\}$$

- Reason about output variables $x \in \mathcal{X}$ given one particular model instantiation

Structured Output Learning:

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

- Inputs $x \in \mathcal{X}$ can be any kind of objects
- Outputs $y \in \mathcal{Y}$ are complex (structured) objects
  - images, text, parse trees, folds of a protein, computer programs, ...

Conditional Random Fields

Markov Random Field:

$$p(x) = \frac{1}{Z} \exp\left\{ \sum_{i=1}^{100} \psi_i(x_i) + \alpha \sum_{i \sim j} \psi_{ij}(x_i, x_j) \right\}$$

- Reason about output variables $x \in \mathcal{X}$ given one particular model instantiation

Conditional Random Field:

$$p(y|x, w) = \frac{1}{Z} \exp\left\{ \sum_{i=1}^{100} \psi_i(x, y_i) + \alpha \sum_{i \sim j} \psi_{ij}(x, y_i, y_j) \right\}$$

- Make conditioning of variables y on data x and parameters w explicit (here $w = \alpha$)
- MRF notation: outputs $x \in \mathcal{X} \Rightarrow$ CRF notation: inputs $x \in \mathcal{X}$, outputs $y \in \mathcal{Y}$
- Learning: Estimate w from dataset $\mathcal{D} = \{(x^1, y^1), \ldots, (x^N, y^N)\}$

Conditional Random Fields

Conditional Random Field – General Form:

$$p(y|x, w) = \frac{1}{Z(x, w)} \exp\{\langle w, \psi(x, y)\rangle\}$$

- Feature function: $\psi(x, y) : \mathcal{X} \times \mathbb{R}^M \to \mathbb{R}^K$
- Parameter vector: $w \in \mathbb{R}^K$ (M: num. output nodes, K: dim. feat. space)
- Partition function: $Z(x, w) = \sum_{y \in \mathcal{Y}} \exp\{\langle w, \psi(x, y)\rangle\}$
- Note: much more flexible than just a single α parameter!
- Learning: Estimate w from dataset $\mathcal{D} = \{(x^1, y^1), \dots, (x^N, y^N)\}$
- GM specifies decomposition of ψ into potentials (=log factors) ψ_f:

$$\psi(x, y) = \left(\psi_f(x, y_f), \dots, \psi_{|\mathcal{F}|}(x, y_{|\mathcal{F}|})\right)$$

Parameter Estimation

Goal: Maximize likelihood of outputs x conditioned on inputs y wrt. w:

$$w^* = \underset{w \in \mathbb{R}^K}{\operatorname{argmax}}\, p(y|x,w) \quad \text{with} \quad p(y|x,w) = \prod_{n=1}^{N} p(y^n|x^n,w)$$

This is equivalent to minimizing the negative conditional log-likelihood:

$$w^* = \underset{w \in \mathbb{R}^K}{\operatorname{argmin}}\, \mathcal{L}(w) \quad \text{with} \quad \mathcal{L}(w) = -\sum_{n=1}^{N} \log p(y^n|x^n,w)$$

Parameter Estimation

Goal: Minimize negative conditional log-likelihood $\mathcal{L}(w)$

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

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

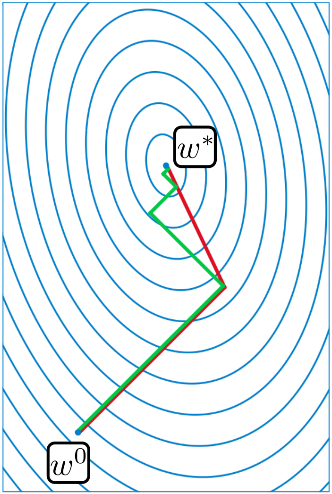
Optimization

Gradient Descent:

- Pick step size η, tolerance ϵ
- Initialize $w^0 = 0$
- Repeat until $\|v\| < \epsilon$
  - $v = \nabla_w \mathcal{L}(w)$
  - $\eta = \mathrm{argmin}_{\eta \in \mathbb{R}} \mathcal{L}(w^t - \eta v)$
  - $w^{t+1} = w^t - \eta v$

Alternatives:

- Conjugate gradient
- L-BFGS
- All require gradient!



Gradient of Negative Conditional Log-Likelihood

$$
\begin{aligned}
\mathcal{L}(w) &= -\sum_{n=1}^{N}\left[\langle w, \psi(x^n, y^n)\rangle - \log \sum_{y\in\mathcal{Y}} \exp\left\{\langle w, \psi(x^n, y)\rangle\right\}\right] \\
\nabla_w \mathcal{L}(w) &= -\sum_{n=1}^{N}\left[\psi(x^n, y^n) - \frac{\sum_{y\in\mathcal{Y}} \exp\left\{\langle w, \psi(x^n, y)\rangle\right\}\psi(x^n, y)}{\sum_{y\in\mathcal{Y}} \exp\left\{\langle w, \psi(x^n, y)\rangle\right\}}\right] \\
&= -\sum_{n=1}^{N}\left[\psi(x^n, y^n) - \sum_{y\in\mathcal{Y}} \frac{\exp\left\{\langle w, \psi(x^n, y)\rangle\right\}}{\sum_{y'\in\mathcal{Y}} \exp\left\{\langle w, \psi(x^n, y')\rangle\right\}}\psi(x^n, y)\right] \\
&= -\sum_{n=1}^{N}\left[\psi(x^n, y^n) - \sum_{y\in\mathcal{Y}} p(y|x^n, w)\psi(x^n, y)\right] \\
&= -\sum_{n=1}^{N}\left[\psi(x^n, y^n) - \mathbb{E}_{y\sim p(y|x^n, w)}\psi(x^n, y)\right]
\end{aligned}
$$

Gradient of Negative Conditional Log-Likelihood

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

When is $\mathcal{L}(w)$ minimal?

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

- Interpretation: we aim at **expectation matching**: $\mathbb{E}_{y \sim p} \psi(x, y) = \psi(x, y^{\text{obs}})$, but discriminatively: only for $x \in \{x^1, \dots, x^N\}$

Note:

- Hessian is pos. definite $\Rightarrow \mathcal{L}(w)$ convex $\Rightarrow \nabla_w \mathcal{L}(w) = 0$ implies global optimum!
- Only true as $p(y|x, w)$ is log-linear in $w \in \mathbb{R}^K$

Computational Complexity

Tasks for gradient descent with line search:

- Evaluate $\mathcal{L}(w)$
- Compute $v = \nabla_w \mathcal{L}(w)$

$$\mathcal{L}(w) = -\sum_{n=1}^{N} \left[\langle w, \psi(x^n, y^n) \rangle - \log \sum_{y \in \mathcal{Y}} \exp \{ \langle w, \psi(x^n, y) \rangle \} \right]$$

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

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

- Binary image segmentation: $|\mathcal{Y}| = 2^{640 \times 480} \approx 10^{92475}$
- We must use the structure in $\mathcal{Y}$, or we are lost!

Computational Complexity

$$
\begin{aligned}
\mathcal{L}(w) &= -\sum_{n=1}^{N}\left[\langle w, \psi(x^n, y^n)\rangle - \log \sum_{y\in\mathcal{Y}} \exp\left\{\langle w, \psi(x^n, y)\rangle\right\}\right] \\
\nabla_w \mathcal{L}(w) &= -\sum_{n=1}^{N}\left[\psi(x^n, y^n) - \sum_{y\in\mathcal{Y}} p(y|x^n, w)\psi(x^n, y)\right]
\end{aligned}
$$

Computational complexity: $O(ND^M K)$

- N: number of samples in dataset ($\approx$ 100 to 1,000,000)
- M: number of output nodes ($\approx$ 100 to 1,000,000)
- D: maximal number of labels per output node ($\approx$ 2 to 100)
- K: dimensionality of feature space

Computational Complexity

$$\mathcal{L}(w) = -\sum_{n=1}^{N} \left[\langle w, \psi(x^n, y^n) \rangle - \log \sum_{y \in \mathcal{Y}} \exp \{ \langle w, \psi(x^n, y) \rangle \} \right]$$

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

Computational complexity: $O(N D^M K)$

- N: number of samples in dataset ($\approx$ 100 to 1,000,000)
- M: number of output nodes ($\approx$ 100 to 1,000,000)
- D: maximal number of labels per output node ($\approx$ 2 to 100)
- K: dimensionality of feature space

Probabilistic Inference to the Rescue

Remember: in a graphical model, features decompose as follows

$$\psi(x, y) = \big(\psi_f(x, y_f), \ldots, \psi_{|\mathcal{F}|}(x, y_{|\mathcal{F}|})\big)$$

Thus:

$$\begin{aligned} \sum_{y \in \mathcal{Y}} \exp\{\langle w, \psi(x^n, y)\rangle\} &= \sum_{y \in \mathcal{Y}} \exp\left\{\sum_{f \in \mathcal{F}} \langle w_f, \psi_f(x^n, y_f)\rangle\right\} \\ &= \sum_{y \in \mathcal{Y}} \prod_{f \in \mathcal{F}} \underbrace{\exp\{\langle w_f, \psi_f(x^n, y_f)\rangle\}}_{\text{factor } f(\cdot)} \end{aligned}$$

- Can be efficiently calculated/approximated using message passing (run unnormalized sum-product BP, sum over any unnormalized marginal)

Probabilistic Inference to the Rescue

Furthermore:

$$
\begin{aligned}
\sum_{y\in\mathcal{Y}} p(y|x^n,w)\psi(x^n,y) &= \mathbb{E}_{y\sim p(y|x^n,w)} \sum_{f\in\mathcal{F}} \psi_f(x^n,y_f) \\
&= \sum_{f\in\mathcal{F}} \mathbb{E}_{y\sim p(y|x^n,w)} \psi_f(x^n,y_f) \\
&= \sum_{f\in\mathcal{F}} \mathbb{E}_{y_f\sim p(y_f|x^n,w)} \psi_f(x^n,y_f) \\
&= \underbrace{\sum_{f\in\mathcal{F}}}_{|\mathcal{F}|} \underbrace{\sum_{y_f\in\mathcal{Y}_f}}_{D^F} \underbrace{p(y_f|x^n,w)}_{\text{marginal}} \psi_f(x^n,y_f)
\end{aligned}
$$

- $|\mathcal{F}|$: number of factors, D: max. number of labels, F: order of largest factor
- Marginals can be calculated efficiently in polynomial time (*e.g.*, with BP)

Computational Complexity

$$\mathcal{L}(w) = -\sum_{n=1}^{N} \left[\langle w, \psi(x^n, y^n) \rangle - \log \sum_{y \in \mathcal{Y}} \exp\{\langle w, \psi(x^n, y) \rangle\} \right]$$

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

Computational complexity: $\cancel{O(N D^M K)} \quad \rightarrow \quad O(N|\mathcal{F}|D^F K)$

- N: number of samples in dataset ($\approx$ 100 to 1,000,000)
- M: number of output nodes ($\approx$ 100 to 1,000,000)
- D: maximal number of labels per output node ($\approx$ 2 to 100)
- K: dim. of feature space, $|\mathcal{F}|$: number of factors, F: order of largest factor

Computational Complexity

$$\mathcal{L}(w) = -\sum_{n=1}^{N} \left[\langle w, \psi(x^n, y^n) \rangle - \log \sum_{y \in \mathcal{Y}} \exp \{ \langle w, \psi(x^n, y) \rangle \} \right]$$

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

Computational complexity: $O(N|\mathcal{F}|D^F K)$

- N: number of samples in dataset (≈ 100 to 1,000,000)
- M: number of output nodes (≈ 100 to 1,000,000)
- D: maximal number of labels per output node (≈ 2 to 100)
- K: dim. of feature space, $|\mathcal{F}|$: number of factors, F: order of largest factor

Computational Complexity

Large Datasets

- Processing all N training samples for one gradient update is slow

How can we estimate parameters efficiently?

- Simplify model to make gradient updates faster $\Rightarrow$ results get worse
- Train model on subsampled dataset $\Rightarrow$ ignores information
- Parallelize across CPUs/GPUs $\Rightarrow$ bottlenecks, doesn't save computation
- Stochastic gradient descent

Stochastic Gradient Descent (SGD)

Stochastic Gradient Descent:

- Keep maximizing $p(w|\mathcal{D})$
- In each gradient step:
  - Create random subset $\mathcal{D}' \subset \mathcal{D}$ (typically $1 \leq \mathcal{D}' \leq 64$)
  - Follow approximate gradient:

$$\nabla_w \approx - \sum_{(x^n, y^n) \in \mathcal{D}'} \left[\psi(x^n, y^n) - \mathbb{E}_{y \sim p(y|x^n, w)} \psi(x^n, y) \right]$$

Comments:

- Line search no longer possible $\Rightarrow$ extra hyper-parameter η
- SGD converges to $\operatorname{argmin}_w \mathcal{L}(w)$! (if η chosen right)
- SGD needs more iterations, but each one is faster
- See also: Bottou & Bousquet: The Tradeoffs of Large Scale Learning, NIPS 2007

Computational Complexity

$$
\begin{aligned}
\mathcal{L}(w) &= -\sum_{n=1}^{N}\left[\langle w, \psi(x^n, y^n)\rangle - \log \sum_{y\in\mathcal{Y}} \exp\{\langle w, \psi(x^n, y)\rangle\}\right] \\
\nabla_w \mathcal{L}(w) &= -\sum_{n=1}^{N}\left[\psi(x^n, y^n) - \sum_{y\in\mathcal{Y}} p(y|x^n, w)\psi(x^n, y)\right]
\end{aligned}
$$

Computational complexity: $O(N|\mathcal{F}|D^F K)$

- N: number of samples in dataset ($\approx$ 100 to 1,000,000)
- M: number of output nodes ($\approx$ 100 to 1,000,000)
- D: maximal number of labels per output node ($\approx$ 2 to 100)
- K: dim. of feature space, $|\mathcal{F}|$: number of factors, F: order of largest factor

Computational Complexity

$$\mathcal{L}(w) = -\sum_{n=1}^{N} \left[\langle w, \psi(x^n, y^n) \rangle - \log \sum_{y \in \mathcal{Y}} \exp \{ \langle w, \psi(x^n, y) \rangle \} \right]$$

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

Computational complexity: $O(N|\mathcal{F}|D^F K)$

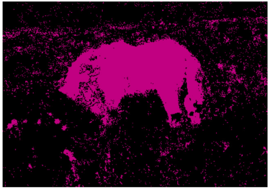
- N: number of samples in dataset ($\approx$ 100 to 1,000,000)
- M: number of output nodes ($\approx$ 100 to 1,000,000)
- D: maximal number of labels per output node ($\approx$ 2 to 100)
- K: dim. of feature space, $|\mathcal{F}|$: number of factors, F: order of largest factor

Feature Functions

Semantic Segmentation:

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

original



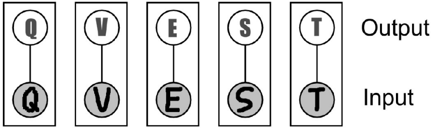
local classification

local + smoothness

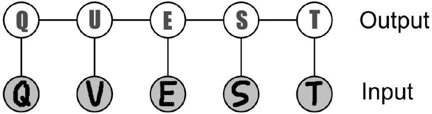
Feature Functions

Handwriting Recognition:

- $\psi_i(x, y_i) \in \mathbb{R}^{\approx 1000}$: image representation (*e.g.*, pixels, gradients)
  $\rightarrow \langle w_i, \psi_i(x, y_i) \rangle$: local classifier for letters
- $\psi_{i,j}(y_i, y_j) \in \mathbb{R}^{26 \times 26}$: letter/letter indicator
  $\rightarrow \langle w_{i,j}, \psi_{i,j}(y_i, y_j) \rangle$: encourage/suppress letter combinations
- Combined: $\operatorname{argmax}_y p(y|x, w)$ is "corrected" version of local cues



local classification



local + "correction"

Feature Functions

Pose Estimation:

- $\psi_i(x, y_i) \in \mathbb{R}^{\approx 1000}$: image representation (*e.g.*, HoG)
  $\rightarrow \langle w_i, \psi_i(x, y_i) \rangle$: local confidence map
- $\psi_{i,j}(y_i, y_j) = \text{fit}(y_i, y_j) \in \mathbb{R}^1$: test for geometric fit
  $\rightarrow \langle w_{i,j}, \psi_{i,j}(y_i, y_j) \rangle$: penalizer for unrealistic poses
- Combined: $\text{argmax}_y \, p(y|x, w)$ is sanitized version of local cues

original local classification local + geometry

Feature Functions

Typical feature functions for CRFs in computer vision:

- Unary terms $\psi_i(x, y_i)$: local representation, high-dimensional
  $\rightarrow \langle w_i, \psi_i(x, y_i) \rangle$: local classifier
- Pairwise terms $\psi_{i,j}(y_i, y_j)$: prior knowledge, typically low-dimensional
  $\rightarrow \langle w_{i,j}, \psi_{i,j}(y_i, y_j) \rangle$: penalize inconsistencies
- Pairwise terms sometimes also depend on x: $\psi_{i,j}(x, y_i, y_j)$

Learning adjusts parameters:

- Unary weights w_i: learn local linear classifiers
- Pairwise weights $w_{i,j}$: learn importance of smoothing/penalization
- $\operatorname{argmax}_y p(y|x, w)$ is cleaned up version of local prediction

Piece-wise Training

Sometimes, training the entire model at once is not easy:

- If terms depend on parameters in non-linear fashion
- If feature representations are high-dimensional, learning can be very slow

Alternative: Piece-wise Training

- Pre-train classifiers $p(y_i|x)$; set $\psi_i(x, y_i) = \log p(y_i|x) \in \mathbb{R}$
- Learn one-dimensional weight per classifier: $\langle w_i, \psi_i(x, y_i) \rangle$

Advantage:

- Lower dimensional feature vector during training/inference $\rightarrow$ faster
- $\log p(y_i|x)$ can be stronger classifiers, *e.g.*, non-linear SVMs, CNNs, ..

Disadvantage

- If local classifiers are bad, CRF training cannot fix this

Summary

Given:

- Training set $\mathcal{D} = \{(x^1, y^1), \dots, (x^N, y^N)\}$ with $(x^n, y^n) \overset{\text{i.i.d.}}{\sim} d(x, y)$
- Feature function: $\psi(x, y) : \mathcal{X} \times \mathbb{R}^M \to \mathbb{R}^K$

Task:

- Find parameter vector w such that $p(y|x, w) = \frac{1}{Z(x,w)} \exp\{\langle w, \psi(x, y)\rangle\} \approx d(y|x)$

Minimize negative conditional log-likelihood:

$$\mathcal{L}(w) = -\sum_{n=1}^{N} \left[\langle w, \psi(x^n, y^n)\rangle - \log \sum_{y \in \mathcal{Y}} \exp\{\langle w, \psi(x^n, y)\rangle\} \right]$$

- Convex optimization problem $\to$ gradient descent works
- Training needs repeated runs of probabilistic inference, faster is better

Summary

Gradient of negative conditional log-likelihood:

$$\mathcal{L}(w) = -\sum_{n=1}^{N} \left[\langle w, \psi(x^n, y^n) \rangle - \log \sum_{y \in \mathcal{Y}} \exp \{ \langle w, \psi(x^n, y) \rangle \} \right]$$

Problem	**Solution**	**Method**
$\|\mathcal{Y}\|$ too large	exploit structure	belief propagation
N too large	mini-batches	stochastic gradient descent
K too large	trained ψ	piece-wise training

Deep Structured Models

Motivation

Log-Linear Models:

$$p(y|x,w) = \frac{1}{Z(x,w)} \exp\{\langle w, \psi(x,y)\rangle\}$$

- Log-linear in the parameters $w \Rightarrow$ features must do all the heavy lifting
- Only linear combination of features is learned

Deep Structured Models:

$$p(y|x,w) = \frac{1}{Z(x,w)} \exp\{\psi(x,y,w)\}$$

- Potential functions directly parametrized via w
- Results in a much more flexible model (ψ can represent, *e.g.*, a neural network)

Deep Structured Models

Negative Log-Likelihood and its Gradient:

$$\begin{aligned}
\mathcal{L}(w) &= -\sum_{n=1}^{N}\left[\psi(x^n, y^n, w) - \log\sum_{y\in\mathcal{Y}}\exp\left\{\psi(x^n, y, w)\right\}\right]\\
\nabla_w\mathcal{L}(w) &= -\sum_{n=1}^{N}\left[\nabla_w\psi(x^n, y^n, w) - \sum_{y\in\mathcal{Y}}p(y|x^n, w)\nabla_w\psi(x^n, y, w)\right]
\end{aligned}$$

- Similar form as for log-linear models
- Differences to log-linear model highlighted in red

Deep Structured Models

Negative Log-Likelihood and its Gradient:

$$\mathcal{L}(w) = -\sum_{n=1}^{N}\left[\psi(x^n, y^n, w) - \log \sum_{y\in\mathcal{Y}} \exp\{\psi(x^n, y, w)\}\right]$$

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

- Again, sums can be efficiently computed as features decompose

$$\psi(x, y, w) = \sum_{f\in\mathcal{F}} \psi_f(x, y_f, w)$$

- Let us now represent $\psi_f(x, y_f, w)$ using deep neural networks

Deep Structured Models

Algorithm:

- Forward pass to compute $\psi_f(x, y_f, w)$
- Backward pass to obtain gradients $\nabla_w \psi(x^n, y, w)$
- Compute marginals using message passing
- Update parameters w

What is the problem with this approach?

- Very slow as inference in GM is required for every gradient update

Alternatives

- Interleave learning and inference [Chen *et al*., ICML 2015]
- But still only applicable to very small scale problems

Inference Unrolling

Inference Unrolling

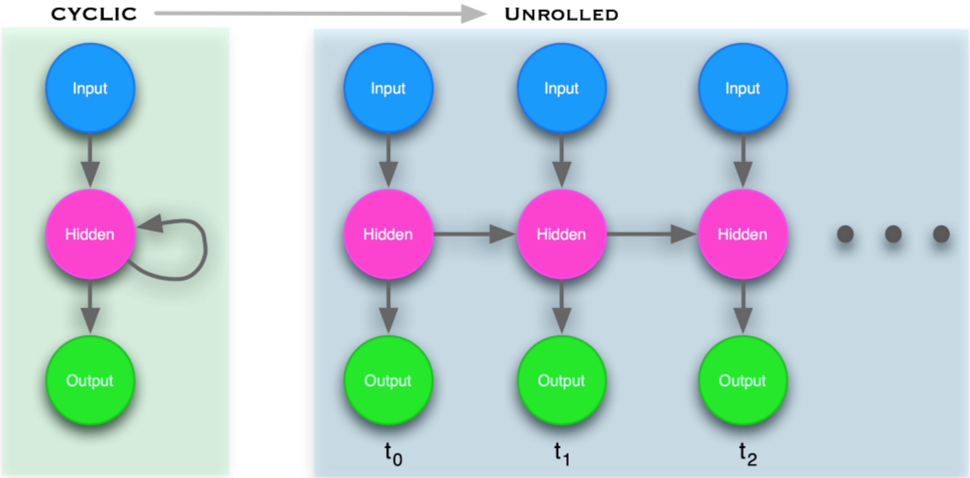
Idea:

- Consider inference as sequence of small computations
- "Unroll" a fixed number of inference iterations and consider as RNN
- Compute gradients, *e.g.*, using automatic differentiation

Warning:

- Now: empirical risk minimization
- Thus purely deterministic approach, giving up probabilistic viewpoint
- But often fast enough for efficient training in deep models
- Effectively integrates structure of the problem into architecture of the network
- Can be thought of as a form of regularization (hard constraint)

Inference Unrolling



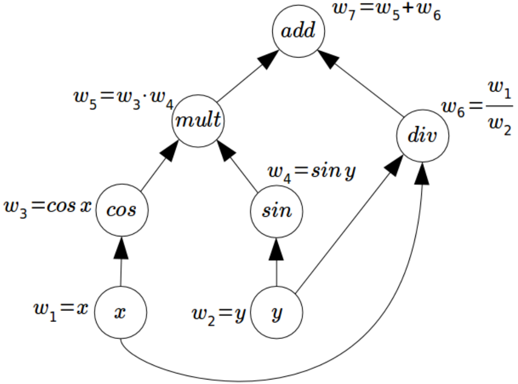
Automatic Differentiation

Idea:

- Rewrite complicated function as composition of simple functions: $f = f_0 \circ f_1 \circ \cdots \circ f_n$
- Each simple function f_k has a simple derivative
- Use chain rule: $\frac{\partial f_0}{\partial f_1} \frac{\partial f_1}{\partial f_2} \ldots \frac{\partial f_n}{\partial x}$
- Example:

$$f(x, y) = \cos(x)\sin(y) + \frac{x}{y}$$

Computation Graph:



http://www.columbia.edu/~ahd2125/post/2015/12/5/

Examples

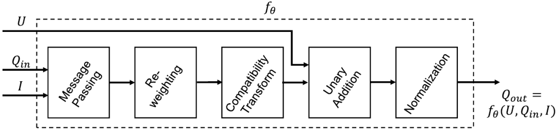
Conditional Random Fields as Recurrent Neural Networks

[Zheng *et al.*, ICCV 2015]

Conditional Random Fields as Recurrent Neural Networks

$$E(\mathbf{x}) = \sum_{i} \psi_u(x_i) + \sum_{i<j} \psi_p(x_i, x_j), \qquad (1)$$

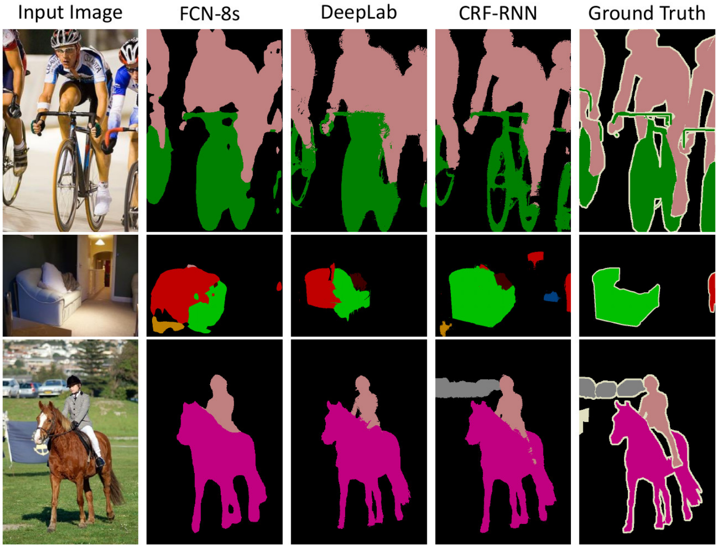
$$\psi_p(x_i, x_j) = \mu(x_i, x_j) \sum_{m=1}^{M} w^{(m)} k_G^{(m)}(\mathbf{f}_i, \mathbf{f}_j), \qquad (2)$$



Algorithm 1 Mean-field in dense CRFs [29], broken down to common CNN operations.

$Q_i(l) \leftarrow \frac{1}{Z_i} \exp\left(U_i(l)\right)$ for all i ▷ Initialization
while not converged **do**
$\tilde{Q}_i^{(m)}(l) \leftarrow \sum_{j \neq i} k^{(m)}(\mathbf{f}_i, \mathbf{f}_j) Q_j(l)$ for all m ▷ Message Passing
$\check{Q}_i(l) \leftarrow \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$ ▷ Weighting Filter Outputs
$\hat{Q}_i(l) \leftarrow \sum_{l' \in \mathcal{L}} \mu(l, l') \check{Q}_i(l')$ ▷ Compatibility Transform
$\breve{Q}_i(l) \leftarrow U_i(l) - \hat{Q}_i(l)$ ▷ Adding Unary Potentials
$Q_i \leftarrow \frac{1}{Z_i} \exp\left(\breve{Q}_i(l)\right)$ ▷ Normalizing
end while

Conditional Random Fields as Recurrent Neural Networks



RayNet: Learning Volumetric 3D Reconstruction with Ray Potentials

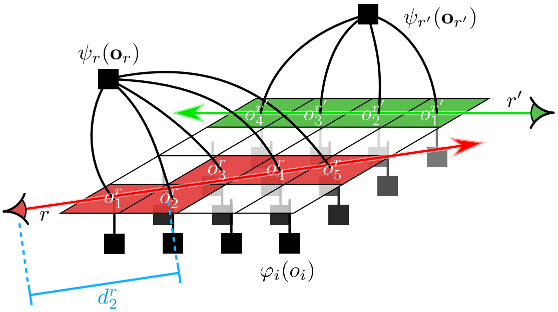
[Paschalidou, Ulusoy, Schmitt, van Gool & Geiger, CVPR 2018]

RayNet: Learning Volumetric 3D Reconstruction

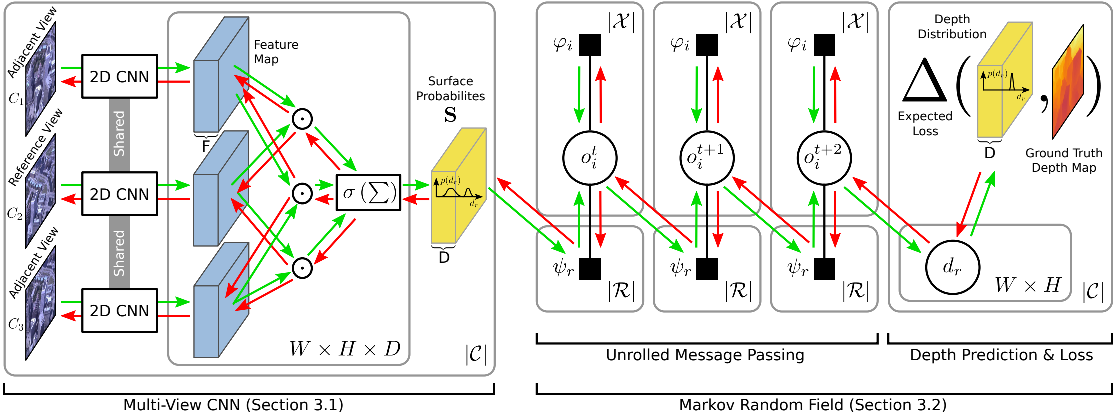
Distribution over voxel occupancies:

$$
\begin{aligned}
p(\mathbf{o}) &= \frac{1}{Z} \prod_{i \in \mathcal{X}} \underbrace{\varphi_i(o_i)}_{\text{unary}} \prod_{r \in \mathcal{R}} \underbrace{\psi_r(\mathbf{o}_r)}_{\text{ray}} \\
\varphi_i(o_i) &= \gamma^{o_i}(1-\gamma)^{1-o_i} \\
\psi_r(\mathbf{o_r}) &= \sum_{i=1}^{N_r} o_i^r \prod_{j<i} (1 - o_j^r) s_i^r
\end{aligned}
$$

Corresponding factor graph:

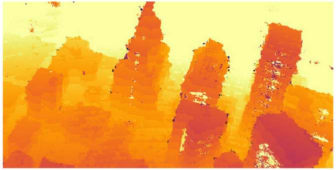


RayNet: Learning Volumetric 3D Reconstruction

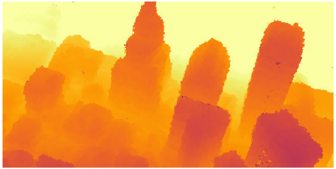


RayNet: Learning Volumetric 3D Reconstruction

(a) Image

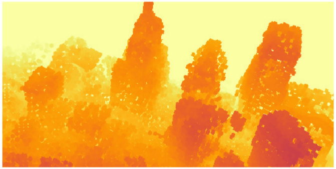


(b) Ours (CNN)

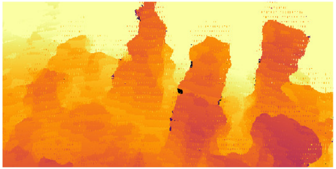


(c) Ours (CNN+MRF)

(d) ZNCC



(e) Ulusoy et al. [35]



(f) Hartmann et al. [14]

Q & A Session