Computer Vision

Lecture 7 – Learning in Graphical Models

Prof. Dr.-Ing. Andreas Geiger

Autonomous Vision Group

University of Tübingen / MPI-IS

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

ellis
European Laboratory for Learning and Intelligent Systems

Agenda

7.1 Conditional Random Fields

7.2 Parameter Estimation

7.3 Deep Structured Models

7.1
Conditional Random Fields

Inference vs. Learning

Markov Random Field:

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

- So far: **Inference**
  - Marginal distributions: $p(x_i) = \sum_{x \setminus x_i} p(x_1, \dots x_{100})$
  - MAP solution: $x_1^*, \dots, x_{100}^* = \operatorname{argmax}_{x_1, \dots, x_{100}} p(x_1, \dots x_{100})$
- Now: **Learning**
  - Estimate parameters (here regularization strength λ) from dataset
- Remark: In the literature, potentials are sometimes defined as the negative log factors, but here we will consider them as generic features and omit the sign

Conditional Random Fields

Markov Random Field:

$$p(\mathcal{X}) = \frac{1}{Z} \exp\left\{ \sum_i \psi_i(x_i) + \lambda \sum_{i \sim j} \psi_{ij}(x_i, x_j) \right\}$$

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

Structured Output Learning:

$$f_{\mathbf{w}} : \mathbb{X} \to \mathbb{Y}$$

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

Conditional Random Fields

Markov Random Field:

$$p(\mathcal{X}) = \frac{1}{Z} \exp\left\{ \sum_i \psi_i(x_i) + \lambda \sum_{i \sim j} \psi_{ij}(x_i, x_j) \right\}$$

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

Conditional Random Field:

$$p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) = \frac{1}{Z} \exp\left\{ \sum_i \psi_i(\mathcal{X}, y_i) + \lambda \sum_{i \sim j} \psi_{ij}(\mathcal{X}, y_i, y_j) \right\}$$

- Make conditioning of output $\mathcal{Y}$ on input $\mathcal{X}$ and parameters $\mathbf{w}$ explicit (here $\mathbf{w} = \lambda$)
- MRF notation: outputs $\mathcal{X} \in \mathbb{X} \Rightarrow$ CRF notation: inputs $\mathcal{X} \in \mathbb{X}$, outputs $\mathcal{Y} \in \mathbb{Y}$
- Learning: Estimate $\mathbf{w}$ from dataset $\mathcal{D} = \{(\mathcal{X}^1, \mathcal{Y}^1), \ldots, (\mathcal{X}^N, \mathcal{Y}^N)\}$

Conditional Random Fields

Conditional Random Field – General Form:

$$p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) = \frac{1}{Z(\mathcal{X}, \mathbf{w})} \exp\{\langle \mathbf{w}, \psi(\mathcal{X}, \mathcal{Y}) \rangle\}$$

- **Feature function:** $\psi(\mathcal{X}, \mathcal{Y}) : \mathbb{X} \times \mathbb{R}^M \to \mathbb{R}^D$ (concatenates potentials/features)
  Graphical model specifies decomposition of ψ into potentials (=log factors) ψ_k:
  $$\psi(\mathcal{X}, \mathcal{Y}) = \big(\psi_1(\mathcal{X}, \mathcal{Y}_1), \ldots, \psi_K(\mathcal{X}, \mathcal{Y}_K)\big)$$
- **Parameter vector:** $\mathbf{w} \in \mathbb{R}^D$ (M: num. output nodes, D: dim. of feature space)
  Note that this model is much more flexible than a model with a single λ
- **Partition function:** $Z(\mathcal{X}, \mathbf{w}) = \sum_{\mathcal{Y}} \exp\{\langle \mathbf{w}, \psi(\mathcal{X}, \mathcal{Y}) \rangle\}$
- **Learning:** Estimate $\mathbf{w}$ from dataset $\mathcal{D} = \{(\mathcal{X}^1, \mathcal{Y}^1), \ldots, (\mathcal{X}^N, \mathcal{Y}^N)\}$

7.2
Parameter Estimation

Parameter Estimation

Goal: Maximize likelihood of outputs $\mathcal{Y}$ conditioned on inputs $\mathcal{X}$ wrt. $\mathbf{w}$, assuming independent and identically distributed (IID) data (likelihood factorizes):

$$\hat{\mathbf{w}}_{ML} = \underset{\mathbf{w} \in \mathbb{R}^D}{\operatorname{argmax}} \; p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) \quad \text{with} \quad p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) = \prod_{n=1}^{N} p(\mathcal{Y}^n|\mathcal{X}^n, \mathbf{w})$$

In other words, find parameter vector $\hat{\mathbf{w}}_{ML}$ such that $p_{model}(\mathcal{Y}|\mathcal{X}, \hat{\mathbf{w}}_{ML}) \approx p_{data}(\mathcal{Y}|\mathcal{X})$.

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

$$\hat{\mathbf{w}}_{ML} = \underset{\mathbf{w} \in \mathbb{R}^D}{\operatorname{argmin}} \; \mathcal{L}(\mathbf{w}) \quad \text{with} \quad \mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N} \log p(\mathcal{Y}^n|\mathcal{X}^n, \mathbf{w})$$

Parameter Estimation

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

$$\hat{\mathbf{w}}_{ML} = \underset{\mathbf{w}\in\mathbb{R}^D}{\operatorname{argmin}}\ \mathcal{L}(\mathbf{w})$$

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

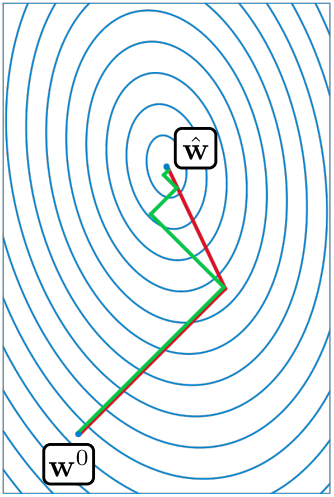
Optimization

Gradient Descent:

- Pick step size η and tolerance ϵ
- Initialize $\mathbf{w}^0$
- Repeat until $\|\mathbf{v}\| < \epsilon$
  - $\mathbf{v} = \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = \sum_{i=1}^{N} \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w})$
  - $\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \mathbf{v}$

Variants:

- Line search (green)
- Conjugate gradients (red)
- All require gradients, some (e.g., line search) require function evaluation



Gradient of Negative Conditional Log-Likelihood

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

Gradient of Negative Conditional Log-Likelihood

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

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

$$\mathbb{E}_{y \sim p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w})} \psi(\mathcal{X}^n, \mathcal{Y}) = \psi(\mathcal{X}^n, \mathcal{Y}^n) \Rightarrow \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = 0$$

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

Note:

- $\mathcal{L}(\mathbf{w})$ convex (Hessian positive semi-definite) $\Rightarrow \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) = 0 \Rightarrow$ **global optimum**
- Only true as $p(\mathcal{Y}|\mathcal{X}, \mathbf{w})$ is log-linear in $\mathbf{w} \in \mathbb{R}^D$ (we will also see non-linear models)

Computational Complexity

Task: For gradient descent with line search we must evaluate $\mathcal{L}(\mathbf{w})$ and $\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w})$:

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

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

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

Computational Complexity

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

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N}\left[\psi(\mathcal{X}^n, \mathcal{Y}^n) - \sum_{\mathcal{Y}\in\mathbb{Y}} p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w})\psi(\mathcal{X}^n, \mathcal{Y})\right]$$

Computational complexity: $O(NC^M D)$

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

Computational Complexity

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

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N}\left[\psi(\mathcal{X}^n, \mathcal{Y}^n) - \sum_{\mathcal{Y}\in\mathbb{Y}} p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w})\psi(\mathcal{X}^n, \mathcal{Y})\right]$$

Computational complexity: $O(NC^MD)$

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

Probabilistic Inference to the Rescue

Remember: in a graphical model, **features and weights decompose** as follows

$$\psi(\mathcal{X}, \mathcal{Y}) = (\psi_1(\mathcal{X}, \mathcal{Y}_1), \ldots, \psi_K(\mathcal{X}, \mathcal{Y}_K)) \qquad \mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_K)$$

Thus, the **partition function simplifies** as:

$$\begin{aligned} \sum_{\mathcal{Y}} \exp\left\{\langle \mathbf{w}, \psi(\mathcal{X}^n, \mathcal{Y})\rangle\right\} &= \sum_{\mathcal{Y}} \exp\left\{\sum_k \langle \mathbf{w}_k, \psi_k(\mathcal{X}^n, \mathcal{Y}_k)\rangle\right\} \\ &= \sum_{\mathcal{Y}} \prod_k \underbrace{\exp\left\{\langle \mathbf{w}_k, \psi_k(\mathcal{X}^n, \mathcal{Y}_k)\rangle\right\}}_{\text{k'th factor}} \end{aligned}$$

- Can be efficiently calculated/approximated using **message passing**
  (run sum-product belief propagation, sum over any of the unnorm. marginals)

Probabilistic Inference to the Rescue

Similarly, the **feature expectation simplifies** as:

$$
\begin{aligned}
\sum_{\mathcal{Y}} p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w})\psi(\mathcal{X}^n, \mathcal{Y}) &= \mathbb{E}_{\mathcal{Y}\sim p(\mathcal{Y}|\mathcal{X}^n,\mathbf{w})}\psi(\mathcal{X}^n, \mathcal{Y}) \\
&= \left(\mathbb{E}_{\mathcal{Y}\sim p(\mathcal{Y}|\mathcal{X}^n,\mathbf{w})}\psi_k(\mathcal{X}^n, \mathcal{Y}_k) \right)_{k\in 1,\ldots,K} \\
&= \left(\mathbb{E}_{\mathcal{Y}_k\sim p(\mathcal{Y}_k|\mathcal{X}^n,\mathbf{w})}\psi_k(\mathcal{X}^n, \mathcal{Y}_k) \right)_{k\in 1,\ldots,K} \\
&= \left(\sum_{\mathcal{Y}_k} p(\mathcal{Y}_k|\mathcal{X}^n, \mathbf{w})\psi_k(\mathcal{X}^n, \mathcal{Y}_k) \right)_{k\in 1,\ldots,K}
\end{aligned}
$$

- Now only C^F terms in sum over $\mathcal{Y}_k$ (C: max. number of labels, F: largest order)
- Marginals $p(\mathcal{Y}_k|\mathcal{X}^n, \mathbf{w})$ can be calculated efficiently (e.g., with BP)

Computational Complexity

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

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N}\left[\psi(\mathcal{X}^n, \mathcal{Y}^n) - \sum_{\mathcal{Y}} p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w})\psi(\mathcal{X}^n, \mathcal{Y})\right]$$

Computational complexity: ~~$O(NC^MD)$~~ $\rightarrow$ $O(NKC^FD)$

- N: number of samples in dataset ($\approx$ 100 to 1,000,000)
- M: number of output nodes ($\approx$ 100 to 1,000,000)
- C: maximal number of labels per output node ($\approx$ 2 to 100)
- D: dim. of feature space, K: number of factors, F: order of largest factor ($\approx$ 2-3)

Computational Complexity

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

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N}\left[\psi(\mathcal{X}^n, \mathcal{Y}^n) - \sum_{\mathcal{Y}} p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w})\psi(\mathcal{X}^n, \mathcal{Y})\right]$$

Computational complexity: $O(NKC^F D)$

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

Computational Complexity

Learning on large datasets:

- Processing all N training samples for one gradient update is slow
- Furthermore, often not all data fits into memory (as in deep learning)

How can we estimate parameters in this setting?

- 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:

- In each gradient step:
  - Create random subset $\mathcal{D}' \subset \mathcal{D}$ (typically $\mathcal{D}' \leq 256$)
  - Follow approximate gradient:

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

Comments:

- Line search no longer possible $\Rightarrow$ extra step-size hyper-parameter η
- SGD converges to $\operatorname{argmin}_{\mathbf{w}} \mathcal{L}(\mathbf{w})$! (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

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

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N}\left[\psi(\mathcal{X}^n, \mathcal{Y}^n) - \sum_{\mathcal{Y}} p(\mathcal{Y}|\mathcal{X}^n, \mathbf{w})\psi(\mathcal{X}^n, \mathcal{Y})\right]$$

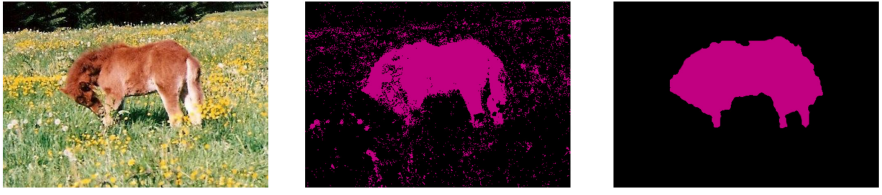
Computational complexity: $O(NKC^F D)$

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

Applications / Feature Functions

Semantic Segmentation:

- $\psi_i(\mathcal{X}, y_i) \in \mathbb{R}^{\approx 1000}$: local image features (e.g., bag of words, deep features)
  $\rightarrow \langle \mathbf{w}_i, \psi_i(\mathcal{X}, y_i) \rangle$: local classifier (like logistic regression)
- $\psi_{ij}(y_i, y_j) = [y_i = y_j] \in \mathbb{R}^1$: test for same label
  $\rightarrow \langle w_{ij}, \psi_{ij}(y_i, y_j) \rangle$: penalizer for label changes (if $w_{ij} > 0$)
- combined: $\operatorname{argmax}_{\mathcal{Y}} p(\mathcal{Y}|\mathcal{X}, \mathbf{w})$ is smoothed version of local cues



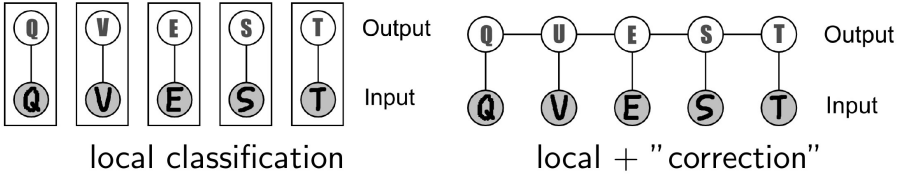
original | local classification | local + smoothness

Shotton, Winn, Rother and Criminisi: TextonBoost: Joint Appearance, Shape and Context Modeling for Multi-class Object Recognition and Segmentation. ECCV, 2006.23

Applications / Feature Functions

Handwriting Recognition:

- $\psi_i(\mathcal{X}, y_i) \in \mathbb{R}^{\approx 1000}$: image representation (e.g., pixels, gradients)
  $\rightarrow \langle \mathbf{w}_i, \psi_i(\mathcal{X}, y_i) \rangle$: local classifier for letters
- $\psi_{ij}(y_i, y_j) = \mathbf{e}_{y_i} \mathbf{e}_{y_j}^\top \in \mathbb{R}^{26 \times 26}$: letter/letter indicator (matrix with one element = 1)
  $\rightarrow \langle \mathbf{w}_{ij}, \psi_{ij}(y_i, y_j) \rangle$: encourage/suppress letter combinations
- Combined: $\operatorname{argmax}_{\mathcal{Y}} p(\mathcal{Y}|\mathcal{X}, \mathbf{w})$ is "corrected" version of local cues



Applications / Feature Functions

Pose Estimation:

- $\psi_i(\mathcal{X}, y_i) \in \mathbb{R}^{\approx 1000}$: image representation (e.g., HoG, deep features)
  $\rightarrow \langle \mathbf{w}_i, \psi_i(\mathcal{X}, y_i) \rangle$: local confidence map
- $\psi_{ij}(y_i, y_j) = \text{fit}(y_i, y_j) \in \mathbb{R}^1$: test for geometric fit / pose prior
  $\rightarrow \langle w_{ij}, \psi_{ij}(y_i, y_j) \rangle$: penalizer for unrealistic poses
- Combined: $\text{argmax}_{\mathcal{Y}}\, p(\mathcal{Y}|\mathcal{X}, \mathbf{w})$ is sanitized version of local cues

original

local classification

local + geometry

Ferrari, Marin-Jimenez and Zisserman: Progressive Search Space Reduction for Human Pose Estimation. CVPR, 2008

Applications / Feature Functions

Typical feature functions for CRFs in computer vision:

- Unary terms $\psi_i(\mathcal{X}, y_i)$: local representation, high-dimensional
  $\rightarrow \langle \mathbf{w}_i, \psi_i(\mathcal{X}, y_i) \rangle$: local classifier
- Pairwise terms $\psi_{ij}(y_i, y_j)$: prior knowledge, typically low-dimensional
  $\rightarrow \langle w_{ij}, \psi_{ij}(y_i, y_j) \rangle$: penalize inconsistencies
- Pairwise terms sometimes also depend on $\mathcal{X}$: $\psi_{ij}(\mathcal{X}, y_i, y_j)$

Learning adjusts parameters:

- Unary weights $\mathbf{w}_i$: learn local linear classifiers
- Pairwise weights w_{ij}: learn importance of smoothing/penalization
- $\operatorname{argmax}_{\mathcal{Y}} p(\mathcal{Y}|\mathcal{X}, \mathbf{w})$ is cleaned up version of local prediction

Piece-wise Training

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

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

Alternative: **Piece-wise Training**

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

Advantage:

- Lower dimensional feature vector during training/inference $\rightarrow$ faster
- $\log p(y_i|\mathcal{X})$ 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 $\mathcal{D} = \{(\mathcal{X}^1, \mathcal{Y}^1), \ldots, (\mathcal{X}^N, \mathcal{Y}^N)\}$ with $(\mathcal{X}^n, \mathcal{Y}^n) \overset{\text{i.i.d.}}{\sim} p_{data}(\mathcal{X}, \mathcal{Y})$
- Feature function: $\psi(\mathcal{X}, \mathcal{Y}) : \mathbb{X} \times \mathbb{R}^M \to \mathbb{R}^D$

Task:

- Find parameter vector $\hat{\mathbf{w}}_{ML}$ such that
  $p_{model}(\mathcal{Y}|\mathcal{X}, \hat{\mathbf{w}}_{ML}) = \frac{1}{Z(\mathcal{X}, \hat{\mathbf{w}}_{ML})} \exp\{\langle \hat{\mathbf{w}}_{ML}, \psi(\mathcal{X}, \mathcal{Y})\rangle\} \approx p_{data}(\mathcal{Y}|\mathcal{X})$

Minimize negative conditional log-likelihood:

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

- Convex optimization problem $\to$ gradient descent leads to global optimum
- Training needs repeated runs of probabilistic inference $\Rightarrow$ must be fast

Summary

Gradient of negative conditional log-likelihood:

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

Problem	Solution	Method
$\lvert\mathbb{Y}\rvert$ too large	exploit structure	belief propagation
N too large	mini-batches	stochastic gradient descent
D too large	trained ψ	piece-wise training

7.3
Deep Structured Models

Motivation

Log-Linear Models:

$$p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) = \frac{1}{Z(\mathcal{X}, \mathbf{w})} \exp\{\langle \mathbf{w}, \psi(\mathcal{X}, \mathcal{Y}) \rangle\}$$

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

Deep Structured Models:

$$p(\mathcal{Y}|\mathcal{X}, \mathbf{w}) = \frac{1}{Z(\mathcal{X}, \mathbf{w})} \exp\{\psi(\mathcal{X}, \mathcal{Y}, \mathbf{w})\}$$

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

Deep Structured Models

Negative Log-Likelihood and its Gradient:

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

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = -\sum_{n=1}^{N}\left[\nabla_{\mathbf{w}}\psi(\mathcal{X}^n,\mathcal{Y}^n,\mathbf{w}) - \sum_{\mathcal{Y}}p(\mathcal{Y}|\mathcal{X}^n,\mathbf{w})\nabla_{\mathbf{w}}\psi(\mathcal{X}^n,\mathcal{Y},\mathbf{w})\right]$$

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

Deep Structured Models

Negative Log-Likelihood and its Gradient:

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

- Again, sums can be efficiently computed as features decompose

$$\psi(\mathcal{X},\mathcal{Y},\mathbf{w}) = \big(\psi_1(\mathcal{X},\mathcal{Y}_1,\mathbf{w}),\ldots,\psi_K(\mathcal{X},\mathcal{Y}_K,\mathbf{w})\big)$$

Deep Structured Models

Algorithm:

- Forward pass to compute $\psi_k(\mathcal{X}, \mathcal{Y}_k, \mathbf{w})$
- Backward pass to obtain gradients $\nabla_{\mathbf{w}} \psi(\mathcal{X}^n, \mathcal{Y}, \mathbf{w})$
- Compute marginals using message passing
- Update parameters $\mathbf{w}$

What is the problem with this approach?

- Very slow as forward and backward pass are required to calculate features and gradients for GM inference in every gradient update step

Alternatives:

- Interleave learning and inference [Chen et al., ICML 2015], but still slow
- Unrolled inference (simple, but we loose probabilistic interpretation)

Chen, Schwing, Yuille and Urtasun: Learning Deep Structured Models. ICML, 2015.

Inference Unrolling

Inference Unrolling

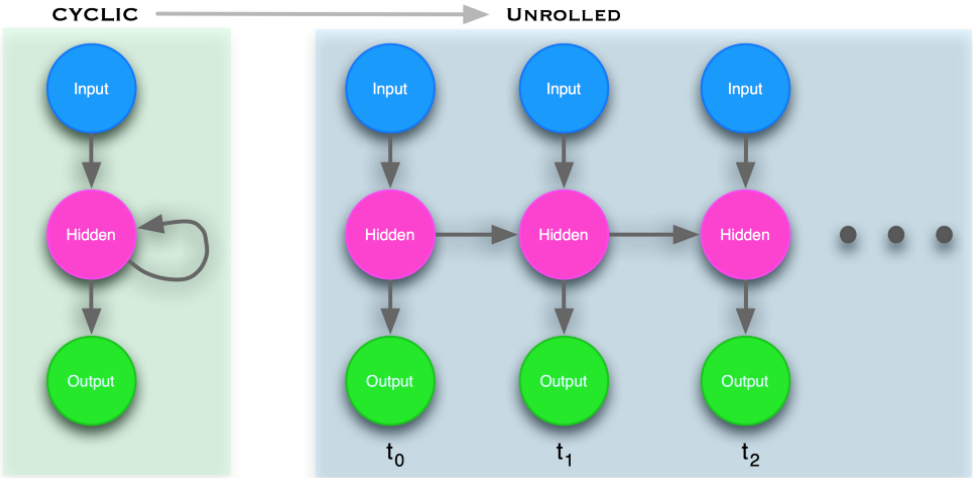
Idea:

- Consider inference as sequence of small computations
- "Unroll" a **fixed** number of inference iterations similar to RNN
- Compute gradients using automatic differentiation

Remarks:

- 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



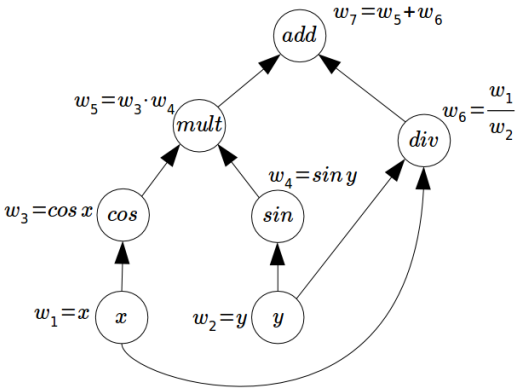
Automatic Differentiation

Automatic Differentiation:

- 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: $\frac{\partial f_0}{\partial f_1}\frac{\partial f_1}{\partial f_2}\dots\frac{\partial f_n}{\partial x}$
- **Example:**

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

Computation Graph:

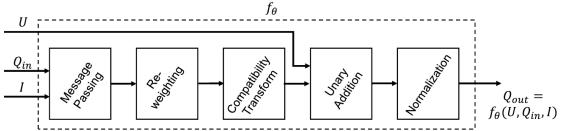


Examples

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.

$Q_i(l) \leftarrow \frac{1}{Z_i} \exp(U_i(l))$ 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

$\breve{Q}_i(l) \leftarrow \sum_m w^{(m)} \tilde{Q}_i^{(m)}(l)$ ▷ Weighting Filter Outputs

$\hat{Q}_i(l) \leftarrow \sum_{l' \in \mathcal{L}} \mu(l, l') \breve{Q}_i(l')$ ▷ Compatibility Transform

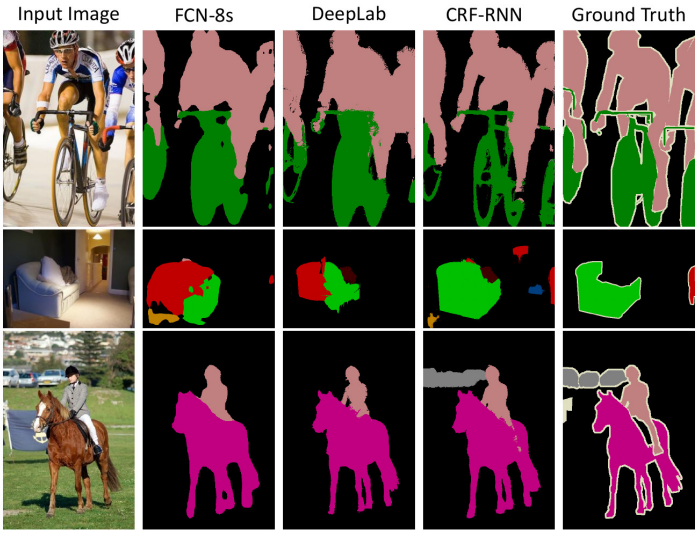
$\breve{Q}_i(l) \leftarrow U_i(l) - \hat{Q}_i(l)$ ▷ Adding Unary Potentials

$Q_i \leftarrow \frac{1}{Z_i} \exp\left(\breve{Q}_i(l)\right)$ ▷ Normalizing

end while

Zheng et al.: Conditional Random Fields as Recurrent Neural Networks. ICCV, 2015.

Conditional Random Fields as Recurrent Neural Networks



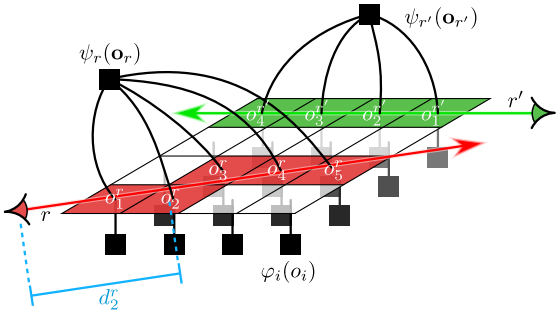
Zheng et al.: Conditional Random Fields as Recurrent Neural Networks. ICCV, 2015.

RayNet: Learning Volumetric 3D Reconstruction

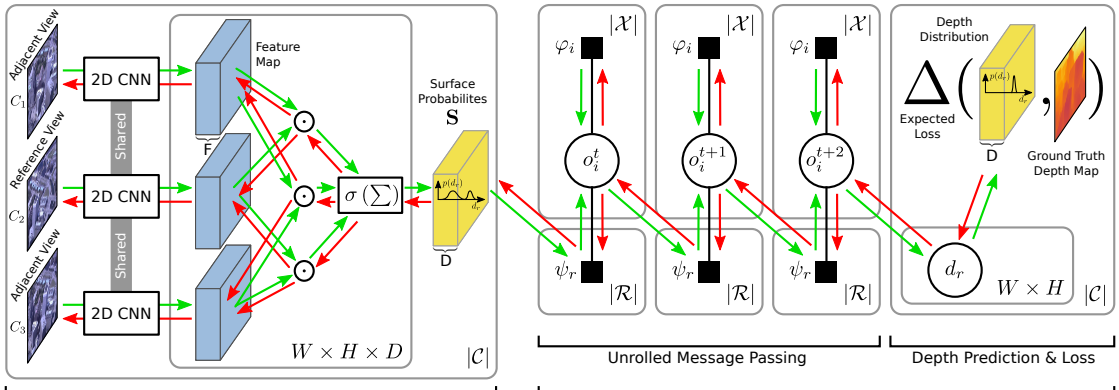
Distribution over voxel occupancies:

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

Corresponding factor graph:

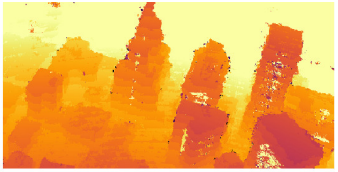


RayNet: Learning Volumetric 3D Reconstruction

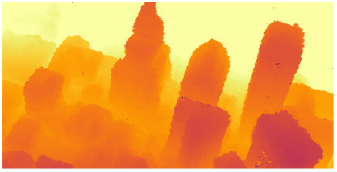


RayNet: Learning Volumetric 3D Reconstruction

(a) Image

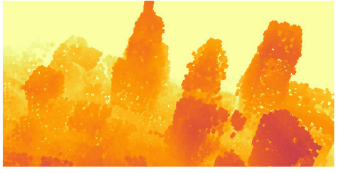


(b) Ours (CNN)

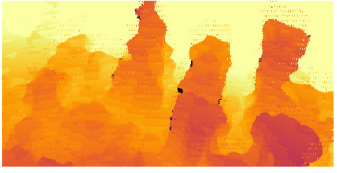


(c) Ours (CNN+MRF)

(d) ZNCC



(e) Ulusoy et al. [35]



(f) Hartmann et al. [14]