Introducing Structure in Deep Learning

Raquel Urtasun

Computer Science
UNIVERSITY OF TORONTO
Deep Learning is Everywhere!

What can I help you with?
Deep Learning is Everywhere!

ALPHAGO
00:10:29

AlphaGo
Google DeepMind

LEE SEDOL
00:01:00
Deep Learning is Everywhere!
Deep Learning is Everywhere!

360° Velodyne Laserscanner

Stereo Camera Rig

Monochrome  Color

GPS
Current Status of many AI Communities
I’m also guilty ...
And other people you might know ...
And other people you might know ...
And other people you might know ...
What’s the difference between Computer Vision and Machine Learning?
What’s the difference between Computer Vision and Machine Learning?

CV: what can a NNet do for you?
What’s the difference between Computer Vision and Machine Learning?

CV: what can a NNet do for you?

ML: What can you do for a NNet?
Today’s Talk

Will be a machine learning talk

1. Structure in the Outputs
2. Structure in the Loss
3. Structure in the Embedding
Structure in the Output
Many problems are complex and involve predicting many random variables that are statistically related.

**Scene understanding**

$x =$ image

$y :$ room layout

**Tag prediction**

$x =$ image

$y :$ tag "combo"

**Segmentation**

$x =$ image

$y :$ segmentation
Deep Learning

- Complex mapping $F(x, y, w)$ to predict output $y$ given input $x$ through a series of matrix multiplications, non-linearities and pooling operations.

Figure: Imagenet CNN [Krizhevsky et al. 12]
Complex mapping $F(x, y, w)$ to predict output $y$ given input $x$ through a series of matrix multiplications, non-linearities and pooling operations.

Figure: Imagenet CNN [Krizhevsky et al. 12]

We typically train the network to predict one random variable (e.g., ImageNet) by minimizing loss (e.g., cross-entropy).
Deep Learning

- Complex mapping $F(x, y, w)$ to predict output $y$ given input $x$ through a series of matrix multiplications, non-linearities and pooling operations

Figure: Imagenet CNN [Krizhevsky et al. 12]

- We typically train the network to predict one random variable (e.g., ImageNet) by minimizing loss (e.g., cross-entropy)
- Multi-task extensions: sum the loss of each task, and share part of the network (e.g., segmentation)
Deep Learning

- Complex mapping $F(x, y, w)$ to predict output $y$ given input $x$ through a series of matrix multiplications, non-linearities and pooling operations

Figure: Imagenet CNN [Krizhevsky et al. 12]

- We typically train the network to predict one random variable (e.g., ImageNet) by minimizing loss (e.g., cross-entropy)
- Multi-task extensions: sum the loss of each task, and share part of the network (e.g., segmentation)
- Use an MRF as a post processing step
**PROBLEM:** How can we take into account complex dependencies when predicting multiple variables?
**PROBLEM:** How can we take into account complex dependencies when predicting multiple variables?

**SOLUTION:** Graphical models
Graphical Models

- Convenient tool to illustrate dependencies among random variables

\[ E(y) = - \sum_i f_i(y_i) - \sum_{i,j \in \mathcal{E}} f(y_i, y_j) - \sum_{\alpha} f_{\alpha}(y_{\alpha}) \]

- Widespread usage among different fields: vision, NLP, comp. bio, …
In Computer Vision we usually express

\[ E(y) = - \sum_{i} f_{i}(y_{i}) - \sum_{i,j \in \mathcal{E}} f(y_{i}, y_{j}) - \sum_{\alpha} f_{\alpha}(y_{\alpha}) \]

- unaries
- pairwise
- high-order

For the purpose of this talk we are going to use a more compact notation

\[ E(y, w) = - \sum_{r \in \mathcal{R}} f_{r}(y_{r}, w) \]

with \( y_{r} \) is of any order, and \( f_{r} \) are a function of parameters \( w \).
In Computer Vision we usually express

\[ E(y) = - \sum_i f_i(y_i) - \sum_{i,j \in E} f(y_i, y_j) - \sum_{\alpha} f_\alpha(y_\alpha) \]

unaries \hspace{1cm} pairwise \hspace{1cm} high-order

For the purpose of this talk we are going to use a more compact notation

\[ E(y, w) = - \sum_{r \in R} f_r(y_r, w) \]

with \( y_r \) is of any order, and \( f_r \) are a function of parameters \( w \)
Probabilistic Discriminative Models

- In Computer Vision we usually express

\[
E(y) = - \sum_i f_i(y_i) - \sum_{i,j \in E} f(y_i, y_j) - \sum_{\alpha} f_{\alpha}(y_{\alpha})
\]

- For the purpose of this talk we are going to use a more compact notation

\[
E(y, w) = - \sum_{r \in \mathcal{R}} f_r(y_r, w)
\]

with \(y_r\) is of any order, and \(f_r\) are a function of parameters \(w\)

- Probabilistic discriminative models

\[
p(y|x; w) = \frac{1}{Z(x)} \exp \left( \sum_{r \in \mathcal{R}} f_r(x, y_r, w) \right)
\]

with \(Z(x, w) = \sum_y \exp \left( \sum_{r \in \mathcal{R}} f_r(x, y_r, w) \right)\) the partition function
MAP: maximum a posteriori estimate, or minimum energy configuration

\[ y^* = \arg \max_y \sum_{r \in R} f_r(y_r, w) \]

Probabilistic Inference: We might want to compute \( p(y_r) \) for any possible subset of variables \( r \), or \( p(y_r|y_p) \) for any subset \( r \) and \( p \)

Very difficult tasks in general (i.e., NP-hard). Some exceptions, e.g., low-tree width models and binary MRFs with sub-modular energies
Learning in CRFs

- Given pairs \((x, y) \in D\) we want to estimate \(f_r(x, y_r, w)\), i.e., the weights \(w\).
Learning in CRFs

- Given pairs $(x, y) \in \mathcal{D}$ we want to estimate $f_r(x, y, w)$, i.e., the weights $w$.
- We would like to do this by minimizing the empirical task loss:
  \[
  \min_w \frac{1}{N} \sum_{(x, y) \in \mathcal{D}} \ell_{\text{task}}(x, y, w)
  \]
Learning in CRFs

- Given pairs \((x, y) \in D\) we want to estimate \(f_r(x, y_r, w)\), i.e., the weights \(w\).
- We would like to do this by minimizing the empirical task loss

\[
\min_w \frac{1}{N} \sum_{(x,y) \in D} \ell_{task}(x, y, w)
\]

- Very difficult, instead we minimize a surrogate (typically convex) loss, e.g., hinge loss, log-loss

\[
\bar{\ell}_{log}(x, y, w) = -\ln p_{x,y}(y; w).
\]
\[
\bar{\ell}_{hinge}(x, y, w) = \max_{\hat{y} \in Y} \{\ell(y, \hat{y}) - w^T \Phi(x, \hat{y}) + w^T \Phi(x, y)\}
\]
Learning in CRFs

- Given pairs \((x, y) \in \mathcal{D}\) we want to estimate \(f_r(x, y_r, w)\), i.e., the weights \(w\).

- We would like to do this by minimizing the empirical task loss

\[
\min_w \frac{1}{N} \sum_{(x, y) \in \mathcal{D}} \ell_{task}(x, y, w)
\]

- Very difficult, instead we minimize a surrogate (typically convex) loss, e.g., hinge loss, log-loss

\[
\bar{\ell}_{log}(x, y, w) = -\ln p_{x,y}(y; w).
\]

\[
\bar{\ell}_{hinge}(x, y, w) = \max_{\hat{y} \in \mathcal{Y}} \{ \ell(y, \hat{y}) - w^T \Phi(x, \hat{y}) + w^T \Phi(x, y) \}
\]

- The assumption is that the model is log-linear

\[
F(x, y, w) = \sum_{r \in \mathcal{R}} w_r^T \phi(x, y)
\]

therefore they are very shallow.
**PROBLEM:** How can we make MRFs less shallow?
PROBLEM: How can we make MRFs less shallow?

SOLUTION: Deep Structured Models
Standard CNN

Deep Structured Models: sum of complex functions of subsets of variables

\[ F(x, y, w) = \sum_{r \in \mathcal{R}} f_r(x, y_r, w) \]
- Standard CNN

- Deep Structured Models: sum of complex functions of subsets of variables

\[
F(x, y, w) = \sum_{r \in \mathcal{R}} f_r(x, y_r, w)
\]

[Diagram of deep structured models with CNN nodes and connections between nodes labeled y1, y2, y3, y1,2, CNN1, CNN2, CNN3, CNN4, CNN5.]
Probability of a configuration $y$:

$$p(y \mid x; w) = \frac{1}{Z(x, w)} \exp F(x, y, w)$$

with $Z(x, w)$ the partition function
Learning

- **Probability of a configuration** \( y \):

\[
p(y | x; w) = \frac{1}{Z(x, w)} \exp F(x, y, w)
\]

with \( Z(x, w) \) the partition function

- **Maximize the likelihood** of training data via

\[
\begin{align*}
\mathbf{w}^* &= \arg \max_w \prod_{(x,y) \in D} p(y|x; w) \\
&= \arg \max_w \sum_{(x,y) \in D} \left( F(x, y, w) - \ln \sum_{\hat{y} \in Y} \exp F(x, \hat{y}, w) \right)
\end{align*}
\]

- Maximum likelihood is equivalent to maximizing cross-entropy when the target distribution \( p(x, y), \text{tg}(\hat{y}) = \delta(\hat{y} = y) \)
Learning

- Probability of a configuration $y$:

$$p(y \mid x; w) = \frac{1}{Z(x, w)} \exp F(x, y, w)$$

with $Z(x, w)$ the partition function

- Maximize the likelihood of training data via

$$w^* = \arg \max_w \prod_{(x,y) \in \mathcal{D}} p(y \mid x; w)$$

$$= \arg \max_w \sum_{(x,y) \in \mathcal{D}} \left( F(x, y, w) - \ln \sum_{\hat{y} \in \mathcal{Y}} \exp F(x, \hat{y}, w) \right)$$

- Maximum likelihood is equivalent to maximizing cross-entropy when the target distribution $p_{(x,y),tg}(\hat{y}) = \delta(\hat{y} = y)$
Program of interest:

\[
\max_w \sum_{(x,y) \in \mathcal{D}, \hat{y}} p(x,y), \text{tg} (\hat{y}) \ln p(\hat{y} \mid x; w)
\]

Optimize via gradient ascent

\[
\frac{\partial}{\partial w} \sum_{(x,y) \in \mathcal{D}, \hat{y}} p(x,y), \text{tg} (\hat{y}) \ln p(\hat{y} \mid x; w)
\]

\[
= \sum_{(x,y) \in \mathcal{D}} \left( \mathbb{E}_{p(x,y), \text{tg}} \left[ \frac{\partial}{\partial w} F(\hat{y}, x, w) \right] - \mathbb{E}_{p(x,y)} \left[ \frac{\partial}{\partial w} F(\hat{y}, x, w) \right] \right)
\]

- Compute predicted distribution \( p(\hat{y} \mid x; w) \)
- Use chain rule to pass back difference between prediction and observation
Repeat until stopping criteria

1. Forward pass to compute $F(y, x, w)$
2. Compute $p(y \mid x, w)$
3. Backward pass via chain rule to obtain gradient
4. Update parameters $w$
Deep Structured Learning (algo 1)

Repeat until stopping criteria

1. Forward pass to compute $F(y, x, w)$
2. Compute $p(y \mid x, w)$
3. Backward pass via chain rule to obtain gradient
4. Update parameters $w$

What is the PROBLEM?
Repeat until stopping criteria

1. Forward pass to compute $F(y, x, w)$
2. Compute $p(y | x, w)$
3. Backward pass via chain rule to obtain gradient
4. Update parameters $w$

What is the PROBLEM?

- How do we even represent $F(y, x, w)$ if $Y$ is large?
- How do we compute $p(y | x, w)$?
1. Use the graphical model $F(y, x, w) = \sum_r f_r(y_r, x, w)$

$$\frac{\partial}{\partial w} \sum_{(x,y) \in D, \hat{y}} p(x,y),tg(\hat{y}) \ln p(\hat{y} | x; w)$$

$$= \sum_{(x,y) \in D, r} \left( \mathbb{E}_{p(x,y),r,tg} \left[ \frac{\partial}{\partial w} f_r(\hat{y}_r, x, w) \right] - \mathbb{E}_{p(x,y),r} \left[ \frac{\partial}{\partial w} f_r(\hat{y}_r, x, w) \right] \right)$$
1. Use the graphical model \( F(y, x, w) = \sum_r f_r(y_r, x, w) \)

\[
\frac{\partial}{\partial w} \sum_{(x,y) \in \mathcal{D}, \hat{y}} p(x,y),tg(\hat{y}) \ln p(\hat{y} | x; w) \\
= \sum_{(x,y) \in \mathcal{D}, r} \left( \mathbb{E}_{p(x,y),r,tg} \left[ \frac{\partial}{\partial w} f_r(\hat{y}_r, x, w) \right] - \mathbb{E}_{p(x,y),r} \left[ \frac{\partial}{\partial w} f_r(\hat{y}_r, x, w) \right] \right)
\]

2. Approximate marginals \( p_r(\hat{y}_r | x, w) \) via beliefs \( b_r(\hat{y}_r | x, w) \) computed e.g. by:
   - Sampling methods
   - Variational methods
Repeat until stopping criteria

1. Forward pass to compute the $f_r(y_r, x, w)$
2. Compute the $b_r(y_r \mid x, w)$ by running approximated inference
3. Backward pass via chain rule to obtain gradient
4. Update parameters $w$
Repeat until stopping criteria

1. Forward pass to compute the $f_r(y_r, x, w)$
2. Compute the $b_r(y_r \mid x, w)$ by running approximated inference
3. Backward pass via chain rule to obtain gradient
4. Update parameters $w$

PROBLEM: We have to run inference in the graphical model every time we want to update the weights
Dealing with large number $|\mathcal{D}|$ of training examples:

- Parallelized across samples (any number of machines and GPUs)
- Usage of mini batches
How to deal with Big Data

Dealing with large number $|\mathcal{D}|$ of training examples:

- Parallelized across samples (any number of machines and GPUs)
- Usage of mini batches

Dealing with large output spaces $\mathcal{Y}$:

- Variational approximations treated as RNN (use GPU!)
- Blending of learning and inference
Sample parallel implementation:

<table>
<thead>
<tr>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Each compute node uses GPU for CNN Forward pass to compute $f_r(y_r, x, w)$</td>
</tr>
<tr>
<td>2. Each compute node estimates beliefs $b_r(y_r</td>
</tr>
<tr>
<td>3. Backpropagation of difference using GPU to obtain machine local gradient</td>
</tr>
<tr>
<td>4. Synchronize gradient across all machines using MPI</td>
</tr>
<tr>
<td>5. Update parameters $w$</td>
</tr>
</tbody>
</table>
Better Option: Interleaving Learning and Inference

- **Learning objective**

\[
\min_w \sum_{(x, y) \in D} (\ln Z(x, w) - F(x, y; w))
\]
Better Option: Interleaving Learning and Inference

- Learning objective

\[
\min_w \sum_{(x,y) \in D} (\ln Z(x,w) - F(x,y;w))
\]

- Use LP relaxation instead

\[
\min_w \sum_{(x,y) \in D} \left( \max_{b(x,y) \in C(x,y)} \left\{ \sum_{r,\hat{y}_r} b(x,y)_r \hat{y}_r f_r(x,\hat{y}_r;w) + \sum_r c_r H(b(x,y)_r) \right\} - F(x,y;w) \right)
\]

LP relaxation
Better Option: Interleaving Learning and Inference

- Learning objective

\[
\min_w \sum_{(x, y) \in D} (\ln Z(x, w) - F(x, y; w))
\]

- Use LP relaxation instead

\[
\min_w \sum_{(x, y) \in D} \left( \max_{b(x, y) \in C(x, y)} \left\{ \sum_{r, \hat{y}_r} b(x, y, r, \hat{y}_r) f_r(x, \hat{y}_r; w) + \sum_r c_r H(b(x, y, r)) \right\} - F(x, y; w) \right)
\]

- More efficient algorithm by blending min. w.r.t. \(w\) and max. of the beliefs \(b\) by using the dual

\[
\min_{w, \lambda} G(\lambda, w)
\]
Better Option: Interleaving Learning and Inference

- Learning objective
\[
\min_w \sum_{(x,y) \in D} (\ln Z(x,w) - F(x,y;w))
\]

- Use LP relaxation instead
\[
\min_w \sum_{(x,y) \in D} \left( \max_{b(x,y) \in C(x,y)} \left\{ \sum_{r, \hat{y}_r} b_{(x,y),r}(\hat{y}_r)f_r(x,\hat{y}_r;w) + \sum_r c_r H(b_{(x,y),r}) \right\} - F(x,y;w) \right)
\]

- LP relaxation

- More efficient algorithm by blending min. w.r.t. \(w\) and max. of the beliefs \(b\) by using the dual
\[
\min_{w,\lambda} G(\lambda, w)
\]

- After introducing Lagrange multipliers \(\lambda\), we can minimize the dual
Better Option: Interleaving Learning and Inference

- Learning objective

\[
\min_w \sum_{(x,y) \in \mathcal{D}} (\ln Z(x, w) - F(x, y; w))
\]

- Use LP relaxation instead

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

- More efficient algorithm by blending min. w.r.t. \(w\) and max. of the beliefs \(b\) by using the dual

\[
\min_{w, \lambda} G(\lambda, w)
\]

- After introducing Lagrange multipliers \(\lambda\), we can minimize the dual

- We can then do block coordinate descent to solve the minimization problem
Deep Structured Learning (algo 3)

Repeat until stopping criteria

1. Forward pass to compute the $f_r(y_r, x, w)$
2. Update (some) messages $\lambda$
3. Backward pass via chain rule to obtain gradient
4. Update parameters $w$

[Chen & Schwing & Yuille & Urtasun ICML’15]
Sample parallel implementation:

Partition data $\mathcal{D}$ onto compute nodes
Repeat until stopping criteria

1. Each compute node uses GPU for CNN Forward pass to compute $f_r(y_r, x, w)$
2. Each compute node updates (some) messages $\lambda$
3. Backpropagation of difference using GPU to obtain \textit{machine local} gradient
4. Synchronize gradient across all machines using MPI
5. Update parameters $w$
Wake Up Call!
**Application 1: Character Recognition**

- **Task:** Word Recognition from a fixed vocabulary of 50 words, 28 × 28 sized image patches
- Characters have complex backgrounds and suffer many different distortions
- Training, validation and test set sizes are 10k, 2k and 2k variations of words

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>banal</td>
<td>julep</td>
<td>resty</td>
</tr>
<tr>
<td>drein</td>
<td>yojan</td>
<td>mothy</td>
</tr>
<tr>
<td>snack</td>
<td>feize</td>
<td>porer</td>
</tr>
</tbody>
</table>
Graphical model has 5 nodes, MLP for each unary and non-parametric pairwise potentials

Joint training, structured, deep and more capacity helps

<table>
<thead>
<tr>
<th>Grap</th>
<th>MLP</th>
<th>Method</th>
<th>$H_1 = 128$</th>
<th>$H_1 = 256$</th>
<th>$H_1 = 512$</th>
<th>$H_1 = 768$</th>
<th>$H_1 = 1024$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>1lay</td>
<td>Unary only</td>
<td>8.60 / 61.32</td>
<td>10.80 / 64.41</td>
<td>12.50 / 65.69</td>
<td>12.95 / 66.66</td>
<td>13.40 / 67.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JointTrain</td>
<td>16.80 / 65.28</td>
<td>25.20 / 70.75</td>
<td><strong>31.80</strong> / 74.90</td>
<td>33.05 / 76.42</td>
<td>34.30 / 77.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PwTrain</td>
<td>12.70 / 64.35</td>
<td>18.00 / 68.27</td>
<td>22.80 / 71.29</td>
<td>23.25 / 72.62</td>
<td>26.30 / 73.96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PreTrainJoint</td>
<td><strong>20.65</strong> / 67.42</td>
<td><strong>25.70</strong> / 71.65</td>
<td>31.70 / <strong>75.56</strong></td>
<td><strong>34.50</strong> / 77.14</td>
<td><strong>35.85</strong> / 78.05</td>
</tr>
<tr>
<td>2nd</td>
<td>1lay</td>
<td>JointTrain</td>
<td>25.50 / 67.13</td>
<td>34.60 / 73.19</td>
<td>45.55 / 79.60</td>
<td><strong>51.55</strong> / 82.37</td>
<td><strong>54.05</strong> / 83.57</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PwTrain</td>
<td>10.05 / 58.90</td>
<td>14.10 / 63.44</td>
<td>18.10 / 67.31</td>
<td>20.40 / 70.14</td>
<td>22.20 / 71.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PreTrainJoint</td>
<td><strong>28.15</strong> / <strong>69.07</strong></td>
<td><strong>36.85</strong> / <strong>75.21</strong></td>
<td><strong>45.75</strong> / <strong>80.09</strong></td>
<td>50.10 / 82.30</td>
<td>52.25 / 83.39</td>
</tr>
<tr>
<td></td>
<td>2lay</td>
<td>$H_1 = 512$</td>
<td>$H_2 = 32$</td>
<td>$H_2 = 64$</td>
<td>$H_2 = 128$</td>
<td>$H_2 = 256$</td>
<td>$H_2 = 512$</td>
</tr>
<tr>
<td>1st</td>
<td></td>
<td>Unary only</td>
<td>15.25 / 69.04</td>
<td>18.15 / 70.66</td>
<td>19.00 / 71.43</td>
<td>19.20 / 72.06</td>
<td>20.40 / 72.51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JointTrain</td>
<td>35.95 / 76.92</td>
<td>43.80 / 81.64</td>
<td>44.75 / 82.22</td>
<td>46.00 / 82.96</td>
<td><strong>47.70</strong> / 83.64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PwTrain</td>
<td>34.85 / 79.11</td>
<td>38.95 / 80.93</td>
<td>42.75 / 82.38</td>
<td>45.10 / 83.67</td>
<td>45.75 / 83.88</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PreTrainJoint</td>
<td><strong>42.25</strong> / <strong>81.10</strong></td>
<td><strong>44.85</strong> / <strong>82.96</strong></td>
<td><strong>46.85</strong> / <strong>83.50</strong></td>
<td><strong>47.95</strong> / <strong>84.21</strong></td>
<td><strong>47.05</strong> / <strong>84.08</strong></td>
</tr>
<tr>
<td>2nd</td>
<td>2lay</td>
<td>JointTrain</td>
<td>54.65 / 83.98</td>
<td>61.80 / 87.30</td>
<td>66.15 / 89.09</td>
<td>64.85 / 88.93</td>
<td>68.00 / 89.96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PwTrain</td>
<td>39.95 / 81.14</td>
<td>48.25 / 84.45</td>
<td>52.65 / 86.24</td>
<td>57.10 / 87.61</td>
<td>62.90 / 89.49</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PreTrainJoint</td>
<td><strong>62.60</strong> / <strong>88.03</strong></td>
<td><strong>65.80</strong> / <strong>89.32</strong></td>
<td><strong>68.75</strong> / <strong>90.47</strong></td>
<td><strong>68.60</strong> / <strong>90.42</strong></td>
<td><strong>69.35</strong> / <strong>90.75</strong></td>
</tr>
</tbody>
</table>
Learned Weights

Unary weights

distance-1 edges

distance-2 edges
Neural Nets Modeling Pairwise

One-Layer MLP Chain

Two-Layer MLP Chain

R. Urtasun (UofT)
Example 2: Image Tagging

- Flickr dataset: 38 possible tags, $|\mathcal{Y}| = 2^{38}$
- 10k training, 10k test examples

<table>
<thead>
<tr>
<th>Training method</th>
<th>Prediction error [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary only</td>
<td>9.36</td>
</tr>
<tr>
<td>Piecewise</td>
<td>7.70</td>
</tr>
<tr>
<td>Joint (with pre-training)</td>
<td>7.25</td>
</tr>
</tbody>
</table>

R. Urtasun (UofT) Deep Structured Models
Visual results

female/indoor/portrait
female/indoor/portrait
animals/dog/indoor
animals/dog

sky/plant life/tree
sky/plant life/tree
indoor/flower/plant life

water/animals/sea
water/animals/sky
∅
Learned class correlations

<table>
<thead>
<tr>
<th></th>
<th>female</th>
<th>people</th>
<th>indoor</th>
<th>portrait</th>
<th>sky</th>
<th>plant life</th>
<th>male</th>
<th>clouds</th>
<th>tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>female</td>
<td>0.00</td>
<td>0.68</td>
<td>0.04</td>
<td>0.24</td>
<td>-0.01</td>
<td>-0.05</td>
<td>0.07</td>
<td>-0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>people</td>
<td>0.68</td>
<td>0.00</td>
<td>0.06</td>
<td>0.36</td>
<td>-0.05</td>
<td>-0.12</td>
<td>0.74</td>
<td>-0.04</td>
<td>-0.03</td>
</tr>
<tr>
<td>indoor</td>
<td>0.04</td>
<td>0.06</td>
<td>0.00</td>
<td>0.07</td>
<td>-0.35</td>
<td>-0.34</td>
<td>0.02</td>
<td>-0.15</td>
<td>-0.21</td>
</tr>
<tr>
<td>portrait</td>
<td>0.24</td>
<td>0.36</td>
<td>0.07</td>
<td>0.00</td>
<td>-0.02</td>
<td>-0.01</td>
<td>0.12</td>
<td>0.02</td>
<td>0.05</td>
</tr>
<tr>
<td>sky</td>
<td>-0.01</td>
<td>-0.05</td>
<td>-0.35</td>
<td>-0.02</td>
<td>0.00</td>
<td>0.24</td>
<td>-0.00</td>
<td>0.44</td>
<td>0.30</td>
</tr>
<tr>
<td>plant life</td>
<td>-0.05</td>
<td>-0.12</td>
<td>-0.34</td>
<td>-0.01</td>
<td>0.24</td>
<td>0.00</td>
<td>-0.07</td>
<td>0.09</td>
<td>0.68</td>
</tr>
<tr>
<td>male</td>
<td>0.07</td>
<td>0.74</td>
<td>0.02</td>
<td>0.12</td>
<td>-0.00</td>
<td>-0.07</td>
<td>0.00</td>
<td>0.00</td>
<td>-0.02</td>
</tr>
<tr>
<td>clouds</td>
<td>-0.01</td>
<td>-0.04</td>
<td>-0.15</td>
<td>0.02</td>
<td>0.44</td>
<td>0.09</td>
<td>0.00</td>
<td>0.00</td>
<td>0.11</td>
</tr>
<tr>
<td>tree</td>
<td>0.01</td>
<td>-0.03</td>
<td>-0.21</td>
<td>0.05</td>
<td>0.30</td>
<td>0.68</td>
<td>-0.02</td>
<td>0.11</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Only part of the correlations are shown for clarity.
Example 3: Semantic Segmentation

- $|\mathcal{Y}| = 21^{350\cdot500}$, $\approx 10k$ training, $\approx 1500$ test examples
- Oxford-net pre trained on PASCAL, predicts $40 \times 40$ + upsampling
- The graphical model is a fully connected CRF with Gaussian potentials
- Inference using (algo2), with mean-field as approx. inference
Pascal VOC 2012 dataset

- $|\mathcal{Y}| = 21^{350\cdot500}$, $\approx 10k$ training, $\approx 1500$ test examples
- Oxford-net pre trained on PASCAL, predicts $40 \times 40 +$ upsampling
- The graphical model is a fully connected CRF with Gaussian potentials
- Inference using (algo2), with mean-field as approx. inference

<table>
<thead>
<tr>
<th>Training method</th>
<th>Mean IoU [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary only</td>
<td>61.476</td>
</tr>
<tr>
<td>Joint</td>
<td>64.060</td>
</tr>
</tbody>
</table>
Pascal VOC 2012 dataset

- $|\mathcal{Y}| = 21^{350 \cdot 500}$, $\approx 10k$ training, $\approx 1500$ test examples
- Oxford-net pre trained on PASCAL, predicts $40 \times 40 +$ upsampling
- The graphical model is a fully connected CRF with Gaussian potentials
- Inference using (algo2), with mean-field as approx. inference

<table>
<thead>
<tr>
<th>Training method</th>
<th>Mean IoU [%]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary only</td>
<td>61.476</td>
</tr>
<tr>
<td>Joint</td>
<td>64.060</td>
</tr>
</tbody>
</table>

- **Disclaimer**: Much better results by Zheng et al. 15 is now at 74.7%!
Example 4: More Precise Grouping

- Given a single image, we want to infer **Instance-level Segmentation** and **Depth Ordering**.

- Use deep convolutional nets to do both tasks simultaneously.
  - Trick: Encode both tasks with a single parameterization.
  - Run the conv. net at multiple resolutions.
  - Use MRF to form a single coherent explanation across all the image combining the conv nets at multiple resolutions.

- **Important**: we do not use a single pixel-wise training example!
Results on KITTI

[Z. Zhang, S. Fidler and R. Urtasun, CVPR’16]
Example 5: Enhancing freely-available maps

- Fine-grained categorization

(a) Intersection with tram line

(b) Small town

(c) A road with three lanes

(d) Two roads with tram stop in between
Some Previous Work

- Use the hinge loss to optimize the unaries only which are neural nets (Li and Zemel 14). Correlations between variables are not used for learning.
- If inference is tractable, **Conditional Neural Fields** (Peng et al. 09) use back-propagation on the log-loss.
- **Decision Tree Fields** (Nowozin et al. 11), use complex region potentials (decision trees), but given the tree, it is still linear in the parameters.
- **Restricted Bolzmann Machines** (RBMs): Generative model that has a very particular architecture so that inference is tractable via sampling (Salakhutdinov 07). Problems with partition function.
- (Domke 13) treat the problem as learning a set of logistic regressors.
- **Fields of experts** (Roth et al. 05), not deep, use CD training.
- Many ideas go back to (Boutou 91).
- Very popular these days.
Structure in the Loss
To train networks we minimize the loss function w.r.t. the parameters

\[ w^* = \arg \min_w \mathbb{E}[\ell_{\text{task}}(y, y_w)] , \]

where \( \mathbb{E}[\cdot] \) denotes an expectation taken over the given dataset, \( y \) the ground truth and \( y_w \) the prediction of the model.
To train networks we minimize the loss function w.r.t. the parameters

\[ w^* = \arg \min_w \mathbb{E}[\ell_{task}(y, y_w)], \]

where \( \mathbb{E}[\cdot] \) denotes an expectation taken over the given dataset, \( y \) the ground truth and \( y_w \) the prediction of the model.

Supervised learning algorithms involve computing the gradient of the loss function wrt parameters of the model \( w \).
To train networks we minimize the loss function w.r.t. the parameters

$$w^* = \arg\min_w \mathbb{E}[\ell_{\text{task}}(y, y_w)],$$

where $\mathbb{E}[:]$ denotes an expectation taken over the given dataset, $y$ the ground truth and $y_w$ the prediction of the model.

Supervised learning algorithms involve computing the gradient of the loss function wrt parameters of the model $w$.

Thus they require the loss function to be differentiable.
To train networks we minimize the loss function w.r.t. the parameters

$$w^* = \arg \min_w \mathbb{E}[\ell_{task}(y, y_w)],$$

where $\mathbb{E} [\cdot]$ denotes an expectation taken over the given dataset, $y$ the ground truth and $y_w$ the prediction of the model.

- Supervised learning algorithms involve computing the gradient of the loss function wrt parameters of the model $w$.
- Thus they require the loss function to be differentiable.
- Many loss functions are non-differentiable with respect to the output of the network and non-decomposable, e.g., Average precision (AP), intersection over union (IOU).
Approximate with a surrogate loss functions that is differentiable, e.g., cross-entropy, log-likelihood, (structured) hinge loss

\[ \mathbb{E} [\ell_{\text{task}}(y, y_w)] \approx \mathbb{E} [\ell(y, y_w)], \]

where \( \mathbb{E} [\cdot] \) denotes an expectation taken over the given dataset.
Approximate with a **surrogate loss functions** that is differentiable, e.g., cross-entropy, log-likelihood, (structured) hinge loss

\[ \mathbb{E} [\ell_{\text{task}}(y, y_w)] \approx \mathbb{E} [\bar{\ell}(y, y_w)], \]

where \( \mathbb{E} [\cdot] \) denotes an expectation taken over the given dataset

- **Structured SVMs** minimize an upper bound on the task loss
  - upper bound is not always very tight
  - particularly in the presence of **noise**
Optimizing the Task Loss

- Approximate with a **surrogate loss functions** that is differentiable, e.g., cross-entropy, log-likelihood, (structured) hinge loss

\[
\mathbb{E}[\ell_{\text{task}}(y, y_w)] \approx \mathbb{E}[\ell(y, y_w)],
\]

where \(\mathbb{E}[\cdot]\) denotes an expectation taken over the given dataset

- **Structured SVMs** minimize an upper bound on the task loss
  - upper bound is not always very tight
  - particularly in the presence of **noise**

- **Cross entropy** is agnostic to the task loss
Approximate with a **surrogate loss functions** that is differentiable, e.g., cross-entropy, log-likelihood, (structured) hinge loss

\[ \mathbb{E} [\ell_{task}(y, y_w)] \approx \mathbb{E} [\bar{\ell}(y, y_w)] , \]

where \( \mathbb{E} [\cdot] \) denotes an expectation taken over the given dataset

**Structured SVMs** minimize an upper bound on the task loss

- upper bound is not always very tight
- particularly in the presence of noise

**Cross entropy** is agnostic to the task loss

How can we derive learning algorithms that **directly minimize** the loss that we care about for our application?
Direct Loss Minimization

Under some mild regularity conditions the direct loss gradient is:

$$\nabla_w \mathbb{E} [\ell_{task}(y, y_w)] = \pm \lim_{\epsilon \to 0} \frac{1}{\epsilon} \mathbb{E} [\nabla_w F(x, y_{direct}, w) - \nabla_w F(x, y_w, w)],$$

with

$$y_w = \arg \max_{\hat{y} \in \mathcal{Y}} F(x, \hat{y}, w),$$

$$y_{direct} = \arg \max_{\hat{y} \in \mathcal{Y}} F(x, \hat{y}, w) \pm \epsilon \ell_{task}(y, \hat{y}).$$
Under some mild regularity conditions the direct loss gradient is:

$$\nabla_w \mathbb{E}[\ell_{\text{task}}(y, y_w)] = \pm \lim_{\epsilon \to 0} \frac{1}{\epsilon} \mathbb{E} [\nabla_w F(x, y_{\text{direct}}, w) - \nabla_w F(x, y_w, w)] ,$$

with

$$y_w = \arg \max_{\hat{y} \in \mathcal{Y}} F(x, \hat{y}, w),$$

$$y_{\text{direct}} = \arg \max_{\hat{y} \in \mathcal{Y}} F(x, \hat{y}, w) \pm \epsilon \ell_{\text{task}}(y, \hat{y}).$$

- $y_w$: standard inference task
Direct Loss Minimization

- Under some mild regularity conditions the direct loss gradient is:

\[
\nabla_w \mathbb{E} [\ell_{task}(y, y_w)] = \pm \lim_{\epsilon \to 0} \frac{1}{\epsilon} \mathbb{E} [\nabla_w F(x, y_{direct}, w) - \nabla_w F(x, y_w, w)],
\]

with

\[
y_w = \arg \max_{\hat{y} \in \mathcal{Y}} F(x, \hat{y}, w),
\]

\[
y_{direct} = \arg \max_{\hat{y} \in \mathcal{Y}} F(x, \hat{y}, w) \pm \epsilon \ell_{task}(y, \hat{y}).
\]

- \(y_w\): standard inference task
- \(y_{direct}\): prediction of a scoring function perturbed by the task loss \(\ell_{task}(y, \hat{y})\) This is loss augmented inference; can be non-trivial to solve
Direct Loss Minimization

- Under some mild regularity conditions the direct loss gradient is:

\[
\nabla_w \mathbb{E} \left[ \ell_{\text{task}}(y, y_w) \right] = \pm \lim_{\epsilon \to 0} \frac{1}{\epsilon} \mathbb{E} \left[ \nabla_w F(x, y_{\text{direct}}, w) - \nabla_w F(x, y_w, w) \right],
\]

with

\[
y_w = \arg \max_{\hat{y} \in \mathcal{Y}} F(x, \hat{y}, w),
\]

\[
y_{\text{direct}} = \arg \max_{\hat{y} \in \mathcal{Y}} F(x, \hat{y}, w) \pm \epsilon \ell_{\text{task}}(y, \hat{y}).
\]

- \(y_w\): standard inference task
- \(y_{\text{direct}}\): prediction of a scoring function perturbed by the task loss \(\ell_{\text{task}}(y, \hat{y})\) This is loss augmented inference; can be non-trivial to solve

- Similarity to hinge loss, difference is that the second term does not have the ground truth, but the prediction instead, and we have negative update
Direct Loss Minimization

Under some mild regularity conditions the direct loss gradient is:

$$\nabla_w \mathbb{E}[\ell_{task}(y, y_w)] = \pm \lim_{\epsilon \to 0} \frac{1}{\epsilon} \mathbb{E}[\nabla_w F(x, y_{direct}, w) - \nabla_w F(x, y_w, w)],$$

with

$$y_w = \arg \max_{\hat{y} \in Y} F(x, \hat{y}, w),$$

$$y_{direct} = \arg \max_{\hat{y} \in Y} F(x, \hat{y}, w) \pm \epsilon \ell_{task}(y, \hat{y}).$$

- $y_w$: standard inference task
- $y_{direct}$: prediction of a scoring function perturbed by the task loss $\ell_{task}(y, \hat{y})$ This is loss augmented inference; can be non-trivial to solve

Similarity to hinge loss, difference is that the second term does not have the ground truth, but the prediction instead, and we have negative update

Extension of [McAllester 10] from linear to non-linear functions
Direct Loss Minimization

- Under some mild regularity conditions the direct loss gradient is:

\[
\nabla_w \mathbb{E} [\ell_{\text{task}}(y, y_w)] = \pm \lim_{\epsilon \to 0} \frac{1}{\epsilon} \mathbb{E} [\nabla_w F(x, y_{\text{direct}}, w) - \nabla_w F(x, y_w, w)],
\]

with

\[
y_w = \arg \max_{\hat{y} \in \mathcal{Y}} F(x, \hat{y}, w),
\]

\[
y_{\text{direct}} = \arg \max_{\hat{y} \in \mathcal{Y}} F(x, \hat{y}, w) \pm \epsilon \ell_{\text{task}}(y, \hat{y}).
\]

- \(y_w\): standard inference task
- \(y_{\text{direct}}\): prediction of a scoring function perturbed by the task loss \(\ell_{\text{task}}(y, \hat{y})\) This is loss augmented inference; can be non-trivial to solve

- Similarity to hinge loss, difference is that the second term does not have the ground truth, but the prediction instead, and we have negative update

- Extension of [McAllester 10] from linear to non-linear functions

- Allow us to train neural nets with arbitrarily complex loss functions
Algorithm: Direct Loss Minimization for Deep Networks
Repeat until stopping criteria

1. Forward pass to compute $F(x, \hat{y}; w)$
2. Obtain $y_w$ and $y_{\text{direct}}$ via inference and loss-augmented inference
3. Single backward pass via chain rule to obtain gradient
   \[
   \nabla_w \mathbb{E} [\ell_{\text{task}}(y, y_w)] = \pm \lim_{\epsilon \to 0} \frac{1}{\epsilon} \mathbb{E} [\nabla_w F(x, y_{\text{direct}}, w) - \nabla_w F(x, y_w, w)]
   \]
4. Update parameters using stepsize $\eta$:
   \[
   w \leftarrow w - \eta \nabla_w \mathbb{E} [L(y, y_w)]
   \]

Figure: Our algorithm for direct loss minimization.
Average precision is **non-decomposable** and **non-differentiable**

- $y_w$ is sorting, and new **dynamic programing algorithm** for $y_{direct}$
Average Precision for Ranking

[Y. Song, A. Schwing, R. Zemel and R. Urtasun, ICML’16]

- Average precision is **non-decomposable** and **non-differentiable**
- \( y_w \) is sorting, and new **dynamic programing algorithm** for \( y_{direct} \)
- Summary of synthetic experiments:
  - **Known network architecture**: direct loss minimization much better
Average precision is non-decomposable and non-differentiable.

\( y_w \) is sorting, and new dynamic programing algorithm for \( y_{\text{direct}} \).

Summary of synthetic experiments:

- Known network architecture: direct loss minimization much better
- Unknown network architecture: not so clear winner.
Average precision is non-decomposable and non-differentiable

\( y_w \) is sorting, and new dynamic programing algorithm for \( y_{direct} \)

Summary of synthetic experiments:

- **Known network architecture**: direct loss minimization much better
- **Unknown network architecture**: not so clear winner.
- Noise free case: model error dominates loss error
Average precision is non-decomposable and non-differentiable

- $y_w$ is sorting, and new **dynamic programing algorithm** for $y_{direct}$

Summary of synthetic experiments:

- **Known network architecture**: direct loss minimization much better
- **Unknown network architecture**: not so clear winner.
- Noise free case: **model error dominates loss error**
- This is different when having **label noise**
Full batch training of AlexNet (10 classes, 6278 examples)

Direct loss much more robust to label noise (i.e., flips)
R-CNN [Girshick 14] with AlexNet fine-tuned for our task

- We use the **AP on each mini-batch** to approximate the overall AP

- A batch size of 512 balances computational complexity and performance; larger batch size (such as 2048) generally results in better performance

<table>
<thead>
<tr>
<th></th>
<th>aeroplane</th>
<th>bicycle</th>
<th>bird</th>
<th>boat</th>
<th>bottle</th>
<th>bus</th>
<th>car</th>
<th>cat</th>
<th>chair</th>
<th>cow</th>
<th>dog</th>
<th>horse</th>
<th>motorbike</th>
<th>person</th>
<th>pottedplant</th>
<th>sheep</th>
<th>sofa</th>
<th>train</th>
<th>tvmonitor</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>x-ent</strong></td>
<td>63.8</td>
<td>61.0</td>
<td>42.6</td>
<td>30.7</td>
<td>23.5</td>
<td>63.2</td>
<td>51.7</td>
<td>58.5</td>
<td>20.1</td>
<td>37.0</td>
<td>32.0</td>
<td>52.8</td>
<td>50.8</td>
<td>62.5</td>
<td>50.1</td>
<td>23.5</td>
<td>48.3</td>
<td>33.1</td>
<td>48.5</td>
<td>57.4</td>
</tr>
<tr>
<td><strong>hinge-AP</strong></td>
<td>67.5</td>
<td>60.6</td>
<td>43.6</td>
<td>30.8</td>
<td>25.3</td>
<td>64.5</td>
<td>54.9</td>
<td>64.4</td>
<td>21.9</td>
<td>34.5</td>
<td>34.2</td>
<td>57.0</td>
<td>48.8</td>
<td>63.9</td>
<td>56.3</td>
<td>25.1</td>
<td>49.6</td>
<td>37.4</td>
<td>54.3</td>
<td>57.3</td>
</tr>
<tr>
<td><strong>pos-AP</strong></td>
<td>65.1</td>
<td>59.8</td>
<td>43.7</td>
<td>31.4</td>
<td>27.7</td>
<td>64.6</td>
<td>53.1</td>
<td>63.7</td>
<td>25.6</td>
<td>40.2</td>
<td>36.2</td>
<td>58.1</td>
<td>52.8</td>
<td>63.6</td>
<td>56.2</td>
<td>28.1</td>
<td>50.0</td>
<td>38.9</td>
<td>61.3</td>
<td>48.5</td>
</tr>
</tbody>
</table>

---

[Y. Song, A. Schwing, R. Zemel and R. Urtasun, ICML'16]
PASCAL VOC2012 Object Detection

[Y. Song, A. Schwing, R. Zemel and R. Urtasun, ICML'16]

<table>
<thead>
<tr>
<th></th>
<th>aeroplane</th>
<th>bicycle</th>
<th>bird</th>
<th>boat</th>
<th>bottle</th>
<th>bus</th>
<th>car</th>
<th>cat</th>
<th>chair</th>
<th>cow</th>
<th>dog</th>
<th>horse</th>
<th>motorbike</th>
<th>person</th>
<th>pottedplant</th>
<th>sheep</th>
<th>sofa</th>
<th>train</th>
<th>tvmonitor</th>
<th>mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-ent</td>
<td>63.8</td>
<td>42.6</td>
<td>30.7</td>
<td>23.5</td>
<td>63.2</td>
<td>51.7</td>
<td>58.5</td>
<td>20.1</td>
<td>37.0</td>
<td>32.0</td>
<td>52.8</td>
<td>50.8</td>
<td>62.5</td>
<td>50.1</td>
<td>23.5</td>
<td>48.3</td>
<td>33.1</td>
<td>48.5</td>
<td>57.4</td>
<td>45.6</td>
</tr>
<tr>
<td>hinge-AP</td>
<td>67.5</td>
<td>60.6</td>
<td>43.6</td>
<td>30.8</td>
<td>25.3</td>
<td>64.5</td>
<td>54.9</td>
<td>64.4</td>
<td>21.9</td>
<td>34.5</td>
<td>34.2</td>
<td>57.0</td>
<td>48.8</td>
<td>63.9</td>
<td>56.3</td>
<td>49.6</td>
<td>37.4</td>
<td>54.3</td>
<td>57.3</td>
<td>47.6</td>
</tr>
<tr>
<td>pos-AP</td>
<td>65.1</td>
<td>59.8</td>
<td>43.7</td>
<td>31.4</td>
<td>27.7</td>
<td>64.6</td>
<td>53.1</td>
<td>63.7</td>
<td>25.6</td>
<td>40.2</td>
<td>36.2</td>
<td>58.1</td>
<td>52.8</td>
<td>63.6</td>
<td>56.2</td>
<td>50.0</td>
<td>38.9</td>
<td>61.3</td>
<td>48.5</td>
<td></td>
</tr>
</tbody>
</table>

- R-CNN [Girshick 14] with AlexNet fine-tuned for our task
- We use the AP on each mini-batch to approximate the overall AP
- A batch size of 512 balances computational complexity and performance; larger batch size (such as 2048) generally results in better performance
- With 20% label noise, performance is 0% for x-ent/hinge and 20% for us
Structure in the Embedding
Deep learning has been very popular to learn embeddings

- sentence embeddings
- image embeddings
- multi-modal embeddings (text+images)
Deep Embeddings

- Deep learning has been very popular to learn embeddings
  - sentence embeddings
  - image embeddings
  - multi-modal embeddings (text+images)
- Oftentimes we learned them in a supervised fashion to solve a specific task
Deep Embeddings

- Deep learning has been very popular to learn embeddings
  - sentence embeddings
  - image embeddings
  - multi-modal embeddings (text + images)
- Oftentimes we learned them in a supervised fashion to solve a specific task
- These embeddings have been learned in an unsupervised fashion
  - to reconstruct the input
  - to predict the context around them
Deep Embeddings

- Deep learning has been very popular to learn embeddings
  - sentence embeddings
  - image embeddings
  - multi-modal embeddings (text+images)

- Oftentimes we learned them in a supervised fashion to solve a specific task

- These embeddings have been learned in an unsupervised fashion
  - to reconstruct the input
  - to predict the context around them

- Sometimes we have access to prior knowledge that we would like to exploit in order to learn better embeddings
Visual-semantic hierarchy

- Partial order over images and language
  - hypernym relation between words
  - textual entailment among phrases
  - captions are simply abstractions of images

![Diagram showing the partial order between words and images](image_url)
Visual-semantic hierarchy

- Partial order over images and language
  - hypernym relation between words
  - textual entailment among phrases
  - captions are simply abstractions of images

- Create order-embeddings that respect this partial order (i.e., abstraction)
Reversed product order

- Reversed product order on $\mathbb{R}_+^N$,

$$ x \preceq y \text{ if and only if } \bigwedge_{i=1}^{N} x_i \geq y_i $$

for all vectors $x, y$ with nonnegative coordinates

- Reverse direction: smaller coordinates imply higher position in partial order

- The origin is the top element of the order, i.e., most general concept
Order Embeddings

- Imposing the order as a hard constraint is too restrictive.
- Instead, use a soft-loss that measures the degree of violation. For an ordered pair \((x, y)\) of points in \(\mathbb{R}^N_+\), we define

\[
E(x, y) = \| \max(0, y - x) \|^2
\]

\(E(x, y) = 0 \iff x \preceq y\) according to the reversed product order; if the order is not satisfied, \(E(x, y)\) is positive.
Toy 2D Example on Wordnet Hypernym Prediction

- **Hypernym:** first concept is abstraction of second, e.g., (women, person)
- Max-margin loss with random negative pairs

\[
\sum_{(u,v) \in \text{WordNet}} E(f(u), f(v)) + \max\{0, \alpha - E(f(u'), f(v'))\}
\]

**Figure:** 2D order-embeddings on Wordnet subset. True (green), bad (pink)
Quantitative Analysis: 50D embedding

- **Transitive closure**: classifies hypernyms pairs as positive if they are in the transitive closure of the union of edges in the training and validation sets.

- **Word2gauss**: baseline evaluates the approach of [Vilnis & McAllum 15] to represent words as Gaussian densities rather than points. This allows a natural representation of hierarchies using the KL divergence.

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>transitive closure</td>
<td>88.2</td>
</tr>
<tr>
<td>word2gauss</td>
<td>86.6</td>
</tr>
<tr>
<td>order-embeddings (symmetric)</td>
<td>84.2</td>
</tr>
<tr>
<td>order-embeddings (bilinear)</td>
<td>86.3</td>
</tr>
<tr>
<td>order-embeddings</td>
<td><strong>90.6</strong></td>
</tr>
</tbody>
</table>
Image-Caption Retrieval

- Microsoft COCO: training (113,287 images), validation (5000 images), and test (5000 images).

- Standard loss (but with our asymmetric score) that encourages \( S(c, i) \) for ground truth caption-image pairs to be greater than all other pairs:

\[
\sum_{(c,i)} \left( \sum_{c'} \max\{0, \alpha - S(c, i) + S(c', i)\} + \sum_{i'} \max\{0, \alpha - S(c, i) + S(c, i')\} \right)
\]

where \((c, i)\) is a ground truth caption-image pair, \(c'\) goes over captions that no describe \(i\), and \(i'\) goes over image not described by \(c\).

- Use our order-violation penalty \( E \)

\[
S(c, i) = -E(f_c(c), f_i(i))
\]

with \( E \) our order-violation penalty and \( f_c, f_i \) are embedding functions from captions and images into \( \mathbb{R}_{+}^N \).

- \( f_c \) and \( f_i \) are VGG and GRU, but use absolute value to ensure positiveness
### COCO Caption Retrieval

[I. Vendrov, S. Fidler and R. Urtasun, ICLR’16]

<table>
<thead>
<tr>
<th>Model</th>
<th>Caption Retrieval</th>
<th>1k Test Images</th>
<th>5k Test Images</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R@1</td>
<td>R@10</td>
<td>Med</td>
</tr>
<tr>
<td>MNLM [Kiros 14]</td>
<td>43.4</td>
<td>85.8</td>
<td>2</td>
</tr>
<tr>
<td>m-RNN [Mao 15]</td>
<td>41.0</td>
<td>83.5</td>
<td>2</td>
</tr>
<tr>
<td>DVSA [Karpathy 15]</td>
<td>38.4</td>
<td>80.5</td>
<td>1</td>
</tr>
<tr>
<td>STV [Kiros 15]</td>
<td>33.8</td>
<td>82.1</td>
<td>3</td>
</tr>
<tr>
<td>FV [Klein 15]</td>
<td>39.4</td>
<td>80.9</td>
<td>2</td>
</tr>
<tr>
<td>m-CNN [Ma 15]</td>
<td>38.3</td>
<td>81.0</td>
<td>2</td>
</tr>
<tr>
<td>m-CNN&lt;sub&gt;ENS&lt;/sub&gt;</td>
<td>42.8</td>
<td>84.1</td>
<td>2</td>
</tr>
<tr>
<td>order-embed (reversed)</td>
<td>11.2</td>
<td>44.0</td>
<td>14.2</td>
</tr>
<tr>
<td>order-embed (1-crop)</td>
<td>41.4</td>
<td>84.2</td>
<td>2.0</td>
</tr>
<tr>
<td>order-embed (symm.)</td>
<td>45.8</td>
<td>88.2</td>
<td>2.0</td>
</tr>
<tr>
<td>order-embed</td>
<td>46.7</td>
<td>88.9</td>
<td>2.0</td>
</tr>
<tr>
<td>DVSA</td>
<td>11.8</td>
<td>45.4</td>
<td>12.2</td>
</tr>
<tr>
<td>FV</td>
<td>17.3</td>
<td>50.2</td>
<td>10.0</td>
</tr>
<tr>
<td>order-embed (symm.)</td>
<td>22.5</td>
<td>63.2</td>
<td>6.0</td>
</tr>
<tr>
<td>order-embed</td>
<td>23.3</td>
<td>65.0</td>
<td>5.0</td>
</tr>
</tbody>
</table>
Figure: Multimodal regularities: Elementwise max of two vectors gives their greatest common descendant, and min gives their lowest common ancestor.
Conclusions and Future Work

Deep Structured Models:

- Structure in the Output
- Structure in the Loss
- Structure in the Embedding

To appear at NIPS:
Continuous-valued deep Structured Models

Future work:
- Learning deep structured models with latent variables
- Learning deep structured models with asynchronous updates
- Direct loss minimization for other loss-functions: e.g., IoU.
- Other orderings and/or constraints in the embeddings
- Many many many more applications
Conclusions and Future Work

Deep Structured Models:

- Structure in the Output
- Structure in the Loss
- Structure in the Embedding

To appear at NIPS:

- Continuous-valued deep Structured Models
Conclusions and Future Work

Deep Structured Models:

- Structure in the Output
- Structure in the Loss
- Structure in the Embedding

To appear at NIPS:

- Continuous-valued deep Structured Models

Future work:

- Learning deep structured models with latent variables
- Learning deep structured models with asynchronous updates
- Direct loss minimization for other loss-functions: e.g., IoU.
- Other orderings and/or constraints in the embeddings
- Many many many more applications
Acknowledgements

Team behind what I showed you today:

- Liang-Chieh Chen (visiting PhD student)
- Sanja Fidler (faculty)
- Gellert Mattyus (visiting PhD student)
- Alex Schwing (postdoc)
- Yang Song (Undergrad student)
- Ivan Vendrov (Master student)
- Shenlong Wang (PhD student)
- Richard Zemel (faculty)
- Allan Yuille (faculty)
- Ziyu Zhang (Master student)