Learning Deep Structured Models


Discussion by R. Henao

Duke University

May 27, 2016
**Motivation**

**Problem:**
- Predicting statistically related random variables.
- Developing scalable algorithms that learn high-order knowledge taking into account output dependencies.
- Existing algorithms use a two-stage process.

**Motivation:**
- Developing scalable algorithms that learn high-order knowledge (deep learning) taking into account output dependencies.
- Existing algorithms use a two-stage process.
- Features from these approaches are suboptimal.

**Contributions:**
- Extending deep learning algorithms to allow for complex representations that account for dependencies between outputs.
- Features of the deep model form potentials of a Markov random field (MRF).
- The approach is efficient: joint learning/inference, single loop and GPUs.
- Joint learning of features and MRF parameters yields big performance gains.
PROBLEM SETTING

Assume we have data:

- Outputs: $y \in \mathcal{Y}, \ y = (y_1, \ldots, y_N), \ \mathcal{Y} = \prod_{i=1}^{N} \mathcal{Y}_i$ and $\mathcal{Y}_i = \{1, \ldots, |\mathcal{Y}_i|\}$.
- Inputs: $x \in \mathcal{X}$
- Parameters: $w \in \mathbb{R}^A$
- Function: $F(x, y; w) : \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^A \rightarrow \mathbb{R}$.

Inference amounts to find the highest scoring function

$$y^* = \arg\max_y F(x, y; w), \quad (1)$$

where $F(\cdot)$ is a deep network.

Inference: forward pass.
Learning: backward pass.

Simple case: $N$ disconnected discrete outputs (unary potentials).
General case: search over $\prod_{i=1}^{N} |\mathcal{Y}_i|$ is NP-hard.
Special case: log-linear model, $F(x, y; w) = w^\top \phi(x, y)$ with features $\phi(x, y)$. 

Duke University
**Problem Setting**

In the paper, $F(x, y; w)$ is an arbitrary function of $w$ and $(x, y)$.

Let the probability of a configuration $\hat{y}$ be

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

where:

- $Z_\epsilon(x, w) = \sum_{\hat{y} \in Y} \exp\{F(x, \hat{y}; w)\}^{1/\epsilon}$ is the partition function.
- $p(x, y) \in \Delta$, the simplex.
- $\epsilon \geq 0$, the annealing parameter.
LEARNING VIA ML

Given a set $D$ consisting of pairs $(x, y) \in D$, find $w$ by ML

$$
\min_w \sum_{(x,y) \in D} \epsilon \log Z_\epsilon(x, w) - F(x, y; w). \tag{3}
$$

This is equivalent to maximize the cross-entropy between $p(x,y),tg(\hat{y}) = \delta(\hat{y} = y)$ (target) and the model $p(x,y)(\hat{y}|w, \epsilon)$.

As a result, (3) is equivalent to

$$
\max_w \sum_{(x,y) \in D, \hat{y} \in Y} p(x,y),tg(\hat{y}) \log p(x,y)(\hat{y}|w, \epsilon). \tag{4}
$$

Maximizing (3) w.r.t. $w$ requires the gradient

$$
\sum_{(x,y) \in D} \sum_{\hat{y} \in Y} \frac{\partial}{\partial w} F(x, \hat{y}; w) \left( p(x,y)(\hat{y}|w, \epsilon) - p(x,y),tg(\hat{y}) \right). \tag{5}
$$
Learning Deep Structured Models

Algorithm: Deep Structured Learning
Repeat until stopping criteria

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

Figure 1. Gradient descent for learning deep structured models.
**Approximate Learning**

In general, computing $p(x,y)(\hat{y}|w,\epsilon)$ is not possible, $\mathcal{Y} = \prod_{n=1}^{N} \mathcal{Y}_i$ is exponentially large.

Use identity from literature on approximations for log-linear models

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

where $\mathbb{E}[\cdot]$ is over $p(x,y)(\hat{y})$ and $H(\cdot)$ its entropy.

In most applications, $F(x,y; w)$ decomposes as

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

where $r$ restricts $y = (y_1, \ldots, y_N)$ to the subset $r \subset \{1, \ldots, N\}$.

Important: note that $f_r (x,y_r; w)$ can be non-linear w.r.t. $w$. 
Approximate Learning

Using the partition defined by $\mathcal{R}$, we can write

$$
\epsilon \log Z_\epsilon(w, w) = \max_{p(x, y)} \sum_{r, \hat{y}_r} p(x, y, r) f_r(x, \hat{y}_r; w) + \epsilon H(p(x, y)(\hat{y}))
$$

where $p(x, y, r)(\hat{y}_r) = \sum_{\hat{y} \neq \hat{y}_r} p(x, y)(\hat{y})$ (marginals).

Problem: The scoring function is now local but $H(p(x, y)(\hat{y}))$ is still hard to compute.

Approach: approximate marginals with local beliefs $b(x, y, r)$. Fulfill marginalization constrains locally, not globally, thus

- $b(x, y, r)$ are not required to arise from a common $p(x, y)$.
- Approximate via fractional entropy, $H(p(x, y)) \approx \sum_r c_r H(b(x, y, r))$, where $c_r$ are counting numbers to weight marginal entropies.
Approximate Learning

Putting everything together

\[ \epsilon \log Z_\epsilon(w, w) = \max_{b(x, y) \in C(x, y)} \sum_{r, \hat{y}_r} b(x, y), r(\hat{y}_r) f_r(x, \hat{y}_r; w) + \epsilon \sum_r c_r H(b(x, y), r), \]

where beliefs are constrained to the local polytope

\[ C_{(x, y)} = \begin{cases} \forall r & b(x, y), r \in \Delta \\ \forall r, \hat{y}_r, p \in P(r) & \sum_{\hat{y}_p \backslash \hat{y}_r} b(x, y), p(\hat{y}_p) = b(x, y), r(\hat{y}_r) \end{cases} \]

where

- \( P(r) \), the set of parents of region \( r \), i.e., \( P(r) \subset \{ p \in \mathcal{R} : r \in p \} \).
- \( C(r) \), the set of children \( C(r) = \{ c \in \mathcal{R} : r \in P(c) \} \).
Approximate Learning

Replacing (9) into (3) (approximation into ML objective)

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

Figure 2. The approximated non-linear structured prediction task.

We need:

- Subgradient w.r.t. \( w \).
- Maximize w.r.t. beliefs \( b \) exactly.
- Typically NP-hard, use iterative message passing instead.
- As a result, slow double-loop algorithm (when combined with Algorithm 1).
**Efficient Approximate Learning**

Use principle of blending learning with inference.
- Single message passing iteration before updating $w$.
- One iteration does not yield accurate beliefs $b$.
- Assume all counting numbers $c_r > 0$, it is possible to derive algorithm to interleave $\min_w$ and $\max_b$.
- More efficient algorithm, more frequent updates of $w$.

Convert $\max_b$ into minimization using a dual program. The conversion is possible since the maximization is concave in $b(x,y)$ if $\forall r, \epsilon c_r \geq 0$. 
Efficient Approximate Learning

Claim 1 Assume \( \epsilon c_r \geq 0 \ \forall r \), and let \( \bar{F}(w) = \sum_{(x,y) \in D} F(x,y;w) \) denote the sum of empirical function observations. Let \( \lambda(x,y), r \rightarrow p(\hat{y}_r) \) be the Lagrange multipliers for each marginalization constraint \( \sum_{\tilde{y}_p \not= \hat{y}_r} b(x,y), p(\hat{y}_p) = b(x,y), r(\hat{y}_r) \) within the polytope \( \mathcal{C}(x,y) \). Then the approximated general structured prediction task shown in Fig. 2 is equivalent to

\[
\min_{w,\lambda} \sum_{(x,y), r} \epsilon c_r \ln \sum_{\hat{y}_r} \exp \frac{f_r(x, \hat{y}_r; w, \lambda)}{\epsilon c_r} - \bar{F}(w), \tag{5}
\]

where we employed the re-parameterization score

\[
\hat{f}_r(x, \hat{y}_r; w, \lambda) = f_r(x, \hat{y}_r; w) + \sum_{c \in C(r)} \lambda(x,y), c \rightarrow r(\hat{y}_c) - \sum_{p \in P(r)} \lambda(x,y), r \rightarrow p(\hat{y}_r).
\]

Duality allows to convert the min-max task into the single minimization in (5).

- (5) can be minimized by block coordinate descent.
- We can interleave between learning (update \( w \)) and inference (update messages).
Efficient Approximate Learning

Algorithm: Efficient Deep Structured Learning
Repeat until stopping criteria
1. Forward pass to compute \( f_r(x, \hat{y}_r; w) \) \( \forall (x, y), r, y_r \)
2. Compute approximate beliefs \( b_{(x,y),r} \propto \exp \frac{f_r(x, \hat{y}_r; w) + \lambda}{\epsilon c_r} \) by iterating for a fixed number of times over \( r \):
\[
\forall (x, y), p \in P(r), \hat{y}_r
\mu_{(x,y),p \rightarrow r}(\hat{y}_r) = \frac{c_p}{\epsilon c_r} \ln \sum_{\hat{y}_p \neq \hat{y}_r} \exp \frac{f_p(x, \hat{y}_p; w) - \sum p' \in P(p) \lambda_{(x,y),p \rightarrow p'}(\hat{y}_p) + \sum r' \in C(p) \setminus r \lambda_{(x,y),r' \rightarrow p}(\hat{y}_{r'})}{\epsilon c_p}
\lambda_{(x,y),r \rightarrow p}(\hat{y}_r) \propto \frac{c_p}{c_r} + \sum_{p \in P(r)} c_p \left( f_r(x, \hat{y}_r; w) + \sum_{c \in C(r)} \lambda_{(x,y),c \rightarrow r}(\hat{y}_c) + \sum_{p \in P(r)} \mu_{(x,y),p \rightarrow r}(\hat{y}_r) \right) - \mu_{(x,y),p \rightarrow r}(\hat{y}_r)
\]
3. Backward pass via chain-rule to obtain gradient \( g = \sum_{(x,y),r,\hat{y}_r} b_{(x,y),r}(\hat{y}_r) \nabla_w f_r(x, \hat{y}_r; w) - \nabla_w F(w) \)
4. Update parameters \( w \) using stepsize \( \eta \) via \( w \leftarrow w - \eta g \)

Figure 3. Efficient learning algorithm that blends learning and inference.

Implementation:
- C++ library with GPU support.
- Standard gradient descent plus mini-batch, moments and regularization.
- Schedule for step-size via log-likelihood or validation set performance.
- \( F(\cdot) \) is specified as a DAG.
- HDF5 storage.
**Word Recognition: Word50**

Settings:

- Word recognition from noisy images (28 × 28).
- 50 random words of length 5.
- Generate variations: characters from Chars74K + random background patches + scale-rotation-translation.
- Training, validation, test sizes: 10K, 2K and 2K, respectively.
- Graphical models: unary and pairwise regions defined over 5 random variables.
- Unary potentials: MLP with ReLU.
- Pairwise potentials: 1st and 2nd order Markov model.
- Weights across nodes and pairwise interactions are shared.
- Mini-batch of size 100, step-size 0.01, momentum 0.95.
- Validation set is used to tune (decrease) step-size.
- Set $\epsilon = 1$ and $c_r = 1$, $\forall r$. 
- 10 message passing iterations to compute marginal beliefs.
Word Recognition: Word50

Figure 4. Samples from the Word50 dataset. Note the high degree of rotation, scaling and translation.

<table>
<thead>
<tr>
<th>Graph</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 order Markov</td>
<td>One Layer</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>31.80 / 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>20.65 / 67.42</td>
<td>25.70 / 71.65</td>
<td>31.70 / 75.56</td>
<td>34.50 / 77.14</td>
<td>35.85 / 78.05</td>
</tr>
<tr>
<td>2nd order Markov</td>
<td>One Layer</td>
<td>JointTrain</td>
<td>25.50 / 67.13</td>
<td>34.60 / 73.19</td>
<td>45.55 / 79.60</td>
<td>51.55 / 82.37</td>
<td>54.05 / 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>28.15 / 69.07</td>
<td>36.85 / 75.21</td>
<td>45.75 / 80.09</td>
<td>50.10 / 82.30</td>
<td>52.25 / 83.39</td>
</tr>
<tr>
<td>1st order Markov</td>
<td>Two Layer</td>
<td>$H_1 = 512$</td>
<td>H2 = 32</td>
<td>H2 = 64</td>
<td>H2 = 128</td>
<td>H2 = 256</td>
<td>H2 = 512</td>
</tr>
<tr>
<td></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>47.70 / 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>42.25 / 81.10</td>
<td>44.85 / 82.96</td>
<td>46.85 / 83.50</td>
<td>47.95 / 84.21</td>
<td>47.05 / 84.08</td>
</tr>
<tr>
<td>2nd order Markov</td>
<td>Two Layer</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>62.60 / 88.03</td>
<td>65.80 / 89.32</td>
<td>68.75 / 90.47</td>
<td>68.60 / 90.42</td>
<td>69.35 / 90.75</td>
</tr>
</tbody>
</table>

Table 1. Word / Character accuracy. Performance improves as (1) joint-training is employed, (2) the model is more structured, and (3) deeper unary classifiers are utilized. The number of hidden units for the first and second layer are denoted as $H_1$ and $H_2$ respectively.
**Word Recognition: Word50**

*Figure 5.* (left) Subset of the learned unary weights. Pairwise weights (middle two panels), the darker, the larger the weight. (right) Negative log-likelihood for different learning approaches.
Image Tagging: Flickr

Settings:

- 10K training and 10K testing images from Flickr.
- Predict which of 38 possible tags should be assigned to an image.
- Graphical model: 38 binary random variables.
- Non-linear unaries: 8-layer deep-net with 76-dimensional top layer.
- Pairwise potential: any pair of outputs.
- Mini-batch of size 300, step-size 0.0001, momentum 0.95.
- Initialize deep-net using pre-trained on ImageNet.
- Set $\epsilon = 1$ and $c_r = 1$, $\forall r$. 
Experiments

Figure 7. Flickr test set images and a subset of the assigned tags as well as our predictions (bottom row).

<table>
<thead>
<tr>
<th>Method</th>
<th>Mean error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unary only</td>
<td>9.36</td>
</tr>
<tr>
<td>PwTrain</td>
<td>7.70</td>
</tr>
<tr>
<td>PreTrainJoint</td>
<td><strong>7.25</strong></td>
</tr>
</tbody>
</table>

Table 2. Flickr Hamming loss: Joint training of deep features and the MRF improves performance.
EXPERIMENTS

Figure 8. Correlation matrix (i.e., pairwise potentials) learned on the Flickr dataset.