Today: latent variable SVMstruct - CCCP

Deep learning - RNN

Latent variables

Motivation: semantic segmentation

Segmentation is expensive \( \rightarrow Z \) "latent variable"

Perhaps only have class labels \( \rightarrow y \)

Also: Felzenszwalb, 2010, IPAM 2010

"deformable parts models" for object recognition

\( \rightarrow Z \) time was an object part approximation

Before, we had \( s(x, y, z; w) = \langle w, y(x, z) \rangle \)

Now, consider \( s(x, y, z; w) = \langle w, y(x, z) \rangle \)

As before, could predict with any max \( s(x, y, z; w) \)

\( y, z \in Z \)

Learning

CRF approach

hidden CRF \( p(y | z, x) \)

Similarly to latent variable model

with graph model

\( \rightarrow \) marginals \( Z \) out

ML \( \rightarrow \) EM (expectation-maximization)

Analogy for latent SVMstruct is CCCP

(majorization minimization procedure)

Latent SVMstruct

\( l(y, (y, z)) \)

Generalized structured hinge loss

\( \mathcal{L}(x, y, w) \leq \max_{y, z} \langle w, Q(x, y, z) \rangle + l(y, (y, z)) \)

\( \leq u(w) \)

Here, \( \mathcal{L}(x, y, w) = u(w) - v(w) \) where \( u, v \) are convex for \( w \)
CCCP procedure:

- Linearize $v(w)$ at $w_t$ to get an upper bound on $-v(w)$
- $w_{t+1}$ is obtained by minimizing the upper bound $u(w_t) + $ linearize piece $v(w)$
- repeat 7, a majorization-minimization procedure (EM is another example)

\[
\begin{align*}
S_t(w) &= u(w_t) - \langle v(w_t) + \nabla v(w_t), w - w_t \rangle \\
S_t(w) &= \frac{S_t(w)}{w_{t+1}} \\
S_t(w) &= \text{argmin}_w S_t(w)
\end{align*}
\]

Properties of procedure:
- like EM, descent procedure i.e. $S(w_{t+1}) \leq S(w_t)$
- $S_t(w_t) = S_t(w_t) \geq S_t(w_{t+1}) \\
\text{upper bound}

* local linear convergence to a station pt.,

CCCP algorithm for latent SV$\mu$struct:

\[
\begin{align*}
&\text{repeat: } \\
&\quad \text{fill in } z_i^{(t)} \text{ for all grand truth } y^{(t)} \text{ using } w_t \\
&\quad \text{solve a standard SV$\mu$struct b get } w_{t+1}
\end{align*}
\]

Deep Learning

\[
\text{go from } \langle w, \ell(x,y) \rangle \text{ to } \langle w, \ell(x,y), \delta \rangle
\]
I) plug in "deep learning" features in a structured prediction model

- Example: OCR
  \[ y_1 \rightarrow y_2 \rightarrow y_3 \]
  
  so far \[ p^0(x_t; y_t) = \begin{pmatrix} 0 \\ x_t \\ 0 \end{pmatrix} \cdot y_t \]

  instead \[ p^0(x_t; y_t) = \begin{pmatrix} W(y_t) \\ x_t \end{pmatrix} \cdot y_t \]

  pre-trained on \{\text{e.g.}\} imagine e.g.

II) "end-to-end" training

- Structured prediction energy networks (SPENs)

\[
y \rightarrow \begin{pmatrix} l_k \text{ [RNN]} \text{ [CWS]} \end{pmatrix} \rightarrow s(z, y, w, \theta)\]

- \[ p^1(x; y, \theta) = \begin{pmatrix} \text{spatial memory} \text{ [CWS]} \end{pmatrix} \]

III) Recurrent neural networks (RNN)

- Motivation: \[ p(y \mid x) = \frac{1}{Z} \prod_{t=1}^{T} p(y_t \mid y_{t-1}, x) \]

- Chain rule

- RNN = "structured parameterization" with no conditional assumption

- of \[ p(y \mid y_{1:t-1}; x) \]

  using a NN

\[
h_{t+1} = f(h_t, x, y_t, \theta)\]

\[
h_t = f(f(\cdots f(\cdots f(h_0, x, y_1, y_2, \theta), x, y_2, \theta), \cdots), x, y_{t-1}, \theta)\]

- Define \[ p(y_t \mid y_{1:t-1}; x) \propto \exp(c^{y_t}v^T h_t) \]

E.g.

- Word embedding in HMM and can be fine-tuned

- Standard learning: using MCL
Standard training: using MCL
\[
\text{min}_{w, \tilde{w}} \sum_{i=1}^{n} \log p(y_i^{(i)} | z_i^{(i)})
\]
"backward forcing"

\[
\leq \log p(y_i^{(i)} | y_{i-1}^{(i)}, z_i^{(i)})
\]
output of a deep NN

For ML, do SGD on obj.

\[
\text{gradient of } \log p(y_i^{(i)} | y_{i-1}^{(i)}, z_i^{(i)}; w, \tilde{w})
\]

→ use backpropagation

\[
\text{decoding } \sum_{s \in S} \text{argmax}_{s} \log p(s_{t+1}^{(i)} | y_{t}^{(i)}, z_{t}^{(i)}; \hat{w})
\]
→ NP hard?

→ need approximation

\[
\text{greedy decoding } y_{t}^{(i)} \leftarrow \text{argmax}_{y_{t}^{(i)}} p(y_{t}^{(i)} | y_{1:t-1}^{(i)}, z_{t}^{(i)}; \hat{w})
\]

beams search

"greedy decoding with memory of size k" → size of beam

beam search:

\[
\text{goal: } \hat{y}_{i}^{(i)} \ldots \hat{y}_{t}^{(i)}
\]

beam of size L (memory)

→ at step t, you have L candidate solution prefixes \(y_{1:t}^{(i)}\)

\[
\hat{y}_{i}^{(i)} \leftarrow \text{argmax}_{y_{i}^{(i)}} p(y_{i}^{(i)} | y_{1:i-1}^{(i)}, z_{i}^{(i)}; \hat{w})
\]

→ expand possible next choices \(1 \times \Sigma_{t+1} \times L\)

score them (e.g., \(\log p(y_{t+1}^{(i)} | y_{1:t}^{(i)}; x_{t}) + \log p(y_{t}^{(i)}; z_{t})\))

→ keep top \(L\) candidates as \(\Sigma_{t} y_{t:i}^{(i)} \leq \ell_{s}\)

vs. Viterbi alg., which does "backtracking" to correct past mistakes

seq2seq align. encoder/decoder architecture

\[\text{Useful way to get } p(y_{t}^{(i)} | y_{t:n}^{(i)}, x) \text{ for a RNN when } x \text{ has variable length}\]
Issues:

a) Variable length output? → end-of-sequence symbol

b) How to handle long input sequence $x$?

  problem: Need to summarize input sentence in one context vector of fixed length

  solution: "Attention mechanism"

c) Vanishing gradient?

  - LSTM
  - Gated recurrent unit (GRU)
  - etc..

d) Sequential processing → fixed by "transformer" architecture "Attention is all you need"

  → (Vaswani et al.)

  "Self-attention"

  - Quadratic cost
  → Need tricks to handle long context/sequence