Today: Learning to Search

• SEARN
• SPENs

Learning to search (L2S)

SEARN: Hal Daumé's PhD Thesis

- \( h_w: X \rightarrow Y \)
  - \( h_w(x) = \arg \max_y s(x, y; w) \)

**I generated search procedure**

Today: Special case of SEARN: learning to do greedy search

1. Set \( y \) in ordered list of decisions \( (y_1, y_2, \ldots, y_T) \)
2. Learning a classifier \( \hat{T}_w \) (feature \( (y_1, \hat{y}_2, \ldots, \hat{y}_{T-1}, x) \) = \( \hat{y}_T \)
   - classifier “decoding policy”

**L2S Framework**

From \( (x^{(i)}, y^{(i)}) \), \( i = 1 \) and \( l(\cdot, \cdot) \)

Learn a good classifier/policy \( \hat{T}_w \) s.t.

- \( \hat{y}_T = \tau_w (\ell(h_w(x), y_T)) \)
- \( h_w(x) = \{y_1, \ldots, y_T\} \) “greedy decoder” s.t.
  - \( l(y^{(i)}; \hat{y}_w(x)) \) is small

Central idea: “reduction” where replaces structured prediction problem to problem of cost-sensitive classification learning of \( \tau_w \)

Method: Generate training data for classifier \( \tau_w \)

- \( (x^{(i)}, y^{(i)}, \text{cost}(y^{(i)})) \)

Prefer sequence to make best prediction

“Roll-in” policy → determines how \( \hat{y}_{i-1} \) context

“Roll-out” policy → \( \hat{y}_{i-1} \) \( y_{i-1}, \ldots, y_T \) “forget computation” to get

- \( \ell(y^{(i)}; \hat{y}_{i-1}, y_{i-1}, \ldots, y_T) \)
- From roll-out
- From roll-in

IFT6132 Page 1
The idea is to define surrogate loss for \( \hat{w} \) as function of (cost of\( \hat{w} \)).

\[
\text{loss} = \text{argmin}_{\hat{y}} \left( \frac{1}{m} \sum_{i=1}^{m} l(y_i, \hat{y}_i) \right)
\]

in practice: use hammers to approximate \( \hat{w} \)

but sometimes, can compute exactly

\( \hat{w} \) for Hamming loss: is just copying ground truth

\[\hat{w} = \text{learned true}\]

Example of approximate \( \hat{w} \): is like score

consider all possible offsets of ground truth
add pick best (hueristic) score

\[\text{EARHN} : \text{ apply LAS to RNN training}
\]

\[L(\hat{y}; h, x) \rightarrow \text{RNN cell}
\]

\[p(\text{ground truth}; y; h, x) \text{ of a (bi-directional) RNN}
\]

\[\text{if you use} -\text{log} p(\text{ground truth}; y; h, x) \text{ as a cost surrogate}
\]
If you use \(-\log p(y_{\text{target}} | y_{1:t-1}, z)\) as a cost surrogate

and \(y_{\text{target}} = \text{ground truth}\)

then L2S with \(\text{target mlog-} \) is standard MLE

\[ \text{Cost-sensitive surrogate loss choice} \]

a) structured SVM:
\[
\min_y \left\{ \Delta(y) - c(y) \right\} - S(y_{\text{target}})
\]
\[
\Delta(y) = \max_y \left\{ c(y) - c(y') \right\}
\]

b) "target log-loss": \(-\log p_0(y_{\text{target}} | y_{1:t-1}, z)\)

L2 differences with MLE:
- add additional bias using learned rel-fun
- make use of structured loss \(S(.)\) to predict \(y_{\text{target}}\) vs. ground truth

Structured prediction energy networks (SPENs)

ICML 2010:
\[
E(x; y; w) \ (\sim \sigma(x; y; w))
\]

\(\sigma(x; y; w)\) is the loss of a few steps of projected GD on \(E(x; y; w)\) (approximate prediction procedure)

2016 paper:
SSVM loss \(\Rightarrow \) ("subgradient" method on \(w\))

huge loss \(\Rightarrow\) max \(\sum_{y \in \Omega(y)} \left( \sigma_{\text{max}}(y) - E(x^{(i)}, y^{(i)}; w) + E(x^{(i)}, y^{(i)}; w) \right)\)

approximate "subgradient" is \(-\nabla_w E(x^{(i)}, y^{(i)}; w) + \nabla_w E(x^{(i)}, y^{(i)}; w)\)

because \(\hat{y}\) is approximate max

2017 paper: end-to-end training

\[
\hat{y}_w(x) = \arg\max_{y} \sum_{t=1}^{T} \alpha_t \Delta E(x, y; w)
\]

"unrolled optimization"