The Perceptron for Structured Prediction

Stephen Clark

Lent 2013

Machine Learning for Language Processing: Lecture 5

MPhil in Advanced Computer Science

Local and Global Feature Representations

- We have already seen local feature representations for the maxent tagger:

$$f_i(C,t) = \begin{cases} 1 & \text{if } \texttt{word}(C) = \texttt{Moody} \ \ \& \ \ t = \text{I-ORG} \\ 0 & \text{otherwise} \end{cases}$$

where C is a context and t a tag

- These can be extended to the whole sequence (as in a CRF):

$$F_i(W,T) = \sum_{j=1}^{n} f_i(C_j, t_j)$$

where W is the sentence and T the tag sequence; C_j is the context for the jth word; t_j is the tag for the jth word

Decoding for the CRF Tagger

$$
\begin{aligned}
T_{\text{max}}(W) &= \operatorname*{argmax}_T \ \exp\Big(\sum_i \lambda_i \, F_i(W,T)\Big) \ / \ Z(W) \\
&= \operatorname*{argmax}_T \sum_i \lambda_i \, F_i(W,T) - \log Z(W) \\
&= \operatorname*{argmax}_T \sum_i \lambda_i \, F_i(W,T)
\end{aligned}
$$

- Viterbi finds this argmax efficiently (assuming each contex C_j contains only the previous m tags; i.e. assuming an m-gram tagger)

Perceptron Training for a Linear Model

$$\text{Score}(T, W) = \sum_i \lambda_i \, F_i(W, T) = \overline{\lambda} \cdot \Phi(W, T)$$

Some notation:

- $\phi : (C, t) \rightarrow \mathbb{R}^d$ where d is the number of features
- $\Phi : (W, T) \rightarrow \mathbb{R}^d$
- $\Phi(W, T) = \sum_j \phi(C_j, t_j)$

ϕ is the local feature vector; Φ is the global feature vector; $\overline{\lambda}$ is the corresponding global weight vector

The Perceptron Training Algorithm

Inputs: Training examples (x_k, y_k)
Initialization: $\overline{\lambda} = 0$
Algorithm:
  For $l = 1$ to L, $k = 1$ to n
    Use Viterbi to get $z_k = \text{argmax}_z \, \overline{\lambda} \cdot \Phi(x_k, z)$
    If $z_k \neq y_k$ then $\overline{\lambda} = \overline{\lambda} + \Phi(x_k, y_k) - \Phi(x_k, z_k)$
Output: weights $\overline{\lambda}$

The Perceptron Training Algorithm (with words)

Inputs: Training examples (x_k, y_k)
Initialization: $\overline{\lambda} = 0$
Algorithm:
For $l = 1$ to L, $k = 1$ to n
Use Viterbi to get $z_k = \text{argmax}_z \, \overline{\lambda} \cdot \Phi(x_k, z)$
If $z_k \neq y_k$ then $\overline{\lambda} = \overline{\lambda} + \Phi(x_k, y_k) - \Phi(x_k, z_k)$
Output: weights $\overline{\lambda}$

Assume n tagged sentences for training
Initialise weights to zero
Do L passes over the training data
For each tagged sentence in the training data, find the highest scoring tag sequence using the current weights
If the highest scoring tag sequence matches the gold, move to next sentence
If not, for each feature in the gold but not in the output, add 1 to its weight; for each feature in the output but not in the gold, take 1 from its weight
Return weights

Averaging Parameters

Perceptron training can suffer from over-fitting; averaging the parameters is a simple addition which works well in practice:

$$\lambda_i^{av} = \sum_{l=1 \text{ to } L,\ k=1 \text{ to } n} \lambda_i^{l,k} / Ln$$

(Based on the voted perceptron, which has some theory associated with it)

Theory of the Perceptron

Simple additive update seems intuitive, but do we have any guarantees? Collins (2002) has some proofs showing that:

- If the data is separable with some margin, then the algorithm will converge on weights which give zero error on the training data
- If the training data is not separable, but "close" to being separable, then the algorithm will make a small number of mistakes (on the training data)
- If the algorithm makes a small number of errors on the training data, it is likely to generalise well to unseen data

Michael Collins, Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms, 2002