Machine Learning for Language Processing
ACS 2015/16

Stephen Clark

L4: The Perceptron for Structured Prediction

UNIVERSITY OF CAMBRIDGE

Local and Global Features

- 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

UNIVERSITY OF CAMBRIDGE

A Linear Tagging Model

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

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; $\bar{\lambda}$ is the corresponding global weight vector

UNIVERSITY OF CAMBRIDGE

Decoding for a Global Linear Model

$$T_{\text{max}}(W) = \underset{T}{\operatorname{argmax}} \sum_i \lambda_i F_i(W, T)$$

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

UNIVERSITY OF CAMBRIDGE

The Perceptron Training Algorithm

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

UNIVERSITY OF CAMBRIDGE

The Training Algorithm (with words)

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

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

UNIVERSITY OF CAMBRIDGE

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)

UNIVERSITY OF CAMBRIDGE

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

A *Ranking* Perceptron

Some notation:

- Assume training data $\{(s_i, t_i)\}$ (e.g. s_i is a sentence and t_i the correct tree for s_i)
- $\mathbf{x}_{ij}$ is the jth candidate for example i (e.g. the jth tree for sentence i)
- Assume (w.l.o.g.) that $\mathbf{x}_{i1}$ is the correct output for input s_i (i.e. $\mathbf{x}_{i1} = t_i$)
- $\mathbf{h}(\mathbf{x}_{ij}) \in \mathbb{R}^d$ is the feature vector for $\mathbf{x}_{ij}$
- $\mathbf{w} \in \mathbb{R}^d$ is the corresponding weight vector
- Output of the model on example s (train or test) is $\mathrm{argmax}_{\mathbf{x} \in \mathcal{C}(s)} \mathbf{w} \cdot \mathbf{h}(\mathbf{x})$
- $\mathcal{C}(s)$ is the set of candidate outputs for input s

UNIVERSITY OF CAMBRIDGE

Training (with the new notation)

```
**Define:**
$F(\mathbf{x}) = \mathbf{w} \cdot \mathbf{h}(\mathbf{x})$
**Initialisation:** Set parameters $\mathbf{w} = 0$
**For** $i = 1$ to $n$
    $j = \mathrm{argmax}_{\{1,\ldots,n_i\}} F(\mathbf{x}_{ij})$
    **If** $j \neq 1$ **then** $\mathbf{w} = \mathbf{w} + \mathbf{h}(\mathbf{x}_{i1}) - \mathbf{h}(\mathbf{x}_{ij})$
**Output on test sentence** $s$**:**
    $\mathrm{argmax}_{\mathbf{x} \in \mathcal{C}(s)} F(\mathbf{x})$
```

- For simplicity, only showing one pass over the data and no averaging
- The argmax can be obtained just though enumeration (i.e. we have a *ranking* problem, so no need for dynamic programming)

UNIVERSITY OF CAMBRIDGE

Perceptron Training (a duel form)

```
**Define:**
$G(\mathbf{x}) = \sum_{(i,j)} \alpha_{i,j}(\mathbf{h}(\mathbf{x}_{i1}) \cdot \mathbf{h}(\mathbf{x}) - \mathbf{h}(\mathbf{x}_{ij}) \cdot \mathbf{h}(\mathbf{x}))$
**Initialisation:** Set dual parameters $\alpha_{i,j} = 0$
**For** $i = 1$ to $n$
    $j = \text{argmax}_{\{1,\ldots,n_i\}} G(\mathbf{x}_{ij})$
    **If** $j \neq 1$ **then** $\alpha_{i,j} = \alpha_{i,j} + 1$
**Output on test sentence** $s$**:**
    $\text{argmax}_{\mathbf{x} \in \mathcal{C}(s)} G(\mathbf{x})$
```

- Notice there is a dual parameter $\alpha_{i,j}$ for each training example $\mathbf{x}_{i,j}$

UNIVERSITY OF CAMBRIDGE

Equivalence of the two Forms

- $\mathbf{w} = \sum_{(i,j)} \alpha_{i,j}(\mathbf{h}(\mathbf{x}_{i1}) - \mathbf{h}(\mathbf{x}_{ij}))$; therefore $G(\mathbf{x}) = F(\mathbf{x})$ throughout training
- Why is this useful? Consider the complexity of the two algorithms

UNIVERSITY OF CAMBRIDGE

Complexity of the two Forms

- Assume T is the size of the training set; i.e. $T = \sum_i n_i$
- Take d to be the size of the parameter vector $\mathbf{w}$
- Vanilla perceptron takes $O(Td)$ time (time taken to compute F is $O(d)$)
- Assume time taken to compute the inner product between examples is k
- Running time of the dual-form perceptron is $O(Tnk)$
- Dual-form is therefore more efficient when $nk << d$ (i.e. when time taken to compute inner products between examples is much less than $O(d)$)

UNIVERSITY OF CAMBRIDGE

Complexity of Inner Products

- Can the time to calculate the inner product between two examples $\mathbf{h}(\mathbf{x}) \cdot \mathbf{h}(\mathbf{y})$ ever be less than $O(d)$?
- Yes! For certain high-dimensional feature representations
- Examples include feature representations which track all sub-trees in a tree, or all sub-sequences in a tag sequence

Tree Kernels

- Tree kernels count the numbers of shared subtrees between trees $\mathcal{T}_1$ and $\mathcal{T}_2$
  - the feature-space, $\mathbf{h}\left(\mathcal{T}_1\right)$, can be defined as

$$\mathbf{h}_i\left(\mathcal{T}_1\right) = \sum_{n \in \mathcal{V}_1} I_i(n); \quad I_i(n) = \begin{cases} 1 & \text{if sub-tree } i \text{ rooted at node } n \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{V}_j$ is the set of nodes in tree $\mathcal{T}_j$

Computation of Subtree Kernel

- Can be made computationally efficient by recursively using a counting function:

$$k(\mathcal{T}_1, \mathcal{T}_2) = \mathbf{h}(\mathcal{T}_1)^{\mathsf{T}}\mathbf{h}(\mathcal{T}_2) = \sum_{n_1 \in \mathcal{V}_1} \sum_{n_2 \in \mathcal{V}_2} f(n_1, n_2);$$

  - if productions from n_1 and n_2 differ $f(n_1, n_2) = 0$
  - for pre-terminals $f(n_1, n_2) = \begin{cases} 1 & \text{if productions are the same} \\ 0 & \text{otherwise} \end{cases}$
  - for non-pre-terminals and productions the same
    $f(n_1, n_2) = \prod_{i=1}^{|\mathsf{ch}(n_1)|}(1 + f(\mathsf{ch}(n_1, i), \mathsf{ch}(n_2, i)))$

where $\mathsf{ch}(n_j)$ is the set of children of n_j and $\mathsf{ch}(n_j, i)$ is the ith child of n_j

- Algorithm runs in linear time w.r.t. the size of each tree

UNIVERSITY OF CAMBRIDGE