The Dual-Form Perceptron (leading to Kernels)

Stephen Clark

Lent 2013

Machine Learning for Language Processing: Lecture 6

MPhil in Advanced Computer Science

Ranking Structures with the 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 $\operatorname{argmax}_{\mathbf{x} \in \mathcal{C}(s)} \mathbf{w} \cdot \mathbf{h}(\mathbf{x})$
- $\mathcal{C}(s)$ is the set of candidate outputs for input s

Perceptron 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

$j = \operatorname{argmax}_{\{1,\ldots,n_i\}} F(\mathbf{x}_{ij})$
If $j \neq 1$ **then** $\mathbf{w} = \mathbf{w} + \mathbf{h}(\mathbf{x}_{i1}) - \mathbf{h}(\mathbf{x}_{ij})$

Output on test sentence s**:**

$\operatorname{argmax}_{\mathbf{x} \in \mathcal{C}(s)} F(\mathbf{x})$

- 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)

Perceptron Training (a dual 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
  $j = \text{argmax}_{\{1,\dots,n_i\}} G(\mathbf{x}_{ij})$
  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 $\mathbf{x}_{i,j}$

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

Computational 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 $O(d)$)

Computational 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, $\mathbf{h}(\mathcal{T}_1)$, can be defined as

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

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

Tree Kernels in Practice

- Data-Oriented Parsing (Rens Bod) is a parsing model which uses a similar all-subtrees representation (but without the efficient computation)
- Collins and Duffy report a 0.6% absolute improvement over the generative models of Collins
- Alessandro Moschitti has done a lot of work on using various kernels (including tree kernels) for various tasks (including some parsing tasks)

References

Michael Collins and Nigel Duffy (2002)
New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron

Rens Bod (2003)
Do All Fragments Count?
Natural Language Engineering, 9(4), 307-323.

Moschitti Tutorial at ACL 2012: State-of-the-Art Kernels for Natural Language Processing