Softmax Tree: An Accurate, Fast Classifier When the Number of Classes Is Large

Arman Zharmagambetov, Magzhan Gabidulla and Miguel Á. Carreira-Perpiñán
Dept. Computer Science & Engineering, University of California, Merced, USA

1 Abstract

Classification problems having thousands or more classes naturally occur in NLP, for example language models or document classification. A softmax or one-vs-all classifier is typically handled many classes, but is very slow at inference time, because every class score must be calculated to find the top class.

We propose the “softmax tree”, consisting of a binary tree having sparse hyper-planes at the decision nodes (which make hard, not soft, decisions) and small softmax classifiers at the leaves. This is much faster at inference because the input instance follows a single path to a leaf (whose length is logarithmic on the number of leaves) and the softmax classifier at each leaf operates on a small subset of the classes. Although learning accurate tree-based models has proven difficult in the past, we are able to overcome this by using a variation of a recent algorithm, tree alternating optimization (TAO). Compared to a softmax and other classifiers, the resulting softmax trees are both more accurate and faster in inference, as shown in NLP problems having from one thousand to one hundred thousand classes.

Work supported by NSF award IIS–2007147

2 Softmax Tree (ST): motivation

A large number of classes (K) are quite common in NLP problems:

- Language modeling: ~171k words in the Oxford English Dictionary -> 171k classes and grows as we include all forms of a word, names, acronyms, etc.
- Webpage classification giving its content: ODP contains ~1M website categories. So, automatically tagging a web page will require identifying a large number of classes, conditional on the input instance, in order to determine the (top-n) predicted classes.
- Obviously too slow to speed-up -> use decision trees, e.g. CART: typically perform poorly.

Proposed model: Softmax Tree

\[ E^{\text{ST}}(\theta) = \sum_{i=1}^{n} \left[ \log \sum_{k=1}^{K} \exp \left( \theta_k \sum_{j=1}^{d} \pi_{ij} (x_j) \right) \right] \]

\[ \theta \sim \text{weighted linear softmax classifier} \]

\[ \text{Result trained tree } T^{(\theta)} \]

\[ \text{input training set } \{ (x_i, y_i) \}_{i=1}^{n}, \text{ initial tree } T^{(\theta)} \]

\[ \text{for } d = 1 \text{ down to } 0 \]

\[ \text{if } i \text{ is a leaf then } \]

\[ \pi_{ij} \text{, instances of the most populous class in } T^{(\theta)} \]

\[ \text{else } \]

\[ \text{generate } \pi_{ij} \text{, for each node under the current } T^{(\theta)} \]

\[ \text{for } d = 1 \text{ down to } 0 \]

\[ \text{if } i \text{ is a leaf then } \]

\[ \pi_{ij} \text{, instances of the most populous class in } T^{(\theta)} \]

\[ \text{else } \]

\[ \text{generate } \pi_{ij} \text{, for each node under the current } T^{(\theta)} \]

\[ \text{for } d = 1 \text{ down to } 0 \]

\[ \text{if } i \text{ is a leaf then } \]

\[ \pi_{ij} \text{, instances of the most populous class in } T^{(\theta)} \]

\[ \text{else } \]

\[ \text{generate } \pi_{ij} \text{, for each node under the current } T^{(\theta)} \]

Practicalities:

- Dealing with zero probabilities – problematic during decision node optimization

\[ \frac{\pi_{ij}}{\sum_{j=1}^{d} \pi_{ij}} \text{, for all classes } j \]

- Sparse oblique decision nodes: \((x_k, y_k) \in \text{ in the above figure} \)

- Sparse linear softmax leaves where each leaf focuses only on \( k \) classes (K total number of classes).

3 Softmax Tree (ST): learning

- The proposed model provides speedup of \( \text{O}(K^d) \) compared to one-vs-all while still being accurate.

- However, STs are hard to train: nonconvex, nondifferentiable, discontinuous.

- We use Tree Alternating Optimization (TAO): non-greedy, generally finds better optima, has shown a huge success in training various tree-based models.

Assuming a tree structure \( T \) is given (say, binary complete of depth \( D \)), consider the following regularized objective:

\[ \mathcal{L}(\theta) = \mathcal{L}(\theta) + \lambda \sum_{t=1}^{D} \| \theta_t \|_2 \]

\[ \text{Model summary:} \]

- \( D \text{ = number of decision nodes} \)
- \( K \text{ = total number of classes} \)
- \( \lambda \text{ = regularization parameter} \)

4 Experiments: Document Classification

<table>
<thead>
<tr>
<th>Method</th>
<th>RecalC@1</th>
<th>P@1</th>
<th>Precision@1</th>
<th>Recall@1</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-vs-all</td>
<td>86.9</td>
<td>80.0</td>
<td>87.0</td>
<td>83.9</td>
</tr>
<tr>
<td>HSM</td>
<td>87.1</td>
<td>70.5</td>
<td>84.4</td>
<td>81.0</td>
</tr>
<tr>
<td>MACH</td>
<td>84.0</td>
<td>72.6</td>
<td>82.6</td>
<td>79.6</td>
</tr>
</tbody>
</table>

5 Experiments: Language Modeling

<table>
<thead>
<tr>
<th>Method</th>
<th>top-1</th>
<th>P@1</th>
<th>Coverage</th>
<th>( \Delta )</th>
<th>Int (ms)</th>
<th>size (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSM</td>
<td>87.3</td>
<td>80.0</td>
<td>91.0</td>
<td>84.0</td>
<td>18.0</td>
<td>0.097</td>
</tr>
<tr>
<td>Softmax</td>
<td>85.9</td>
<td>72.8</td>
<td>81.0</td>
<td>79.0</td>
<td>8.0</td>
<td>0.367</td>
</tr>
<tr>
<td>ST+OVR</td>
<td>86.5</td>
<td>72.5</td>
<td>81.0</td>
<td>79.0</td>
<td>6.0</td>
<td>0.357</td>
</tr>
<tr>
<td>ST+HOVR</td>
<td>86.4</td>
<td>64.8</td>
<td>81.0</td>
<td>79.0</td>
<td>4.0</td>
<td>0.366</td>
</tr>
</tbody>
</table>

Table 1: Results on text classification datasets.

We report the top-1 test error, maximum depth (\( \Delta \)), average inference time per test sample (in ms) and uncompressed model sizes (in GB). ST=(x→y) indicates our method which used at most K classes at each leaf. The results in brackets are taken from corresponding datasets.

**Fig. 1:** Avg. int. time tradeoff (top figure) and Top-1 errors (bottom) of the ST for various settings of \( \Delta \) and K on the ODP dataset.

Table 2: Like Table 1 but on PTB-language modeling task. We also report the test Perplexity (with percentage of the covered points) and top-5 error. **"** indicates that smoothing was applied to replace 0 probabilities with some small epsilon and renormalize the output.

**Fig. 2:** Like Table 1 but on the ODP dataset. Results are trained on the recurrent neural network (LSTM).