Probabilistic Label Trees for Efficient Large Scale Image Classification

Baoyuan Liu, Fereshteh Sadeghi, Marshall Tappen
University of Central Florida
{bliu, fsadeghi, mtappen}@eecs.ucf.edu

Ohad Shamir, Ce Liu
Microsoft Research, New England
{ohadsh, celiu}@microsoft.com

Abstract

Large-scale recognition problems with thousands of classes pose a particular challenge because applying the classifier requires more computation as the number of classes grows. The label tree model integrates classification with the traversal of the tree so that complexity grows logarithmically. In this paper, we show how the parameters of the label tree can be found using maximum likelihood estimation. This new probabilistic learning technique produces a label tree with significantly improved recognition accuracy.

1. Introduction

In this paper, we present an improved probabilistic model for distinguishing between large numbers of categories with a label tree model. Developing a large-scale vision system that can recognize a large number of categories poses a number of challenges[7, 13, 14, 15, 19]. In addition to scaling the training algorithms to accommodate all of the examples and features in a large scale problem, a system designed for many categories will also face scaling issues when classifying novel data. The parameters for the popular SVM and multinomial logistic regression models contain one vector per possible class. Thus, if there are K possible classes, assigning a label to a new feature vector $\mathbf{x}$ will require the computation of K dot products between $\mathbf{x}$ and the vectors defining the classifier. For a relatively small number of categories, this computational cost is not significant enough to require attention.

However, as both the number of categories and need for fast recognition increases, this linear relationship between the complexity of recognition and the number of classes can become problematic. This issue is compounded if the classification process involves computations that are more complex than a dot product.

In [1], Bengio et al. introduce the label tree model for reducing the complexity of recognition in a problem with a larger number of classes. In the label tree model, a feature vector is assigned a label by traversing a tree. At each node visited, the classifier computes the dot product between the feature vector and a small number of vectors. This tree structure causes the classification complexity to grow logarithmically, rather than linearly, with the number of classes. More recently, Deng et al. proposed an optimization-based scheme that maximizes accuracy under constraints on ambiguity [8].

In this paper, we present a novel, probabilistic approach to learning the parameters of a label tree. We show how a recursive process learns the tree parameters. As the results in Section 7 will show, this approach produces significantly improved accuracies over previous results in [8]. The probabilistic model also makes it possible to tune accuracy versus efficiency without having to retrain the tree.

From a broader perspective, formulating the label tree in a probabilistic framework provides a straight-forward avenue for integrating more complex, accurate classification models into the label tree framework. With this probabilistic formulation, any classifier that can be expressed probabilistically can be integrated into the label tree.

2. Overview of a Label Tree

This section will briefly review how a label tree operates and previous work on learning the parameters of the tree. Following previous work in [1, 8], we will focus on a label tree that uses linear classifiers at each node of the tree.

Algorithm 1 Classifying a test example with the label tree algorithm.

```
Input: Test example $\mathbf{x}$, label tree parameters $T, \sigma, l$
1: Initialize $s$ to the root node
2: while $\sigma(s) \neq \emptyset$ do
3:     $s \leftarrow \operatorname{argmax}_{c \in \sigma(s)} \mathbf{w}_c^\top \mathbf{x}$
4: end while
5: Assign the label $l(s)$ to the test example
```

Following the notation in [8], a label tree is a tree with nodes V and edges E, such that the tree $T = (V, E)$. The children of a node r are contained in the set $\sigma(r)$. Every child node, c, is associated with a vector of weights w_c that are used to choose which child node will be visited during the classification process. Each leaf node is also associated with a label, $l(s)$, for a node s, that specifies the label that should be assigned to the example if that node s is reached as a leaf node.

As shown in Algorithm 1, a new test example is classified by traversing the tree, beginning at the root node. Except for the leaf nodes, each node is treated identically. At a node s, assuming it is not a leaf node, classification scores are computed for every child node of s using the weights w_c for the child node c. The child node with the highest classification score becomes the next node to be visited. As described above, if the classification algorithm arrives at a leaf node, then the test example is assigned the label $l(s)$, where s is the leaf node visited.

2.1. Previous Work on Learning Label Trees

Most of the research on accelerating large scale classification problems focuses on tree-based models[1, 2, 3, 4, 6, 8, 10, 11]. Learning the label tree parameters requires finding the classifier weights for each node and the labels for the leaf nodes. The efficiency benefits of the tree structure will be maximized when the examples or classes are balanced among the leaf nodes.

In [3], Beygelzimer et al. build a randomized tree at test time, and each sample traverse the tree from the root to single leaf node which corresponds to a single class. Conditional Probability Tree(CPT) [11] adopts a similar idea as [3], but they build the tree at training time based on optimizing conditional probability. In [10], Gao et al. propose to formulate class hierarchy in DAG structure and learn a unified max-margin classifier together with splitting the classes.

Bengio et al. propose to learn a label embedding tree for multi-class tasks [1]. In this method, the tree structure is determined by the spectral clustering of the confusion matrix, which is generated by a one-against-all classifier. Then they learn the tree classifier based on a tree based loss function that outperforms independent nodes optimization. In addition, label embedding is integrated into the tree framework to further boost their accuracy and speed.

In [8], Deng et al. propose a margin-based model that finds the label tree parameters with two alternating optimization steps. In the first step, the system learns the linear classifiers that choose the child node to be visited by maximizing a multi-class margin score. This score is based on a list of classes assigned to each of the child nodes. The list of classes assigned to each node is determined in a second optimization that attempts to create a balanced tree by assigning classes to different child nodes. This assignment, which is found by optimizing a linear program, attempts to minimize the number of classes assigned to any node. This makes it possible to determine the class while visiting as few nodes as possible.

3. Learning a Probabilistic Label Tree

We propose a probabilistic approach for learning the label tree parameters. As will be shown in the experiments in Section 7, this probabilistic model produces improved recognition accuracy.

Each node, s, in the probabilistic label tree is associated with a categorical probability distribution $p[y|S]$ where y denotes the label of the test example and S denotes the event that the inference process has arrived at node s.

This categorical distribution can be combined with a probabilistic classifier defined by the classification vectors at each node to compute the probability of a particular label being assigned to the test example, given the feature vector **x**.

3.1. Defining the Learning Criterion

Defining the label tree as a probabilistic model makes it natural to take a maximum likelihood approach to learning the parameters. The first step is to define the probability that the example should receive the label y, beginning at the root node, r. This can be expressed as

$$p(y|\mathbf{x}) = \sum_{c \in \sigma(r)} p[y|S_c] P[S_c|\mathbf{x}] \qquad (1)$$

where $\sigma(r)$ is the set of all child nodes of r, as in Algorithm 1, S_C denotes the event that the system chooses to visit the child node c next; $P[S_c|\mathbf{x}]$ denotes the probability that the system chooses this node to visit, and $p[y|S_c]$ denotes the probability of label y given the system chooses node S_c.

Consistent with our focus on linear classifiers, the conditional distribution $P[S_c|\mathbf{x}]$ is defined using a multinomial logistic regression model:

$$P[S_c|\mathbf{x}] = \frac{e^{\mathbf{w}_c^\top \mathbf{x}}}{\sum\limits_{i \in \sigma(r)} e^{\mathbf{w}_i^\top \mathbf{x}}}. \qquad (2)$$

It should be noted, though, that any conditional probability distribution could be subsituted here.

This model can be thought of as breaking classification into two steps. At the root node, the classification vectors are applied to the feature vector **x** to determine which child node should be visited. The final label of the example is then determined by the categorical distribution associated with the child node.

3.2. Recursively Expanding the Model

If there are K classes and each node has n children, then the tree will need at least $\log_n K$ levels of child nodes so that there is at least one leaf node for each possible class. Having fewer leaf nodes than classes will guarantee ambiguous results as some leaf nodes will be forced to represent two classes.

New levels can be added to the distribution in Equation

(1) by recursively expanding each $p[y|S_c]$ term. If c^1 represents the child node chosen at the first level and c^2 represents a child node of c^1, a two level model would have the form

$$p(y|\mathbf{x}) = \sum_{c^1 \in \sigma(r)} \sum_{c^2 \in \sigma(c^1)} p[y|S_{c^2}] P[S_{c^2}|S_{c^1}, \mathbf{x}] P[S_{c^1}|\mathbf{x}]. \tag{3}$$

In this case $P[S_{c^2}|S_{c^1}, \mathbf{x}]$ has the same multinomial logistic regression form as Equation (2). Additional levels can be added to this model in a similar fashion.

3.3. Building the Tree Stagewise

Given unlimited computing resources, the label tree parameters could be found by expanding Equation (3) to the desired number of levels and optimizing the log of Equation (3) with a continuous optimization algorithm. However, as the number of levels grows, the number of parameters and complexity of training will grow exponentially. Thus, for practical reasons it is useful to train the classifiers at each node independently.

Returning to the probability at the root node in Equation (1), once the classifier vectors and parameters of the categorical distributions $p[y|S_c]$ have been found, Jensen's Inequality can be applied to the log of Equation (1) to compute a lower bound on $p(y|\mathbf{x})$.

$$\begin{aligned} \log p(y|\mathbf{x}) &= \log \left[\textstyle\sum_{c \in \sigma(r)} p(y|S_c) P[S_c|\mathbf{x}] \right] \\ &\geq \sum_{c \in \sigma(r)} P[S_c|\mathbf{x}] \log \left[p(y|S_c) \right] \end{aligned} \tag{4}$$

Defining this lower bound as L,

$$L = \sum_{c \in \sigma(r)} P[S_c|\mathbf{x}] \log \left[p(y|S_c) \right], \tag{5}$$

the categorical distribution at each of the child nodes can be expanded recursively to have classifiers and a new set of child nodes. Using this bound, the same two level expansion shown in Equation (3) becomes

$$L = \sum_{c^1 \in \sigma(r)} P[S_{c^1}|\mathbf{x}] \log \left[\sum_{c^2 \in \sigma(c^1)} p(y|S_{c^2}) P[S_{c^2}|S_{c^1}, \mathbf{x}] \right] \tag{6}$$

for a single training example.

Each of the terms inside the log in this equation corresponds to one child node. Because each of the terms in the summation in the right hand side of Equation (6) has its own set of parameters, L will be maximized by individually maximizing each of the terms. This decouples the child nodes during training and makes it possible to independently learn the parameters for each of the child nodes.

This also shows that learning the parameters of the label tree can be viewed as a stagewise lower-bound maximization of the log likelihood function for the classification problem.

4. Final Algorithm for Learning a Probabilistic Label Tree

The process of learning the label tree can be viewed as a recursive expansion of nodes. Starting with the root node, each non-leaf node is expanded into a branch node by iteratively performing: (1)Learning the maximum likelihood classifiers based on the categorical distribution of each child node (2) Learning the categorical distribution associated with each child node.

4.1. Learning Parameters for an Expanded Node

For a training set with N training pairs, $(y_i, \mathbf{x}_i)$, this is equivalent to optimizing the log of Equation (1) over all examples, for an arbitrary node.

The overall training process is more easily specified by defining a general loss for expanding a node s, given N training pairs of the form $(y_i, \mathbf{x}_i)$, expressed as:

$$L = \sum_{i=1}^{N} \alpha_i \log \left[\sum_{c \in \sigma(s)} p(y_i|S_c) P[S_c|\mathbf{x}_i] \right] \tag{7}$$

This log likelihood can be maximized using two alternating convex optimizations. In the first step, $p(y_i|S_c)$ is held constant and the optimization is similar to training a multinomial logistic regression model, or softmax classifier. Next, $P[S_c|\mathbf{x}_i]$ is held constant, and the optimization is essentially equivalent to maximum-likelihood estimation of the categorical distribution parameters. We have found that running this optimization for a fixed number of iterations, typically under 10, works well. In our implementation, we use a weighted k-means algorithm to generate an initial set of clusters that can be used to initialize the categorical distributions.

4.2. Formal Specification of the Algorithm

Our formal specification of the algorithm for constructing the label tree will involve two sets of functions. First, we will define $\psi(s)$ to be the set of nodes that must be traversed before arriving at node s, or the path to s. Second, we will define $\lambda()$ to be the set of nodes at a given level of the tree. This will start at 0, with $\lambda(0)$ only containing the root node, $\lambda(1)$ containing the next level of branch nodes, and so on. Algorithm 2 shows the recursive process for expanding the branch nodes.

5. Understanding the Expansion Process

Figure 1 shows examples of how the expansion process operates on the ILSVRC2010 database discussed in Section 7. Figure 1(a) shows the categorical distribution at a branch node in the second level of a T6,4 tree that has four levels, not counting the root node. Figure 1(b) - 1(d) show several of the six child nodes of this branch node. In this next level, the classes with the highest probability at the branch

Class	Prob.
chickpea	0.019
hazelnut	0.019
lentil	0.017
kidney bean	0.016
brussels sprouts	0.016
coffee bean	0.015
clam	0.015
peanut	0.015
plum	0.015
mashed potato	0.014
soy	0.014
orange	0.014
cashew	0.014
lemon	0.014
walnut	0.014
Rem. 985 classes	0.771

(a) Distribution at Parent Node before Expansion

Class	Prob.
hazelnut	0.075
lentil	0.067
kidney bean	0.064
peanut	0.062
cashew	0.057
soy	0.056
coffee bean	0.055
peanut	0.055
green pea	0.055
pumpkin seed	0.054
pistachio	0.051
walnut	0.040
pea	0.038
corn	0.032
bean	0.028
Rem. 985 classes	0.210

(b) Distribution at a Child Node after Expansion

Class	Prob.
chickpea	0.107
mashed potato	0.084
clam	0.081
brussels sprouts	0.074
shrimp	0.071
okra	0.038
french fries	0.035
acorn squash	0.035
broccoli	0.034
cucumber	0.031
spaghetti squash	0.028
celery	0.025
shiitake	0.025
black olive	0.020
bok choy	0.019
Rem. 985 classes	0.292

(c) Distribution at a Child Node after Expansion

Class	Prob.
lemon	0.061
orange	0.060
plum	0.056
persimmon	0.056
guava	0.055
mango	0.055
shallot	0.052
kumquat	0.052
Granny Smith	0.049
turnip	0.047
quince	0.046
bell pepper	0.045
butternut squash	0.032
fig	0.025
spaghetti squash	0.023
Rem. 985 classes	0.285

(d) Distribution at a Child Node after Expansion

Figure 1. This figure visualizes the expansion process for one part of the tree. The table in (a) shows a portion of the categorical distribution at a branch node in the second level of a T6,4 tree that has four levels, excluding the root node. The tables in (b), (c), and (d) show the categorical distributions learned for three of the child nodes. The probability in these child nodes is more concentrated on a subset of the classes than in the parent node. In this result, the tree has not been trained with the pruning techniques in Section.6.1, so the distributions accurately represent the behavior of the maximum-likelihood criterion.

Algorithm 2 Algorithm for Learning Node Parameters

```
Input: $N$ training pairs $(y_i, \mathbf{x}_i)$, maximum number of levels, $L$, branching factor $B$, weight $\alpha$ Test example $\mathbf{x}$, label tree parameters $T, \sigma, l$
1:  for $l = 0 \dots L-1$ do
2:      for all nodes $s$ in $\lambda(l)$ do
3:          Create $B$ child nodes, except at final level (see Sec. 6.2)
4:          $\alpha_i \leftarrow 1 \cdot \prod_{t \in \psi(l)} P[S_t|\mathbf{x}_i], \forall i \in \{1, \dots, N\}$
5:          for all nodes $c$ in $\sigma(s)$ do
6:              for $M$ iterations do
7:                  Fix the parameters for $P[y|S_c]$, maximize Equation (7) over classifier parameters.
8:                  Fix the parameters for classifier parameters $\mathbf{w}$, maximize Equation (7) over the parameters of $P[y|S_c]$.
9:              end for
10:         end for
11:     end for
12: end for
```

node are distributed among the children. The final row in these tables also shows that probability has become more concentrated in the most likely classes.

5.1. Balanced Trees and Efficiency

Increasing classification efficiency is the primary motivation behind the label tree model. For the models based on linear classifiers, this efficiency is best measured by the average number of dot products needed to produce a final label. The number of dot products depends on the number of leaf nodes, which is itself dependent on how balanced the tree is. A perfect balancing of the probabilities for each class across the nodes of the tree would minimize the number of leaf nodes needed.

This raises the question of whether the maximum-likelihood approach proposed here will find a balanced tree. A maximum-likelihood approach attempts to learn a model which induces a label distribution similar to the one in the data. Depending on the underlying distribution, this will not always produce a balanced label tree, and one can certainly construct artificial counter-examples. However, in most natural applications, it is reasonable to assume that the label distribution can be well approximated by a reasonably-balanced label tree, where the leaf nodes distribution concentrate on individual classes.

A simple way to demonstrate this is to show that if the data distribution indeed corresponds to a balanced label tree, then the maximum-likelihood approach would learn a similar balanced tree. Note that this is not altogether trivial: There might be a very unbalanced label tree, which induces the exact same conditional distribution $p(y|\mathbf{x})$ as the balanced label tree, so a maximum-likelihood approach might learn the unbalanced tree instead. To show that this isn't the case, we prove below that our model is *identifiable* - namely, that under mild conditions, given enough data from a distribution induced by a given label tree, then our algorithm would learn the exact same tree.

Theorem 1. *Suppose the training data is sampled i.i.d. from a distribution, such that $p(y|\mathbf{x})$ is generated by some* *label tree, for which* $\mathbf{w}_c \neq w_{c'}$ *for any two sibling nodes* c, c'*. Also, suppose that the support of* $p(\boldsymbol{x})$ *is continuous in some part of the domain. Then as the dataset size increases, the structure and weights of the label tree learned by our algorithm converges to those of the true label tree*

Proof. It is enough to show that we can perfectly reconstruct the root node of the tree - the reconstruction of its child nodes and other nodes in the tree would follow in a similar way by induction. In the limit of infinite data, this boils down to showing that the label distribution $p(y|\mathbf{x})$, which can be written as

$$\sum_{c\in\sigma(r)} p(y|S_c)P[S_c|\mathbf{x}] = \sum_{c\in\sigma(r)} p(y|S_c)\frac{e^{\mathbf{w}_c^\top \mathbf{x}}}{\sum_{i\in\sigma(r)} e^{\mathbf{w}_i^\top \mathbf{x}}}$$

can be induced only by a single choice of the parameters $\{\mathbf{w}_c, p(y|S_c)\}_{c\in\sigma(r)}$. Let us assume on the contrary that there exist some other set of parameters $\{\mathbf{w}'_{c'}, p'(y|S_{c'})\}_{c'\in\sigma'(r)}$ (possibly corresponding to a different number of child nodes), which induce the same distribution, namely

$$\sum_{c\in\sigma(r)} p(y|S_c)\frac{e^{\mathbf{w}_c^\top \mathbf{x}}}{\sum_{i\in\sigma(r)} e^{\mathbf{w}_i^\top \mathbf{x}}} = \sum_{c'\in\sigma'(r)} p'(y|S_{c'})\frac{e^{\mathbf{w}'^\top_{c'} \mathbf{x}}}{\sum_{i'\in\sigma'(r)} e^{\mathbf{w}'^\top_{i'} \mathbf{x}}}$$

for any $\mathbf{x}$ in the support of $p(\mathbf{x})$. Taking a common denominator, and switching sides, this is equivalent to requiring

$$\frac{\sum_{c\in\sigma(r),c'\in\sigma'(r)}(p(y|S_c) - p'(y|S_{c'}))e^{(\mathbf{w}_c+\mathbf{w}'_{c'})^\top \mathbf{x}}}{\sum_{c\in\sigma(r),c'\in\sigma'(r)} e^{(\mathbf{w}_c+\mathbf{w}'_{c'})^\top \mathbf{x}}} = 0.$$

on the support. Since the denominator is always positive, this is equivalent to

$$\sum_{c\in\sigma(r),c'\in\sigma'(r)}(p(y|S_c) - p'(y|S_{c'}))e^{(\mathbf{w}_c+\mathbf{w}'_{c'})^\top \mathbf{x}} = 0.$$

The left hand side is identically zero if $|\sigma(r)| = |\sigma'(r)|$ (namely, there are the same number of child nodes), $\mathbf{w}_c = \mathbf{w}'_{c'}$, and $p(y|S_c) = p'(y|S_{c'})$ for all c, c' (up to permutation of the c, c' labels). If this is not the case, then the equation above can be rewritten as

$$\sum_{i=1}^{N} a_i e^{b_i^\top \mathbf{x}} = 0,$$

for some finite N, distinct $\{b_i\}$, and $\{a_i\}$ such that $a_i \neq 0$. However, if the support of $p(\mathbf{x})$ is dense in some neighborhood, and the equation holds in that domain, then the left hand side can be shown to equal 0 for *all* $\mathbf{x}$ in Euclidean space, which is easily seen to be impossible. Therefore, the parameters that we will learn indeed correspond to the actual label tree. $\square$

6. Implementation

As described in Section 4.1, learning the label tree parameters at each node consists of two alternating steps: learning the classifier weights, then learning the categorical distribution. The classification weights were optimized using the limited memory BFGS (L-BFGS) algorithm. We also experimented with the stochastic meta-descent (SMD) algorithm [5], that has been found to perform better than traditional stochastic gradient descent [17]. We found that using L-BFGS converged faster and found better values for the training criteria.

6.1. Eliminating Samples During Training

In [8], each node is assigned a specific set of classes during the training process. An advantage of this approach is that only the training examples from those classes need to be considered when learning the classifier parameters for that node and children of the node.

In the probabilistic model proposed here, the learning criterion depends on the probability that each example arrives at the node where the parameters are being learned. In practice, this probability is often quite small, but still non-zero, so the learning could require processing all of the training examples at every node in the label tree. To increase the speed of training, the training examples used at a node are pruned to eliminate examples that have a very low probability of reaching the node. This pruning is done independently for each node.

6.2. Fixing the Number of Leaf Nodes

Following [8], the label tree is constructed with a fixed number of levels. Ideally, the learning system would be able to find a perfectly balanced tree and each leaf node would correspond to one class. In practice, it is difficult to find a perfectly balanced tree, so the number of leaf nodes must be determined individually for each branch in the next to last level of the tree. A number of training examples could be assigned to a leaf node with very low probability, so a natural criterion is to assign one leaf per class for the set of classes that account for some percentage, such as 90%, of the probability in the categorical distribution.

For comparison or performance evaluation, it is also useful to have more direct control over the number of leaf nodes. While this threshold could be adjusted to achieve a desired number of leaf-nodes, we found it easier to directly control this number by imposing a hard cap on the number of leaf nodes per branch node. In our experiments, this greatly simplified controlling how many dot products were necessary and performed well.

7. Results

We evaluated our algorithms on both the ILSVRC2010 and ImageNet10k databases used in [8]. In ILSVRC2010, there are 1.2M images from 1k classes for training, 50k images for validation, and 150k images for test. Due to mem-

Method	Flat		T32,2		T10,3		T6,4		T4,5	
	Acc%	S_{te}	Acc%	S_{te}	Acc%	S_{te}	Acc%	S_{te}	Acc%	S_{te}
Trained with ML	-	-	21.38	10.42	20.54	17.85	17.02	31.25	14.98	41.67
Trained with Hard Partition	-	-	20.70	10.37	19.15	17.90	15.76	31.27	14.85	33.84
Results in [8]	-	-	11.9	10.3	8.92	18.2	5.62	31.3	-	
LIBLINEAR	24.84	1	-							

Table 1. Comparison of our method and [8] with different tree configuration on ILSVRC 2010 dataset.Tm, n denotes the tree that has m children per node when branching and n levels. We show the classification accuracy(Acc) and the test time speedup S_{te}. The first row shows the result using our ML based method. The second row is the result using hard label partition and our probabilistic framework as explained in Section 7.2. Our method significantly outperforms [8]. The accuracy of maximum likelihood (ML) is consistently better than the hard label partition with similar speedup. For reference, the last row shows the accuracy produced by a multi-class SVM trained with LIBLINEAR.

ory limitations, we used the first 300 images of each category for training. In ImageNet10k, there are 9M images from 10184 classes. We randomly picked 100 images from each category for training and 50 for testing. This is fewer than used in [8], but demonstrates that our method can scale to large problems.

The features were generated in the same fashion as [8] using the LLC coding strategy outlined in [18]. For each image in ILSVRC2010, we used the VLFeat toolbox [16], which was also used in [8] to extract dense SIFT features from the image. The features were encoded with a codebook with 10,000 entries and the image was encoded using a two-level spatial pyramid[12] with 1×1 and 2×2 grids. This resulted in a feature vector with approximately 50,000 dimensions. In the experiments with the ImageNet10K database, images were represented with vector encoded using LLC, but without a spatial pyramid. The number of image classes in ImageNet10K makes reducing the number of dot-products beneficial, even considering the time needed to generate features. In our experiments computing the classification scores dominated the computation time. The average time for generating features from one image was approximately 0.59 seconds, while computing the one-versus all linear classifier required approximately 2.09 seconds.

In our experiments, we did not make an effort to strictly control the computational complexity during training. Training a tree generally took less than a day on a multi-core machine with 48GB of memory.

7.1. Comparison with Previous Work

Table 1 summarizes the accuracy of our system, compared with the trees trained using the method in [8]. The various columns of Table 1 represent different tree structures. The tree denoted by Tm, n has m children per node when branching and n levels, not including the root node.

To make a fair comparison, we tuned the number of leaf nodes, using the approach described in Section 6.2, so that the trees trained using our approach used a similar average number of dot-products to classify each example. As Table 1 shows, the accuracy rate is significantly higher for the trees that are trained using our approach. Depending on the depth of the tree, our approach classifies images with an accuracy rate that is nearly double or triple the accuracy rates produced using the method from [8]. Our approach is also superior at learning deeper trees. Comparing the accuracies for the $T32, 2$ and $T6, 4$ trees, the reduction in accuracy is less significant in the trees trained using the probabilistic approach.

Method	T101,2		T10,4	
	Acc%	S_{te}	Acc%	S_{te}
Trained with ML	4.77	33.22	3.08	204.54
Results reported in [8]	3.4	32.40	-	

Table 2. Result on Imagenet 10K. While the accuracy numbers cannot be compared directly because of differences in the test set(see text), these results show that our method can scale to large numbers of classes and performs reasonably.

For reference, the last row of Table 1 also reports the classification accuracy of a multi-class one-against-all SVM classifier trained using LIBLINEAR [9]. The trees trained using maximum likelihood produce competitive results while requiring 18 to 30 times less dot products at test time.

We show the results of our method on the ImageNet10K dataset in Table 2. While the accuracies cannot be compared directly because of differences in the test set, these results show that this approach to learning trees can scale to larger problems and that our method produces reasonable performance.

7.2. Evaluating Hard Label Partitioning

One of the most significant differences between this approach and the approach in [8] lies in how the labels are assigned to the leaf nodes. In our probabilistic approach, the training process maintains the probability of each sample reaching a particular node when optimizing that node's parameters. In contrast, the learning approach in [8] uses a partition matrix that is rounded so that all examples from a class either reach the node or do not. We refer to this as *hard label partitioning* because every class, and all of its examples, are assigned a hard binary label describing whether they can reach a node. This partition matrix is found by optimizing a measure of accuracy with hard constraints on an ambiguity measure.

To explore whether this style of partitioning can produce improved results, we evaluated the combination of our probabilistic system with strategy used in [8] to measure whether hard label partitioning is a superior strategy for learning the label tree. This was implemented by replacing the categorical distributions with distributions derived from the partition matrix computed using OP3' in [8]. This matrix can be used to generate new distribution $p(y|S_c)$ by creating a distribution where every node assigned to a specific class is equally likely. For numerical reasons, classes not assigned to a node are given a very small probability. The result is a two-stage alternating algorithm that consists of learning a linear classifier in the form of a multinomial logistic regression classifier, then using the classifications from that classifier to learn the partition matrix.

The ambiguity parameters during learning were manually tuned so that the trees used a comparable average number of dot products to classify examples. Below, Section 7.3 will discuss performance across various parameter settings. As the second row of Table 1 shows, training the tree just using the maximum likelihood criterion consistently outperforms this approach. In addition, as the trees become deeper, the advantage of the maximum likelihood approach increases.

This result also provides insight into the significant differences in performance in Table 1. Given that using OP3' from [8] reduced accuracy, but not did dramatically change results, the combination of the multinomial logistic regression models with an L-BFGS optimization system likely accounts for the largest increase in performance over the results in [8]. The system in [8] uses several passes of parallel Stochastic Gradient Descent [20] to learn the classifiers. In our experiments, we found that using a stochastic descent algorithms led to far worse classification weights than when the quasi-Newton L-BFGS algorithm was used. It should be noted, though, that the criterion in OP2 in [8] is non-differentiable at points, due to the use of a max operator, so the system in [8] cannot be simply modified to use L-BFGS instead of stochastic gradient optimization.

7.3. Exploring the Accuracy-Efficiency Trade-off

One of the weaknesses of the classification procedure in a label tree is that the classifier must make a hard choice to decide which branch to follow. In cases where there is ambiguity in the correct branch, this hard choice will degrade recognition accuracy if the wrong branch is chosen. An advantage of the probabilistic formulation is that it facilitates performing a limited search over multiple branches of the label tree.

In this limited search, the classifier does not just follow the best-scoring branch. Instead, it makes a decision at each branch whether to follow just the highest-scoring branch or both the highest-scoring and the second highest-scoring branches. This decision is made by comparing the difference in scores between the two branches. If this difference is below a threshold, both branches are followed. The final classification is found by adding together the categorical distributions at each of the leaf nodes that the classification algorithms reaches.

As this threshold varies, the average number of dot-products needed to produce a classification also varies. The green curves in Figures 2 and 3 show the relationship in different tree models between the average number of dot-products needed to classify a sample and the resulting accuracy. Notice that as the classification process explores more branches, the accuracy rises.

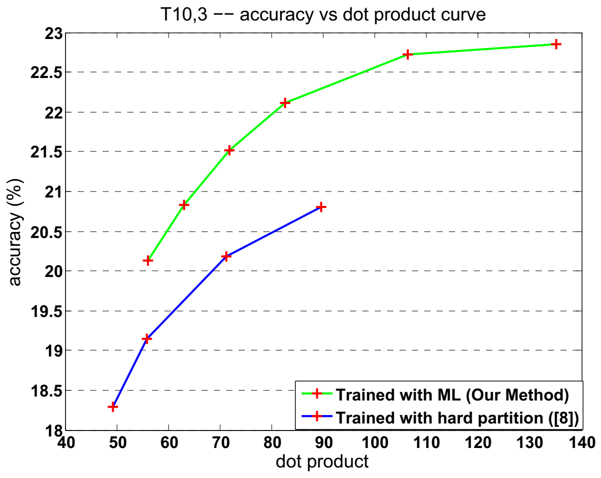

Figure 2. Accuracy vs dot product curve for our T10,3 tree. The green curve shows our method using maximum likelihood with multinomial estimation. The blue curve shows the result using hard label partition[8] with our framework. The maximum likelihood method has higher classification accuracy with less average dot products needed at test time

.

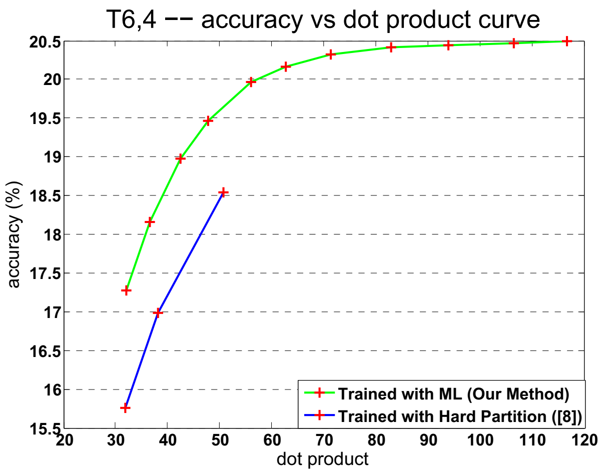

Figure 3. Accuracy vs dot product curve for our T6,4 tree. Again, the tree trained with the maximum likelihood approach has consistently higher accuracy.

For comparison, the blue curves in these figures show a similar curve that is computed by using the hybrid systems described in Section 7.2 and varying the ambiguity limits. These curves is shorter because we found that as the ambiguity limits increased, the number of classes at deep levels of the tree increased dramatically and the time required to train the system became untenable. In both figures, the tree learned with the probabilistic approach provides better accuracy for a similar average number of dot products.

It is also important to note that creating this curve with our model does not require that tree parameters be retrained. This approach can be used to adjust the accuracy/efficiency trade-off on an existing tree. On the other hand, using the optimization with ambiguity constraints requires that the tree be re-trained for different parameters.

8. Conclusion

In this work, we propose a probabilistic label tree framework to accelerate large scale classification problems. Since our method is totally based on probability and maximum likelihood optimization, it can be adapted to different types of probabilistic classifiers. Our experiments show that learning a label tree in this fashion can improve recognition accuracy with comparable speedups to previous work.

Acknowledgements

This work was supported by NSF grants IIS-0905387, IIS-0916868, and IIS-1212948. Thanks to the UCF STOKES High Performance Computing Cluster for providing powerful computing resources.

References

[1] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, pages 163–171, 2010. 1, 2

[2] A. Beygelzimer, J. Langford, Y. Lifshits, G. Sorkin, and A. Strehl. Conditional probability tree estimation analysis and algorithms. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 51–58. AUAI Press, 2009. 2

[3] A. Beygelzimer, J. Langford, and P. Ravikumar. Multiclass classification with filter trees. *Preprint, June*, 2007. 2

[4] J. K. Bradley and C. Guestrin. Learning tree conditional random fields. In *Proceedings of the International Conference on Machine Learning (ICML-2010)*, 2010. 2

[5] M. Bray, E. Koller-Meier, P. Müller, L. Van Gool, and N. Schraudolph. 3d hand tracking by rapid stochastic gradient descent using a skinning model. In *In 1st European Conference on Visual Media Production (CVMP*. Citeseer, 2004. 5

[6] M. J. Choi, J. J. Lim, A. Torralba, and A. S. Willsky. Exploiting hierarchical context on a large database of object categories. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 129–136. IEEE, 2010. 2

[7] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 1

[8] J. Deng, S. Satheesh, A. C. Berg, and F.-F. Li. Fast and balanced: Efficient label tree learning for large scale object recognition. In *Advances in Neural Information Processing Systems*, pages 567–575, 2011. 1, 2, 5, 6, 7

[9] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. 6

[10] T. Gao and D. Koller. Discriminative learning of relaxed hierarchy for large-scale visual recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2072–2079. IEEE, 2011. 2

[11] G. Griffin and P. Perona. Learning and using taxonomies for fast visual categorization. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 2

[12] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2169–2178. IEEE, 2006. 6

[13] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, K. Yu, L. Cao, and T. Huang. Large-scale image classification: fast feature extraction and svm training. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1689–1696. IEEE, 2011. 1

[14] J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1665–1672. IEEE, 2011. 1

[15] A. Torralba, R. Fergus, and W. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(11):1958–1970, 2008. 1

[16] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. `http://www.vlfeat.org/`, 2008. 6

[17] S. Vishwanathan, N. Schraudolph, M. Schmidt, and K. Murphy. Accelerated training of conditional random fields with stochastic meta-descent. In *International Conference on Machine Learning (ICML '06)*, 2006. 5

[18] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3360–3367. IEEE, 2010. 6

[19] X. Zhou, K. Yu, T. Zhang, and T. S. Huang. Image classification using super-vector coding of local image descriptors. In *Computer Vision–ECCV 2010*, pages 141–154. Springer, 2010. 1

[20] M. Zinkevich, M. Weimer, A. Smola, and L. Li. Parallelized stochastic gradient descent. *Advances in Neural Information Processing Systems*, 23(23):1–9, 2010. 7