arXiv
(2017)

Hierarchical loss for classification

Cinna Wu, Mark Tygert

Facebook Artificial Intelligence Research
1 Facebook Way
Menlo Park, CA 94025
e-mail: cinna@fb.com*;* tygert@fb.com

Yann LeCun

Facebook Artificial Intelligence Research
770 Broadway
New York, NY 10003
e-mail: yann@fb.com

Abstract: Failing to distinguish between a sheepdog and a skyscraper should be worse and penalized more than failing to distinguish between a sheepdog and a poodle; after all, sheepdogs and poodles are both breeds of dogs. However, existing metrics of failure (so-called "loss" or "win") used in textual or visual classification/recognition via neural networks seldom view a sheepdog as more similar to a poodle than to a skyscraper. We define a metric that, inter alia, can penalize failure to distinguish between a sheepdog and a skyscraper more than failure to distinguish between a sheepdog and a poodle. Unlike previously employed possibilities, this metric is based on an ultrametric tree associated with any given tree organization into a semantically meaningful hierarchy of a classifier's classes.

MSC 2010 subject classifications: Primary 62H30; secondary 68T05, 65K10.
Keywords and phrases: Hierarchy, taxonomy, ontology, tree, classifier, categorization.

1. Introduction

Metrics for classifier accuracy used in the neural network methods of LeCun, Bengio and Hinton (2015) seldom account for semantically meaningful organizations of the classes; these metrics neglect, for instance, that sheepdogs and poodles are dogs, that dogs and cats are mammals, that mammals, birds, reptiles, amphibians, and fish are vertebrates, and so on. Below, we define a metric — the amount of the "win" or "winnings" for a classification — that accounts for a given organization of the classes into a tree. During an optimization (also known as "training"), we want to maximize the win or, equivalently, minimize the "loss" (loss is the negative of the win). We caution that some of our experiments indicate that plain stochastic gradient descent optimization with random starting points can get stuck in local optima; even so, the hierarchical win can serve as a good metric of success or figure of merit for the accuracy of the classifier.

imsart ver. 2014/10/16 file: arxiv.tex date: September 6, 2017

The approach detailed below is a special case of the general methods of Cai and Hofmann (2004), Kosmopoulos et al. (2015), and their references. The particular special cases discussed by Binder, Kawanabe and Brefeld (2009), Cai and Hofmann (2004), Chang and Lee (2015), Costa et al. (2007), Deng et al. (2010), Deng et al. (2011), Kosmopoulos et al. (2015), and Wang, Zhou and Liew (1999) allocate different weights to different leaves, not leveraging "ultrametric trees" as in the present paper (an ultrametric tree is a tree with a so-called ultrametric distance metric such that all leaves are the same distance from the root, as in the phylogenetics discussed, for example, by Reece et al. (2013)). A related topic is hierarchical classification, as reviewed by Silla, Jr. and Freitas (2011) (with further use more recently by Kosmopoulos, Paliouras and Androutsopoulos (2015) and Redmon and Farhadi (2017), for example); however, the present paper considers only classification into a given hierarchy's finest-level classes. The design options for classification into only the finest-level classes are more circumscribed, yet such classification is easier to use, implement, and interface with existing codes.

The remainder of the present paper has the following structure: Section 2 constructs the hierarchical loss and win. Section 3 details refinements useful when optimizing based on hierarchical loss or win. Section 4 illustrates and evaluates the hierarchical loss and win via several numerical experiments. Section 5 draws conclusions and proposes directions for future work.

2. Construction and calculation of the hierarchical loss or win

Concretely, suppose that we want to classify each input into one of many classes, and that these classes are the leaves in a tree which organizes them into a semantically meaningful hierarchy. Suppose further that a classifier maps an input to an output probability distribution over the leaves, hopefully concentrated on the leaf corresponding to the correct class for the input. We define the probability of any node in the tree to be the sum of the probabilities of all leaves falling under the node (the node represents an aggregated class consisting of all these leaves); a leaf falls under the node if the node is on the path from the root to the leaf. We then define the amount of the "win" or "winnings" to be the weighted sum (with weights as detailed shortly) of the probabilities of the nodes along the path from the root to the leaf corresponding to the correct class.

To calculate the win, we sum across all nodes on the path from the root to the leaf corresponding to the correct class, including both the root and the leaf, weighting the probability at the first node (that is, at the root) by $1/2$, weighting at the second node by $1/2^2$, weighting at the third node by $1/2^3$, weighting at the fourth node by $1/2^4$, and so on until the final leaf. We then add to this sum the probability at the final leaf, weighted with the same weight as in the sum, that is, we double-count the final leaf. We justify the double-counting shortly.

To compute the probability of each node given the probability distribution over the leaves, we propagate the leaf probabilities through the tree as follows. We begin by storing a zero at each node in the tree. Then, for each leaf, we add

Fig 1. *An algorithm for propagating probabilities*

Input: a discrete probability distribution over the leaves of a tree

Output: a scalar value (the total probability) at each node in the tree

Procedure:
Store the value 0 at each node in the tree.
For each leaf,
  for each node in the path from the root to the leaf (including both the root and the leaf),
    add to the value stored at the node the probability of the leaf.

Fig 2. *An algorithm for computing the "win" or "winnings"*

Input: two inputs, namely (1) a discrete probability distribution over the leaves of a tree, and (2) which leaf corresponds to a completely correct classification

Output: a single scalar value (the "win" or "winnings")

Procedure:
Run the procedure in Figure 1 to obtain a scalar value at each node in the tree.
Store the value 0 in an accumulator.
For $j = 1, 2, \ldots, \ell$,
  where ℓ is the number of nodes in the path from the root (node 1) to the leaf (node ℓ),
  add to the accumulated value 2^{-j} times the value stored at the path's jth node.
Add to the accumulated value $2^{-\ell}$ times the value stored at the path's ℓth node (the leaf).
Return the final accumulated value.

the probability associated with the leaf to the value stored at each node on the path from the root to the leaf, including at the root and at the leaf. We save these accumulated values as the propagated probabilities (storing the value 1 at the root — the sum of all the probabilities is 1 of course).

Thus, if the probability of the final leaf is 1, then the win is 1. The win can be as large as 1 (this happens when the classification is completely certain and correct) or as small as 1/2 (this happens when the classification is as wrong as possible). The win being 1 whenever the probability of the final leaf is 1 — irrespective of which is the final leaf — means that the weights form an "ultrametric tree," as in the phylogenetics discussed, for example, by Reece et al. (2013). This justifies double-counting the final leaf.

Figures 1 and 2 summarize in pseudocode the algorithms for propagating the probabilities and for calculating the win, respectively (the latter algorithm runs the former as its initial step).

To facilitate optimization via gradient-based methods (such as the stochastic gradient descent used by Joulin et al. (2017)), we now detail how to compute the gradient of the hierarchical win with respect to the input distribution: Relaxing the constraint that the distribution over the leaves be a probability distribution, that is, that the "probabilities" of the leaves be nonnegative and sum to 1, the algorithms specified above yield a value for the win as a function of any distribution over the leaves. Without the constraint that the distribution over the

FIG 3. *An algorithm for choosing the single best leaf*

Input: a discrete probability distribution over the leaves of a tree

Output: a single leaf of the tree

Procedure:
Run the procedure in Figure 1 to obtain a scalar value at each node in the tree.
Move to the root.
Repeating until at a leaf,
  follow (to the next finer level) the branch containing the greatest among all scalar values stored at the current level in the current subtree.
Return the final leaf.

leaves be a probability distribution, the win is actually a linear function of the distribution, that is, the win is the dot product between the input distribution and another vector (where this other vector depends on which leaf corresponds to the correct classification); the gradient of the win with respect to the input is therefore just the vector in the dot product. An entry of this gradient vector, say the jth entry, is equal to the win for the distribution over the leaves that consists of all zeros except for one value of one on the jth leaf; this win is clearly equal to the sum of the series $1/2 + 1/2^2 + 1/2^3 + 1/2^4 + \dots$, truncated to the number of terms equal to the number of nodes for which the path from the root to the correct leaf and the path from the root to the jth leaf coincide (or not truncated at all if the jth leaf happens to be the same as the leaf for a correct classification).

Remark 1. If forced to choose a single class corresponding to a leaf of the given hierarchy (rather than classifying into a probability distribution over all leaves) when optimizing the hierarchical loss, we first identify the node having greatest probability among all nodes (including leaves) at the coarsest level, then the node having greatest probability among all nodes (including leaves) falling under the first node selected, then the node having greatest probability among all nodes (including leaves) falling under the second node selected, and so on, until we select a leaf. The leaf we select corresponds to the class we choose. Figure 3 summarizes in pseudocode this procedure for selecting a single best leaf.

3. Logarithms when using a softmax or independent samples

In order to provide appropriately normalized results, the input to the hierarchical loss needs to be a (discrete) probability distribution, not just an arbitrary collection of numbers. A "softmax" provides a good, standard means of converting any collection of real numbers into a proper probability distribution. Recall that the softmax of a sequence x_1, x_2, …, x_n of n numbers is the normalized sequence $\exp(x_1)/Z$, $\exp(x_2)/Z$, …, $\exp(x_n)/Z$, where $Z = \sum_{k=1}^{n} \exp(x_k)$. Notice that each of the normalized numbers lies between 0 and 1, and the sum of the numbers in the normalized sequence is 1 — the normalized sequence is a proper probability distribution.

When generating the probability distribution over the leaves via a softmax, we should optimize based on the logarithm of the "win" introduced above rather than the "win" itself. In this case, omitting the contribution of the root to the objective value and its gradient makes the most sense, ensuring that a flat hierarchy (that is, a hierarchy which has only one level aside from the root's) results in the same training as with the usual cross-entropy loss. Henceforth, we omit the contribution of the root to the hierarchical wins and losses that we report, and we multiply by 2 the resulting win, so that its minimal and maximal possible values become 0 and 1 (with 0 corresponding to the most incorrect possible classification, and with 1 corresponding to the completely correct classification). Taking the logarithm also makes sense because the joint probability of stochastically independent samples is the product of the probabilities of the individual samples, making averaging (across the different samples) the logarithm of a function (the function could be the win) make more sense than averaging (across the samples) the function directly. That said, taking the logarithm emphasizes highly misclassified samples, which may not be desirable if misclassifying a few samples (while simultaneously reporting high confidence in their classification) should be acceptable.

Indeed, if the logarithm of the win for even a single sample is infinite, then the average of the logarithm of the win is also infinite, irrespective of the values for other samples. Whether the hierarchy is full or flat, training on the logarithms of wins is very stringent, whereas the wins without the logarithms can be more meaningful as metrics of success or figures of merit. It can make good sense to train on the logarithm, which works really hard to accommodate and learn from the hardest samples, but to make the metric of success or figure of merit be robust against such uninteresting outliers. Thus, training with the logarithm of the win can make good sense, where the win — without the logarithm — is the metric of success or figure of merit for the testing or validation stage.

4. Numerical experiments

We illustrate the hierarchical loss and its performance using supervised learning for text classification with fastText of Joulin et al. (2017). For all experiments, we hashed bigrams (pairs of words) into a million buckets and trained using stochastic gradient descent, setting the learning rate to start at the values indicated in the tables (these worked about as well as any other settings), then decaying linearly to 0 over the course of training, as in the work of Joulin et al. (2017); the starting point for the stochastic gradient descent was random. The "learning rate" is also known as the "step size" or "step length" in the update for each iteration (step) of stochastic gradient descent. When using the hierarchy to inform the training, we follow Section 3, maximizing the hierarchical win (or its logarithm) calculated without any contribution from the root (excluding the root makes a small difference when taking the logarithm).

Our implementation couples the C++ software of Joulin et al. (2017) with a Python prototype. An industrial deployment would require acceleration of the Python prototype (rewriting in C++, for instance), but our codes are sufficient for estimating the ensuing gains in accuracy and illustrating the figure of merit, providing a proof of principle. In particular, our experiments indicate that the gains in accuracy due to training with the hierarchical loss are meager except in special circumstances detailed below and summarized in the conclusion, Section 5. Pending further development as suggested in Section 5, the main present use for the hierarchical win should be as a metric of success or figure of merit — a good notion of "accuracy" — at least when training with plain stochastic gradient descent coupled to backpropagation.

In the tables, the columns "training loss, rate, epochs" list the following three parameters for training: (1) the form of the loss function used during training (as explained shortly), (2) the initial learning rate which tapers linearly to 0 over the course of training, and (3) the total number of sweeps through the data performed during training (too many sweeps results in overfitting). The training loss "flat" refers to training using the usual cross-entropy loss, which is the same as the negative of the natural logarithm of the hierarchical win when using a flat hierarchy in which all labelable classes are leaves attached to the root (as discussed in Section 3). The training loss "raw" refers to training using the hierarchical loss, using the full hierarchy. The training loss "log" refers to training using the negative of the natural logarithm of the hierarchical win, using the full hierarchy. The training loss "coarse" refers to training using the usual cross-entropy loss for classification into only the coarsest (aggregated) classes in the hierarchy (based on a suitably smaller softmax for input to the loss). The values reported in the tables for the learning rate and number of epochs yielded among the best results for the accuracies discussed in the following paragraphs.

The columns "one-hot win via hierarchy" display the average over all testing samples of the hierarchical win fed with the results of a one-hot encoding of the class chosen according to Remark 1 of Section 2. (The one-hot encoding of a class is the vector whose entries are all zeros aside from a single entry of one in the position corresponding to the class.) The columns "softmax win via hierarchy" display the average over all testing samples of the hierarchical win fed with the results of a softmax from fastText of Joulin et al. (2017) (Section 3 above reviews the definition of "softmax"). The columns "$-$log of win via hierarchy" display the average over all testing samples of the negative of the natural logarithm of the hierarchical win fed with the results of a softmax from fastText (Section 3 above reviews the definition of "softmax"). The columns "cross entropy" display the average over all testing samples of the usual cross-entropy loss, which is the same as the negative of the natural logarithm of the hierarchical win fed with the results of a softmax when using a "flat" hierarchy in which all labelable classes are leaves attached to the root (as discussed in Section 3).

The columns "coarsest accuracy" display the fraction of testing samples for which the coarsest classes containing the fine classes chosen during the classification are correct, when classifying each sample into exactly one class, as in Remark 1 of Section 2. The columns "parents' accuracy" display the fraction of

TABLE 1
Results on RCV1-v2, tested on 5,000 samples

training loss, rate, epochs	one-hot win via hierarchy	softmax win via hierarchy	$-$log of win via hierarchy	cross entropy
flat, 2, 4	.85	.80	.52	.95
raw, 12, 4	.51	.50	4.2	∞
log, 4, 4	.74	.72	.76	4.4
the ideal	higher	higher	lower	lower

training loss, rate, epochs	coarsest accuracy	parents' accuracy	finest accuracy
flat, 2, 4	.88	.82	.80
raw, 12, 4	.74	.26	.21
log, 4, 4	.87	.63	.53
coarse, .05, 100	.88		
the ideal	higher	higher	higher

TABLE 2
Results on RCV1-v2 with at most one training sample per class, tested on 5,000 samples

training loss, rate, epochs	one-hot win via hierarchy	softmax win via hierarchy	$-$log of win via hierarchy	cross entropy
flat, .04, 9	.14	.07	2.8	5.5
raw, 45, 500	.16	.15	7.2	13
log, 3, 40	.17	.09	2.7	5.5
the ideal	higher	higher	lower	lower

training loss, rate, epochs	coarsest accuracy	parents' accuracy	finest accuracy
flat, .04, 9	.209	.086	.051
raw, 45, 500	.235	.095	.052
log, 3, 40	.258	.089	.064
coarse, 4, 1000	.324		
the ideal	higher	higher	higher

testing samples for which the parents of the classes chosen during the classification are correct, when classifying each sample into exactly one class; the parents are the same as the coarsest classes in Tables 3 and 4, as the experiments reported in Tables 3 and 4 pertain to hierarchies with only two levels (excluding the root). The columns "finest accuracy" display the fraction of testing samples classified correctly, when classifying each into exactly one finest-level class, again as in Remark 1.

The last lines of the tables remind the reader that the best classifier would have higher one-hot win via hierarchy, higher softmax win via hierarchy, lower $-$log of win via hierarchy, lower cross entropy, higher coarsest accuracy, higher parents' accuracy, and higher finest accuracy.

Sections 4.1–4.6 detail our experiments and data sets.

Table 3
Results on Yahoo Answers, tested on 60,000 samples

training loss, rate, epochs	one-hot win via hierarchy	softmax win via hierarchy	−log of win via hierarchy	cross entropy
flat, .1, 4	.76	.67	.67	.91
raw, 1, 4	.65	.64	2.6	∞
log, .1, 4	.76	.67	.68	1.0
the ideal	higher	higher	lower	lower

training loss, rate, epochs	coarsest accuracy	finest accuracy
flat, .1, 4	.80	.72
raw, 1, 4	.79	.50
log, .1, 4	.80	.71
coarse, .1, 4	.80	
the ideal	higher	higher

Table 4
Results on DBpedia, tested on 70,000 samples

training loss, rate, epochs	one-hot win via hierarchy	softmax win via hierarchy	−log of win via hierarchy	cross entropy
flat, .5, 4	.99	.99	.034	.054
raw, 1, 4	.81	.81	.312	6.06
log, .5, 4	.99	.99	.036	.063
the ideal	higher	higher	lower	lower

training loss, rate, epochs	coarsest accuracy	finest accuracy
flat, .5, 4	.992	.986
raw, 1, 4	.989	.636
log, .5, 4	.992	.985
coarse, .3, 4	.992	
the ideal	higher	higher

Table 5
Results on DBpedia fish, tested on 6,000 samples

training loss, rate, epochs	one-hot win via hierarchy	softmax win via hierarchy	−log of win via hierarchy	cross entropy
flat, 5, 25	.27	.25	4.4	16
raw, 200, 200	.15	.15	21	∞
log, 5, 50	.38	.36	3.5	19
the ideal	higher	higher	lower	lower

training loss, rate, epochs	coarsest accuracy	parents' accuracy	finest accuracy
flat, 5, 25	.38	.26	.065
raw, 200, 200	.28	.03	.000
log, 5, 50	.60	.22	.014
coarse, 3, 70	.60		
the ideal	higher	higher	higher

Table 6
Results on a subset of LSHTC1, tested on 2,000 samples

training loss, rate, epochs	one-hot win via hierarchy	softmax win via hierarchy	−log of win via hierarchy	cross entropy
flat, 6, 1000	.43	.36	1.6	5.0
raw, 20000, 1000	.23	.23	.35	∞
log, 15, 15	.36	.30	1.7	5.5
the ideal	higher	higher	lower	lower

training loss, rate, epochs	coarsest accuracy	parents' accuracy	finest accuracy
flat, 6, 1000	.61	.33	.23
raw, 20000, 1000	.46	.24	.01
log, 15, 15	.66	.12	.01
coarse, 5, 10000	.69		
the ideal	higher	higher	higher

4.1. RCV1-v2

Table 1 reports results on RCV1-v2 of Lewis et al. (2004). This dataset includes a hierarchy of 364 classes (semantically, these are classes of industries); each sample from the dataset comes associated with at least one of these 364 class labels, whether or not the class is an internal node of the tree or a leaf. Each sample from the dataset consists of filtered, tokenized text from Reuters news articles ("article" means the title and body text). As described by Lewis et al. (2004), labels associated with internal nodes in the original hierarchy may be viewed as leaves that fall under those internal nodes while not classifying into any of the lower-level nodes. In our hierarchy, we hence duplicate every internal node into an "other" class under that node, such that the "other" class is a leaf.

We discard every sample from the dataset associated with more than one label, and swap the training and testing sets (since the original training set is small, whereas the original testing set is large). Furthermore, we randomly permute all samples in both the training and testing sets, and subsample to 5,000 samples for testing and 200,000 for training. The hierarchy has 10 coarsest classes, 61 parents of the leaves that were used in the training set, and 254 leaves that were used. We embedded the words and bigrams (words are unigrams) into a 1,000-dimensional space, following Joulin et al. (2017) (the results display little dependence on this dimension). For training the classifier into only the coarsest (aggregated) classes, we embedded the words and bigrams into a 20-dimensional space.

For this data set, optimizing based on the hierarchical loss (with or without a logarithm) yields worse accuracy according to all metrics considered compared to optimizing based on the standard cross-entropy loss (cross entropy is the same as the negative of the logarithm of the hierarchical win with a flat hierarchy).

4.2. Subsampled RCV1-v2

Table 2 reports results on the same RCV1-v2 of Section 4.1, but retaining only one training sample for each class label. The training set thus consists of 254 samples (many of the 364 possible class labels had no corresponding samples in the training set from Section 4.1). The hierarchy has 10 coarsest classes, 61 parents of the leaves that were used in the training set, and 254 leaves that were used. We embedded the words and bigrams (words are unigrams) into a 1,000-dimensional space, following Joulin et al. (2017) (the results display little dependence on this dimension). For training the classifier into only the coarsest (aggregated) classes, we embedded the words and bigrams into a 20-dimensional space.

For this data set, optimizing based on the negative of the natural logarithm of the hierarchical win yields better accuracy according to all metrics considered compared to optimizing based on the standard cross-entropy loss (cross entropy is the same as the negative of the logarithm of the hierarchical win with a flat hierarchy), except on the negative of the natural logarithm of the hierarchical win and the cross-entropy loss (for which the accuracies are similar).

4.3. Yahoo Answers

Table 3 reports results on the Yahoo Answers subset introduced by Zhang, Zhao and LeCun (2015). This dataset includes 10 classes (semantically, these are classes of interest groups); each sample from the dataset comes associated with exactly one of these 10 class labels. Each sample from the dataset consists of normalized text from questions and answers given on a website devoted to Q&A. For the nontrivial hierarchy, we grouped the 10 classes into 4 superclasses. We embedded the words and bigrams (words are unigrams) into a 20-dimensional space, following Joulin et al. (2017) (the results display little dependence on this dimension). For training the classifier into only the coarsest (aggregated) classes, we embedded the words and bigrams into a 10-dimensional space.

With only 10 classes and two levels for the classification hierarchy, Table 3 indicates that training with or without the hierarchical loss makes little difference.

4.4. DBpedia

Table 4 reports results on the DBpedia subset introduced by Zhang, Zhao and LeCun (2015). This dataset includes 14 classes (semantically, these are categories from DBpedia); each sample from the dataset comes associated with exactly one of these 14 class labels. Each sample from the dataset consists of normalized text from DBpedia articles ("article" means the title and body text). For the nontrivial hierarchy, we grouped the 14 classes into 6 superclasses. We embedded the words and bigrams (words are unigrams) into a 20-dimensional space, following Joulin et al. (2017) (the results display little dependence on this dimension). For training the classifier into only the coarsest (aggregated) classes, we embedded the words and bigrams into a 10-dimensional space.

With only 14 classes and two levels for the classification hierarchy, Table 4 indicates that training with or without the hierarchical loss makes little difference.

4.5. DBpedia fish

Table 5 reports results on the subset corresponding to fish from the DBpedia of Lehmann et al. (2015). This dataset includes 1,298 classes (semantically, these are taxonomic groups of fish, such as species containing sub-species, genera containing species, or families containing genera — DBpedia extends to different depths of taxonomic rank for different kinds of fish; our classes are the parents of the leaves in the DBpedia tree). Each sample from the dataset consists of normalized text from the lead section (the introduction) of the Wikipedia article on the associated type of fish, with all sub-species, species, genus, family, and order names removed from the associated Wikipedia article (DBpedia derives from Wikipedia, as discussed by Lehmann et al. (2015)). For each of our finest-level classes, we chose uniformly at random one leaf in the DBpedia taxonomic tree of fish to be a sample in the training set, reserving the other leaves for the testing set (the testing set consists of a random selection of 6,000 of these leaves). The hierarchy has 94 coarsest classes, 367 parents of the leaves in our tree, and 1,298 leaves in our tree. We embedded the articles' words and bigrams (words are unigrams) into a 2,000-dimensional space, following Joulin et al. (2017) (the results display little dependence on this dimension). Optimizing the hierarchical win — without any logarithm — was wholly ineffective, always resulting in assigning the same finest-level class to all input samples (with the particular class assigned varying according to the extent of training and the random starting point). So taking the logarithm of the hierarchical win was absolutely necessary to train successfully. For training the classifier into only the coarsest (aggregated) classes, we embedded the words and bigrams into a 200-dimensional space.

For this data set, optimizing based on the negative of the logarithm of the hierarchical win yields much better coarsest accuracy and hierarchical wins than optimizing based on the standard cross-entropy loss (cross entropy is the same as the negative of the logarithm of the hierarchical win with a flat hierarchy), while optimizing based on the standard cross-entropy loss yields much better finest accuracy and cross entropy. When optimizing based on the negative of the natural logarithm of the hierarchical win, the accuracy on the coarsest aggregates reaches that attained when optimizing the coarse classification directly.

4.6. LSHTC1

Table 6 reports results on a subset of the LSHTC1 dataset introduced by Partalas et al. (2015). The subset considered consists of the subtree for class 3261;

this subtree includes 18 coarsest classes (though 3 of these have no corresponding samples in the testing or training sets) and 364 finest-level classes (with 288 of these having corresponding samples in the testing and training sets). We reserved one sample per finest-level class for training; all other samples were for testing, and we chose 2,000 of these uniformly at random to form the testing set. Each sample from the dataset consists of normalized, tokenized text in extracts from Wikipedia, the popular crowdsourced online encyclopedia. The hierarchy has 18 coarsest classes (with 15 actually used), 111 parents of the leaves that were used in the training set, and 288 leaves that were used. We embedded the words and bigrams (words are unigrams) into a 1,000-dimensional space, following Joulin et al. (2017) (the results display little dependence on this dimension). For training the classifier into only the coarsest (aggregated) classes, we embedded the words and bigrams into a 20-dimensional space.

For this data set, optimizing based on the hierarchical loss (with or without a logarithm) yields worse accuracy according to all metrics except the accuracy on the coarsest aggregates, compared to optimizing based on the standard cross-entropy loss (cross entropy is the same as the negative of the logarithm of the hierarchical win with a flat hierarchy). When optimizing based on the negative of the natural logarithm of the hierarchical win, the accuracy on the coarsest aggregates approaches its maximum attained when optimizing the coarse classification directly.

5. Conclusion

In our experiments, optimizing the hierarchical loss (or, rather, the negative of the logarithm of the hierarchical win) using plain stochastic gradient descent with backpropagation could be helpful relative to optimizing the usual cross-entropy loss, but mainly just when there were at most a few training samples per class (in principle, the training set can still be big, if there are many classes . . . though evaluating the case with many, many classes would require software and algorithmic development well beyond the scope of the present paper). In particular, this provides a very limited partial solution to the problem of "open world" classification, classification in which the testing set includes samples from classes not represented in the training set. This also has direct relevance to the problem of "personalization," which often involves limited data per individual class (even though there may be many, many individuals).

The experiments reported above may be summarized as follows: relative to training on the usual cross-entropy loss, training on the negative of the logarithm of the hierarchical win hurt in all respects in Section 4.1, helped in all respects in Section 4.2, improved coarse accuracy as much as optimizing directly for coarse classification in Section 4.5, hurt in most respects in Section 4.6 while improving coarse accuracy nearly as much as optimizing directly for coarse classification, and made essentially no difference in Sections 4.3 and 4.4. Thus, whether optimizing with a hierarchical loss makes sense depends on the data set and associated hierarchy.

Even so, optimizing hierarchical loss using plain stochastic gradient descent with backpropagation (as we did) is rather ineffective, at least relative to what might be possible. We trained using stochastic gradient descent with a random starting point, which may be prone to getting stuck in local optima. To some extent, hierarchical loss collapses the many classes in the hierarchy into a few aggregate superclasses, and the parameters being optimized within the aggregates should be tied closely together during the optimization — plain stochastic gradient descent is unlikely to discover the benefits of such tying, as plain stochastic gradient descent does not tie together these parameters in any way, optimizing all of them independently. Optimizing the hierarchical loss would presumably be more effective using a hierarchical process for the optimization, which could be a good subject for future work. The hierarchical optimization could alter stochastic gradient descent explicitly into a hierarchical process, or could involve regularization terms penalizing variance in the parameters associated with the leaves in the same coarse aggregate. For the time being, hierarchical loss is most useful as a metric of success, gauging the performance of a fully trained classifier as a semantically meaningful figure of merit.

Acknowledgements

We would like to thank Priya Goyal, Anitha Kannan, and Brian Karrer for their generous and plentiful help.

References

Binder, A., Kawanabe, M. and Brefeld, U. (2009). Efficient classification of images with taxonomies. In *Proc. 9th Asian Conf. Computer Vision. Lecture Notes in Computer Science* **5996** 351–362. Springer.

Cai, L. and Hofmann, T. (2004). Hierarchical document categorization with support vector machines. In *Proc. 13th ACM Internat. Conf. Information and Knowledge Management* 78–87. ACM.

Silla, Jr. , C. N. and Freitas, A. A. (2011). A survey of hierarchical classification across different application domains. *J. Data Mining Knowledge Discovery* **22** 31–72.

Chang, J. Y. and Lee, K. M. (2015). Large margin learning of hierarchical semantic similarity for image classification. *Computer Vision and Image Understanding* **132** 3–11.

Costa, E. P., Lorena, A. C., Carvalho, A. C. P. L. F. and Freitas, A. A. (2007). A review of performance evaluation measures for hierarchical classifiers. In *Evaluation Methods for Machine Learning II: Papers from the AAAI-2007 Workshop* (C. Drummond, W. Elazmeh, N. Japkowicz and S. A. Macskassy, eds.) 182–196. AAAI Press AAAI Technical Report WS-07-05.

Deng, J., Berg, A. C., Li, K. and Li, F.-F. (2010). What does classifying more than 10,000 image categories tell us? In *Proc. 11th European Conf. Computer Vision* **5** 71–84. Springer-Verlag.

Deng, J., Berg, A. C., Li, K. and Li, F.-F. (2011). Hierarchical semantic indexing for large scale image retrieval. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition* 785–792. IEEE.

Joulin, A., Grave, E., Bojanowski, P. and Mikolov, T. (2017). Bag of tricks for efficient text classification. In *Proc. 15th Conf. European Chapter Assoc. Comput. Linguistics* 427–431. ACL.

Kosmopoulos, A., Paliouras, G. and Androutsopoulos, I. (2015). Probabilistic cascading for large-scale hierarchical classification Technical Report No. 1505.02251, arXiv.

Kosmopoulos, A., Partalas, I., Gaussier, E., Paliouras, G. and Androutsopoulos, I. (2015). Evaluation measures for hierarchical classification: a unified view and novel approaches. *Data Mining and Knowledge Discovery* **29** 820–865.

LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning. *Nature* **521** 436–444.

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S. and Bizer, C. (2015). DBpedia — a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* **6** 167–195.

Lewis, D. D., Yang, Y., Rose, T. G. and Li, F. (2004). RCV1: a new benchmark collection for text categorization research. *J. Machine Learning Research* **5** 361–397.

Partalas, I., Kosmopoulos, A., Baskiotis, N., Artieres, T., Paliouras, G., Gaussier, E., Androutsopoulos, I., Amini, M.-R. and Galinari, P. (2015). LSHTC: a benchmark for large-scale text classification Technical Report No. 1503.08581, arXiv.

Redmon, J. and Farhadi, A. (2017). YOLO9000: better, faster, stronger. In *IEEE Conf. Comput. Vision Pattern Recognition* 1–9. IEEE.

Reece, J. B., Urry, L. A., Cain, M. L., Wasserman, S. A., Minorsky, P. V. and Jackson, R. B. (2013). *Campbell Biology*, 10th ed. Pearson.

Wang, K., Zhou, S. and Liew, S. C. (1999). Building hierarchical classifiers using class proximity. In *Proc. 25th Internat. Conf. Very Large Data Bases* 363–374. Morgan Kaufmann Publishers.

Zhang, X., Zhao, J. and LeCun, Y. (2015). Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems* **28** 1–9. Neural Information Processing Systems Foundation.