Simultaneous Learning of Trees and Representations for Extreme Classification and Density Estimation

Yacine Jernite [1] Anna Choromanska [1] David Sontag [2]

Abstract

We consider multi-class classification where the predictor has a hierarchical structure that allows for a very large number of labels both at train and test time. The predictive power of such models can heavily depend on the structure of the tree, and although past work showed how to learn the tree structure, it expected that the feature vectors remained static. We provide a novel algorithm to simultaneously perform representation learning for the input data and learning of the hierarchical predictor. Our approach optimizes an objective function which favors balanced and easily-separable multi-way node partitions. We theoretically analyze this objective, showing that it gives rise to a boosting style property and a bound on classification error. We next show how to extend the algorithm to conditional density estimation. We empirically validate both variants of the algorithm on text classification and language modeling, respectively, and show that they compare favorably to common baselines in terms of accuracy and running time.

1. Introduction

Several machine learning settings are concerned with performing predictions in a very large discrete label space. From extreme multi-class classification to language modeling, one commonly used approach to this problem reduces it to a series of choices in a tree-structured model, where the leaves typically correspond to labels. While this allows for faster prediction, and is in many cases necessary to make the models tractable, the performance of the system can depend significantly on the structure of the tree used, e.g. (Mnih & Hinton, 2009).

Instead of relying on possibly costly heuristics (Mnih & Hinton, 2009), extrinsic hierarchies (Morin & Bengio, 2005) which can badly generalize across different data sets, or purely random trees, we provide an efficient data-dependent algorithm for tree construction and training. Inspired by the LOM tree algorithm (Choromanska & Langford, 2015) for binary trees, we present an objective function which favors high-quality node splits, i.e. balanced and easily separable. In contrast to previous work, our objective applies to trees of arbitrary width and leads to guarantees on model accuracy. Furthermore, we show how to successfully optimize it in the setting when the data representation needs to be learned simultaneously with the classification tree.

Finally, the multi-class classification problem is closely related to that of conditional density estimation (Ram & Gray, 2011; Bishop, 2006) since both need to consider all labels (at least implicitly) during learning and at prediction time. Both problems present similar difficulties when dealing with very large label spaces, and the techniques that we present in this work can be applied indiscriminately to either. Indeed, we show how to adapt our algorithm to efficiently solve the conditional density estimation problem of learning a language model which uses a tree structured objective.

This paper is organized as follows: Section 2 discusses related work, Section 3 outlines the necessary background and defines the flat and tree-structured objectives for multi-class classification and density estimation, Section 4 presents the objective and the optimization algorithm, Section **??** contains theoretical results, Section 5 adapts the algorithm to the problem of language modeling, Section 6 reports empirical results on the Flickr tag prediction dataset and Gutenberg text corpus, and finally Section 7 concludes the paper. Supplementary material contains additional material and proofs of theoretical statements of the paper. We also release the C++ implementation of our algorithm[1].

2. Related Work

The multi-class classification problem has been addressed in the literature in a variety of ways. Some examples include i) clustering methods (Bengio et al., 2010; Madzarov et al., 2009; Weston et al., 2013) ((Bengio et al., 2010)

[1]New York University, New York, New York, USA [2]Massachussets Institute of Technology, Cambridge, Massachussets, USA. Correspondence to: Yacine Jernite <jernite@cs.nyu.edu>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

[1]https://github.com/yjernite/fastTextLearnTree

was later improved in (Deng et al., 2011)), ii) sparse output coding (Zhao & Xing, 2013), iii) variants of error correcting output codes (Hsu et al., 2009), iv) variants of iterative least-squares (Agarwal et al., 2014), v) a method based on guess-averse loss functions (Beijbom et al., 2014), and vi) classification trees (Beygelzimer et al., 2009b; Choromanska & Langford, 2015; Daume et al., 2016) (that includes the Conditional Probability Trees (Beygelzimer et al., 2009a) when extended to the classification setting).

The recently proposed LOM tree algorithm (Choromanska & Langford, 2015) differs significantly from other similar hierarchical approaches, like for example Filter Trees (Beygelzimer et al., 2009b) or random trees (Breiman, 2001), in that it addresses the problem of learning good-quality binary node partitions. The method results in low-entropy trees and instead of using an inefficient enumerate-and-test approach, see e.g: (Breiman et al., 1984), to find a good partition or expensive brute-force optimization (Agarwal et al., 2013), it searches the space of all possible partitions with SGD (Bottou, 1998). Another work (Daume et al., 2016) uses a binary tree to map an example to a small subset of candidate labels and makes a final prediction via a more tractable one-against-all classifier, where this subset is identified with the proposed Recall Tree. A notable approach based on decision trees also include FastXML (Prabhu & Varma, 2014) (and its slower and less accurate at prediction predecessor (Agarwal et al., 2013)). It is based on optimizing the rank-sensitive loss function and shows an advantage over some other ranking and NLP-based techniques in the context of multi-label classification. Other related approaches include the SLEEC classifier (Bhatia et al., 2015) for extreme multi-label classification that learns embeddings which preserve pairwise distances between only the nearest label vectors and ranking approaches based on negative sampling (Weston et al., 2011). Another tree approach (Kontschieder et al., 2015) shows no computational speed up but leads to significant improvements in prediction accuracy.

Conditional density estimation can also be challenging in settings where the label space is large. The underlying problem here consists in learning a probability distribution over a set of random variables given some context. For example, in the language modeling setting one can learn the probability of a word given the previous text, either by making a Markov assumption and approximating the left context by the last few words seen (n-grams e.g. (Jelinek & Mercer, 1980; Katz, 1987), feed-forward neural language models (Mnih & Teh, 2012; Mikolov et al., 2011; Schwenk & Gauvain, 2002)), or by attempting to learn a low-dimensional representation of the full history (RNNs (Mikolov et al., 2010; Mirowski & Vlachos, 2015; Tai et al., 2015; Kumar et al., 2015)). Both the recurrent and feed-forward Neural Probabilistic Language Models (NPLM) (Bengio et al., 2003) simultaneously learn a distributed representation for words and the probability function for word sequences, expressed in terms of these representations. The major drawback of these models is that they can be slow to train, as they grow linearly with the vocabulary size (anywhere between 10,000 and 1M words), which can make them difficult to apply (Mnih & Teh, 2012). A number of methods have been proposed to overcome this difficulty. Works such as LBL (Mnih & Hinton, 2007) or Word2Vec (Mikolov et al., 2013) reduce the model to its barest bones, with only one hidden layer and no non-linearities. Another proposed approach has been to only compute the NPLM probabilities for a reduced vocabulary size, and use hybrid neural-n-gram model (Schwenk & Gauvain, 2005) at prediction time. Other avenues to reduce the cost of computing gradients for large vocabularies include using different sampling techniques to approximate it (Bengio & Sénécal, 2003; Bengio & Senecal, 2008; Mnih & Teh, 2012), replacing the likelihood objective by a contrastive one (Gutmann & Hyvärinen, 2012) or spherical loss (de Brébisson & Vincent, 2016), relying on self-normalizing models (Andreas & Klein, 2015), taking advantage of data sparsity (Vincent et al., 2015), or using clustering-based methods (Grave et al., 2016). It should be noted however that most of these techniques (to the exception of (Grave et al., 2016)) do not provide any speed up at test time.

Similarly to the classification case, there have also been a significant number of works that use tree structured models to accelerate computation of the likelihood and gradients (Morin & Bengio, 2005; Mnih & Hinton, 2009; Djuric et al., 2015; Mikolov et al., 2013). These use various heuristics to build a hierarchy, from using ontologies (Morin & Bengio, 2005) to Huffman coding (Mikolov et al., 2013). One algorithm which endeavors to learn a binary tree structure along with the representation is presented in (Mnih & Hinton, 2009). They iteratively learn word representations given a fixed tree structure, and use a criterion that trades off between making a balanced tree and clustering the words based on their current embedding. The application we present in the second part of our paper is most closely related to the latter work, and uses a similar embedding of the context. However, where their setting is limited to binary trees, we work with arbitrary width, and provide a tree building objective which is both less computationally costly and comes with theoretical guarantees.

3. Background

In this section, we define the classification and log-likelihood objectives we wish to maximize. Let $\mathcal{X}$ be an input space, and $\mathcal{V}$ a label space. Let $\mathcal{P}$ be a joint distribution over samples in $(\mathcal{X}, \mathcal{V})$, and let $f_\Theta : \mathcal{X} \rightarrow \mathbb{R}^{d_r}$ be a function mapping every input $x \in \mathcal{X}$ to a representation $\mathbf{r} \in \mathbb{R}^{d_r}$, and parametrized by Θ (e.g. as a neural network).

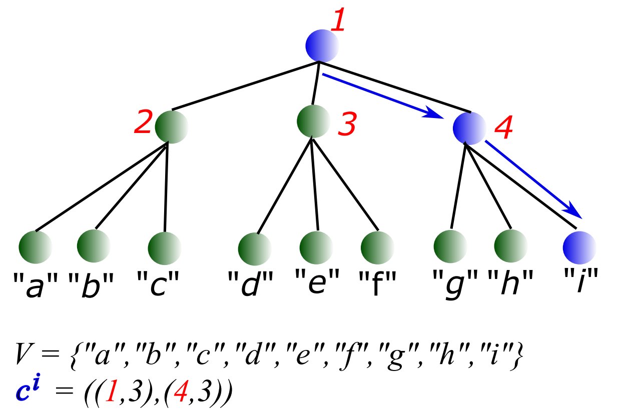
We consider two objectives. Let g be a function that takes an input representation $\mathbf{r} \in \mathbb{R}^{d_r}$, and predicts for it a label $g(\mathbf{r}) \in \mathcal{V}$. The classification objective is defined as the expected proportion of correctly classified examples:

$$\mathcal{O}^{\text{class}}(\Theta, g) = \mathbb{E}_{(x,y)\sim\mathcal{P}}\Big[\mathbb{1}[g \circ f_\Theta(x) = y]\Big] \quad (1)$$

Now, let $p_\theta(\cdot|\mathbf{r})$ define a conditional probability distribution (parametrized by θ) over $\mathcal{V}$ for any $\mathbf{r} \in \mathbb{R}^{d_r}$. The density estimation task consists in maximizing the expected log-likelihood of samples from $(\mathcal{X}, \mathcal{V})$:

$$\mathcal{O}^{\text{ll}}(\Theta, \theta) = \mathbb{E}_{(x,y)\sim\mathcal{P}}\Big[\log p_\theta(y|f_\Theta(x))\Big] \quad (2)$$

Tree-Structured Classification and Density Estimation Let us now show how to express the objectives in Equations 1 and 2 when using tree-structured prediction functions (with fixed structure) as illustrated in Figure 1.



1
2
3
4
"a" "b" "c" "d" "e" "f" "g" "h" "i"
V = {"a","b","c","d","e","f","g","h","i"}
c^i = ((1,3),(4,3))

Figure 1. Hierarchical predictor: in order to predict label "i", the system needs to choose the third child of node 1, then the third child of node 4.

Consider a tree $\mathcal{T}$ of depth D and arity M with $K = |\mathcal{V}|$ leaf nodes and N internal nodes. Each leaf l corresponds to a label, and can be identified with the path $\mathbf{c}^l$ from the root to the leaf. In the rest of the paper, we will use the following notations:

$$\mathbf{c}^l = ((c^l_{1,1}, c^l_{1,2}), \ldots, (c^l_{d,1}, c^l_{d,2}), \ldots, (c^l_{D,1}, c^l_{D,2})), \quad (3)$$

where $c^l_{d,1} \in [1, N]$ correspond to the node index at depth d, and $c^l_{d,2} \in [1, M]$ indicates which child of $c^l_{d,1}$ is next in the path. In that case, our classification and density estimation problems are reduced to choosing the right child of a node or defining a probability distribution over children given $x \in \mathcal{X}$ respectively.

We then need to replace g and p_θ with node decision functions $(g_n)_{n=1}^N$ and conditional probability distributions $(p_{\theta_n})_{n=1}^N$ respectively. Given such a tree and representation function, our objective functions then become:

$$\mathcal{O}^{\text{class}}(\Theta, g) = \mathbb{E}_{(x,y)\sim\mathcal{P}}\Big[\prod_{d=1}^{D} \mathbb{1}[g_{c^l_{d,1}} \circ f_\Theta(x) = c^l_{d,2}]\Big] \quad (4)$$

$$\mathcal{O}^{\text{ll}}(\Theta, \theta) = \mathbb{E}_{(x,y)\sim\mathcal{P}}\Big[\sum_{d=1}^{D} \log p_{\theta_{c^l_{d,1}}}(c^l_{d,2}|f_\Theta(x))\Big] \quad (5)$$

The tree objectives defined in Equations 4 and 5 can be optimized in the space of parameters of the representation and node functions using standard gradient ascent methods. However, they also implicitly depend on the tree structure $\mathcal{T}$. In the rest of the paper, we provide a surrogate objective function which determines the structure of the tree and, as we show theoretically (Section **??**), maximizes the criterion in Equation 4 and, as we show empirically (Sections 5 and 6), maximizes the criterion in Equation 5.

4. Learning Tree-Structured Objectives

In this section, we introduce a per-node objective J_n which leads to good quality trees when maximized, and provide an algorithm to optimize it.

4.1. Objective function

We define the node objective J_n for node n as:

$$J_n = \frac{2}{M} \sum_{i=1}^{K} q_i^{(n)} \sum_{j=1}^{M} |p_j^{(n)} - p_{j|i}^{(n)}|, \quad (6)$$

where $q_i^{(n)}$ denotes the proportion of nodes reaching node n that are of class i, $p_{j|i}^{(n)}$ is the probability that an example of class i reaching n will be sent to its jth child, and $p_j^{(n)}$ is the probability that an example of any class reaching n will be sent to its jth child. Note that we have:

$$\forall j \in [1, M], \;\; p_j^{(n)} = \sum_{i=1}^{K} q_i^{(n)} p_{j|i}^{(n)}. \quad (7)$$

The objective in Equation 6 reduces to the LOM tree objective in the case of $M = 2$.

At a high level, maximizing the objective encourages the conditional distribution for each class to be as different as possible from the global one; so the node decision function needs to be able to discriminate between examples of the different classes. The objective thus favors balanced and pure node splits. To wit, we call a split at node n *perfectly balanced* when the global distribution $p_{\cdot}^{(n)}$ is uniform, and *perfectly pure* when each $p_{\cdot|i}^{(n)}$ takes value either 0 or 1, as all data points from the same class reaching node n are sent to the same child.

In Section **??** we discuss the theoretical properties of this objective in details. We show that maximizing it leads to perfectly balanced and perfectly pure splits. We also derive the boosting theorem that shows the number of internal nodes that the tree needs to have to reduce the classification error below any arbitrary threshold, under the assumption that the objective is "weakly" optimized in each node of the tree.

Remark 1. *In the rest of the paper, we use node functions g_n which take as input a data representation $\mathbf{r} \in \mathbb{R}^{d_r}$ and output a distribution over children of n (for example using a soft-max function). When used in the classification setting, g_n sends the data point to the child with the highest predicted probability. With this notation, and representa-*

Algorithm 1 Tree Learning Algorithm

```
Input Input representation function: $f$ with parameters
      $\Theta_f$. Node decisions functions $(g_n)_{n=1}^K$ with
      parameters $(\Theta_n)_{n=1}^K$. Gradient step size $\epsilon$.
Ouput Learned $M$-ary tree, parameters $\Theta_f$ and $(\Theta_n)_{n=1}^K$.

procedure InitializeNodeStats ()
    for $n = 1$ to $N$ do
        for $i = 1$ to $K$ do
            SumProbas$_{n,i} \leftarrow \mathbf{0}$
            Counts$_{n,i} \leftarrow 0$

procedure NodeCompute ($\mathbf{w}$, $n$, $i$, target)
    $\mathbf{p} \leftarrow g_n(\mathbf{w})$
    SumProbas$_{n,i} \leftarrow$ SumProbas$_{n,i} + \mathbf{p}$
    Counts$_{n,i} \leftarrow$ Counts$_{n,i} + 1$
    // Gradient step in the node parameters
    $\Theta_n \leftarrow \Theta_n + \epsilon \frac{\partial p_{\text{target}}}{\partial \Theta_n}$
    return $\frac{\partial p_{\text{target}}}{\partial \mathbf{w}}$

InitializeNodeStats ()
for Each batch $b$ do
    // AssignLabels () re-builds the tree based on the
    // current node statistics
    AssignLabels ($\{1, \dots, K\}$, root)
    for each example $(\mathbf{x}, i)$ in $b$ do
        Compute input representation $\mathbf{w} = f(\mathbf{x})$
        $\Delta\mathbf{w} \leftarrow \mathbf{0}$
        for $d = 1$ to $D$ do
            // $c^i_{1,\dots,D}$ is the current path from the root to i
            Set node id and target: $(n, j) \leftarrow c^i_d$
            $\Delta\mathbf{w} \leftarrow \Delta\mathbf{w}$ + NodeCompute ($\mathbf{w}$, n, i, j)

        // Gradient step in the parameters of $f$
        $\Theta_f \leftarrow \Theta_f + \epsilon \frac{\partial f}{\partial \Theta_f} \Delta\mathbf{w}$
```

tion function f_Θ, we can write:

$$p_j^{(n)} := \mathbb{E}_{(x,y)\sim\mathcal{P}}[g_n \circ f_\Theta(x)] \qquad (8)$$

and

$$p_{j|i}^{(n)} := \mathbb{E}_{(x,y)\sim\mathcal{P}}[g_n \circ f_\Theta(x) | y = i]. \qquad (9)$$

An intuitive geometric interpretation of probabilities $p_j^{(n)}$ and $p_{j|i}^{(n)}$ can be found in the Supplementary material.

4.2. Algorithm

In this section we present an algorithm for simultaneously building the classification tree and learning the data representation. We aim at maximizing the accuracy of the tree as defined in Equation 4 by maximizing the objective J_n of Equation 6 at each node of the tree (the boosting theorem that will be presented in Section **??** shows the connection between the two).

Algorithm 2 Label Assignment Algorithm

```
Input labels currently reaching the node
      node ID $n$
Ouput Lists of labels now assigned to the node's children

procedure CheckFull (full, assigned, count, $j$)
    if $|\text{assigned}_j| \equiv 2 \mod (M-1)$ then
        count $\leftarrow$ count $- (M-1)$
    if count $= 0$ then
        full $\leftarrow$ full $\cup \{j\}$
    if count $= 1$ then
        count $\leftarrow 0$
        for $j'$ s.t. $|\text{assigned}_{j'}| \equiv 1 \mod (M-1)$ do
            full $\leftarrow$ full $\cup \{j'\}$

procedure AssignLabels (labels, $n$)
    // first, compute $p_j^{(n)}$ and $p_{j|i}^{(n)}$.
    $\mathbf{p}_0^{avg} \leftarrow \mathbf{0}$
    count $\leftarrow 0$
    for $i$ in labels do
        $\mathbf{p}_0^{avg} \leftarrow \mathbf{p}_0^{avg}$ + SumProbas$_{n,i}$
        count $\leftarrow$ count + Counts$_{n,i}$
        $\mathbf{p}_i^{avg} \leftarrow$ SumProbas$_{n,i}$/Counts$_{n,i}$
    $\mathbf{p}_0^{avg} \leftarrow \mathbf{p}_0^{avg}$/count
    // then, assign each label to a child of $n$
    unassigned $\leftarrow$ labels
    full $\leftarrow \emptyset$
    count $\leftarrow$ (|unassigned| $- (M-1)$)
    for $j = 1$ to $M$ do
        assigned$_j \leftarrow \emptyset$
    while unassigned $\neq \emptyset$ do
        // $\frac{\partial J_n}{\partial p_{j|i}^{(n)}}$ is given in Equation 10
        $(i^*, j^*) \leftarrow \underset{i \in \text{unassigned}, j \notin \text{full}}{\text{argmax}} \left( \frac{\partial J_n}{\partial p_{j|i}^{(n)}} \right)$
        if $n$ = root then
            $\mathbf{c}^{i^*} \leftarrow (n, j^*)$
        else
            $\mathbf{c}^{i^*} \leftarrow (\mathbf{c}^{i^*}, (n, j^*))$
        assigned$_{j^*} \leftarrow$ assigned$_{j^*} \cup \{i^*\}$
        unassigned $\leftarrow$ unassigned $\setminus \{i^*\}$
        CheckFull (full, assigned, count, $j^*$)
    for $j = 1$ to $M$ do
        AssignLabels (assigned$_j$, child$_{n,j}$, $d + 1$)
    return assigned
```

Let us now show how we can efficiently optimize J_n. The gradient of J_n with respect to the conditional probability distributions is (see proof of Lemma 1 in the Supplement):

$$\frac{\partial J_n}{\partial p_{j|i}^{(n)}} = \frac{2}{M} q_i^{(n)} (1 - q_i^{(n)}) \operatorname{sign}(p_{j|i}^{(n)} - p_j^{(n)}). \qquad (10)$$

Then, according to Equation 10, increasing the likelihood of sending label i to any child j of n such that $p_{j|i}^{(n)} > p_j^{(n)}$ increases the objective J_n. Note that we only need to consider the labels i for which $q_i^{(n)} > 0$, that is, labels i which reach node n in the current tree.

We also want to make sure that we have a well-formed M-ary tree at each step, which means that the number of labels assigned to any node is always congruent to 1 modulo $(M-1)$. Algorithm 2 provides such an assignment by greedily choosing the label-child pair (i, j) such that j still has room for labels with the highest value of $\frac{\partial J_n}{\partial p_{j|i}^{(n)}}$.

The global procedure, described in Algorithm 1, is then the following.

- At the start of each batch, re-assign targets for each node prediction function, starting from the root and going down the tree. At each node, each label is more likely to be re-assigned to the child it has had most affinity with in the past (Algorithm 2). This can be seen as a form of hierarchical on-line clustering.
- Every example now has a unique path depending on its label. For each sample, we then take a gradient step at each node along the assigned path (see Algorithm 1).

Lemma 1. *Algorithm 2 finds the assignment of nodes to children for a fixed depth tree which most increases J_n under well-formedness constraints.*

Remark 2. *An interesting feature of the algorithm, is that since the representation of examples from different classes are learned together, there is intuitively less of a risk of getting stuck in a specific tree configuration. More specifically, if two similar classes are initially assigned to different children of a node, the algorithm is less likely to keep this initial decision since the representations for examples of both classes will be pulled together in other nodes.*

Next, we provide a theoretical analysis of the objective introduced in Equation 6. Proofs are deferred to the Supplementary material.

5. Theoretical Results

In this section, we first analyze theoretical properties of the objective J_n as regards node quality, then prove a boosting statement for the global tree accuracy.

5.1. Properties of the objective function

We start by showing that maximizing J_n in every node of the tree leads to high-quality nodes, i.e. perfectly balanced and perfectly pure node splits. Let us first introduce some formal definitions.

Definition 1 (Balancedness factor). *The split in node n of the tree is $\beta^{(n)}$-balanced if*

$$\beta^{(n)} \leq \min_{j=\{1,2,\ldots,M\}} p_j^{(n)},$$

where $\beta^{(n)} \in (0, \frac{1}{M}]$ is a balancedness factor.

A split is perfectly balanced if and only if $\beta^{(n)} = \frac{1}{M}$.

Definition 2 (Purity factor). *The split in node n of the tree is $\alpha^{(n)}$-pure if*

$$\frac{1}{M}\sum_{j=1}^{M}\sum_{i=1}^{K} q_i^{(n)} \min\left(p_{j|i}^{(n)}, 1 - p_{j|i}^{(n)}\right) \leq \alpha^{(n)},$$

where $\alpha^{(n)} \in [0, \frac{1}{M})$ is a purity factor.

A split is perfectly pure if and only if $\alpha^{(n)} = 0$.

The following lemmas characterize the range of the objective J_n and link it to the notions of balancedness and purity of the split.

Lemma 2. *The objective function J_n lies in the interval $\left[0, \frac{4}{M}\left(1 - \frac{1}{M}\right)\right]$.*

Let J^* denotes the highest possible value of J_n, i.e. $J^* = \frac{4}{M}\left(1 - \frac{1}{M}\right)$.

Lemma 3. *The objective function J_n admits the highest value, i.e. $J_n = J^*$, if and only if the split in node n is perfectly balanced, i.e. $\beta^{(n)} = \frac{1}{M}$, and perfectly pure, i.e. $\alpha^{(n)} = 0$.*

We next show Lemmas **??** and **??** which analyze balancedness and purity of a node split in isolation, i.e. we analyze resp. balancedness and purity of a node split when resp. purity and balancedness is fixed and perfect. We show that in such isolated setting increasing J_n leads to a more balanced and more pure split.

Lemma 4. *If a split in node n is perfectly pure, then*

$$\beta^{(n)} \in \left[\frac{1}{M} - \frac{\sqrt{M(J^* - J_n)}}{2}, \frac{1}{M}\right].$$

Lemma 5. *If a split in node n is perfectly balanced, then $\alpha^{(n)} \leq (J^* - J_n)/2$.*

Next we provide a bound on the classification error for the tree. In particular, we show that if the objective is "weakly" optimized in each node of the tree, where this weak advantage is captured in a form of the *Weak Hypothesis Assumption*, then our algorithm will amplify this weak advantage to build a tree achieving any desired level of accuracy.

5.2. Error bound

Denote $y(x)$ to be a fixed target function with domain $\mathcal{X}$, which assigns the data point x to its label, and let $\mathcal{P}$ be a fixed target distribution over $\mathcal{X}$. Together y and $\mathcal{P}$ induce a distribution on labeled pairs $(x, y(x))$. Let $t(x)$ be the label assigned to data point x by the tree. We denote as $\epsilon(\mathcal{T})$ the error of tree $\mathcal{T}$, i.e. $\epsilon(\mathcal{T}) := \mathbb{E}_{x\sim\mathcal{P}}\left[\sum_{i=1}^{K} \mathbb{1}[t(x) = i, y(x) \neq i]\right]$ ($1 - \epsilon(\mathcal{T})$ refers to the accuracy as given by Equation 4). Then the following theorem holds

Theorem 1. *The Weak Hypothesis Assumption says that for any distribution $\mathcal{P}$ over the data, at each node n of the tree $\mathcal{T}$ there exists a partition such that $J_n \geq \gamma$, where $\gamma \in \left[\frac{M}{2} \min_{j=1,2,\ldots,M} p_j, 1 - \frac{M}{2} \min_{j=1,2,\ldots,M} p_j\right]$.*

Under the Weak Hypothesis Assumption, for any $\kappa \in [0,1]$, to obtain $\epsilon(\mathcal{T}) \leq \kappa$ it suffices to have a tree with

$$N \geq \left(\frac{1}{\kappa}\right)^{\frac{16[M(1-2\gamma)+2\gamma](M-1)}{\log_2 eM^2\gamma^2} \ln K} \qquad \textit{internal nodes.}$$

The above theorem shows the number of splits that suffice to reduce the multi-class classification error of the tree below an arbitrary threshold κ. As shown in the proof of the above theorem, the *Weak Hypothesis Assumption* implies that all p_js satisfy: $p_j \in [\frac{2\gamma}{M}, \frac{M(1-2\gamma)+2\gamma}{M}]$. Below we show a tighter version of this bound when assuming that each node induces balanced split.

Corollary 1. *The Weak Hypothesis Assumption says that for any distribution $\mathcal{P}$ over the data, at each node n of the tree $\mathcal{T}$ there exists a partition such that $J_n \geq \gamma$, where $\gamma \in \mathbb{R}^+$.*

Under the Weak Hypothesis Assumption and when all nodes make perfectly balanced splits, for any $\kappa \in [0,1]$, to obtain $\epsilon(\mathcal{T}) \leq \kappa$ it suffices to have a tree with

$$N \geq \left(\frac{1}{\kappa}\right)^{\frac{16(M-1)}{\log_2 eM^2\gamma^2} \ln K} \qquad \textit{internal nodes.}$$

6. Extension to Density Estimation

We now show how to adapt the algorithm presented in Section 4 for conditional density estimation, using the example of language modeling.

Hierarchical Log Bi-Linear Language Model (HLBL) We take the same approach to language modeling as (Mnih & Hinton, 2009). First, using the chain rule and an order T Markov assumption we model the probability of a sentence $\mathbf{w} = (w_1, w_2, \ldots, w_n)$ as:

$$p(w_1, w_2, \ldots, w_n) = \prod_{t=1}^{n} p(w_t | w_{t-T,\ldots,t-1})$$

Similarly to their work, we also use a low dimensional representation of the context $(w_{t-T,\ldots,t-1})$. In this setting, each word w in the vocabulary $\mathcal{V}$ has an embedding $U_w \in \mathbb{R}^{d_r}$. A given context $x = (w_{t-T}, \ldots, w_{t-1})$ corresponding to position t is then represented by a context embedding vector r_x such that

$$r_x = \sum_{k=1}^{T} R_k U_{w_{t-k}},$$

where $U \in \mathbb{R}^{|\mathcal{V}| \times d_r}$ is the embedding matrix, and $R_k \in \mathbb{R}^{d_r \times d_r}$ is the transition matrix associated with the k^{th} context word.

The most straight-forward way to define a probability function is then to define the distribution over the next word given the context representation as a soft-max, as done in (Mnih & Hinton, 2007). That is:

$$\begin{aligned} p(w_t = i | x) &= \sigma_i(r_x^\top U + \mathbf{b}) \\ &= \frac{\exp(r_x^\top U_i + b_i)}{\sum_{w \in \mathcal{V}} \exp(r_x^\top U_w + b_w)}, \end{aligned}$$

where b_w is the bias for word w. However, the complexity of computing this probability distribution in this setting is $O(|V| \times d_r)$, which can be prohibitive for large corpora and vocabularies.

Instead, (Mnih & Hinton, 2009) takes a hierarchical approach to the problem. They construct a binary tree, where each word $w \in \mathcal{V}$ corresponds to some leaf of the tree, and can thus be identified with the path from the root to the corresponding leaf by making a sequence of choices of going left versus right. This corresponds to the tree-structured log-likelihood objective presented in Equation 5 for the case where $M = 2$, and $f_\Theta(x) = r_x$. Thus, if $\mathbf{c}^i$ is the path to word i as defined in Expression 3, then:

$$\log p(w_t = i | x) = \sum_{d=1}^{D} \log \sigma_{c^i_{d,2}}((r_x^\top U^{c^i_{d,1}} + \mathbf{b}^{c^i_{d,1}}) \quad (11)$$

In this binary case, σ is the sigmoid function, and for all non-leaf nodes $n \in \{1, 2, \ldots, N\}$, we have $U^n \in \mathbb{R}^{d_r}$ and $\mathbf{b}^n \in \mathbb{R}^{d_r}$. The cost of computing the likelihood of word w is then reduced to $O(\log(|\mathcal{V}|) \times d_r)$. In their work, the authors start the training procedure by using a random tree, then alternate parameter learning with using a clustering-based heuristic to rebuild their hierarchy. We expand upon their method by providing an algorithm which allows for using hierarchies of arbitrary width, and jointly learns the tree structure and the model parameters.

Using our Algorithm We may use Algorithm 1 as is to learn a good tree structure for classification: that is, a model that often predicts w_t to be the most likely word after seeing the context $(w_{t-T}, \ldots, w_{t-1})$. However, while this could certainly learn interesting representations and tree structure, there is no guarantee that such a model would achieve a good average log-likelihood. Intuitively, there are often several valid possibilities for a word given its immediate left context, which a classification objective does not necessarily take into account. Yet another option would be to learn a tree structure that maximizes the classification objective, then fine-tune the model parameters using the log-likelihood objective. We tried this method, but initial tests of this approach did not do much better than the use of random trees. Instead, we present here a small modification of Algorithm 1 which is equivalent to log-likelihood training when restricted to the fixed tree setting, and can be shown to increase the value of the node objectives J_n: by replacing the gradients with respect to p_{target} by those with respect to $\log p_{target}$. Then, for a given tree structure, the algorithm takes a gradient step with respect to the log-likelihood of the samples:

$$\frac{\partial J_n}{\partial \log p_{j|i}^{(n)}} = \frac{2}{M} q_i^{(n)}(1-q_i^{(n)}) \operatorname{sign}(p_{j|i}^{(n)} - p_j^{(n)}) p_{j|i}^{(n)}. \quad (12)$$

Lemma 1 extends to the new version of the algorithm.

7. Experiments

We ran experiments to evaluate both the classification and density estimation version of our algorithm. For classification, we used the YFCC100M dataset (Thomee et al., 2016), which consists of a set of a hundred million Flickr pictures along with captions and tag sets split into 91M training, 930K validation and 543K test examples. We focus here on the problem of predicting a picture's tags given its caption. For density estimation, we learned a log-bilinear language model on the Gutenberg novels corpus, and compared the perplexity to that obtained with other flat and hierarchical losses. Experimental settings are described in greater detail in the Supplementary material.

7.1. Classification

We follow the setting of (Joulin et al., 2016) for the YFCC100M tag prediction task: we only keep the tags which appear at least a hundred times, which leaves us with a label space of size 312K. We compare our results to those obtained with the FastText software (Joulin et al., 2016), which uses a binary hierarchical softmax objective based on Huffman coding (Huffman trees are designed to minimize the expected depth of their leaves weighed by frequencies and have been shown to work well with word embedding systems (Mikolov et al., 2013)), and to the Tagspace system (Weston et al., 2014), which uses a sampling-based margin loss (this allows for training in tractable time, but does not help at test time, hence the long times reported). We also extend the FastText software to use Huffman trees of arbitrary width. All models use a bag-of-word embedding representation of the caption text; the parameters of the input representation function f_Θ which we learn are the word embeddings $U_w \in \mathbb{R}^d$ (as in Section 5) and a caption representation is obtained by summing the embeddings of its words. We experimented with embeddings of dimension $d = 50$ and $d = 200$. We predict one tag for each caption, and report the precision as well as the training and test times in Table 1.

Our implementation is based on the FastText open source version[2], to which we added M-ary Huffman and learned tree objectives. Table 1 reports the best accuracy we obtained with a hyper-parameter search using this version on our system so as to provide the most meaningful comparison, even though the accuracy is less than that reported in (Joulin et al., 2016).

We gain a few different insights from Table 1. First, although wider trees are theoretically slower (remember that the theoretical complexity is $O(M \log_M(N))$ for an M-ary tree with N labels), they run incomparable time in practice and always perform better. Using our algorithm to learn the structure of the tree also always leads to more accurate models, with a gain of up to 3.3 precision points in the smaller 5-ary setting. Further, both the importance of having wider trees and learning the structure seems to be less when the node prediction functions become more expressive. At a high level, one could imagine that in that setting, the model can learn to use different dimensions of the input representation for different nodes, which would minimize the negative impact of having to learn a representation which is suited to more nodes.

d	Model	Arity	P@1	Train	Test
50	TagSpace[1]	-	30.1	3h8	6h
	FastText[2]	2	27.2	**8m**	**1m**
	M-ary Huffman Tree	5	28.3	**8m**	**1m**
		20	29.9	10m	3m
	Learned Tree	5	31.6	18m	**1m**
		20	**32.1**	30m	3m
200	TagSpace[1]	35.6	5h32	15h	
	FastText[2]	2	35.2	**12m**	**1m**
	M-ary Huffman Tree	5	35.8	13m	2m
		20	36.4	18m	3m
	Learned Tree	5	36.1	35m	3m
		20	**36.6**	45m	8m

Table 1. Classification performance on the YFCC100M dataset. [1](Weston et al., 2014). [2](Joulin et al., 2016). M-ary Huffman Tree modifies FastText by adding an M-ary hierarchical softmax.

Another thing to notice is that since prediction time only depends on the expected depth of a label, our models which learned balanced trees are nearly as fast as Huffman coding which is optimal in that respect (except for the dimension 200, 20-ary tree, but the tree structure had not stabilized yet in that setting). Given all of the above remarks, our algorithm especially shines in settings where computational complexity and prediction time are highly constrained at test time, such as mobile devices or embedded systems.

7.2. Density Estimation

We also ran language modeling experiments on the Gutenberg novel corpus[3], which has about 50M tokens and a vocabulary of 250,000 words.

One notable difference from the previous task is that the language modeling setting can drastically benefit from the use of GPU computing, which can make using a flat softmax tractable (if not fast). While our algorithm requires

[2]https://github.com/facebookresearch/fastText

[3]http://www.gutenberg.org/

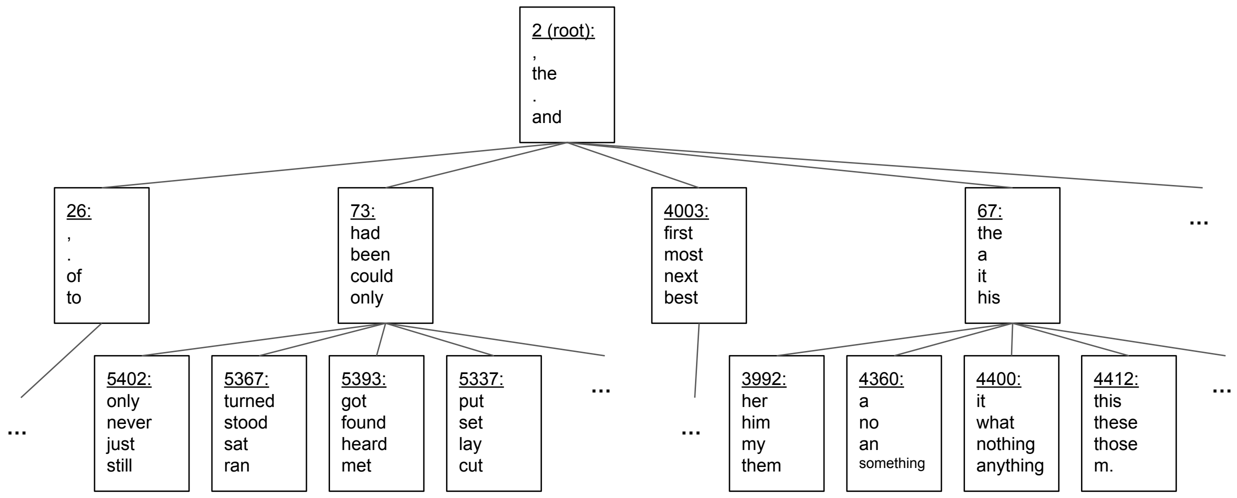

2 (root): , the . and
26: , . of to
73: had been could only
4003: first most next best
67: the a it his
5402: only never just still
5367: turned stood sat ran
5393: got found heard met
5337: put set lay cut
3992: her him my them
4360: a no an something
4400: it what nothing anything
4412: this these those m.
...

Figure 3. Tree learned from the Gutenberg corpus, showing the four most common words assigned to each node.

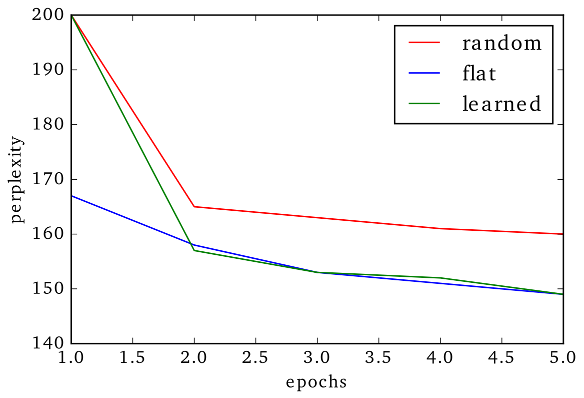


Figure 2. Test perplexity per epoch.

more flexibility and thus does not benefit as much from the use of GPUs, a small modification of Algorithm 2 (described in the Supplementary material) allows it to run under a maximum depth constraint and remain competitive. The results presented in this section are obtained using this modified version, which learns 65-ary trees of depth 3.

Table 2 presents perplexity results for different loss functions, along with the time spent on computing and learning the objective (softmax parameters for the flat version, hierarchical softmax node parameters for the fixed tree, and hierarchical softmax structure and parameters for our algorithm). The learned tree model is nearly three and seven times as fast at train and test time respectively as the flat objective without losing any points of perplexity.

Huffman coding does not apply to trees where all of the leaves are at the same depth. Instead, we use the following heuristic as a baseline, inspired by (Mnih & Hinton, 2009): we learn word embeddings using FastText, perform a hierarchical clustering of the vocabulary based on these, then use the resulting tree to learn a new language model. We call this approach "Clustering Tree". However, for all hyper-parameter settings, this tree structure did worse than a random one. We conjecture that its poor performance is because such a tree structure means that the deepest node decisions can be quite difficult. Finally, we also ran density estimation experiments on the Penn TreeBank data set, which consists of 1M tokens with a vocabulary size of 10,000, with sensibly similar performance results and a speedup factor of two (see supplementary material). It should be noted that running a softmax on a label set of this size (only 10K) fits comfortably on most modern GPUs (hence the comparatively smaller speed gain).

Model	perp.	train ms/batch	test ms/batch
Clustering Tree	212	**2.0**	**1.0**
Random Tree	160	**1.9**	**0.9**
Flat soft-max	149	*12.5*	*6.9*
Learned Tree	**148**	*4.5*	**0.9**

Table 2. Comparison of a flat soft-max to a 65-ary hierarchical soft-max (learned, random and heuristic-based tree).

Figure 2 shows the evolution of the test perplexity for a few epochs. It appears that most of the relevant tree structure can be learned in one epoch: from the second epoch on, the learned hierarchical soft-max performs similarly to the flat one. Figure 3 shows a part of the tree learned on the Gutenberg dataset, which appears to make semantic and syntactic sense.

8. Conclusion

In this paper, we introduced a provably accurate algorithm for jointly learning tree structure and data representation for hierarchical prediction. We applied it to a multi-class classification and a density estimation problem, and showed our models' ability to achieve favorable accuracy in competitive times in both settings.

References

Agarwal, A., Kakade, S. M., Karampatziakis, N., Song, L., and Valiant, G. Least squares revisited: Scalable approaches for multi-class prediction. In *ICML*, 2014.

Agarwal, R., Gupta, A., Prabhu, Y., and Varma, M. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *WWW*, 2013.

Andreas, J. and Klein, D. When and why are log-linear models self-normalizing? In *NAACL HLT*, 2015.

Azocar, A., Gimenez, J., Nikodem, K., and Sanchez, J. L. On strongly midconvex functions. *Opuscula Math.*, 31 (1):15–26, 2011.

Beijbom, O., Saberian, M., Kriegman, D., and Vasconcelos, N. Guess-averse loss functions for cost-sensitive multiclass boosting. In *ICML*, 2014.

Bengio, S., Weston, J., and Grangier, D. Label embedding trees for large multi-class tasks. In *NIPS*, 2010.

Bengio, Y. and Sénécal, J.-S. Quick training of probabilistic neural nets by importance sampling. In *AISTATS*, 2003.

Bengio, Y. and Senecal, J.-S. Adaptive importance sampling to accelerate training of a neural probabilistic language model. *IEEE Trans. Neural Networks*, 19:713–722, 2008.

Bengio, Y., Ducharme, R., V., Pascal, and Janvin, C. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, 2003.

Beygelzimer, A., Langford, J., Lifshits, Y., Sorkin, G. B., and Strehl, A. L. Conditional probability tree estimation analysis and algorithms. In *UAI*, 2009a.

Beygelzimer, A., Langford, J., and Ravikumar, P. D. Error-correcting tournaments. In *ALT*, 2009b.

Bhatia, K., Jain, H., Kar, P., Varma, M., and Jain, P. Sparse local embeddings for extreme multi-label classification. In *NIPS*. 2015.

Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

Bottou, L. Online algorithms and stochastic approximations. In *Online Learning and Neural Networks*. Cambridge University Press, 1998.

Breiman, L. Random forests. *Mach. Learn.*, 45:5–32, 2001.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees*. CRC Press LLC, Boca Raton, Florida, 1984.

Choromanska, A. and Langford, J. Logarithmic time online multiclass prediction. In *NIPS*. 2015.

Choromanska, A., Choromanski, K., and Bojarski, M. On the boosting ability of top-down decision tree learning algorithm for multiclass classification. *CoRR*, abs/1605.05223, 2016.

Daume, H., Karampatziakis, N., Langford, J., and Mineiro, P. Logarithmic time one-against-some. *CoRR*, abs/1606.04988, 2016.

de Brébisson, A. and Vincent, P. An exploration of softmax alternatives belonging to the spherical loss family. In *ICLR*, 2016.

Deng, J., Satheesh, S., Berg, A. C., and Fei-Fei, L. Fast and balanced: Efficient label tree learning for large scale object recognition. In *NIPS*, 2011.

Djuric, N., Wu, H., Radosavljevic, V., Grbovic, M., and Bhamidipati, N. Hierarchical neural language models for joint representation of streaming documents and their content. In *WWW*, 2015.

Grave, E., Joulin, A., Cissé, M., Grangier, D., and Jégou, H. Efficient softmax approximation for gpus. *CoRR*, abs/1609.04309, 2016.

Gutmann, M. U. and Hyvärinen, A. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *J. Mach. Learn. Res.*, 13(1):307–361, 2012.

Hsu, D., Kakade, S., Langford, J., and Zhang, T. Multi-label prediction via compressed sensing. In *NIPS*, 2009.

Jelinek, F. and Mercer, R. L. Interpolated estimation of Markov source parameters from sparse data. In *Proceedings, Workshop on Pattern Recognition in Practice*, pp. 381–397. North Holland, 1980.

Joulin, Armand, Grave, Edouard, Bojanowski, Piotr, and Mikolov, Tomas. Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759, 2016.

Katz, S. M. Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *IEEE Trans. on Acoustics, Speech and Singal proc.*, volume ASSP-35, pp. 400–401, 1987.

Kontschieder, P., Fiterau, M., Criminisi, A., and Bulo', S. Rota. Deep Neural Decision Forests. In *ICCV*, 2015.

Kumar, A., Irsoy, O., Su, J., Bradbury, J., English, R., Pierce, B., Ondruska, P., Gulrajani, I., and Socher, R. Ask me anything: Dynamic memory networks for natural language processing. *CoRR*, abs/1506.07285, 2015.

Madzarov, G., Gjorgjevikj, D., and Chorbev, I. A multi-class svm classifier utilizing binary decision tree. *Informatica*, 33(2):225–233, 2009.

Mikolov, T., Karafit, M., Burget, L., Cernock, J., and Khudanpur, S. Recurrent neural network based language model. In *INTERSPEECH*, 2010.

Mikolov, T., Deoras, A., Kombrink, S., Burget, L., and Cernocky, J. Honza. Empirical evaluation and combination of advanced language modeling techniques. In *INTERSPEECH*, 2011.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

Mirowski, P. and Vlachos, A. Dependency recurrent neural language models for sentence completion. *CoRR*, abs/1507.01193, 2015.

Mnih, A. and Hinton, G. Three new graphical models for statistical language modelling. In *ICML*, 2007.

Mnih, A. and Hinton, G. E. A scalable hierarchical distributed language model. In *NIPS*. 2009.

Mnih, A. and Teh, Y. W. A fast and simple algorithm for training neural probabilistic language models. In *ICML*, 2012.

Morin, F. and Bengio, Y. Hierarchical probabilistic neural network language model. In *AISTATS*, 2005.

Prabhu, Y. and Varma, M. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *ACM SIGKDD*, 2014.

Ram, P. and Gray, A. G. Density estimation trees. In *KDD*, 2011.

Schwenk, H. and Gauvain, J.-L. Connectionist language modeling for large vocabulary continuous speech recognition. In *ICASSP*, 2002.

Schwenk, H. and Gauvain, J.-L. Training neural network language models on very large corpora. In *HLT*, 2005.

Shalev-Shwartz, S. Online learning and online convex optimization. *Found. Trends Mach. Learn.*, 4(2):107–194, 2012.

Tai, K. S., Socher, R., and Manning, C. D. Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, abs/1503.00075, 2015.

Thomee, Bart, Shamma, David A., Friedland, Gerald, Elizalde, Benjamin, Ni, Karl, Poland, Douglas, Borth, Damian, and Li, Li-Jia. YFCC100M: the new data in multimedia research. *Commun. ACM*, 59(2):64–73, 2016.

Vincent, P., de Brébisson, A., and Bouthillier, X. Efficient exact gradient update for training deep networks with very large sparse targets. In *NIPS*, 2015.

Weston, J., Bengio, S., and Usunier, N. Wsabie: Scaling up to large vocabulary image annotation. In *IJCAI*, 2011.

Weston, J., Makadia, A., and Yee, H. Label partitioning for sublinear ranking. In *ICML*, 2013.

Weston, Jason, Chopra, Sumit, and Adams, Keith. #tagspace: Semantic embeddings from hashtags. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pp. 1822–1827, 2014.

Zhao, B. and Xing, E. P. Sparse output coding for large-scale visual recognition. In *CVPR*, 2013.