Neurocomputing 258 (2017) 63–73

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Unsupervised feature extraction with autoencoder trees

Ozan İrsoy[a,*], Ethem Alpaydın[b]

[a] Department of Computer Science, Cornell University, Ithaca, NY 14853, United States
[b] Department of Computer Engineering, Boğaziçi University, Bebek, İstanbul 34342, Turkey

ARTICLE INFO

Article history:
Received 20 May 2016
Revised 31 January 2017
Accepted 18 February 2017
Available online 16 March 2017

MSC:
00-01
99-00

Keywords:
Unsupervised learning
Decision trees
Autoencoders

ABSTRACT

The autoencoder is a popular neural network model that learns hidden representations of unlabeled data. Typically, single- or multilayer perceptrons are used in constructing an autoencoder, but we use soft decision trees (i.e., hierarchical mixture of experts) instead. Such trees have internal nodes that implement soft multivariate splits through a gating function and all leaves are weighted by the gating values on their path to get the output. The encoder tree converts the input to a lower dimensional representation in its leaves, which it passes to the decoder tree that reconstructs the original input. Because the splits are soft, the encoder and decoder trees can be trained back to back with stochastic gradient-descent to minimize reconstruction error. In our experiments on handwritten digits, newsgroup posts, and images, we observe that the autoencoder trees yield as small and sometimes smaller reconstruction error when compared with autoencoder perceptrons. One advantage of the tree is that it learns a hierarchical representation at different resolutions at its different levels and the leaves specialize at different local regions in the input space. An extension with locally linear mappings in the leaves allows a more flexible model. We also show that the autoencoder tree can be used with multimodal data where a mapping from one modality (i.e., image) to another (i.e., topics) can be learned.

1. Introduction

One of the basic tenets of statistics and machine learning is that though the data may be big, it can be explained by a small number of factors; that the data has an underlying pattern or regularity that can be explained by a small number of variables and their interaction, and the rest are variations that have no significant effect. For example, an image may contain a large number of pixels in many colors, but it can be broken into basic shapes such as lines and edges of different orientations. Similarly, a movie recommendations data set with many movies and customers may define a very large database but if we think about customer properties such as age, gender, and so on, and movie properties such as genre, year, and so on, we can posit the existence of simpler dependencies that underlie the large database. Hence, extraction of such hidden features and how they combine to explain the data is one of the most important research areas in statistics and machine learning, and many different methods have been proposed towards this aim [1].

One approach that can be used for this purpose is the autoencoder, which had been originally proposed almost thirty years ago but has recently become popular again with the popularity of deep learning. An autoencoder is composed of two models, the encoder and the decoder placed back to back. The encoder takes the original data as its input (e.g., an image) and maps it to an intermediate or hidden representation, and the decoder takes this hidden representation and reconstructs the original input. When the hidden representation uses fewer dimensions than the input, the encoder performs dimensionality reduction; one may impose additional constraints on the hidden representation, for example, sparsity. When both the encoder and decoder are implemented using differentiable models, they can be trained by stochastic gradient-descent together, backpropagating the reconstruction error at the output of the decoder first to the parameters of the decoder, and then by chain rule further on to the parameters of the encoder.

The idea is that if after training, the decoder can reconstruct the original input faithfully, the hidden representation should be a meaningful and useful one. Conventional autoencoders use a single layer, perceptron-type neural network for encoding and decoding, which implements an affine map followed by a nonlinearity for the encoder [2]. Using multilayer perceptrons with one or more hidden layers as encoder and decoder allow more complex mappings and hence more complex hidden representations to be learned.

One of the nice characteristics of the autoencoder is that it can be trained with unlabeled data, and once such a hidden representation is learned, it can then be taken as input to a supervised

* Corresponding author.
E-mail address: oirsoy@cs.cornell.edu (O. İrsoy).

http://dx.doi.org/10.1016/j.neucom.2017.02.075
0925-2312/

learner that can then be trained with a smaller labeled data set. It is also possible to train autoencoders sequentially where the next autoencoder is trained with the learned hidden representation of the previous autoencoder. Such an approach allows learning a sequence of increasingly complex and abstract representations which then can be stacked as successive layers in a deep network [3].

We explore an autoencoder model where the encoder and decoder are decision trees, instead of single or multilayer perceptrons [4]. We use the soft decision tree model whose internal nodes implement a soft multivariate split as defined by a gating function. The effect of the soft splitting is that the final output is not found by following a single path, but is the average of all the paths to all the leaves weighted by the soft gating values on each path [5,6]. Soft splitting also implies that the output of the tree is continuous and so we can use stochastic gradient-descent to update the parameters of encoder and decoder trees to minimize the reconstruction error. The training of the two trees are done together in a coupled manner: The error terms of the hidden representation of the decoding layer are passed back to the encoding layer as its external error using chain rule. Our simulation results we report below indicate that the autoencoder tree works as well, and sometimes better than the autoencoder perceptrons.

The rest of this paper is organized as follows: we review autoencoder perceptrons in Section 2 and then we discuss autoencoder trees in Section 3. Our experimental results in Section 4 indicate that such autoencoder trees can learn, in an unsupervised manner, hierarchical decompositions of data into subspaces which respect localities in the data. Section 5 concludes and discusses future work.

2. Autoencoder perceptrons

An autoencoder is a composition of two functions, the *encoder* and the *decoder*. The encoder maps the original d-dimensional input $\boldsymbol{x}$ to a k-dimensional intermediate or hidden representation $\boldsymbol{h} = \phi_e(\boldsymbol{x})$. The decoder maps this $\boldsymbol{h}$ back to a d-dimensional vector which we want to be as close as possible to the original input of the encoder: $\hat{\boldsymbol{x}} = \phi_d(\boldsymbol{h})$ [7,8]. We call this process a *reconstruction* and the difference between the output of the decoder and the input to the encoder is called the *reconstruction error*. If $k < d$, the encoder works as a dimensionality reducer, and as such learns the best k features that allow the reconstruction of the original higher dimensional input with minimum reconstruction error.

A simple way to realize an autoencoder is by using a single-layer perceptron for both the encoder and the decoder:

$$\phi_e(\boldsymbol{x}) = \sigma_e(W_e\boldsymbol{x}) = \boldsymbol{h} \tag{1}$$

$$\phi_d(\boldsymbol{h}) = \sigma_d(W_d\boldsymbol{h}) \tag{2}$$

where $\sigma(\cdot)$ is an elementwise nonlinearity function, such as the sigmoid $\sigma(x) = 1/(1+\exp(x))$. Alternatively, the nonlinearity can be omitted by setting $\sigma(\cdot)$ to the identity function, in which case the process simply becomes a linear projection.

Having a continuous definition with respect to the encoder and decoder weight parameters $\{W_e, W_d\}$, autoencoder perceptrons can be trained with a gradient-based optimization procedure using total reconstruction error as the objective function. It is possible to extend the capacity of the autoencoder by using multilayer perceptrons for each of the ϕ_e and ϕ_d. Deep autoencoders have been shown to outperform their shallow counterparts for the purpose of compression [9].

Many variants of the basic autoencoder model have been proposed in the literature: one might impose additional constraints on the hidden representation $\boldsymbol{h}$ to produce additional structural properties. One such soft constraint is the sparsity penalty, which results in a sparse autoencoder [10,11]. The addition of such a penalty encourages the encoding $\boldsymbol{h}$ to activate its features in a sparse fashion, resulting in a potentially useful feature set for some other end task such as classification.

Other modifications of the autoencoder framework include modifying the objective function. Denoising autoencoders, for instance, attempt to reconstruct a corrupted version of the input $\boldsymbol{x}$ [12–14]. Therefore the resulting autoencoder needs to undo this corruption rather than just reconstructing the original input. A possible interpretation of denoising autoencoders is that it attempts to learn the input manifold by realigning points that were perturbed and pushed back onto the manifold.

Besides the most prevalent application of dimensionality reduction, autoencoders can also be used as feature extractors with some other end task in mind, such as classification, information retrieval, or search [9,15,16]. In the case where there are other constraints on $\boldsymbol{h}$, such as the sparse autoencoder, the learned representation can even be overcomplete, i.e., have more dimensions than the input $\boldsymbol{x}$.

Autoencoder perceptrons have also been used with multimodal data where the aim is to learn cross-modality representations of multiple views of data, such as image and text [17–20]. Such multimodal representations are shown to improve upon single modality representations for end tasks such as retrieval or classification. It can also achieve transfer capabilities in the case where only a single modality is available at test time [17]. Similarly, for the case of multitask learning, autoencoders attempt to extract cross-task or cross-domain features [21,22].

Finally, by restructuring the encoder and decoder, one can accommodate the structure in data. For instance, one can use the recurrent neural network to encode an entire sequence into a fixed size vector and decode back. Such a sequence autoencoder can be used for semisupervised learning on natural language sentences, which are naturally modeled as sequences of vectors [23]. Similarly, recursive autoencoders are used to encode a tree structure, such as parse trees of natural language sentences, into a vector [24]. Representations that are learned this way are typically infused with syntactic properties as a byproduct of the usage of parse trees, and again, are useful for end tasks such as sentiment classification [25] or machine translation [26]. Note that in these instances, the additional chain or tree structure is part of the data, extrinsic to the model itself. Both recursive and recurrent neural networks are essentially perceptron layers recursively applied in a sequence or tree type computational graph, with parameter sharing across successive applications [27,28]. This is different from the autoencoder tree model that is presented here, in which the tree structure in intrinsic to the model, not the data. In fact, in all of our experiments, the input data is simply a vector without any extra structure. These concepts of structure are orthogonal: one can, for instance, replace the perceptron computation in a recursive neural network with a soft decision tree to get a recursive autoencoder tree, to apply a tree-structured model on tree-structured data.

3. Autoencoder trees

3.1. The model

In an autoencoder tree, the encoder and decoder are implemented by soft decision trees [4]. As opposed to the hard decision node which implements a hard split, a soft decision node redirects instances to all its children but with different probabilities, as given by a *gating function* $g_m(\boldsymbol{x})$—the hard decision tree is a special case where $g(\boldsymbol{x}) \in \{0, 1\}$. This architecture is equivalent to that of the hierarchical mixture of experts [5,6].

Let us consider a *soft binary tree* where each internal node has two children, named left and right. The response y_m at a node m

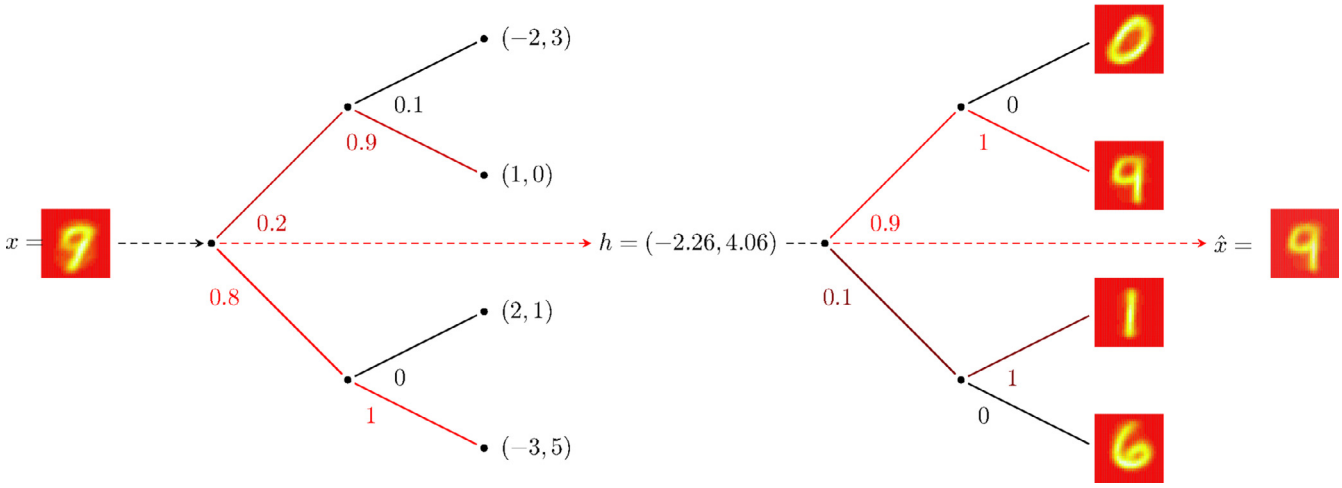

Fig. 1. The autoencoder tree is composed of an encoder tree and a decoder tree put back to back. The encoder tree takes the $28 \times 28 = 784$-dimensional input image x and generates a two-dimensional hidden representation $h = (-2.26, 4.06)$ by making a soft selection among its leaf responses. The decoder tree takes this as input and reconstructs the original 784-dimensional image at its leaves. In the encoder tree, the splits are 784-dimensional and the leaves are two-dimensional; in the decoder tree, the splits are two-dimensional and the leaves are 784-dimensional.

is recursively calculated as the weighted average of the responses of its left child ml and right child mr:

$$y_m(\boldsymbol{x}) = \begin{cases} \rho_m & \text{if } m \text{ is a leaf node} \\ g_m(\boldsymbol{x}) y_{ml}(\boldsymbol{x}) + (1 - g_m(\boldsymbol{x})) y_{mr}(\boldsymbol{x}) & \text{otherwise} \end{cases} \tag{3}$$

The gating function, $g_m(\boldsymbol{x}) \in [0, 1]$, softly chooses among the two outcomes by applying the *sigmoid function* to a linear split:

$$g_m(\boldsymbol{x}) = \frac{1}{1 + \exp[-(\boldsymbol{w}_m^T \boldsymbol{x})]} \tag{4}$$

Because the splits are soft, the output of a soft decision tree is continuous with respect to the parameters for a fixed tree structure. Hence, all the parameters, namely the leaf values and split hyperplane weight vectors at the internal nodes, i.e., $\{\boldsymbol{\rho}_m, \boldsymbol{w}_m\}_m$, can be learned by stochastic gradient-descent. For any error function, gradients with respect to the split and leaf parameters can be calculated over multiple levels using chain rule, i.e., backpropagation.

Using a soft decision tree as the main building block, we can construct an autoencoder tree by putting two soft decision trees, one as encoder and one as decoder, back to back. Let $\boldsymbol{t}(\boldsymbol{x})$ denote a soft decision tree as defined in Eqs. (3) and (4). The encoder tree encodes the d-dimensional input $\boldsymbol{x}$ into a $k < d$-dimensional hidden representation $\boldsymbol{h} = \boldsymbol{t}_e(\boldsymbol{x})$ and the decoder tree decodes the initial input from the hidden representation, $\hat{\boldsymbol{x}} = \boldsymbol{t}_d(\boldsymbol{h})$. This essentially replaces perceptrons with soft decision trees, by setting $\phi_e(\cdot) \equiv t_e(\cdot)$ and $\phi_d(\cdot) \equiv t_d(\cdot)$ in Eqs. (1) and (2). An example is shown in Fig. 1. Here, MNIST dataset is used where the encoder tree takes the $28 \times 28 = 784$-dimensional digit image as its input $\boldsymbol{x}$; it then uses Eq. (3) recursively and gets to all of the leaves—all the gating weight vectors of this encoder tree has $784 + 1$ dimensions. Here, we want to decrease dimensionality to two and so all the leaves store a two-dimensional value. We then calculate the average of all the leaves weighted by their gating values and this gives us the final two-dimensional representation for the digit. We denote this output by $\boldsymbol{h} = \boldsymbol{t}_e(\boldsymbol{x})$.

From this two-dimensional vector, the decoder tree, again uses Eq. (3) recursively, this time with $2 + 1$-dimensional gating weights. Because the aim is reconstruction of the input, the leaves contain 784-dimensional values. Again by calculating a weighted sum of all the leaves we get the final reconstructed image. We denote this by $\hat{\boldsymbol{x}} = \boldsymbol{t}_d(\boldsymbol{h}) = \boldsymbol{t}_d(\boldsymbol{t}_e(\boldsymbol{x}))$.

3.2. Training the autoencoder tree

The aim is to minimize the reconstruction error, that is, we want the output of the decoder tree to be as similar as possible to the input of the encoder tree. Note that this is an unsupervised criterion. The total reconstruction error on a training set $\{\boldsymbol{x}^i\}_{i=1}^N$ is given as:

$$E = \frac{1}{2} \sum_i \|\boldsymbol{x}^i - \hat{\boldsymbol{x}}^i\|^2 = \frac{1}{2} \sum_i \|\boldsymbol{x}^i - \boldsymbol{t}_d(\boldsymbol{t}_e(\boldsymbol{x}^i))\|^2 \tag{5}$$

Both the encoder and decoder trees use soft splits and hence given the reconstruction error at the output of the decoder tree, we can use chain rule and backpropagate this error to all the split and leaf parameters of first the decoder and then the encoder tree. Let us look at Fig. 1: when we use the model for evaluation, given the input, we go in the forward direction and evaluate first the encoder then the decoder tree. In training, we go in the opposite direction. After having evaluated and calculated the output predictions, given the desired output for the decoder tree, we compare the predictions against the desired values and calculate the reconstruction error; using backpropagation, that is, the chain rule, we update the parameters of the decoder tree and then back propagating even further, we update the parameters of the encoder tree.

In order to learn the parameters $\boldsymbol{\rho}$ and $\boldsymbol{w}$ of a single soft decision tree, with a gradient-based method, we use backpropagation. Let E denote the overall error to be minimized and E^i denote the error over a single data instance $\boldsymbol{x}^i$. Let us define $\delta_m = \partial E^i / \partial y_m(\boldsymbol{x}^i)$, which is the *responsibility* of node m. Backpropagating the error from the root towards the leaves, we have

$$\frac{\partial E^i}{\partial \boldsymbol{w}_m} = g_m(\boldsymbol{x}^i)(1 - g_m(\boldsymbol{x}^i)) \left(\delta_m^{iT} (\boldsymbol{y}_{ml}(\boldsymbol{x}^i) - \boldsymbol{y}_{mr}(\boldsymbol{x}^i))\right) \boldsymbol{x}^i \tag{6}$$

$$\frac{\partial E^i}{\partial \boldsymbol{\rho}_m} = \boldsymbol{\delta}_m^i \tag{7}$$

with

$$\boldsymbol{\delta}_m^i = \begin{cases} \boldsymbol{y}_r(\boldsymbol{x}) - \boldsymbol{r}^i & \text{if } m \text{ is root} \\ \boldsymbol{\delta}_{pa(m)}^i g_m(\boldsymbol{x}^i) & \text{if } m \text{ is a left child} \\ \boldsymbol{\delta}_{pa(m)}^i (1 - g_m(\boldsymbol{x}^i)) & \text{if } m \text{ is a right child} \end{cases} \tag{8}$$

where $pa(m)$ is the parent of node m and $\boldsymbol{r}^i$ is the true label of ith instance, represented as a one-hot vector.

We can use these derivations to compute the gradients of an autoencoder tree pair as follows: let $\theta_e \in \{\boldsymbol{\rho}_m, \boldsymbol{w}_m\}_{m \in e}$ and $\theta_d \in \{\boldsymbol{\rho}_m, \boldsymbol{w}_m\}_{m \in d}$ denote a parameter of the encoder and decoder trees respectively. We can update the parameters of both trees by gradient-descent using:

$$\frac{\partial E^i}{\partial \theta_d} = \frac{\partial E^i}{\partial \boldsymbol{t}_d(\boldsymbol{h}^i)} \frac{\partial \boldsymbol{t}_d(\boldsymbol{h}^i)}{\partial \theta_d} = (\hat{\boldsymbol{x}}^i - \boldsymbol{x}^i)^T \frac{\partial \boldsymbol{t}_d(\boldsymbol{h}^i)}{\partial \theta_d} \tag{9}$$

$$\frac{\partial E^i}{\partial \theta_e} = \frac{\partial E^i}{\partial \boldsymbol{h}^i} \frac{\partial \boldsymbol{h}^i}{\partial \theta_e} = (\hat{\boldsymbol{x}}^i - \boldsymbol{x}^i)^T \frac{\partial \boldsymbol{t}_d(\boldsymbol{h}^i)}{\partial \boldsymbol{h}^i} \frac{\partial \boldsymbol{t}_e(\boldsymbol{x}^i)}{\partial \theta_e} \tag{10}$$

where $\partial \boldsymbol{t}_\cdot(\boldsymbol{x})/\partial\theta_\cdot$ can be computed as shown above for a single tree. To be able to calculate $\boldsymbol{\delta h} = \partial E/\partial \boldsymbol{h}$, we need the derivative of a tree response with respect to its input (for the decoder tree):

$$\frac{\partial \boldsymbol{t}(\boldsymbol{x}^i)}{\partial \boldsymbol{x}^i} = \sum_m g_m(\boldsymbol{x}^i)(1 - g_m(\boldsymbol{x}^i))\left(\delta_m^{iT}(y_{ml}(\boldsymbol{x}^i) - y_{mr}(\boldsymbol{x}^i))\right) w \tag{11}$$

δh denotes the error responsibility of the hidden representation backpropagated from the decoder tree to the encoder tree.

With trees having multiple levels, training the encoder and decoder trees together would be rather slow and to accelerate training, we use an incremental procedure, which is also sometimes used in training autoencoder perceptrons. Initially, both trees start with depth two, and after some number of epochs, in both trees, every leaf is replaced by a subtree of depth two with one decision node and two leaves; this adds a level to both trees. We continue doing so until we get to some predefined maximum depth. When the tree grows, we update all the tree parameters thereby finetuning the previous parameters as well. When a leaf is split, its response $\boldsymbol{\rho}$ is copied to its children leaves with some small additive noise.

3.3. Training complexity

What determines the complexity of the autoencoder perceptron is the complexity of a perceptron, which basically is a matrix–vector product. So if we have d inputs and k hidden units, the time complexity is $O(d \cdot k)$. The space complexity is the same.

If the autoencoder uses multilayer perceptrons with one hidden layer of n nodes for encoding and decoding, the encoder has time (and space) complexity of $O(d \cdot n)$ in the first layer and $O(n \cdot k)$ in the second layer, and hence the encoder has a total complexity of $O(n \cdot (d + k))$. We can simplify this as $O(n \cdot d)$ because d will be larger than k in the more frequently encountered use of the autoencoder for dimensionality reduction. The same also holds for the decoder tree when we exchange d and k.

In the case of a tree, the complexity depends on the number of internal nodes. An internal node has time (and space) complexity of $O(d)$ and if the encoder (or decoder) tree has n nodes, the total complexity of the tree is $O(n \cdot d)$. If the hidden dimensionality is k, the two children can be combined in $O(k)$ time with a total of $O(n \cdot k)$ complexity. Summing these, we get $O(n(d + k))$ which again can be taken as $O(n \cdot d)$, assuming $d > k$. Note that again this is true also for the decoder tree, when we exchange d and k.

This analysis indicates that the autoencoder tree and the autoencoder multilayer perceptron have the same time and space complexity if the number of nodes in the trees and the number of hidden units in the multilayer perceptrons are comparable.

To be more concrete, we did timing experiments on an 8-core CPU machine on which matrix multiplications are parallelized. We use the MNIST data which has 60,000 training examples of dimension 784, and update in batches of size 256. A single training epoch for a 6-level deep (63 nodes per encoder and decoder) autoencoder tree takes on the average 2.598 s; an equivalent 2-layer (per encoder and decoder) autoencoder multilayer perceptron which has the same number of parameters (31, 10, and 31 hidden units) takes on the average 2.364 s.

3.4. Autoencoder model trees

The leaves of a soft decision tree ρ_m are constant vectors and hence the response of a (soft) tree can be viewed as a (smoothed) piecewise constant function. We can make the model more flexible by using linear models at the leaves. We change Eq. (3) as follows for leaf node m:

$$\boldsymbol{\rho}_m = \boldsymbol{V}_m \boldsymbol{x} \tag{12}$$

where $\boldsymbol{V}_m$ are the parameters of the linear model. During gradient-descent, we include one more term, $\partial \boldsymbol{\rho}_m / \partial \boldsymbol{V}_m$, to learn $\boldsymbol{V}_m$. The use of linear leaves implies that the overall response of the tree can be viewed as a (smoothed) piecewise linear function.

Having linear response leaves in the encoder and decoder trees adds flexibility to the autonencoder because it allows for a distributed representation at the leaf level—the leaves are (softly) local and specialize to regions in the input space, but in each leaf, we have a distributed model. We expect nearby regions to learn similar linear models and the soft gating mechanism allow a smooth transition from one to the other. When there is no additional nonlinearity, such an autoencoder might be interpreted as a hierarchical mixture of locally linear projections.

4. Experiments

4.1. Results on handwritten digits data

We evaluate the autoencoder tree model on MNIST which contains 60,000 training and 10,000 test examples of handwritten digit images, each of which is 28 by 28 pixels (784-dimensional) [29]. Output labels are the ten digits.

We use two autoencoder perceptrons: One has a single layer, that is, a linear model, with output nonlinearity for the encoder and linear map for the decoder. The second uses the autoencoder with a stacked two-layer perceptron where we first reduce the dimensionality to a larger, intermediate value (such as 50) using the conventional autoencoder, and using the 50-dimensional representation, we once again reduce to the final dimensionality (10 or 2). In both cases, nonlinearity is the hyperbolic tangent.

Both autoencoder perceptrons and autoencoder trees are trained with stochastic gradient-descent. For both, we employ a diagonal variant of AdaGrad [30], which yields smooth and fast convergence. We train for a total of 240 epochs. Both the encoder and the decoder trees start from a depth of two, and the depth is incremented at every 40th epoch, until they reach their final depth (five or six). We employ a simple $L2$ regularization on connection weights for autoencoder perceptrons and hyperplane split parameters ($\boldsymbol{w}_m$) and the leaf responses ($\boldsymbol{\rho}_m$) for autoencoder trees with a fixed penalty of 10^{-5}. Both the autoencoder perceptron and autoencoder tree weights are uniformly randomly initialized in the interval of $[-0.01, 0.01]$—in the case of autoencoder trees, this is also the noise added on top of the inherited weights from the parent nodes. We show the convergence of the reconstruction errors for all models per each epoch of stochastic gradient-descent, in the scale of a single pixel in Fig. 2. We see that the autoencoder trees reaches a lower reconstruction error than the autoencoder perceptron when reducing to two dimensions, whereas the autoencoder perceptron is better than the tree variants when reducing to ten dimensions. For the autoencoder trees, it seems as if the dimensionality of the hidden representation does not have a strong effect. However, the error is less when the depth of tree is increased. For the autoencoder tree therefore, the size of the tree (in terms of its depth) seems critical, and not the dimensionality of the hidden

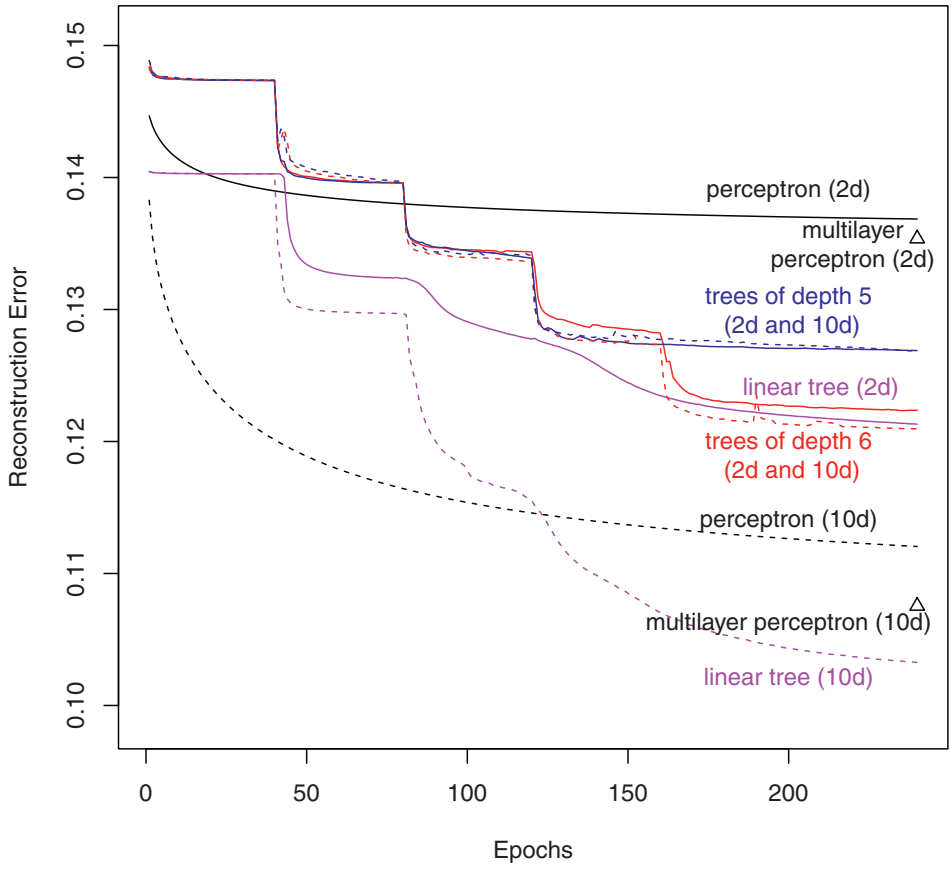

Fig. 2. We compare the reconstruction errors of different autoencoder architectures on the MNIST data set, namely the autoencoder perceptron (black), the autoencoder tree with a depth of five (blue), and the autoencoder tree with a depth of six (red). Purple denote the autoencoder model tree with dimensionality reduced to two. Solid and dash denote dimensionality reduction to two and ten, respectively. Triangles show the multilayer perceptron that first reduces the dimensionality to 50 and then reduces once more to 2 or 10. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

representation. The reconstruction error can be decreased by using larger trees for encoding and decoding but one should be careful because using models that are too complex may lead to overfitting the data. *L2* regularization, as we do, becomes important in such a case. One can try different models and choose the one that performs best on a held-out data. Another factor that determines the reconstruction error is the dimensionality of the hidden representation. In our experiments, we use two or ten which define a serious bottleneck; using a larger hidden representation between the encoder and decoder would both decrease reconstruction error and speed up learning, at the expense of losing from compactness.

We plot the distribution of MNIST digits in the learned two dimensions in Fig. 3. We see that the autoencoder perceptron maps the instances to the corners because of the nonlinearity and hence loses information. With the autoencoder trees, leaves define clusters and we also see points intermediate implying interpolations (due to soft gating) between leaves.

With trees, unlike perceptrons, we learn a localized representation. It is not the directions of the hidden space that is informative, but we learn clusters at different resolutions. We see the autoencoder tree as doing dimensionality reduction together with hierarchical soft clustering. Because of this locality and clustering behavior, it is the structure of the tree and the resulting number of leaves that define the capacity more than the dimensionality of the leaves, whereas for the perceptrons with their distributed representation, it is the number of hidden dimensions that is critical.

In the case of autoencoder model trees, we see that classes are very well-separated even in two dimensions although the training is unsupervised, indicating that the model has effectively captured the underlying distribution. In contrast to other models, autoencoder linear model trees provide a much smoother distribution in the hidden representation space, and move away from the clustering-like behavior to a degree. This validates our intuition that linear models help incorporate a distributed representation in addition to locality. More results on this data set are in [4].

4.2. Results on newsgroup text data

We further evaluate autoencoder trees on 20 Newsgroups data (20News) which contains 18,846 instances of newsgroup documents (partitioned into training and test sets with 60–40% ratio), with output labels denoting the subject matter (category) out of 20 classes [31]. With the bag-of-words representation, the dimensionality is about 60,000, but we sort the words in terms of their frequencies, discard the top 100 (non informative stop words), and use the next 2000.

We compare autoencoder trees to the shallow autoencoder perceptron. We borrow the same learning scheme and set of hyperparameters from MNIST experiments described above. More results on this data set can be found in [4]; here, we will only show a part of the decoder tree learned over the 20News dataset, which we believe is very interesting and informative (Fig. 4). Since the response vector is a bag-of-words representation, we sort the words by their coefficients and show only the top words. We show only some of the paths as an example. In the figure, we observe hierarchies captured by the tree as seen by word distributions; these resemble topics at finer and finer grain as we split the nodes further.

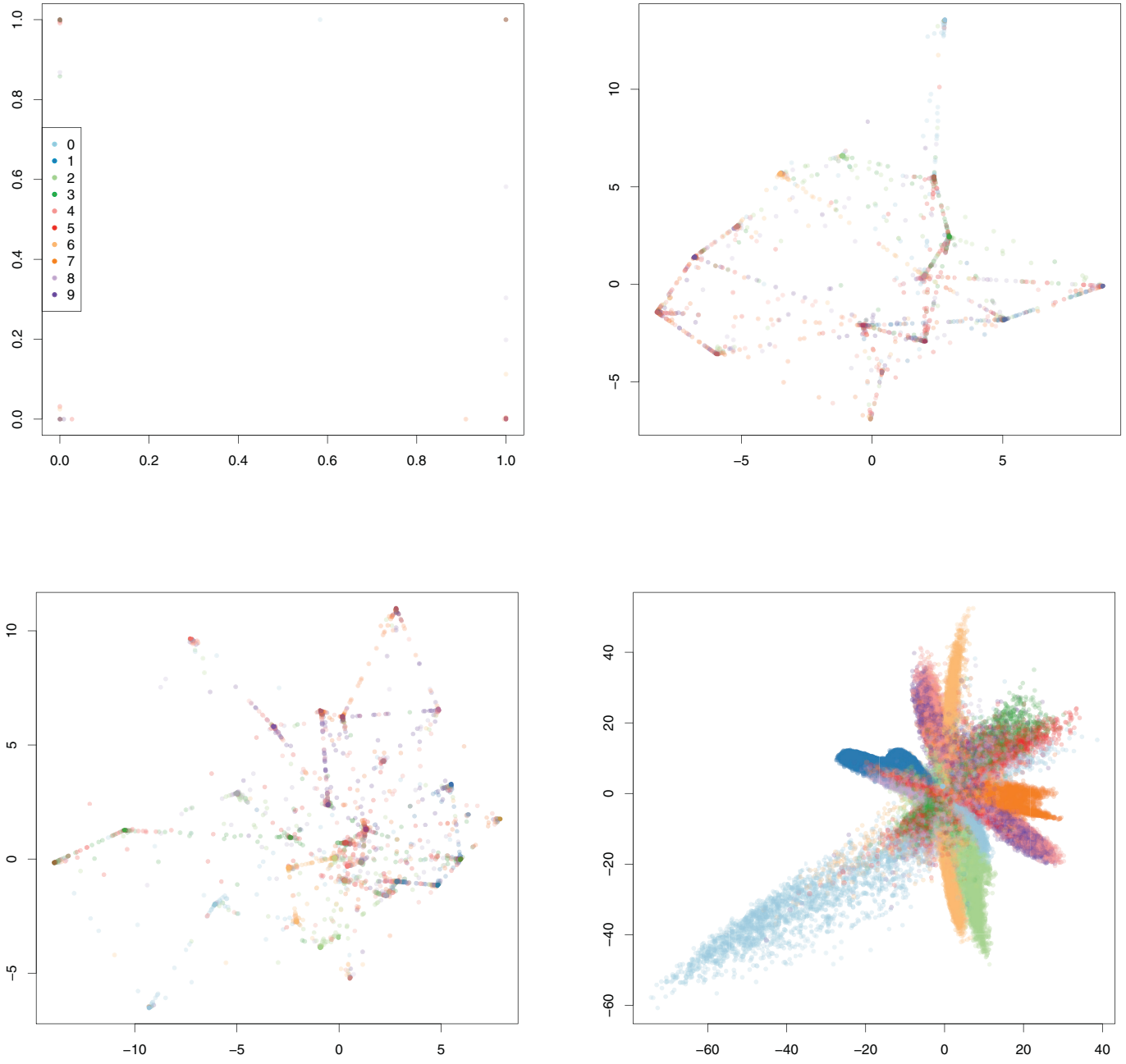

Fig. 3. MNIST digits mapped to two dimensions using the autoencoder perceptron (top left), the autoencoder trees of depth five (top right) and depth six (bottom left), and the autoencoder model tree of depth five (bottom right).

What we see here is very similar to one gets with a hierarchical topic model [32]—we can think of every document represented by a distribution on the leaves of the tree (with the gating function) and every leaf defines a distribution on words (by normalizing leaf responses). In the figure for example, the leftmost leaf roughly corresponds to concepts related to internet anonymity, the next leaf is about programs and output methods, and the last two are about internet publishing of images. We also see that as we get closer to leaves, the resolution increases and the word activations become more specific.

4.3. Results on natural image patches

We evaluate the autoencoder tree on CIFAR which is a labeled set of 50,000 training and 10,000 test images, each of which is 32 by 32 pixels in three channels [33]. Since we intend to qualitatively evaluate and interpret smaller models, we convert the images to grey levels and work at a patch level of size 7 by 7, with an overlap of 3 pixels. We then train autoencoders at the patch level. This is to ensure that even small models can produce reasonable results, so that we can easily interpret learned models.

To report example reconstructions, the autoencoder tree and the autoencoder perceptron reduce dimensionality to ten. We also use the autoencoder model tree that reduces the dimensionality to two for visualization purposes. In the case of the autoencoder perceptron, we employ a pair of single hidden layer perceptrons for encoder and decoder both having 25 hidden units. Autoencoder trees have a maximum depth of six. Other hyperparameters and the learning scheme follow from the previous setting. Fig. 5 shows the convergence of reconstruction errors of the au-

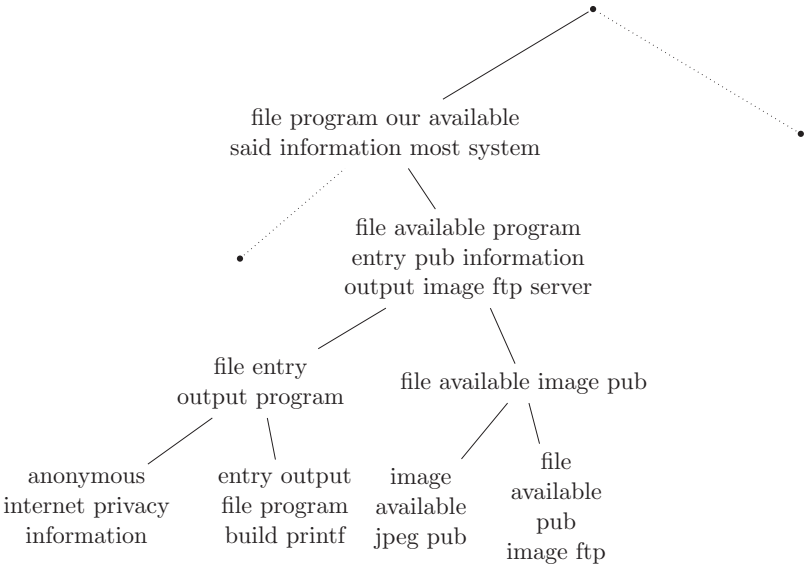

Fig. 4. Response values (the most chosen words) for a subtree of the decoder tree on the 20News data set. Internal nodes show the latest response values before splitting. We see that the leaves contain related words and the tree learns a hierarchical distribution of topics.

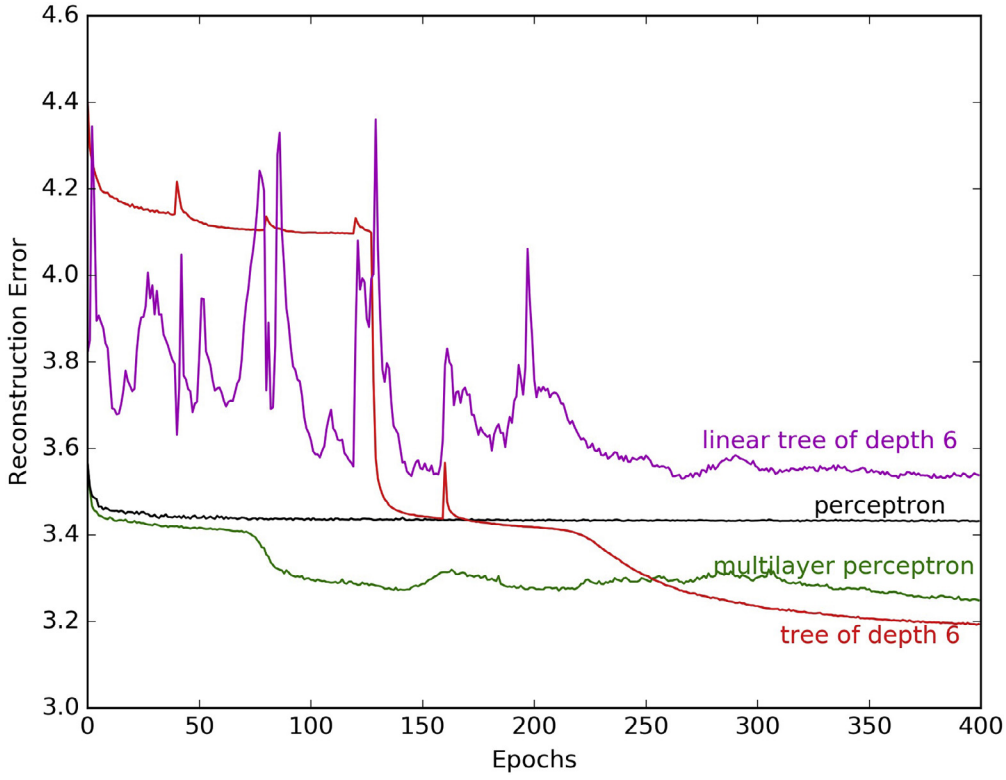

Fig. 5. We compare the reconstruction errors of different autoencoder architectures on image patches from the CIFAR data set, namely the autoencoder perceptron (black), the autoencoder multilayer perceptron with 25 hidden units (green), the autoencoder tree with a depth of six (red) and the autoencoder model tree with a depth of six (purple). All models compared here reduce dimensionality to two. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

toencoder tree and perceptrons. We see that the autoencoder tree and the multilayer perceptron perform better than the autoencoder model tree and the perceptron. This last has the smoothest convergence since it has the smallest number of parameters. The autoencoder model tree performs not as well and oscillates, likely due to high variance because of the larger number of parameters. With trees, we see a spike when we increase the depth by splitting the leaves.

As seen in Fig. 6, both the autoencoder perceptron and the autoencoder trees yield roughly similar reconstructions. The autoencoder model tree has the most blurry reconstructions, likely because reducing to two dimensions results in a very tight bottleneck. Fig. 7 shows the decoder of an autoencoder tree. We see that as the tree grows in size, the representations become more and more specific, in the sense that the leaf nodes specialize to a narrower part of the input space, as we also see in Fig. 4. Deep

Fig. 6. Random sample reconstructions for autoencoder with multilayer perceptron (left), autoencoder tree (middle) and autoencoder model tree (right). Autoencoder model tree reduces to two dimensions whereas the other models reduce to ten.

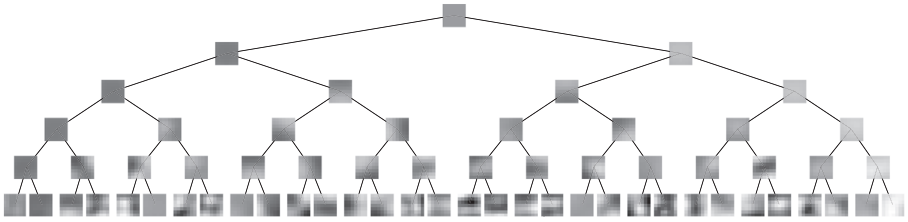

Fig. 7. Decoder tree on CIFAR patches. Image patches denote ρ values of leaves. Internal nodes show the last ρ value before the node is split.

multilayer perceptrons, or in general deep neural networks, may present a similar behavior in which representations become more specialized and specific in a representation hierarchy. The depth of a tree and the depth of a layered perceptron are not immediately interchangable, but an investigation of the effect of depth on the granularity of the representation learned with these two models and their comparison would be an interesting research direction.

We also visualize the resulting two-dimensional scatter of patches for the case of the autoencoder model tree; see Fig. 8. We observe that nearby patches have similar orientations. Additionally, the overall spread in the encoding space seems to correlate with the overall color of patches, such that there is a transition from black to white as we move from left to right.

4.4. Results on multimodal data of images with topics

To show that the autoencoder tree, just like autoencoder perceptron, can be used in a multimodal scenario, we evaluate it on MIRFLICKR 25k data set which has 25,000 Flickr raw images with associated topic labels [34]. We randomly partition the data into training and test sets with 4–1 ratio. We treat topic labels as a modality of the data and attempt to reconstruct them from images (Fig. 9). Topics are represented as a 38-dimensional vector of binary values (not necessarily one-hot). Images are represented as a 3857-dimensional feature vector, using the preprocessed and extracted image features as in [35]. Both the autoencoder tree and the autoencoder perceptron have a hidden dimensionality of ten. Encoder and decoder perceptrons have a single hidden layer of dimension 200 and 25 respectively, and all layers are learned jointly. Autoencoder tree has a maximum depth of six. Because the encoder and decoder models are trained together to generate one modality from the other, the hidden representation in between extracts cross-modalty features.

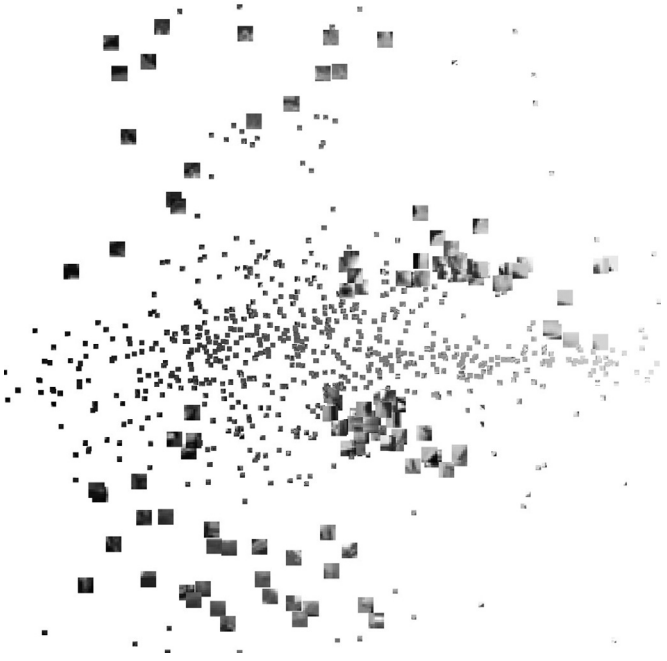

Fig. 8. Two-dimensional scatter of encodings of 1000 random patches with the autoencoder model tree. Certain patches are enlarged for better visibility. We observe an overall transition from black to white as we move from left to right. We also see that similarly oriented patches are placed nearby.

Five randomly chosen samples and their estimated topics (top five predictions) are shown in Table 1 for both autoencoder perceptrons and autoencoder trees. We see that both models produce reasonable results—the autoencoder tree seems to fail on the

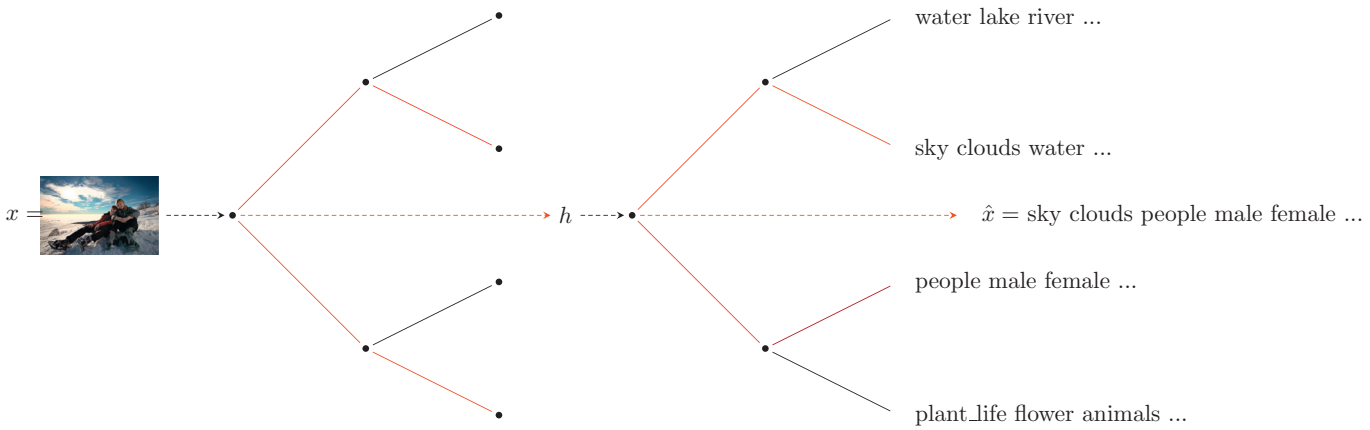

Fig. 9. Operation of an autoencoder tree on multimodal image and text data. Given an input image x, the encoder tree maps it to a hidden representation h. Then, the decoder tree generates the textual topic representation from this h. Because the two trees are trained together, h learns to be a cross-modality representation.

Table 1
Sample images, their topics and topic predictions using autoencoder multilayer perceptron and autoencoder trees on MIRFLICKR.

Image	True labels	MLP	AETR
	people_r1 female_r1 sky structures female people	Indoor structures plant_life animals people	Indoor food animals people plant_life
	people_r1 clouds_r1 male_r1 female_r1 plant_life sky male tree sunset female clouds people	Sky plant_life clouds structures tree	Sky clouds clouds_r1 water plant_life
	Animals river bird_r1 bird water lake	Indoor structures plant_life animals people	plant_life sky animals structures tree
	Flower plant_life sky structures clouds flower_r1	Structures plant_life indoor sky people	People people_r1 portrait portrait_r1 indoor
	people_r1 female_r1 male structures female people	People people_r1 female male portrait	Structures people male people_r1 plant_life

fourth instance; note however that the autoencoder perceptron has a larger number of parameters in these experiments.

Fig. 10 shows the convergence of the reconstruction errors for the autoencoder tree and the autoencoder multilayer perceptron. We see that the perceptron achieves good performance quickly, but quickly starts to overfit (we use early stopping, that is, our qualitative results above uses the best performing iteration on the development set). The autoencoder tree model converges smoothly but starts overfitting too when trained too long. Both models perform similarly at their peak.

5. Conclusions

The autoencoder tree uses two soft decision trees back to back for encoding and decoding. We show that such an autoencoder reaches as low or lower reconstruction error than autoencoder perceptrons on several data sets. A major advantage of the autoencoder tree is that because of the locality of the leaves and its hierarchical structure, it is easier to interpret and hence allows for information extraction.

When used in this unsupervised setting, the autoencoder tree implements soft hierarchical clustering and when the hidden representation uses fewer dimensions than the input, it does dimensionality reduction within the clusters. The autoencoder tree does these two in a coupled manner learning both the hierarhical splitting and the hidden representations in the leaves together. The hierarchical organization of the decoder tree and the soft splitting at the nodes allow for a smooth interpolation between the leaves and makes the model more flexible.

The autoencoder tree is an unsupervised method that finds a short, compressed representation of the input. In a semi-supervised setting, such a learned hidden representation can then be given as input to a supervised learner for a later classification or regression task, which can be trained with a smaller labeled data set. We see that though the training is unsupervised, the autoencoder tree learns internal nodes or leaves that become increasingly responsive to single or few classes, and we expect this to be useful in the later training of a supervised mapping.

The gating values on the path from the root to a leaf softly delimit a local region in the input space which becomes more specific, or in other words, higher resolution, as the tree depth increases. Not only is such a representation very informative, but it can also improve prediction accuracy on supervised tasks.

As is, the structure of the encoder and decoder trees are fixed and pre-defined (as is also the case for autoencoder perceptrons). We have recently proposed the budding decision tree model where the tree structure can be adapted during training by dynamically growing and pruning subtrees [36]; a possible research direction is to use such budding trees as encoder and decoders.

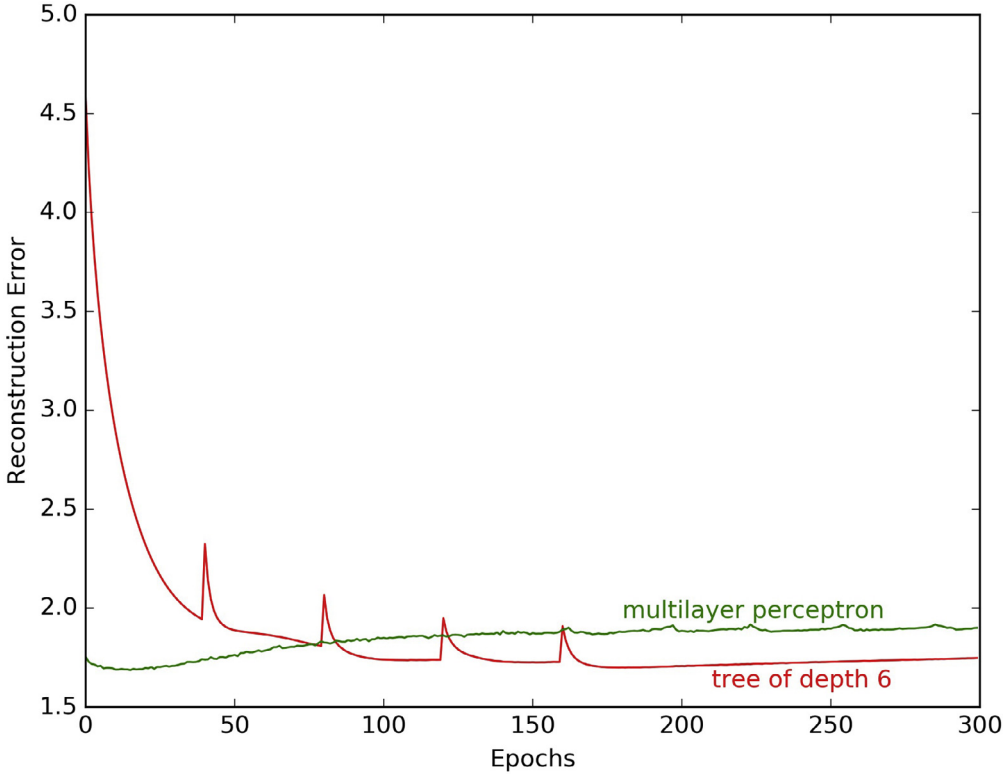

Fig. 10. We compare the reconstruction errors of different autoencoder architectures on the FLICKR data set, namely the autoencoder multilayer perceptron with 200 hidden units in the encoder and 25 hidden units in the decoder (green), and autoencoder tree with a depth of six. Both models reduce to 10 dimensions. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Denoising autoencoders (when there are missing features), recursive autoencoders (when inputs are sequences), and other autoencoder variants in the literature currently use perceptrons as their building blocks; it will be interesting to implement their tree variants. We expect the hierarchical, multi-resolution representation capability of trees will be useful especially in interpretation.

Acknowledgments

This work is partially supported by Boğaziçi University Research Funds with Grant number 14A01P4.

References

[1] E. Alpaydın, Introduction to Machine Learning, third edition, The MIT Press, 2014.
[2] G.W. Cottrell, P. Munro, D. Zipser, Learning internal representations from gray-scale images: an example of extensional programming, in: Proceedings of the Ninth Annual Conference of the Cognitive Science Society, 1987, pp. 462–473.
[3] Y. Bengio, Learning deep architectures for AI, Found. Trends Mach. Learn. 2 (1) (2009) 1–127.
[4] O. İrsoy, E. Alpaydın, Autoencoder trees, in: Proceedings of the 2015 Asian Conference on Machine Learning, 2015, pp. 378–390.
[5] M.I. Jordan, R.A. Jacobs, Hierarchical mixtures of experts and the EM algorithm, Neural Comput. 6 (2) (1994) 181–214.
[6] O. İrsoy, O.T. Yıldız, E. Alpaydın, Soft decision trees, in: Proceedings of the 2012 International Conference on Pattern Recognition, 2012, pp. 1819–1822.
[7] H. Bourlard, Y. Kamp, Auto-association by multilayer perceptrons and singular value decomposition, Biol. Cybern. 59 (4–5) (1988) 291–294.
[8] G.E. Hinton, R.S. Zemel, Autoencoders, minimum description length, and Helmholtz free energy, in: Proceedings of the 1993 Advances in Neural Information Processing Systems, 1993, pp. 3–10.
[9] G.E. Hinton, R.R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (5786) (2006) 504–507.
[10] C.P. Marc'Aurelio Ranzato, S. Chopra, Y. LeCun, Efficient learning of sparse representations with an energy-based model, in: Proceedings of the 2007 Advances in Neural Information Processing Systems, 2007, pp. 1137–1144.
[11] M.A. Ranzato, F.J. Huang, Y.-L. Boureau, Y. LeCun, Unsupervised learning of invariant feature hierarchies with applications to object recognition, in: Proceedings of the 2007 Conference on Computer Vision and Pattern Recognition, IEEE, 2007, pp. 1–8.
[12] G. Alain, Y. Bengio, What regularized auto-encoders learn from the data-generating distribution, J. Mach. Learn. Res. 15 (1) (2014) 3563–3593.
[13] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (2010) 3371–3408.
[14] Y. Bengio, L. Yao, G. Alain, P. Vincent, Generalized denoising auto-encoders as generative models, in: Proceedings of the 2013 Advances in Neural Information Processing Systems, 2013, pp. 899–907.
[15] A. Torralba, R. Fergus, Y. Weiss, Small codes and large image databases for recognition, in: Proceedings of the 2008 Conference on Computer Vision and Pattern Recognition, IEEE, 2008, pp. 1–8.
[16] R. Salakhutdinov, G. Hinton, Semantic hashing, Int. J. Approx. Reason. 50 (7) (2009) 969–978.
[17] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, A.Y. Ng, Multimodal deep learning, in: Proceedings of the 2011 International Conference on Machine Learning, 2011, pp. 689–696.
[18] W. Wang, B.C. Ooi, X. Yang, D. Zhang, Y. Zhuang, Effective multi-modal retrieval based on stacked auto-encoders, in: Proceedings of the 2014 International Conference on Very Large Data Bases (VLDB) Endowment, 2014, pp. 649–660.
[19] C. Hong, J. Yu, J. Wan, D. Tao, M. Wang, Multimodal deep autoencoder for human pose recovery, Trans. Image Process. 24 (12) (2015) 5659–5670.
[20] Y. Liu, X. Feng, Z. Zhou, Multimodal video classification with stacked contractive autoencoders, Signal Process. 120 (2016) 761–766.
[21] F. Zhuang, D. Luo, X. Jin, H. Xiong, P. Luo, Q. He, Representation learning via semi-supervised autoencoder for multi-task learning, in: Proceedings of the 2015 International Conference on Data Mining, IEEE, 2015, pp. 1141–1146.
[22] M. Ghifary, W. Bastiaan Kleijn, M. Zhang, D. Balduzzi, Domain generalization for object recognition with multi-task autoencoders, in: Proceedings of the 2015 International Conference on Computer Vision, IEEE, 2015, pp. 2551–2559.
[23] A.M. Dai, Q.V. Le, Semi-supervised sequence learning, in: Proceedings of the 2015 Advances in Neural Information Processing Systems, 2015, pp. 3061–3069.
[24] R. Socher, E.H. Huang, J. Pennington, C.D. Manning, A.Y. Ng, Dynamic pooling and unfolding recursive autoencoders for paraphrase detection, in: Proceedings of the 2011 Advances in Neural Information Processing Systems, 2011a, pp. 801–809.
[25] R. Socher, J. Pennington, E.H. Huang, A.Y. Ng, C.D. Manning, Semi-supervised recursive autoencoders for predicting sentiment distributions, in: Proceedings of the 2011 Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2011b, pp. 151–161.

[26] P. Li, Y. Liu, M. Sun, Recursive autoencoders for ITG-based translation., in: Proceedings of the 2013 Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2013, pp. 567–577.
[27] J.L. Elman, Finding structure in time, Cognit. Sci. 14 (2) (1990) 179–211.
[28] J.B. Pollack, Recursive auto-associative memory., Neural Netw. 1 (1988) 122.
[29] Y. LeCun, C. Cortes, The MNIST database of handwritten digits, 1998.
[30] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (2011) 2121–2159.
[31] 20Newsgroups, URL: http://qwone.com/~jason/20Newsgroups/.
[32] D.M. Blei, T.L. Griffiths, M.I. Jordan, J.B. Tenenbaum, Hierarchical topic models and the nested chinese restaurant process, in: Proceedings of the 2003 Advances in Neural Information Processing Systems, 2003, pp. 17–24.
[33] A. Krizhevsky, Learning Multiple Layers of Features from Tiny Images, University of Toronto, 2009 Master's thesis.
[34] M.J. Huiskes, M.S. Lew, The MIR Flickr retrieval evaluation, in: Proceedings of the 2008 International Conference on Multimedia Information Retrieval, ACM, 2008, pp. 39–43.
[35] N. Srivastava, R. Salakhutdinov, Multimodal learning with deep Boltzmann machines, J. Mach. Learn. Res. 15 (2014) 2949–2980.
[36] O. İrsoy, O.T. Yıldız, E. Alpaydın, Budding trees, in: Proceedings of the 2014 International Conference on Pattern Recognition, 2014, pp. 3582–3587.

Ozan İrsoy received the B.A. degree in mathematics and the B.Sc. degree in computer engineering (double major) from Bogazici University in 2012 and is currently working toward the Ph.D. degree in computer science at Cornell University.

Ethem Alpaydin received the B.Sc. degree from Bogazici University in 1987 and the Ph.D. degree from EPF Lausanne in 1990. He did his postdoctoral work at ICSI, Berkeley, in 1991 and he visited MIT in 1994, ICSI, Berkeley, in 1997 (as a Fulbright scholar) and IDIAP, Switzerland, in 1998. He has been a professor of computer engineering at Bogazici University since 1991. His book Introduction to Machine Learning was published by The MIT Press in 2004, which since has been translated to German, Chinese, and Turkish. He is senior member of the IEEE.