Neurocomputing 419 (2021) 148–156

Contents lists available at ScienceDirect

Neurocomputing

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

# Dropout regularization in hierarchical mixture of experts

Ozan İrsoy [a], Ethem Alpaydın [b,*]

[a] Bloomberg L.P., 731 Lexington Ave, New York, NY 10022, USA
[b] Department of Computer Science, Özyeğin University, Çekmeköy 34794, İstanbul, Turkey

ARTICLE INFO

*Article history:*
Received 1 October 2019
Revised 8 July 2020
Accepted 28 August 2020
Available online 3 September 2020
Communicated by Li Sheng

*Keywords:*
Mixture of experts
Hierarchical models
Regularization
Dropout

ABSTRACT

Dropout is a very effective method in preventing overfitting and has become the go-to regularizer for multi-layer neural networks in recent years. Hierarchical mixture of experts is a hierarchically gated model that defines a soft decision tree where leaves correspond to experts and decision nodes correspond to gating models that softly choose between its children, and as such, the model defines a soft hierarchical partitioning of the input space. In this work, we propose a variant of dropout for hierarchical mixture of experts that is faithful to the tree hierarchy defined by the model, as opposed to having a flat, unitwise independent application of dropout as one has with multi-layer perceptrons. We show that on a synthetic regression data and on MNIST, CIFAR-10, and SSTB datasets, our proposed dropout mechanism prevents overfitting on trees with many levels improving generalization and providing smoother fits.



## 1. Introduction

In machine learning, we care about the generalization performance of our predictor over unseen data. Regularization is one of the fundamental methods to combat overfitting the training data, by preferring smaller, simpler, or smoother models. Traditional regularizers work by pushing model parameters to smaller values (e.g., L1/L2 regularizers), or increasing the model sparsity.

Deep neural networks composed of many layers with many units can overfit and a number of regularization methods have been proposed. One such method is the dropout in which each unit is randomly dropped (along with its connections) from the network with a certain probability [1]. Intuitively, this reduces the co-adaptation of units by forcing them not to rely too much on one another. This is equivalent to sampling from an exponential number of networks of different complexities at training time.

The dropout method has spurred research and several variants have been proposed, such as drop-connect where each connection can be separately dropped [2], fast-dropout that performs a Gaussian approximation to the implied objective [3], and word-dropout in natural language processing where each word in a given sentence is dropped out (or replaced with the unknown token) [4]. Later work grounded the approach in a stronger theoretical basis [5,6].

The hierarchical mixture of experts (HMoE) is a meta-model that combines several models using a gating function that is defined with respect to a tree hierarchy [7]. When the gating function and the individual experts are differentiable with respect to their parameters, the whole tree can be trained using stochastic gradient-descent (SGD), just like the weights of a neural network given its structure. The hierarchical mixture of experts can be used in supervised learning for regression or classification, and we have recently shown how an autoencoder can be built using such models for unsupervised learning [8]. Shazeer et al. [9] used the (non–hierarchical) mixture of experts model to choose among thousands of neural network models. Their motivation is to decrease the overall complexity by choosing and using only the relevant model; we use much simpler (constant) models at the leaves and we have no such concern.

As we discuss in more detail shortly, the hierarchical mixture model defines a soft decision tree because of the soft gating function and this leads to a smoother output. Still, when the hierarchy has many levels, the number of gating models and experts increase exponentially with depth and we have observed experimentally that the HMoE is prone to overfitting. Our contribution in this work is to propose a dropout mechanism suitable for a tree structure to avoid such overfitting behavior. The original dropout method was proposed for the fully-connected layers of a feed-forward neural network; the novelty of our contribution is in adapting the dropout to a tree structure. Our experimental results show that our proposed dropout method works as a regularizer and indeed improves generalization.

The basic idea behind the original dropout as applied to feed-forward neural networks is that the final decision should be distributed over the whole model, because any part of the model

* Corresponding author.
*E-mail address:* ethem.alpaydin@ozyegin.edu.tr (E. Alpaydın).

https://doi.org/10.1016/j.neucom.2020.08.052
0925-2312/

can be dropped out. The same also holds for the decision tree. With a tree of many levels, every leaf or subtree is dedicated to some small region in the input space and such a tree is likely to learn the noise in the training data and overfit. However, if parts of the tree can be dropped out at random, to be able to still generate the correct output, the rest of the tree should learn to make up for the missing parts, and this forces the response to be distributed over the tree, hence generating a smoother output.

Another important point about the original dropout is that every time a subset of the units is dropped out at random, we get a different network having different complexity, and when we train them all, it is as if we are averaging over all of them, as opposed to "putting all our eggs in the same basket." The same also holds when dropout is applied to a tree; every time some parts of the tree are dropped out, we get a different tree with a different complexity and when we train them all, we are averaging over trees of different complexities. It is this averaging effect that is another explanation for why dropout is a method for regularization, alleviating overfitting and improving generalization.

This paper is organized as follows: We review the basic HMoE model and the dropout method as applied to multilayer neural networks in Section 2. Our proposed dropout method for HMoE is discussed in Section 3. Our experimental results on three image data sets (one toy and two real-world) and one sentiment recognition data set are given in Section 4. We conclude in Section 5.

## 2. Preliminaries

### 2.1. Hierarchical mixture of experts

The mixture of experts architecture (MoE) consists of multiple experts and a gating model. The gating model is basically a classifier that divides up the input space among the experts and each expert is responsible for generating the correct output in its domain of expertise (as defined by the gating model) [10]. More formally, let $f_i(x)$ be the output of expert $i$ for input $x$ and $\alpha(x)$ denote the gating function. Then, the overall response is calculated as:

$$y(x) = \sum_{i=1}^{K} \alpha_i(x) f_i(x) \tag{1}$$

where $\sum_i \alpha_i(x) = 1$ and there are $K$ experts. A simple gating function divides the subspace linearly:

$$\alpha_i(x) = \frac{\exp[w_i^T x]}{\sum_j \exp[w_j^T x]} \tag{2}$$

which reduces to the sigmoid when $K = 2$. Another interpretation of this model is that $\alpha_i(x)$ models the probability of $x$ falling into the $i$th subspace and $y(x)$ computes the expected response with respect to this probability distribution.

If we replace each expert by a mixture of experts recursively, we get the hierarchical mixture of experts (HMoE) [7]. This defines a tree with gating models working as internal decision nodes splitting the input space sequentially as we go down from the root and experts at the lowest level corresponding to the leaves. This defines a *soft decision tree* because the splitting is not hard: Unlike a hard decision tree where we take one path from the root to one of the leaves, we traverse all paths to all the leaves and we take a sum weighted by the gating values on each path. More formally, we have:

$$y_m(x) = \begin{cases} \alpha_m(x) y_{ml}(x) + (1 - \alpha_m(x)) y_{mr}(x) & \text{if } m \text{ is an internal node} \\ f_m(x) & \text{if } m \text{ is a leaf} \end{cases} \tag{3}$$

where $\alpha_m()$ denotes the local gating function for node $m$:

$$\alpha_m(x) = \text{sigmoid}(w_m^T x) \tag{4}$$

Here we assume branching into two and we have a binary tree. For node $m$, $ml$ and $mr$ denote its left and right children respectively.

In the simplest case, $f_m(x)$, the response at leaf node $m$, is a constant value, i.e., $f_m(x) = c_m$; in a multi-class classification task, this would be a vector value of log-odds of classes.

The hierarchical mixture of experts is similar to a feed-forward neural network with one hidden layer, with the difference that there is a functional dependency between each unit determined by the tree structure.

### 2.2. Dropout

Dropout is a regularization method that reduces the co-adaptation of feature learners in a network. It works by randomly dropping out units in a neural network; therefore the network has to learn to work in the absence of any of its units. Let us limit our scope to a feed-forward layer. A single hidden unit $h_i$ is defined as:

$$h_i = \sigma(w_i^T x) \tag{5}$$

where $\sigma(\cdot)$ denotes a squashing nonlinearity such as tanh or sigmoid. If we apply dropout to the hidden layer with a probability value of $p$, the equation is modified as follows (at training time):

$$d_i \sim \text{Bernoulli}(1 - p) \tag{6}$$

and

$$h_i = d_i \cdot \sigma(w_i^T x) \tag{7}$$

So with probability $p$, we assign $h_i$ to be zero, effectively dropping that unit out from the network; with probability $1 - p$, it is kept. In recent work, dropout has shown to improve generalization ability across different architectures and different tasks, and has gained widespread use [1,11–15].

## 3. A dropout method for hierarchically gated models

In a hierarchical mixture of expert, we use dropout on internal gating nodes. Again for each node $m$, we sample from a Bernoulli using a dropout-rate hyperparameter $p$. If we choose to drop, this results in dropping out the entire left subtree, i.e. resorting to only the right subtree for the response function. It is important that we dropout only the left subtree as we will see shortly.

For internal node $m$:

$$d_m \sim \text{Bernoulli}(1 - p) \tag{8}$$

and

$$y_m(x) = \begin{cases} \alpha_m(x) y_{ml}(x) + (1 - \alpha_m(x)) y_{mr}(x) & \text{if } d_m = 0 \\ y_{mr}(x) & \text{if } d_m = 1 \end{cases} \tag{9}$$

At training time, dropout implies (randomly) setting $\alpha_m$ to 0, which implies that $1 - \alpha_m$ is set to 1. At test time, we do not rescale with $(1 - p)$ as in traditional dropout [1]; this is because in both dropped and non-dropped cases, the gating values sum to 1.

This approach is different from the traditional dropout used on perceptrons in which we drop or not a single unit independently of other units. The hierarchical mixture of expert connects each unit to another in a hierarchical gating structure where a parent controls the gating value assigned to its children, in contrast to the flat structure defined by the perceptron layer. The dropout approach being presented here respects this hierarchy: Dropping out one of the children results in turning off the entire path of

responsibility through that child node, resulting in dropping out an entire subtree.

The reason we break the symmetry and drop only the left subtree as opposed to randomly dropping either is as follows: Randomly dropping either subtree would push both subtrees to implement similar functionalities using the same complexity; this is equivalent to ensembling trees with half the number of nodes. Dropping only the left subtree pushes the right subtree to implement a function similar to that of the whole tree while having fewer nodes; it is as if through the right children we generate an ensemble of (unbalanced) trees of different sizes.

This means that we, in turn, force remaining subtrees to be capable on their own in the absence of the dropped subtrees. As seen in our qualitative experiments, this manifests as right subtrees

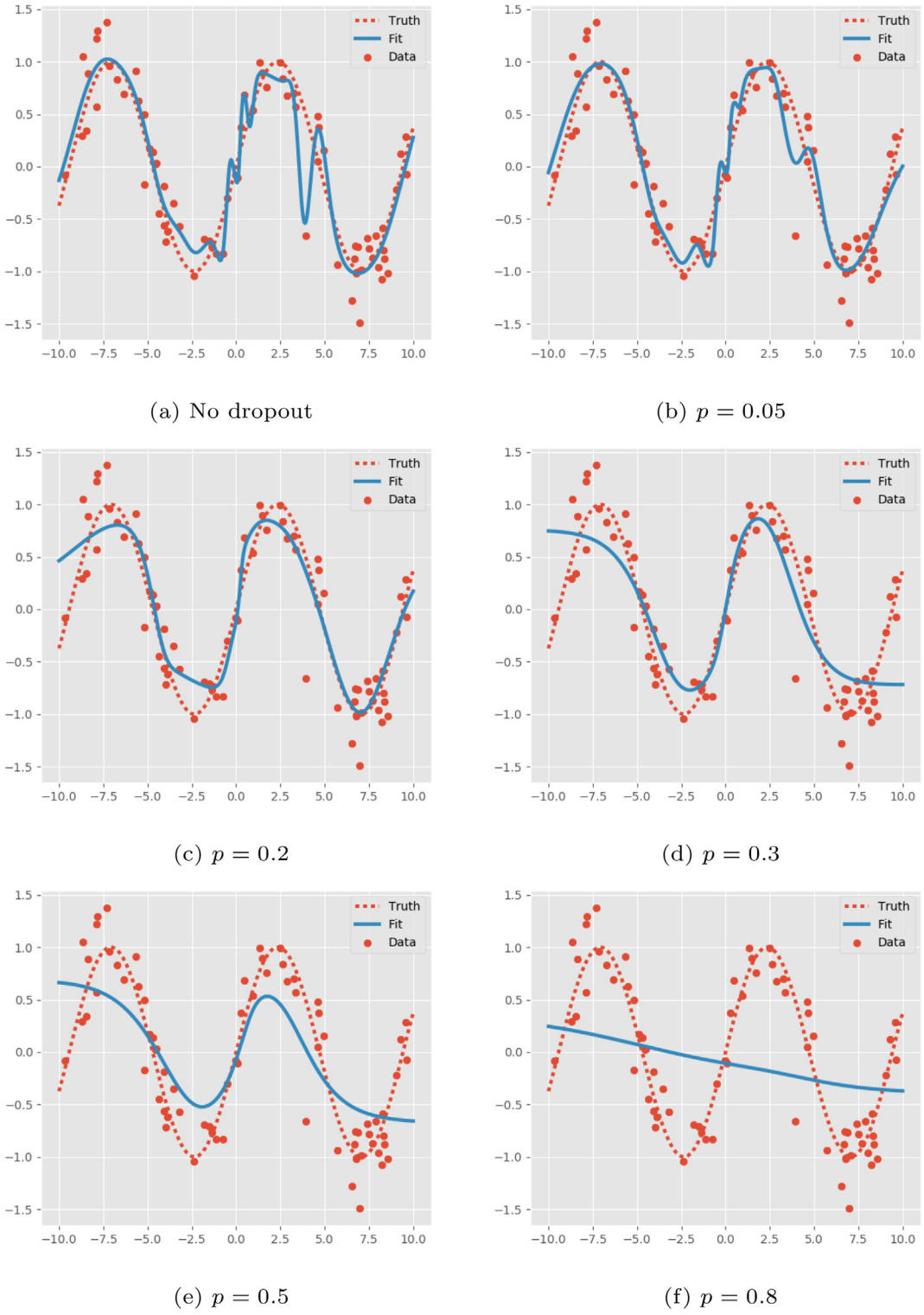



**Fig. 1.** Fits using a HMoE model with depth 10 and 1023 internal nodes for various dropout rates. We see decreasing variance, increasing bias and smoothness as we increase the dropout rate.

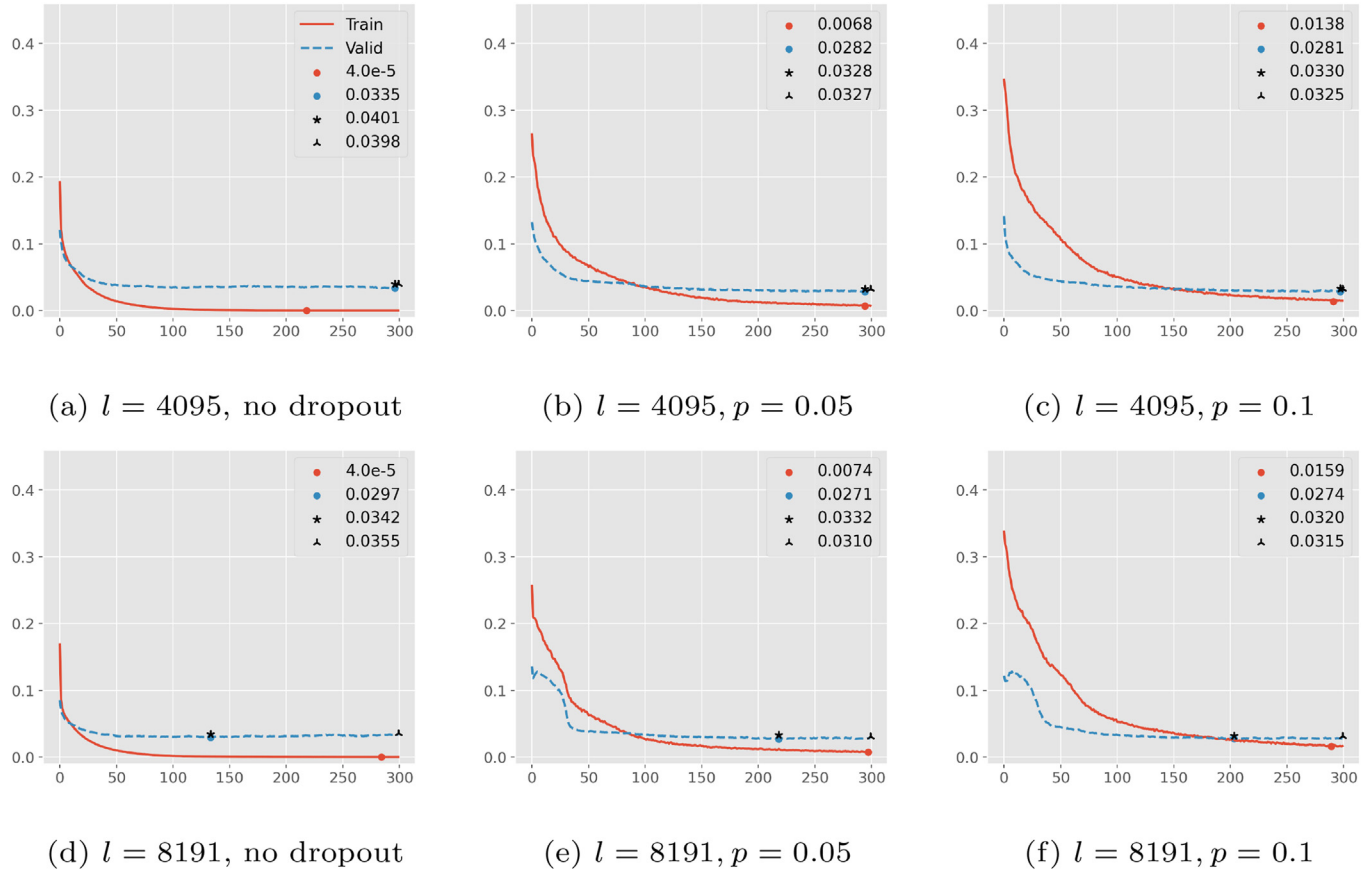



**Fig. 2.** Misclassification errors on MNIST for different number of internal nodes $l$ and different dropout rates $p$. The red and blue (dashed) curves show the convergence of training and validation errors and the red and blue dots denote the best training and validation set scores, respectively. The asterisk depicts the test set performance of the best performing model chosen according to validation set score, and the three-pointed star shows the test set performance of the final model at the end of training (here, after 300 epochs). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

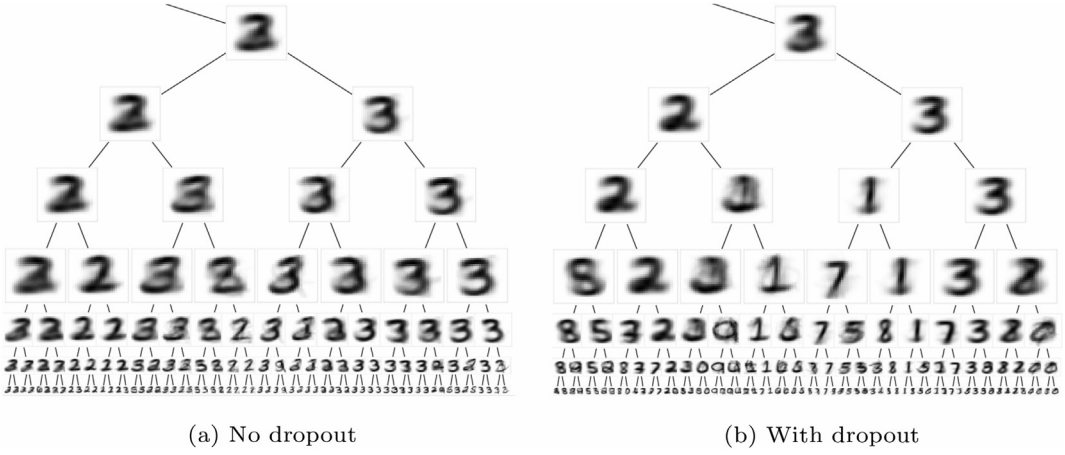



**Fig. 3.** A visualization of the rightmost subtrees of a hierarchical mixture of experts model with 2047 internal nodes.

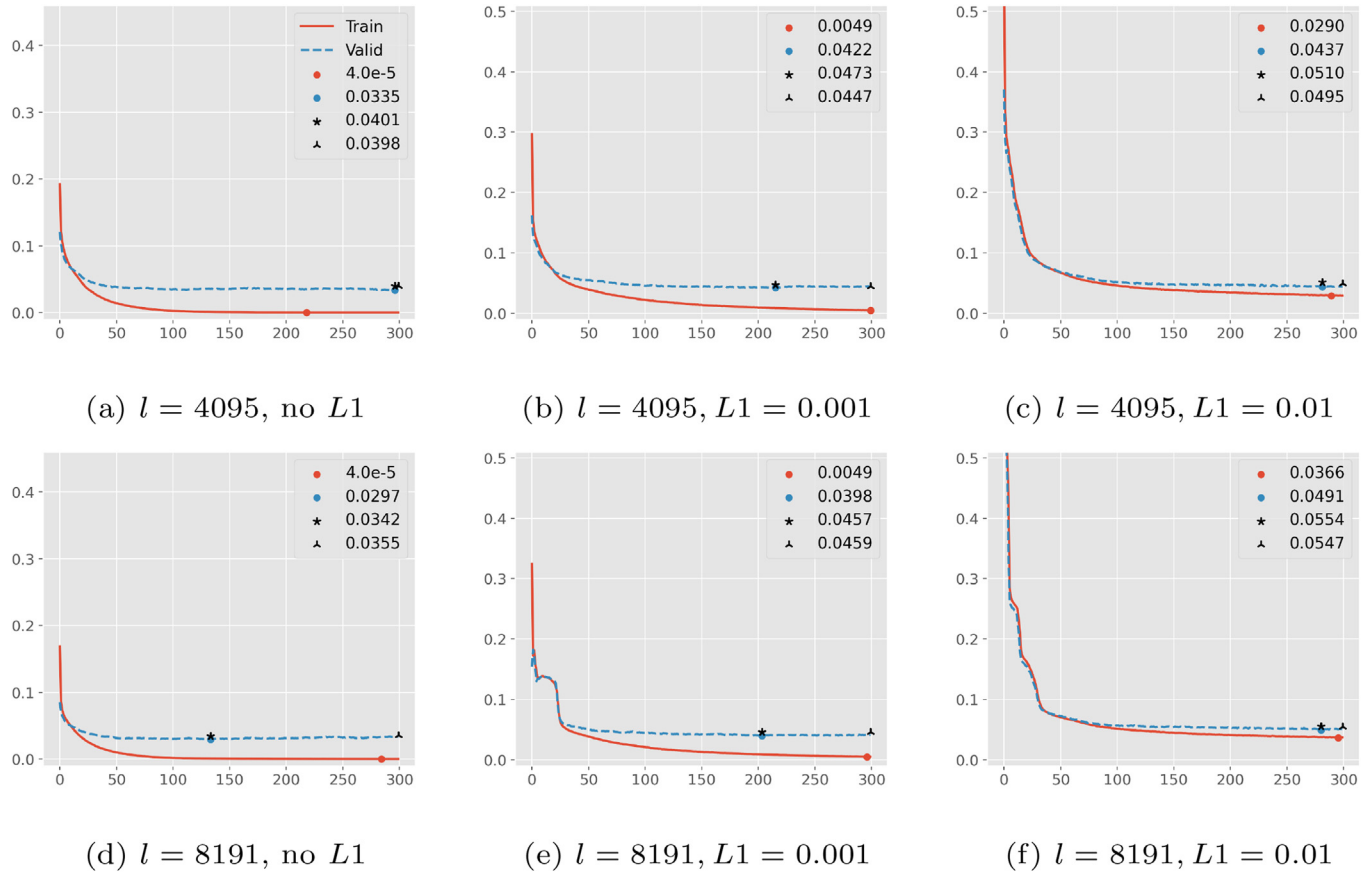
learning diverse sets of features and nodes in the left subtrees introducing additional variations to the features.

## 4. Experiments

### 4.1. Setting

In our experiments, we limit the structure of the hierarchical mixture of expert to a complete binary tree of given depth. The depth of the tree is a hyperparameter and as such it defines the complexity of the model; we investigate the impact of dropout for increasing depths, and hence complexities. We use the cross-entropy loss (with softmax nonlinearity at the top) for classification and squared loss for regression. Training is done using minibatched stochastic gradient-descent with Adam update rule [16] on a fixed structure where we update all parameters (in gatings on all levels and leaves) simultaneously.

The dropout rate values that we investigate are typically small, such as 0.05 or 0.1, as opposed to values used in traditional




(a) $l = 4095$, no $L1$ (b) $l = 4095$, $L1 = 0.001$ (c) $l = 4095$, $L1 = 0.01$

(d) $l = 8191$, no $L1$ (e) $l = 8191$, $L1 = 0.001$ (f) $l = 8191$, $L1 = 0.01$

**Fig. 4.** Misclassification errors on MNIST for different number of internal nodes $l$ and different L1 regularization penalties. The red and blue (dashed) curves show the convergence of training and validation errors and the red and blue dots denote the best training and validation set scores, respectively. The asterisk depicts the test set performance of the best performing model chosen according to validation set score, and the three-pointed star shows the test set performance of the final model at the end of training. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

dropout, such as 0.5 [1]. This is because in the case of dropout on a tree, the expected number of retained units decrease exponentially with the depth of the tree. A dropout value of 0.5 means that at every level we trim approximately half of the remaining subtree, whereas in a flat feed-forward architecture, this value trims approximately half of the overall number of units. We could have, alternatively, used a dropout rate which increases with the depth of the tree, however scheduling the rate as a function of depth would have introduced an additional hyperparameter to optimize.

### 4.2. Results on toy data

We test our approach on a one-dimensional synthetic dataset, which is sampled from a sinusoid function with small Normal noise. The task is posed as a regression problem with single scalar response value, therefore we use the squared loss without any nonlinearity at the output. We use a learning rate of $10^{-3}$ and run a model with 1023 internal nodes (depth 10) for 1000 epochs.

The resulting fits are shown in Fig. 1 for different values of the dropout rate $p$. We observe that for the case of no dropout as well as for small values of dropout rates, such as 0.05, the model overfits; i.e., it learns the noise and exhibits steep jumps. As we increase the dropout rate, the fit becomes smooth, eventually to the point where we get almost a linear fit with very low variance and high bias. This indicates that our proposed method works effectively as a regularizer with the dropout rate as its hyperparameter.

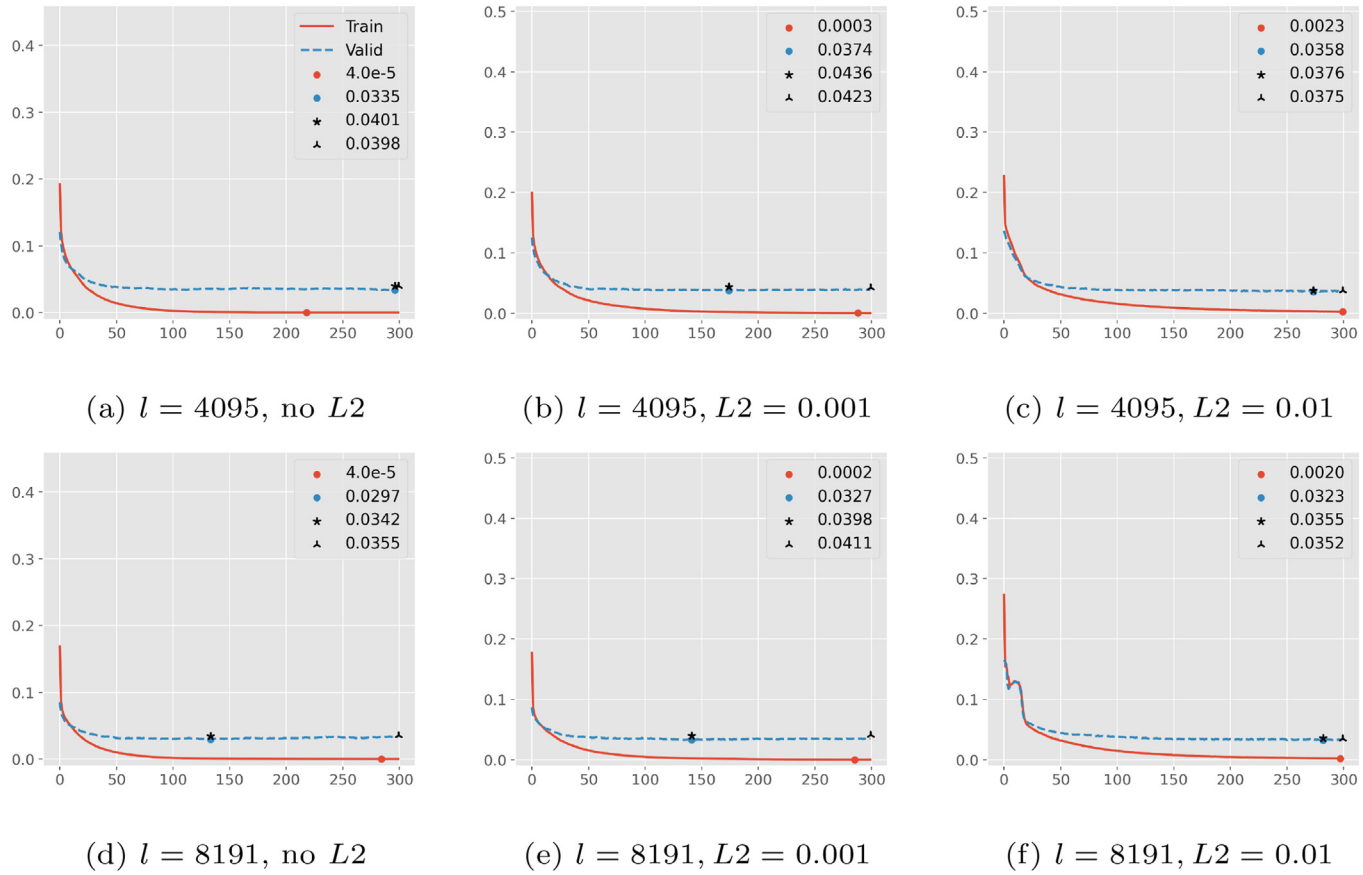
### 4.3. Results on digit recognition

Next, we evaluate our approach on the MNIST data which contains 60,000 training and 10,000 test examples of handwritten digit images [17]. Each image is $28 \times 28$ pixels with single (black & white) channel (of 784 dimensions). The output labels are the ten digits and the task is formulated as a 10-class classification problem. We randomly partition the original training set into training and validation sets with a ratio of 5:1. We use the cross-entropy loss with softmax activation at the top. We train for a total of 300 epochs.

The results are presented in Fig. 2. We show misclassification errors for two architectures that have a total number of internal nodes 4095 (depth 12) and 8191 (depth 13) using dropout rates of 0, 0.05, and 0.1.

We see that for both sizes of 4095 and 8191, with no dropout, there is overfitting – the blue (dashed) curve (validation error) is higher than the red curve (training error). This gap is an indicator of overfitting and we see that it gets smaller with dropout. In terms of validation performance, dropout rates of 0.05 and 0.1 show similar behavior, and they beat the case of no dropout: For the tree with 4095 nonleaves, we have 2.82% and 2.81% versus 3.35%, and for the tree with 8191 nonleaves, we have 2.71% and 2.74% versus 2,97%.

A similar trend is also observed for the test set performance. The best model ends up being the larger tree with a dropout rate of 0.1. In the case of no dropout, not only there is a larger gap between the training and validation performances but for large trees, the valida-




(a) $l = 4095$, no $L2$ (b) $l = 4095$, $L2 = 0.001$ (c) $l = 4095$, $L2 = 0.01$

(d) $l = 8191$, no $L2$ (e) $l = 8191$, $L2 = 0.001$ (f) $l = 8191$, $L2 = 0.01$

**Fig. 5.** Misclassification errors on MNIST for different number of internal nodes $l$ and different L2 regularization penalties. The red and blue (dashed) curves show the convergence of training and validation errors and the red and blue dots denote the best training and validation set scores, respectively. The asterisk depicts the test set performance of the best performing model chosen according to validation set score, and the three-pointed star shows the test set performance of the final model at the end of training. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

tion error also quickly starts curving upwards. In contrast, with dropout, there is both a smaller gap and a consistently improving validation performance – the best model (shown with an asterisk) has lower error and is achieved later.

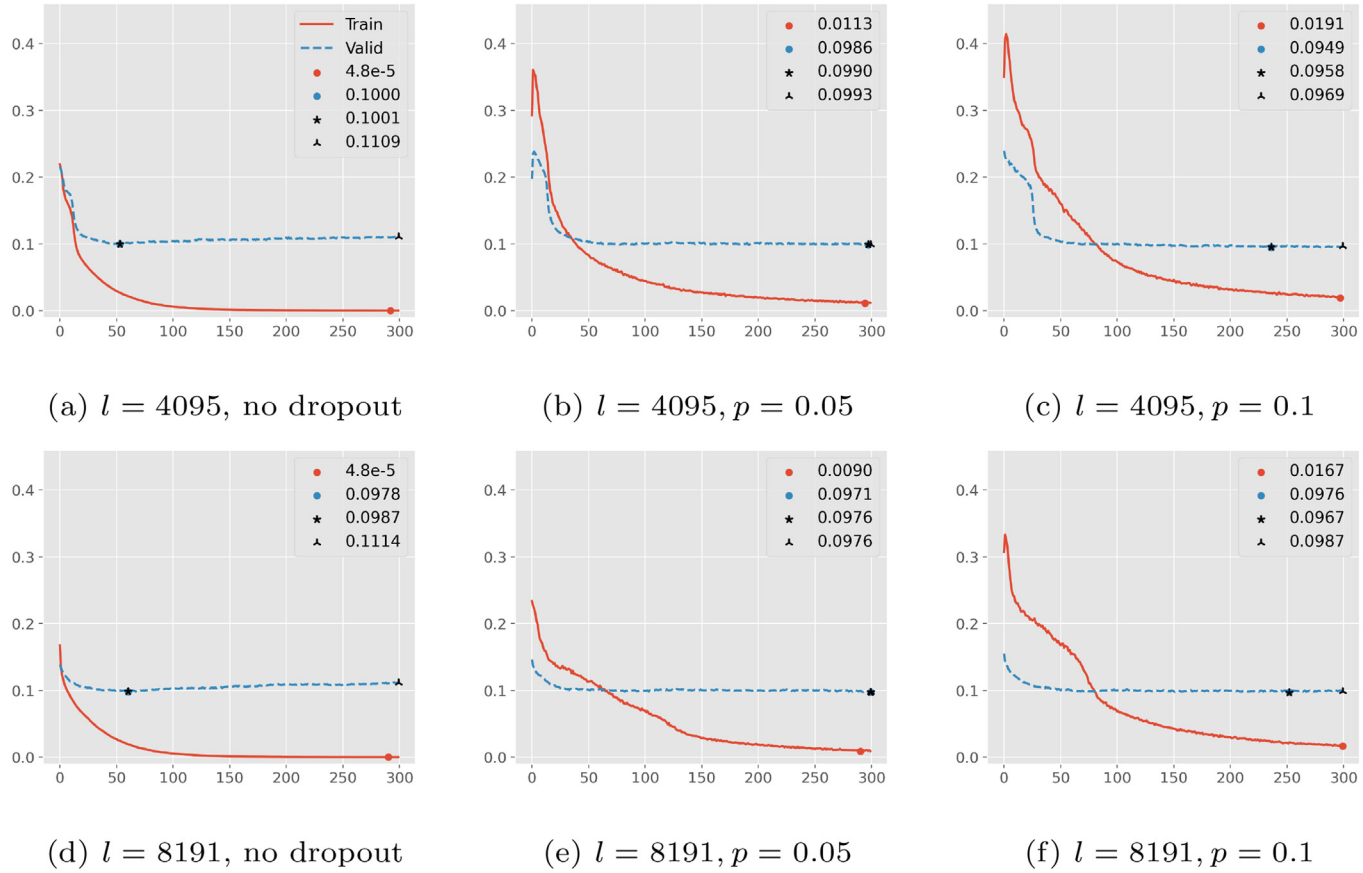
Additionally, to qualitatively inspect our models, we visualize the learned hierarchy by measuring which input instances activate a given node. A subtree of the resulting visualization is given in Fig. 3. We simply take a weighted average over all the input vectors, where weights are determined by the gating activation of that particular node, and treating the resulting average as the representation learned by that node. We observe that with no dropout, the features tend to look very similar for close siblings and vary very little. In contrast, when we apply dropout, there is more variation in the features. This behavior is intuitive: Whenever we choose to drop, we rely on only a single (right) subtree to implement the prediction function, which forces the right subtree to handle more classes which is reflected by higher variance in the features.

When there is no dropout, the task is distributed over the whole tree; with dropout however, with always dropping out the left side branch, we are effectively training an ensemble of trees of different depths and complexities.

To compare with the dropout results, we also tested L1 and L2 regularization, where we apply L1 or L2 penalties in learning the splitting hyperparameter weights ($\{w_m\}_m$). The results on the same size trees again trained on MNIST are presented in Figs. 4 and 5 respectively. We observe that although the gap between training and validation performances shrink with increasing regularization, unlike with dropout, development or test set performance is not always improved. For instance, in L1 regularization a penalty of 0.01 acts very aggressively as seen in the much smaller gaps between training and validation curves in Fig. 4(c) compared to (a), yet, has a higher test error rate (5.1% vs 4.0%). Although less aggressive, a penalty of 0.001 still has a worse test error rate (Fig. 4(b), 4.7%). Similarly for L2 regularization, only test set error rate improvement happens for the case of 4096 nodes and a penalty of 0.01 (Fig. 5, 3.8% vs 4.0%), with a comparatively wider gap in between training and validation curves. On the other hand with dropout, with few exceptions, with higher dropout rates we usually observe both smaller gaps, and smaller or comparable development and test error rates (Fig. 2).

### 4.4. Results on image classification

We evaluate our models also on the CIFAR-10 dataset which has 60,000 images of $32 \times 32$ images with three color channels, each tagged with a class label out of ten classes [18]. We use the training-test set partitioning with the ratio of 5:1 originally recommended. In addition to the existing training-test split, we randomly partition the training set into training and validation (development) sets with a ratio of 4:1. To represent the images, we use the version 3 of the Inception network [19] pretrained on ImageNet [12]. In particular, after running the network on each image, we extract the response from the third pooling layer, which results in a 2,048-dimensional vector for each image instance.




(a) $l = 4095$, no dropout (b) $l = 4095, p = 0.05$ (c) $l = 4095, p = 0.1$

(d) $l = 8191$, no dropout (e) $l = 8191, p = 0.05$ (f) $l = 8191, p = 0.1$

**Fig. 6.** Misclassification errors on CIFAR-10 for different number of internal nodes $l$ and different dropout rates $p$. The red and blue (dashed) curves show the convergence of training and validation errors and the red and blue dots denote the best training and validation set scores, respectively. The asterisk depicts the test set performance of the best performing model chosen according to validation set score, and the three-pointed star shows the test set performance of the final model at the end of training. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Since this task is formulated as a classification problem, we optimize for cross-entropy objective with a softmax nonlinearity at the very top of the model. We train each model for 300 epochs.

The results are presented in Fig. 6. We show misclassification errors for two architectures that have a total number of 4095 and 8191 nonleaves with dropout rates of 0, 0.05 and 0.1.

When there is no dropout, both architectures seem to get around 10% error, and we see that the validation performance quickly starts to worsen and curve upward, while the training error reaches to almost zero[1]. As we increase the dropout rate, we observe that the gap between training and validation performances shrinks.

For the dropout rate values we have investigated (0, 0.05 and 0.1), we see that increasing the dropout rate consistently improves test performance, which shows the effectiveness of using dropout. For dropout rate of 0.05, the best validation set performing model is the final model at the last epoch, which shows that overfitting still has not started after 300 epochs and there might be room for further improvement. In contrast, for the case where dropout is not applied, the final model is noticeably worse than the best validation model which is attained at approximately 50th epoch.

[1] Training error typically starts off worse than development error, which is because the training error is computed in an on-line fashion as the training continues within an epoch. Furthermore, it includes the stochasticity due to dropping out units.

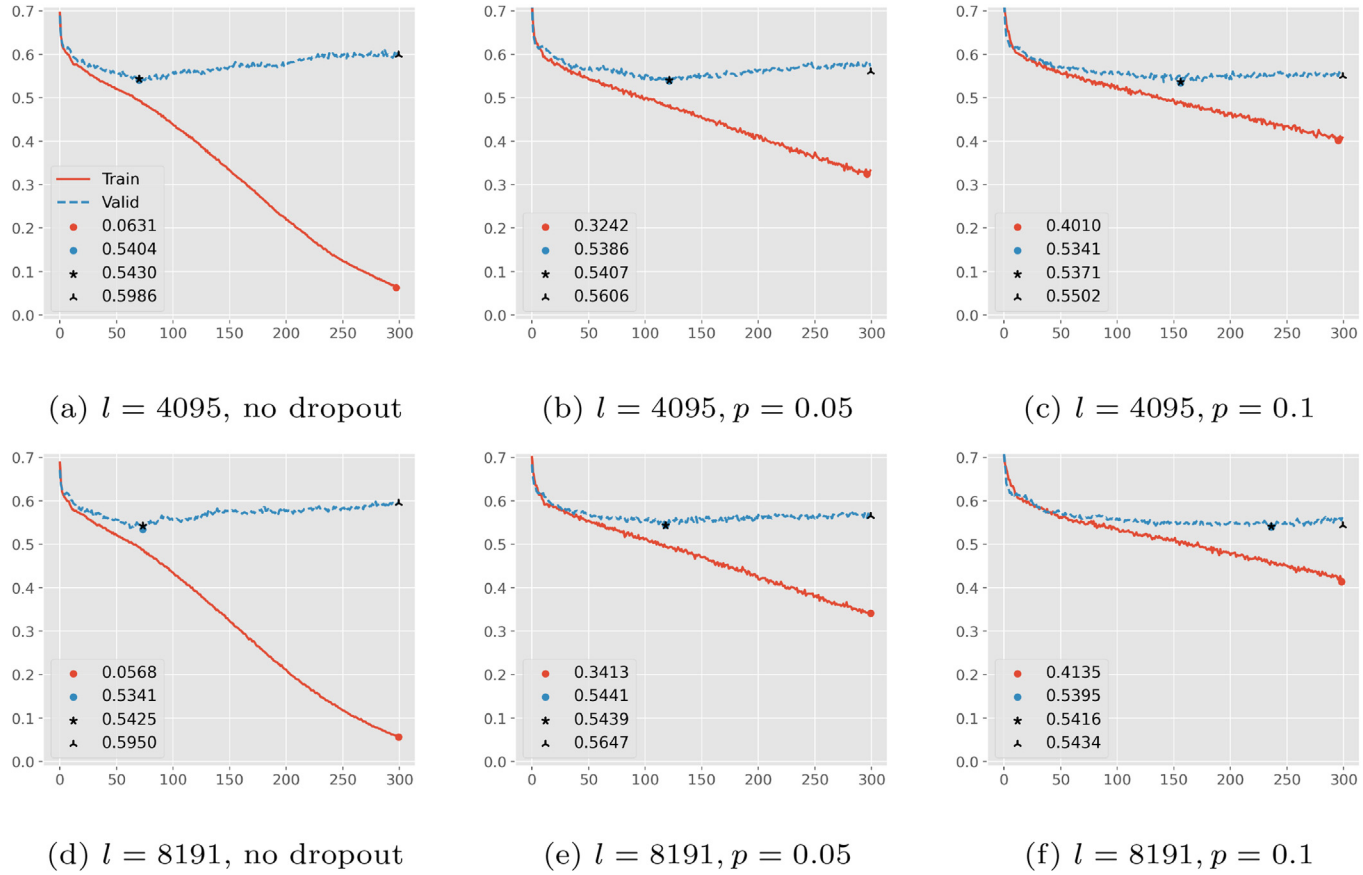
### 4.5. Results on sentiment detection

We further evaluate our models on the SSTB (Stanford Sentiment TreeBank), which includes 11,855 sentences from movie reviews with sentiment labels attached from the set $\{0, \ldots, 4\}$ [20]. Therefore the task is defined as a five-class classification problem. We use the training-validation-test partitioning originally recommended. We represent each sentence as its mean word embeddings, where we use the pre-trained Glove word embeddings [21]. Word embeddings are tuned throughout training, which makes the model prone to overfitting.

The results are presented in Fig. 7. This is a case where we see overfitting very clerly: Training error decreases with more training; validation error curves upward after some training and the gap increases as more and more training is done. With dropout, we see that the gap decreases significantly and this decrease is more apparent when the dropout probability is higher. For trees of size 4095, we observe improving test set performance with more regularization ($54.3\% > 54\% > 53.7\%$). For the larger trees, results are mixed where a dropout rate of 0.05 performs the worst, followed by no dropout and a dropout rate of 0.1 ($54.4\% > 54.3\% > 54.2\%$).

## 5. Conclusions

We provide a novel dropout mechanism that can be applied to the hierarchical mixture of experts method. In contrast to the dropout on flat architectures with units that are conditionally




(a) $l = 4095$, no dropout (b) $l = 4095, p = 0.05$ (c) $l = 4095, p = 0.1$

(d) $l = 8191$, no dropout (e) $l = 8191, p = 0.05$ (f) $l = 8191, p = 0.1$

**Fig. 7.** Misclassification errors on SSTB for different number of internal nodes $l$ and different dropout rates $p$. The red and blue (dashed) curves show the convergence of training and validation errors and the red and blue dots denote the best training and validation set scores, respectively. The asterisk depicts the test set performance of the best performing model chosen according to validation set score, and the three-pointed star shows the test set performance of the final model at the end of training. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

independent, our method is faithful to the gating dependencies that exist in the tree hierarchy of the model.

Note that in our approach, the gatings and the models at the leaves are trained together and our proposed dropout mechanism has the effect of distributing the task to the whole tree. If we have a scenario where we already have sufficiently complex models each responsible from some part of the overall problem and the tree is there just to choose among them suitably, a dropout mechanism would not be appropriate.

We show the effectiveness of our approach on a synthetic toy dataset as well as on three real-world data sets for the tasks of digit, image, and sentiment classification. On all data sets, we see that the hierarchical mixture of experts does overfit when there are too many levels and leaves, but that our proposed method works as an effective regularizer with the dropout rate as the hyperparameter trading off bias and variance. We also see that it is more effective than L1/L2 regularization; this is not surprising since our proposed dropout is better suited to the model unlike L1/L2 regularizaton which is a more general technique.

We also qualitatively evaluate the impact of dropout on the representations learned by the models visualizing the effect of the dropout. Because of the fact that we asymmetrically drop only the left subtree, our dropout method effectively samples from an ensemble of tree-structured models of different complexities; this introduces regularization by acting as averaging over models having varying complexities.

## CRediT authorship contribution statement

**Ozan İrsoy:** Conceptualization, Methodology, Software, Writing - review & editing. **Ethem Alpaydın:** Conceptualization, Writing - review & editing.

## Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References

[1] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research 15 (2014) 1929–1958.
[2] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, R. Fergus, Regularization of neural networks using dropconnect, in: International Conference on Machine Learning, 2013, pp. 1058–1066.
[3] S. Wang, C. Manning, Fast dropout training, in: International Conference on Machine Learning, 2013, pp. 118–126.
[4] M. Iyyer, V. Manjunatha, J. Boyd-Graber, H. Daumé III, Deep unordered composition rivals syntactic methods for text classification, in: 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Vol. 1, 2015, pp. 1681–1691.

[5] Y. Gal, Z. Ghahramani, Dropout as a Bayesian approximation: Representing model uncertainty in deep learning, in: International Conference on Machine Learning, 2016, pp. 1050–1059.
[6] Y. Gal, Z. Ghahramani, A theoretically grounded application of dropout in recurrent neural networks, in: Advances in Neural Information Processing Systems, 2016, pp. 1019–1027.
[7] M.I. Jordan, R.A. Jacobs, Hierarchical mixtures of experts and the EM algorithm, Neural Computation 6 (2) (1994) 181–214.
[8] O. İrsoy, E. Alpaydın, Unsupervised feature extraction with autoencoder trees, Neurocomputing 258 (2017) 63–73.
[9] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, J. Dean, Outrageously large neural networks: The sparsely-gated mixture-of-experts layer, arXiv preprint arXiv:1701.06538.
[10] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, G.E. Hinton, Adaptive mixtures of local experts, Neural Computation 3 (1) (1991) 79–87.
[11] G. E. Dahl, T. N. Sainath, G. E. Hinton, Improving deep neural networks for lvcsr using rectified linear units and dropout, in: International Conference on Acoustics, Speech and Signal Processing, IEEE, 2013, pp. 8609–8613.
[12] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
[13] Y. Kim, Convolutional neural networks for sentence classification, arXiv preprint arXiv:1408.5882.
[14] W. Zaremba, I. Sutskever, O. Vinyals, Recurrent neural network regularization, arXiv preprint arXiv:1409.2329.
[15] O. İrsoy, C. Cardie, Opinion mining with deep recurrent neural networks, in: Empirical Methods in Natural Language Processing, 2014, pp. 720–728.
[16] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.
[17] Y. LeCun, C. Cortes, The MNIST database of handwritten digits, 1998.
[18] A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images, Technical report, University of Toronto.
[19] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z. Wojna, Rethinking the inception architecture for computer vision, in: IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 2818–2826.
[20] R. Socher, A. Perelygin, J. Wu, J. Chuang, C.D. Manning, A.Y. Ng, C. Potts, Recursive deep models for semantic compositionality over a sentiment treebank, in: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013, pp. 1631–1642.
[21] J. Pennington, R. Socher, C.D. Manning, Glove: Global vectors for word representation, in: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543.

**Ozan Irsoy** received the B.A. degree in mathematics and the B.Sc. degree in computer engineering from Bogazici University in 2012. He received the Ph.D. degree in computer science from Cornell University in 2017. He is currently an AI Research Scientist at Bloomberg L.P.

**Ethem Alpaydin** received his Ph.D. degree from Ecole Polytechnique Fédérale de Lausanne, Switzerland, in 1990, and was a postdoc at the International Computer Science Institute, Berkeley in 1991. He is currently a Professor in the Department of Computer Science, Özyegin University, Istanbul and a Member of the Science Academy, Istanbul. He was a Visiting Researcher at MIT in 1994, IDIAP in 1998, and TU Delft in 2014. He was a Fulbright scholar in 1997. The fourth edition of his book Introduction to Machine Learning was published by The MIT Press in 2020.