K-Dropout

Matthew Pauly [*1] **Luke Bailey** [*1]

Abstract

Dropout is a widely used regularization technique that helps improve the generalization capabilities of deep neural networks. In this paper, we introduce k-dropout, a novel generalization of the standard dropout technique. In k-dropout, instead of resampling dropout masks at every training iteration, masks are reused multiple times according to new k and m tunable hyperparameters. Resuing dropout masks leads to a reduced number of unique subnets being trained compared to traditional dropout. We empirically demonstrate that training a multilayer perceptron (MLP) on CIFAR-10 with as few as 50 distinct subnets using k-dropout yields performance comparable to that of regular dropout. Furthermore, we provide detailed analysis of the trade-off between the number of subnets and the model's performance, as well as explore details of the training dynamics that allow the training of few subnets to be competitive with standard dropout.

1. Introduction

Dropout is a tried and tested regularization technique for deep neural networks (Srivastava et al., 2014). When training a network using dropout, neuron activations are kept with probability p and masked otherwise, where p is a tunable hyperparameter. "Masking" is achieved by performing an elementwise multiplication of the activations of each layer with a tensor of zeros and ones. This tensor is referred to as the "mask," and in the basic formulation of dropout is resampled at each training step. During inference, all activations are unmasked and scaled by p. There exists a zoo of dropout variants (Labach et al., 2019; Xie et al., 2021), but standard dropout is still a staple for training many state-of-the-art models (Jumper et al., 2021).

To reason about its effectiveness, some prior work conceives of dropout as a form of ensemble learning (Srivastava et al., 2014; Hara et al., 2016). When training a model with dropout, the loss of each example in a mini-batch is calculated using a randomly sampled sparse subnetwork. This means that during inference with the full model, the predictions of all the subnets used in training are in some sense "averaged."

In this work we present *k-dropout*, a generalized version of dropout inspired by ensemble learning. Standard dropout involves two tunable parameters: p, the probability of masking each activation, and b, the mini-batch size during training. k-dropout introduces two new hyperparameters:

1. k - the expected number of minibatches of training each dropout mask is used in. Increasing k allows the same subnets to be used in more training steps. Any value $1 \leq k \leq T$ (where T is the total number of training steps) is valid.
2. m - the number of distinct subnets to use within the mask for each mini-batch. Choosing smaller $m < b$ means fewer subnets are used within each minibatch, and thus each involved subnetwork is used to calculate the loss for more training examples in said minibatch. m must be chosen such that it divides b and $1 \leq m \leq b$.

Standard dropout implicitly uses $k = 1$ and $m = b$. This means that each sample in a minibatch uses a different dropout mask and masks are resampled before each training step. At the other extreme with $k = T$ and $m = 1$, we use the same dropout mask for each example within a minibatch and do not resample masks for the entire training run. Standard dropout improves model performance, and the extreme case of training a smaller subnetwork in general degrades model performance (Frankle & Carbin, 2018). While it's difficult to predict a priori which hyperparameter settings are optimal, there is no clear reason why the standard dropout settings of $k = 1$ and $m = b$ would be best in all domains. Moreover, achieving comparable performance to dropout whilst using large k values and small m values can be used to decrease both the computational burden of resampling dropout masks during training and reduce the significant memory cost incurred by storing dropout masks in large distributed training, particularly in pipeline parallel training in which nodes must store many dropout masks at any given time (Harlap et al., 2018).

[*]Equal contribution [1]Harvard University. Correspondence to: Matthew Pauly <mpauly@college.harvard.edu>.

Given our formulation of k-dropout, we introduce two practical implementations, *Sequential k-dropout* and *Pooled k-dropout*. In both we only only consider batch size b and m values that are powers of 2 to ensure each subnetwork in a batch is allocated the same number of training examples (which can be easily calculated as b/m without the need for rounding). In Sequential k-dropout, m dropout masks are used for k steps sequentially until they are all resampled, meaning each individual mask is used in exactly k training steps. In pooled k-dropout, a pool of dropout masks is created before training and at each training step, m masks are randomly sampled from this pool and used for that training step. For pooled k-dropout we refer to the number of masks in the pre-generated pool as pool size. pool size can be seen as a reparameterization of k as $pool\ size = \frac{km}{T}$. Surprisingly, from our experimental results in section 2, we find that both methods are able to perform very similarly when using similar k values, albeit with sequential k-dropout being more sensitive to varying m values.

In this paper we explore the properties of training with k-dropout using different hyperparameter settings. We show k dropout with non-trivial k settings performs comparably with standard dropout whilst training up to $248,000$ times fewer subnets. We also find that the performance of sequential k-dropout and pooled-k dropout decrease around incredibly similar k values, defying our expectations that sequential k-dropout would perform worse. To explain this phenomena we further explore the training dynamics of sequential k-dropout, finding that the first few subnets that are trained retain their performance throughout training and later subnets suffer performance degradation once they have stopped being trained. We hypothesise from this that the training of initial subnets plays an important role in dictating the region of weight space the model lies in throughout training.

2. Methods

Standard dropout zeros elements of the activation matrix outputted by layers with independent probability p (Srivastava et al., 2014). This corresponds to training a different subnet for each training example in the minibatch. Note that it is often incorrectly assumed that the same subnetwork is used across all examples in a given minibatch. At test time, no nodes are dropped out and all activations are scaled by p to ensure the expected output of a node during training time (where the expectation is taken over the probability of dropout) is the same as that during test time when the unit is never dropped out. To see how multiple different subnets can be used to process a single minibatch of data, it can be easier to view dropout as imposing a binary mask on activations. Each layer of f will have some number of hidden units. Let us consider some arbitrary layer l with h hidden units. For a minibatch with b training examples, the activation tensor A^l outputted by this layer will have shape $b \times h$. During training, instead of forward propagating A^l to the next layer, we instead propagate $A^l \odot M^l$ where M^l is a $b \times h$ binary dropout mask such that $M^l_{ij} \sim Bernouli(p)$ and $\odot$ is an elementwise product. M^l is resampled for each training step. At test time, we ignore M^l and simply forward propagate $A^l \times p$.

k-dropout is a generalization of standard dropout. Firstly, we introduce k as the expected number of training steps each dropout mask is involved in. Secondly, we introduce m, the number of dropout masks involved in each training step. Finally r is the function used to sample a new dropout mask when resampling is needed. In this work we use the same random Bernoulli method as standard dropout, in which the new dropout mask for layer l is a mask M^l such that $M^l_{ij} \sim Bernouli(p)$. More principled r functions could be used to try and induce modularity in the network, with certain subnets being activated depending on properties of the input. We leave this for future work however.

At each training step in k-dropout, each layer l of the model has some corresponding dropout mask M^l. Just as in regular dropout M^l is a binary matrix of shape $b \times h$ where b is the batch size and h is the hidden layer size, this being the same shape as the activation matrix A^l outputted by layer l. We partition M^l into m vertically stacked submatrices of shape $\frac{b}{m} \times h$. Let the ith of these submatrices be denoted by $M^{l,(i)}$. Each of these submatrices corresponds to an individual subnetwork. Accordingly, every row of $M^{l,(i)}$ is the same binary mask, which we refer to as $mask^l_i$, where $mask_i$ is row vector of length h with every element being independently sampled from a $Bernouli(p)$ distribution. From this, M^l can be decomposed as follows:

$$M^l = \begin{bmatrix} M^{l,(1)} \\ \hline \vdots \\ \hline M^{l,(m)} \end{bmatrix} = \left[\begin{array}{ccc} \leftarrow & mask^l_1 & \rightarrow \\ & \vdots & \\ \hline \leftarrow & mask^l_1 & \rightarrow \\ \hline & \vdots & \\ \hline \leftarrow & mask^l_m & \rightarrow \\ & \vdots & \\ \leftarrow & mask^l_m & \rightarrow \end{array}\right]$$

Just as in regular dropout, at each layer during the forward pass of training we forward propagate $A^l \odot M^l$ and during inference times we forward propagate $A^l \times p$.

2.1. Sequential k-dropout

In this work we explore two implementations of k-dropout, sequential k-dropout and pooled k-dropout. Each simply

Algorithm 1 Sequential k-dropout Resampling

Input: $k, m, p, model$, and batch size b
for dropout_layer **in** model **do**
  h := dropout_layer.hidden_size
  mask_blocks := []
  block_size := b/m
  for $i = 0$ **to** $m - 1$ **do**
    $mask^i$:= bernoulli_sample(p, shape=(1, h))
    $M^{(i)}$:= repeat($mask^i$, block_size)
    mask_blocks.append($M^{(i)}$)
  end for
  M := concatenate(mask_blocks)
  dropout_layer.mask := M
end for

differs in the way that we reuse a certain mask M^l. In sequential k-dropout, each M^l for every layer is generated and used for k sequential steps, and then resampled. Thus resampling the dropout masks simply takes the form of resampling $mask_i^l$ for all layers l reforming the M^l matrices using these masks. This resampling algorithm is outlined in algorithm 1.

2.2. Pooled k-dropout

In Pooled k-dropout, for each layer l, an ordered list of $pool\ size$ binary masks are created before training, which we refer to as $pool^l$. At each training step, m indexes are sampled with replacement from $\{0, 1, \ldots, pool\ size - 1\}$. Let these indexes be $idx_1, \ldots, idx_m$. Then at each layer, $M^{l,(i)}$ is created by stacking b/m copies of $pool^l[idx_i]$ on top of each other. M^l is then formed by stacking all m $M^{l,(i)}$ on top of each other.

Note the same indexes $idx_1, \ldots idx_m$ are used to choose the masks present in every layer. By using the same indexes across layers, we ensures that Pooled k-dropout is independently sampling subnets from a pool at each training step, where subnetwork i is defined as the nodes that are not dropped out when using the $pool^l[i]$ mask at every layer.

The two distinct phases of pooled k-dropout, *initialization* of the mask pools and *resampling* are outlined in algorithm 2. Note that one of the key differences between sequential k-dropout and pooled k-dropout is in sequential k-dropout, the resampling function is called every k training steps, whereas in pooled k-dropout the resampling function is called at every training step.

3. Experimental Results

We split our experimented analysis of k-dropout up into three sections. We firstly explored overall performance of using k-dropout across a wide range of hyperparameter

Algorithm 2 Pooled k-dropout Initialization and Resampling

Initialization
Input: $k, m, p, model$, and batch size b
for dropout_layer **in** model **do**
  h := dropout_layer.hidden_size
  mask_pool := []
  block_size := b/m
  for $i = 0$ **to** $m - 1$ **do**
    $mask^i$:= bernoulli_sample(p, shape=(1, h))
    mask_pool.append($mask^i$)
  end for
  dropout_layer.pool := mask_pool
end for

Resampling
Input: $k, m, p, model$, and batch size b
for $i = 0$ **to** $m - 1$ **do**
  idx_i = randint(0, $m - 1$)
end for
for dropout_layer **in** model **do**
  h := dropout_layer.hidden_size
  mask_blocks := []
  block_size := b/m
  for $i = 0$ **to** $m - 1$ **do**
    $mask^i$:= dropout_layer.pool[idx_i]
    $M^{(i)}$:= repeat($mask^i$, block_size)
    mask_blocks.append($M^{(i)}$)
  end for
  M := concatenate(mask_blocks)
  dropout_layer.mask := M
end for

settings (3.1). We then investigated the subnet training dynamics for pooled (3.3) and sequential (3.2) k-dropout to provide explanations for the trends in performance seen. Unless otherwise stated, all experiments were conducting by training a 2 layer MLP with hidden size 2000 on CIFAR 10 (Krizhevsky et al., 2010) using a learning rate of 0.0005, ADAM optimizer, and batch size of 512 for 300 epochs (which in all cases lead to training loss convergence).

3.1. k-dropout Performance

We trained a number of different models using the aforementioned hyperparameters along with pooled and sequential k-dropout applied to every MLP hidden layer. We tested a number of different p values and found in almost all cases $p = 0.5$ to be the most performant and thus use this setting for all of our experiments. The performance of the models is shown in Figures 1 and 2.

We begin by examining the results for pooled dropout as seen in the right panel of Figure 1. We notice a number

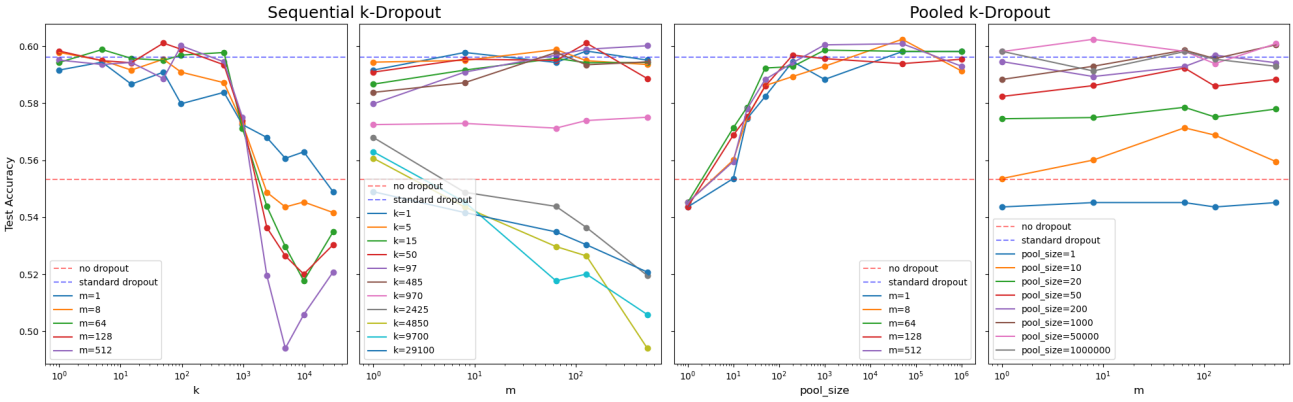

Figure 1. Hyperparameter sweeps of sequential (across k and m) and pooled (across $pool\_size$ and m) k-dropout on the CIFAR-10 dataset.

k	Training Epochs Per Subnet	Total Subnets
1	0.01	29100
5	0.05	5820
15	0.15	1940
50	0.52	582
97	1	300
485	5	60
970	10	30
2425	25	12
4850	50	6
9700	100	3
29100	300	1

Table 1. k values for sequential k-dropout and associated subnets (assuming $m = 1$).

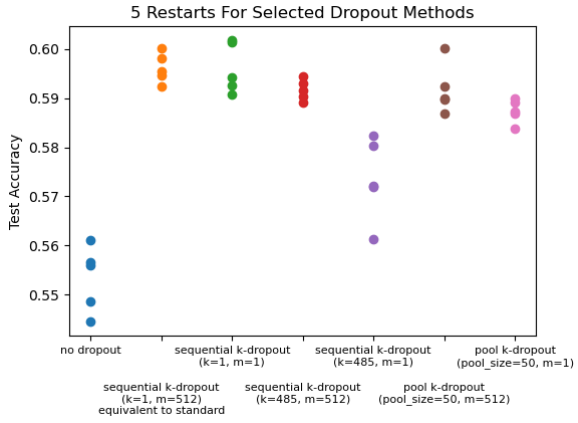

Figure 2. Performance of a model trained on CIFAR-10 for 300 epochs and a batch size of 512 using different dropout methods.

of interesting results, Firstly, a pool size of as low as 50 is able to perform within 1.5% of standard dropout across all m values, and increasing pool size to 200 achieves comparable performance. For this experiment, standard dropout trains 512 separate subnets at each training step (equal to the batch size used), for 97 batches in the dataset, for 300 epochs. This corresponds to 14,899,200 separate subnets. Here we show we can train roughly 75,000 times fewer subnets and achieve comparable performance. We also note that for every pool size above 1, we outperform the corresponding no dropout model, showing that pooled k dropout essentially never leads to performance degredation, except for the degenerate case when training with pool size 1 which simple corresponds to training a sparse subnetwork. Next, we notice an interesting phase transition around pool size 50, above which we get performance comparable to dropout, and below which we suffer fast degradation. We also see for large pool sizes we occasionally outperform normal dropout. Figure 2 shows some selected hyperparameter settings trained 5 times and confirms that some settings occasionally outperform standard dropout (with standard dropout results are shown in orange) and in general are distributed similarly to standard dropout. Finally, we find that m has little effect on pooled k-dropout performance.

Results for sequential k-dropout can be seen in the left panel of Figure 1. Note that standard dropout corresponds to $k = 1$ and $m = 512$. Table 1 shows conversions between corresponding k values and total trained subnets (i.e. pool size if using pooled k-dropout) when using $m = 1$. Firstly, we note low k values also perform similarly or in some cases outperform standard dropout (depending on weight initialization). Surprisingly, for sequential k-dropout there is also a phase transition in network performance around $k = 485$. $k = 485$ and $m = 1$ corresponds to training a total of 60 subnets across training, making the performance phase transition in sequential k-dropout remarkably similar

to that of pooled k-dropout (that occured when training a total of 50 subnets). This result seems particularly interesting as we originally hypothesized that that for a given small number of total networks trained, sequential k-dropout would underperform pooled k-dropout because subnets that were trained early in training would not be relevant by the end of training (as their parameters would never be directly updated together later in training), leading to a lower number of effective subnets contributing to the final model. We explore why this hypothesis is incorrect and explanations behind the surprisingly high sequential k-dropout performance for large k values in section 3.2. Finally, unlike in pooled k-dropout, we see a perculiar bi-modal behaviour in the effect of altering m on model performance. For low k values, m has little effect on model performance, much like in pooled k-dropout, however for large k values, larger m values significantly degrade model performance, from values that are above or similar to a standard non-dropout model, to ones that are roughly 5% lower in terms of test accuracy. We hypothesize that this may be due to interference between subnets prevents convergence when the total number of subnets being trained is low (as is the case for large k values) however more investigation is necessary to fully understand this phenomena.

3.2. Training Dynamics With Sequential k-Dropout

Our performance results showed k-dropout performing surprisingly well for large values of k. To investigate how this was possible, we conducted a number of experiments tracking how each subnets evolved during training for large k values and $m = 1$. We choose $m = 1$ because we were still able to outperform a no dropout model for all but the very largest k value, and setting $m = 1$ allows us to better reason about how subnets evolve as only a single subnet is updated at each training step.

We began by training a range of models with large k values and $m = 1$ and extracting the subnets that were trained after training of the full model was complete. We then recorded the test accuracy of these individual subnets (scaling activations in the same manner as was done during training). The results are shown in Figure 3, where the subnet index reflects the order in which subnets were trained. Unexpectadly we see a U shape. subnets that were trained early on perform the best (in general the very first subnetwork does the best or very close to best), subnets trained in the middle of training perform worst, and then performance goes up again for subnets trained at the end of training. Figure 4 shows plots of the test accuracy for each subnet during the entire network training for $k = 2425$ and $m = 1$. Points where dropout masks are resampled are denoted with black vericle lines, and the red band on each subplot shows the training steps when that specific subnetwork was trained. We see a number of interesting behaviours. Firstly, subnet 0, the first subnet to be trained, achieves high performance during training and then retains this high performance as other subnets are trained. This is not true for every other subnet, that get a small amount of accuracy gain before they are trained directly, a large gain when they are trained, and then get a degradation in performance as following subnets are trained. We note that this behaviour holds across k values, with additional plots for $k = 4850$ and $k = 970$ available in the appendix A. The only change we see for lower k values such as $k = 970$ (which corresponds to the second rightmost subplot in Figure 3) is that not only the first subnets to be trained, but a small number of the first subnets to be trained retain high performance as other subnets are later trained.

We hypothesise that this behaviour could be due to the earliest subnets forcing their weights to be in an area of weight space corresponding to certain low loss neighborhood for themselves that other later subnets are unable to escape from. Instead later subnets tune their parameters within this loss neighborhood, finding specific local optima for themselves that are then quickly overwritten by later subnets training, leading to performance degradation. The initial subnets never experience loss degradation however because their weights are never moved out of the preferable loss nieghorhood they were able to find early on in training. We plan on investigating this hypothesis in a number of ways. Firstly, we could view other subnet training as SGD noise and check if early subnets immediately after training are mode connected by linear paths to the same subnets at snapshots during training and at the end of the entire training process (a technique inspired by (Frankle et al., 2020)), suggesting early networks never left a linear low loss neighborhood of the loss landscape. More simply we also want to investigate how the weights and average activations of early subnets change during model training using some metric, such as L2 norm. We leave conducting these experiments to future work.

Finally, we also investigated how the performance of subnets that were directly trained during sequential dropout differs from randomly selected subnets. The results are shown in Figure 5. We denote randomly selected subnets with the suffix "random subnets" (where each plot corresponds to extracting 20 random subnets) and subnets that were directly trained, and thus featured in a dropout mask during sequential k-dropout training, with the suffix "dropout subnets" (where each plot corresponds to extracting all of the subnets that were directly trained). We see that as k increases, the performance difference between dropout subnets and random subnets increases, showing that large k values do not effectively train random subnets. We note that normal dropout does effectively train random subnets, which can be seen in the second the left most boxplot. We also see that as k increases, dropout subnetwork performance does

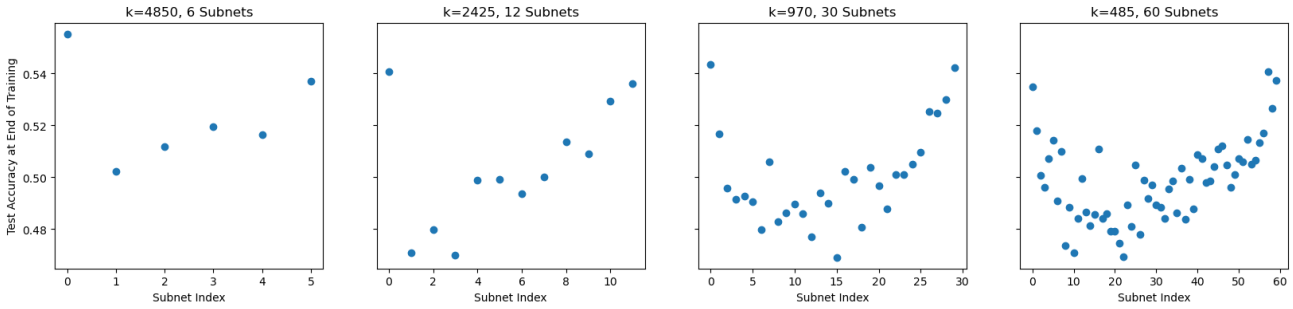

Figure 3. Performance at the end of training for each dropout subnetwork in sequential k-dropout models with $m = 1$.

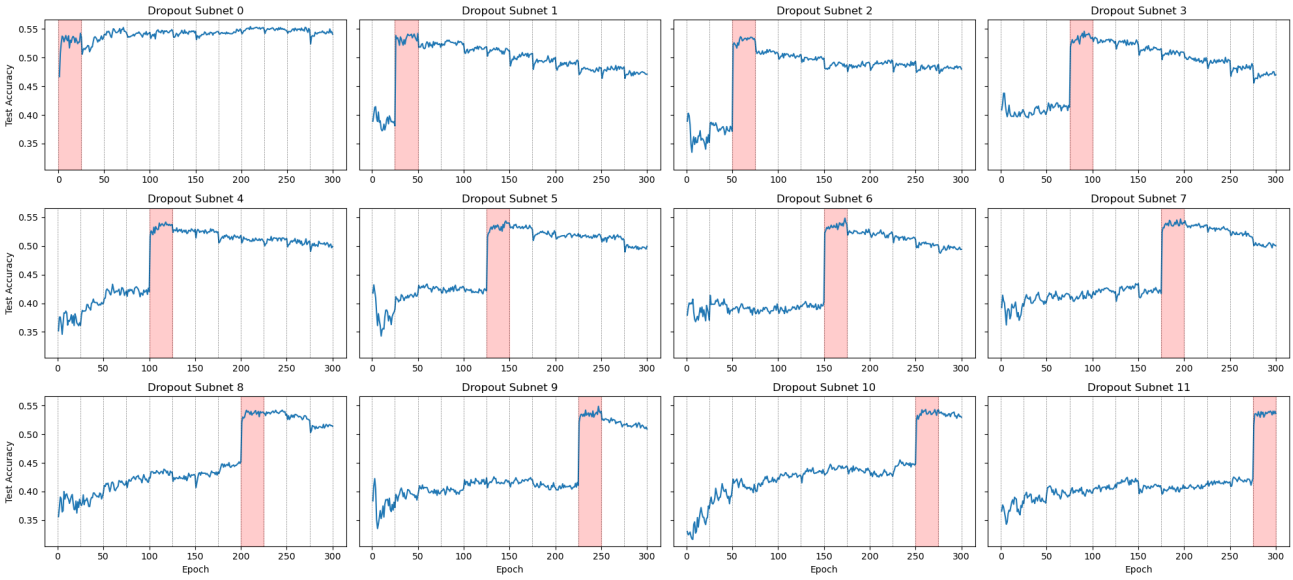

Figure 4. Accuracy over training for dropout subnets in a sequential k-dropout network with $k = 2425$ (12 total subnets).

not decline like full network performance does, showing that the degradation of model performance seen for large k values is not a result of underperformant training of subnets, but rather a lack of the number of said subnets and corresponding lack of robustness in random subnet performance.

3.3. Training Dynamics With Pooled k-Dropout

As both sequential and pooled k-dropout perform well when training a surprisingly low number of subnets, we performed similar experiments on the training dynamics of pooled k-dropout. Using a pool size of 50, we compared the performance at the end of training of random subnets to the dropout subnets (where "dropout subnet" now refers to those from the pool that were directly trained). We chose to use a pool size of 50, which experimentally was the lowest number of subnets while retaining near-standard dropout performance and $m = 512$ as we found m had negligible impact on performance for pooled k-dropout. Note that unlike sequential k-dropout, when using pooled k-dropout different values of m do not change the total number of subnets trained, as this is dictated by the pool size hyperparameter.

We found that the dropout subnets in pooled k-dropout performed better than the randomly sampled subnets in the standard dropout model despite the full pooled k-dropout network performing slightly worse than standard dropout. Each pool subnetwork for the pooled k-dropout model was directly trained while the random subnets, for both standard and pooled, were not.

We also trained an ensemble of 50 subnets independently (simply 50 half-size networks with independent weight initialization) and found that while the ensemble performed much better than standard dropout and pooled k-dropout, the individual members of the ensemble performed about as well as the subnets from the pooled k-dropout model (Figure 6). Additionally, when we treat the subnets from the pooled and standard models as ensembles (running each

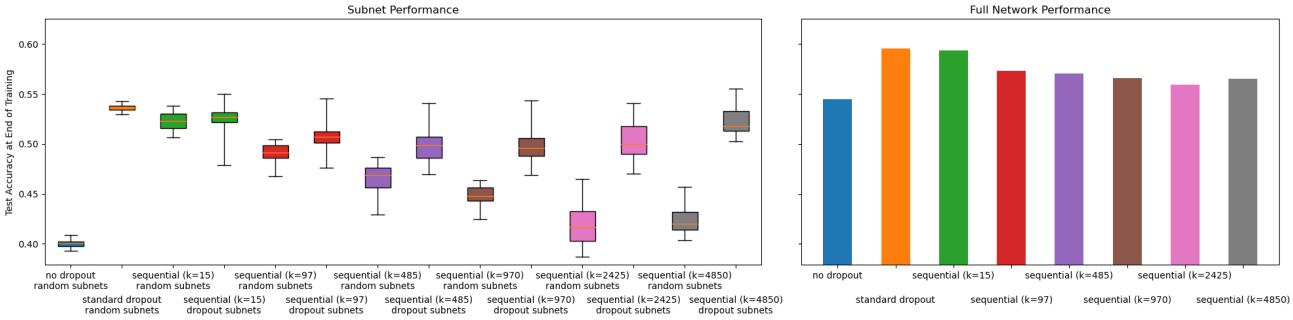

Figure 5. Performance of random and dropout subnets as well as the full models for no dropout, standard dropout, and sequential k-dropout with various values for k and $m = 1$.

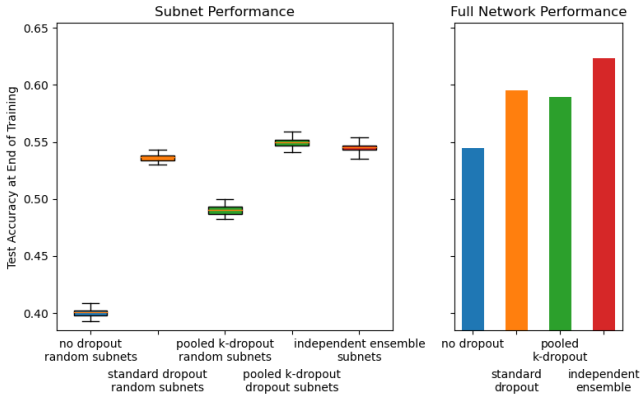

Figure 6. Performance of random, dropout, and independently trained subnets and full models for no dropout, standard dropout, pooled k-dropout (with $pool\_size = 50$), and an ensemble of 50 individually trained subnets.

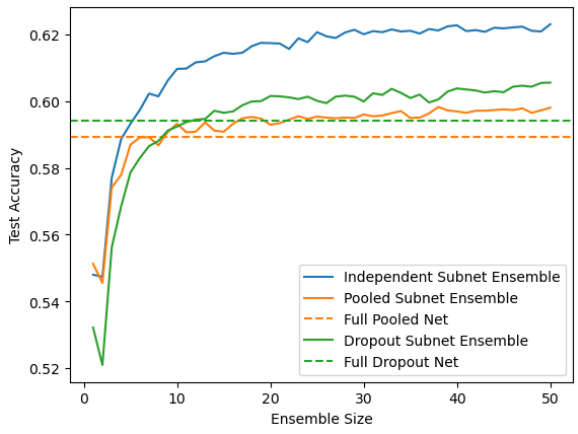

Figure 7. Performance of ensembles of subnets in pooled k-dropout and standard dropout models compared to an ensemble where each network is independently trained.

individually and averaging the outputs), we find that the performance is worse than the independently trained ensemble (Figure 7). We also found that the ensemble of random subnets from the standard dropout model outperformed the ensemble of dropout subnets from pooled k-dropout, despite the pooled k-dropout subnets performing better individually. Though the gap between the ensemble performance for the subnets of the standard and pooled models tracks with the difference in overall model performance (Figure 7), it will take more work to understand the discrepancy in ensemble performance between the subnets of these models.

4. Related Works

In this section, we review the related works on dropout techniques, including the theoretical foundations of dropout, different dropout methods for convolutional networks and transformers, dropout for compressing neural networks, Monte Carlo dropout methods for estimating model uncertainty, and applications of dropout in continual learning.

4.1. Theoretical Foundations of Dropout

Dropout was first introduced as a technique for preventing overfitting and improving generalization in deep neural networks (Srivastava et al., 2014). The method involves randomly dropping units and their connections from the neural network during training, effectively creating a new subnetwork at each training step. Dropout has been interpreted in two main ways: as an ensemble technique and as a Bayesian approximation. Under the ensemble, paradigm Dropout can be seen as a model averaging technique that trains an ensemble of subnets, where each subnetwork is created by randomly dropping out units from the original network (Warde-Farley et al., 2013; Baldi & Sadowski, 2013). Dropout has also been interpreted as a Bayesian approximation to the posterior distribution over network weights. Gal and Ghahramani introduced this interpretation and showed that dropout can be viewed as a variational approximation to the true Bayesian posterior (Gal & Ghahramani, 2016).

4.2. Dropout Methods for Convolutional Networks and Transformers

Different dropout methods have been proposed for convolutional networks and transformers. For convolutional networks, spatial dropout has been proposed, where entire feature maps are dropped instead of individual units (Tompson et al., 2014). This technique encourages the network to learn more robust and invariant features. For transformers, Variational Dropout was introduced, which applies the same dropout mask to all the time steps in the sequence (Kingma et al., 2015). This technique has been used in the BERT model for improved pre-training of language representations (Devlin et al., 2018). The k-dropout generalization applies to both these more specific dropout implementations, by simply altering the method in which new dropout masks are sampled. Investigating how these techniques could benefit by tuning k and m is left to future work.

4.3. Other Applications Of Dropout

Although dropout is most commonly used to increase model performance through regularization, there are a number of other interesting applications for the technique. Dropout has been utilized for compressing neural networks by identifying and removing less important connections or neurons. (Han et al., 2015). Compression poses another possible interesting application for k-dropout, with the possibility that larger k values induce more modularity in the network and thus allow for more aggressive pruning.

Monte Carlo dropout (MC dropout) has been proposed as a method for estimating model uncertainty using dropout at test time (Gal & Ghahramani, 2016). The method involves performing multiple forward passes through the network with dropout enabled during inference and averaging the predictions to obtain a more reliable and informative estimate. Similar techniques could be used with a model trained using k-dropout.

Finally, dropout has also been applied in the context of continual learning. PathNet can be viewed as a version of dropout applied to continual learning, where a fixed-size network is used to learn multiple tasks (Fernando et al., 2017). In PathNet, each task is learned by a subnetwork composed of a subset of the network's parameters, effectively dropping out the unused parameters. This allows the network to learn multiple tasks without catastrophic forgetting. This fits within the k-dropout generalization through a more complicated mask resampling strategy and k values that are dynamic depending on the specific task that is being trained on.

5. Conclusion

Inspired by the ensemble interpretation of dropout training, we introduced k-dropout, a generalization of dropout that allows for individual subnets to be trained for more than a single step. We explore two different implemenetation of k-dropout, sequential and pooled. For both, when training an MLP on CIFAR-10, we show the number of subnets trained can be reduced to values below 100 and achieve comparable performance to standard dropout. We also explored the training dynamics of these techniques, finding the early subnets trained in sequential dropout retain high performance even after they have stopped being trained, inspiring future work to better understand this phenomena.

References

Baldi, P. and Sadowski, P. J. Understanding dropout. *Advances in neural information processing systems*, 26, 2013.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A. A., Pritzel, A., and Wierstra, D. Pathnet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Frankle, J., Dziugaite, G. K., Roy, D., and Carbin, M. Linear mode connectivity and the lottery ticket hypothesis. In *International Conference on Machine Learning*, pp. 3259–3269. PMLR, 2020.

Gal, Y. and Ghahramani, Z. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059. PMLR, 2016.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015.

Hara, K., Saitoh, D., and Shouno, H. Analysis of dropout learning regarded as ensemble learning. In *Artificial Neural Networks and Machine Learning–ICANN 2016: 25th International Conference on Artificial Neural Networks, Barcelona, Spain, September 6-9, 2016, Proceedings, Part II 25*, pp. 72–79. Springer, 2016.

Harlap, A., Narayanan, D., Phanishayee, A., Seshadri, V., Devanur, N., Ganger, G., and Gibbons, P. Pipedream: Fast and efficient pipeline parallel dnn training. *arXiv preprint arXiv:1806.03377*, 2018.

Jumper, J., Evans, R., Pritzel, A., Green, T., Figurnov, M., Ronneberger, O., Tunyasuvunakool, K., Bates, R., Žídek, A., Potapenko, A., et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

Kingma, D. P., Salimans, T., and Welling, M. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 (canadian institute for advanced research), 2010.

Labach, A., Salehinejad, H., and Valaee, S. Survey of dropout methods for deep neural networks. *arXiv preprint arXiv:1904.13310*, 2019.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Tompson, J. J., Jain, A., LeCun, Y., and Bregler, C. Joint training of a convolutional network and a graphical model for human pose estimation. *Advances in neural information processing systems*, 27, 2014.

Warde-Farley, D., Goodfellow, I. J., Courville, A., and Bengio, Y. An empirical analysis of dropout in piecewise linear networks. In *arXiv preprint arXiv:1312.6197*, 2013.

Xie, J., Ma, Z., Lei, J., Zhang, G., Xue, J.-H., Tan, Z.-H., and Guo, J. Advanced dropout: A model-free methodology for bayesian dropout optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9):4605–4625, 2021.

A. Appendix

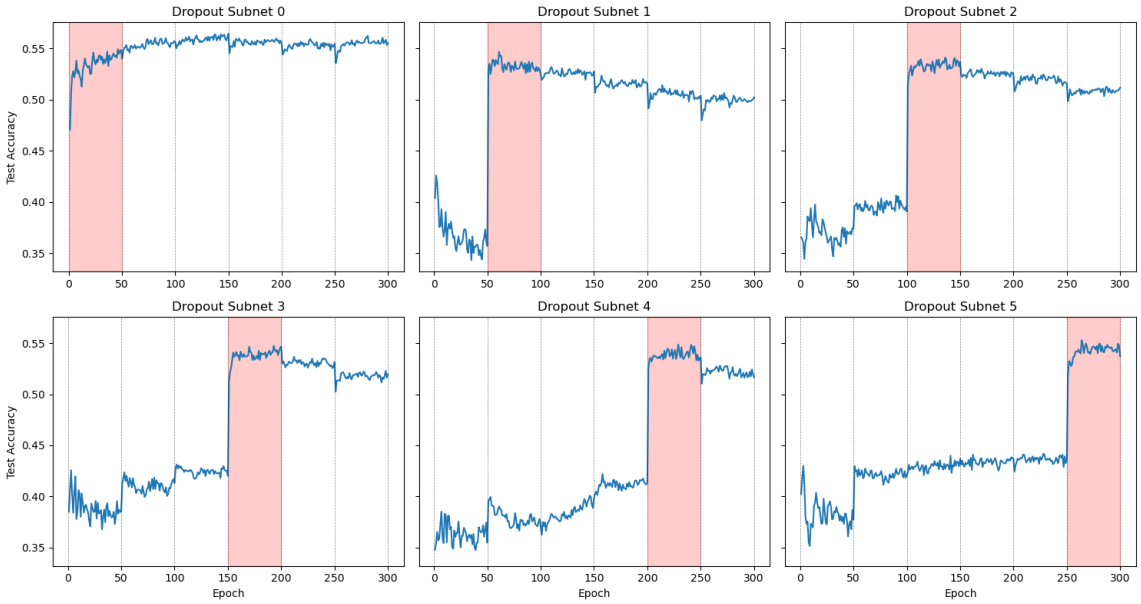

Figure 8. Accuracy over training for dropout subnets in a sequential k-dropout network with $k = 4850$ (6 total subnets).

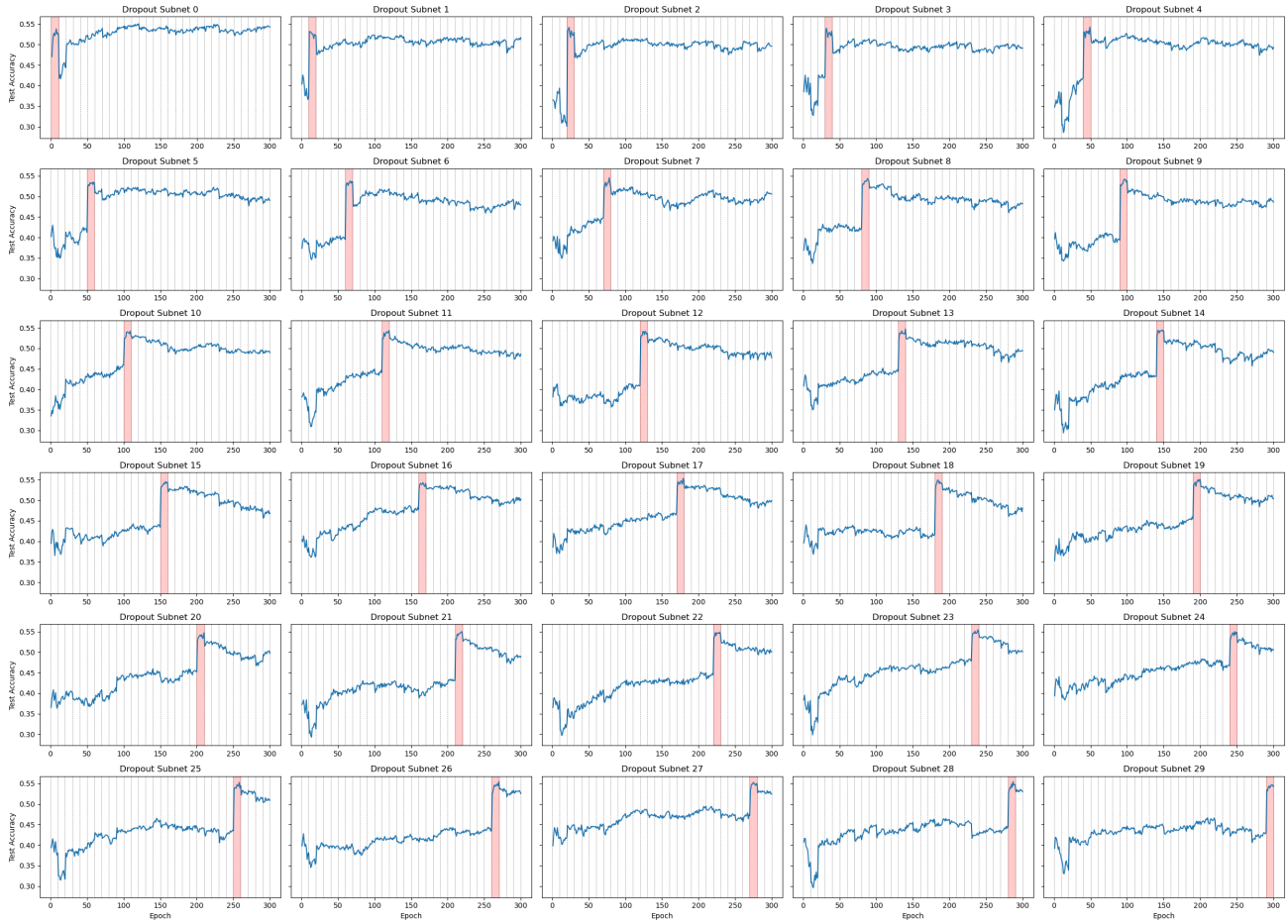

Figure 9. Accuracy over training for dropout subnets in a sequential k-dropout network with $k = 970$ (30 total subnets).

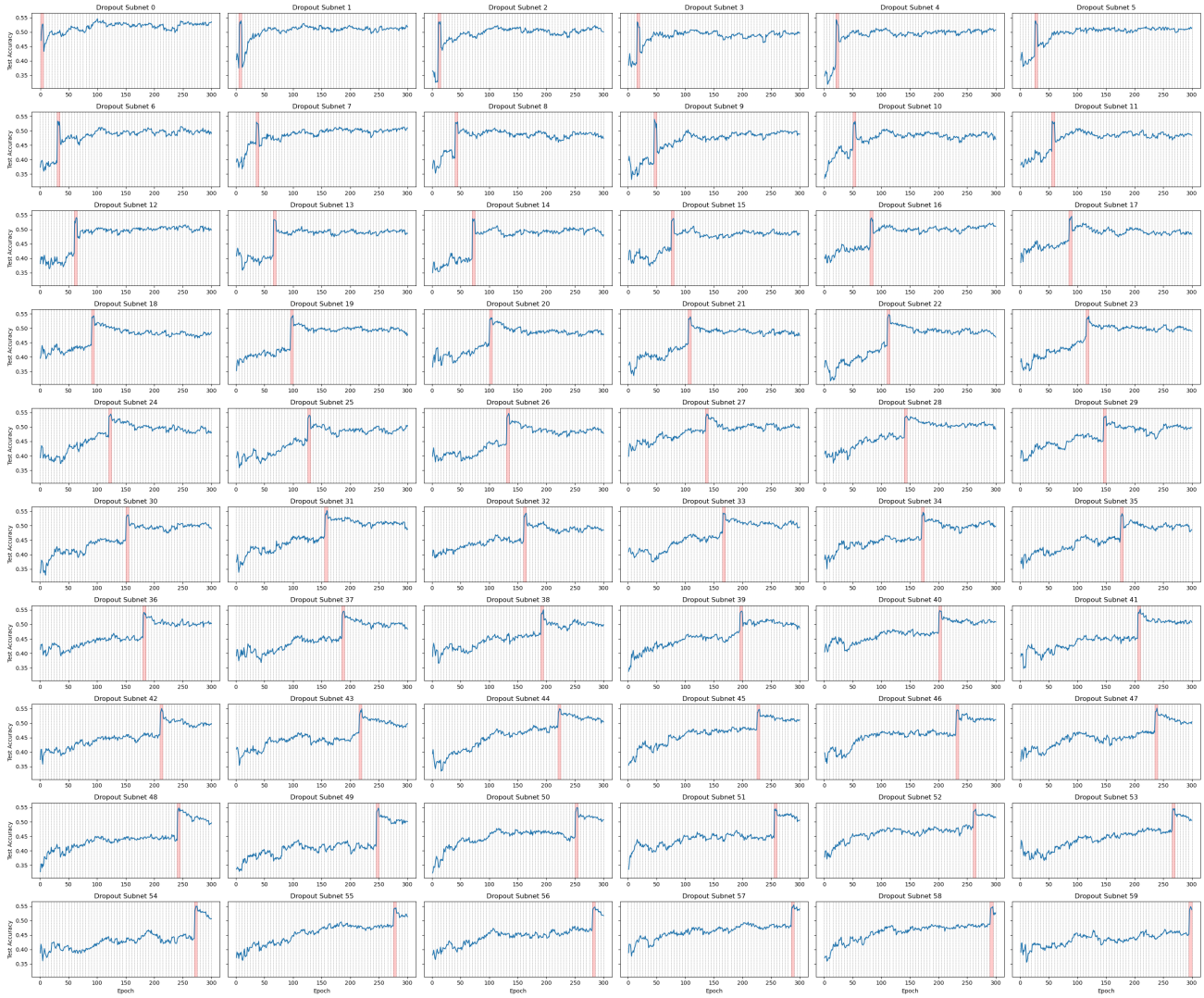

Figure 10. Accuracy over training for dropout subnets in a sequential k-dropout network with $k = 485$ (60 total subnets).