Deep networks with probabilistic gates

Charles Herrmann
Cornell University
cih5@cornell.edu

Richard Strong Bowen
Cornell University
rsb349@cornell.edu

Ramin Zabih
Cornell University, Google Research
rdz@cs.cornell.edu

December 12, 2018

Abstract

We investigate learning to probabilistically bypass computations in a network architecture. Our approach is motivated by AIG [43], where layers are conditionally executed depending on their inputs, and the network is trained against a target bypass rate using a per-layer loss. We propose a per-batch loss function, and describe strategies for handling probabilistic bypass during inference as well as training. Per-batch loss allows the network additional flexibility. In particular, a form of mode collapse becomes plausible, where some layers are nearly always bypassed and some almost never; such a configuration is strongly discouraged by AIG's per-layer loss. We explore several inference-time strategies, including the natural MAP approach. With data-dependent bypass, we demonstrate improved performance over AIG. With data-independent bypass, as in stochastic depth [18], we observe mode collapse and effectively prune layers. We demonstrate our techniques on ResNet-50 and ResNet-101 [11] for ImageNet [3], where our techniques produce improved accuracy (.15–.41% in precision@1) with substantially less computation (bypassing 25–40% of the layers).

1 Introduction

Despite the enormous success of convolutional networks [11, 23, 36], they remain poorly understood and difficult to optimize. A natural line of investigation, which [1] called conditional computation, is to conditionally bypass parts of the network. While inference-time efficiency could obviously benefit [1], bypassing computations can improve training time or test performance [4, 18, 43], and can provide insight into network behavior [18, 43].

In this paper we investigate several new probabilistic bypass techniques. We focus on ResNet [11] architectures since these are the mainstay of current deep learning techniques for image classification. The general architecture of a ResNet with probabilistic bypass gates is shown in figure 1. The main idea is that a residual layer, such as f_1, can potentially be bypassed depending on the results of the gating computation g_1, which controls the gate. The gating computation g_i can be data-independent, or it can depend on its input. If a g_i always executes its layer, this describes a conventional ResNet. A more interesting data-independent architecture is stochastic depth [18], where at training time g_i executes a layer with probability defined by a hyperparameter. At inference time the stochastic depth network is deterministic, though the frequency with which a layer was bypassed during training is used to scale down its weight.

With the introduction of Gumbel-Softmax (GS) [2, 7, 20, 30] it became possible to train a network to bypass computations, given some target bypass rate that trades off against training set accuracy. In this

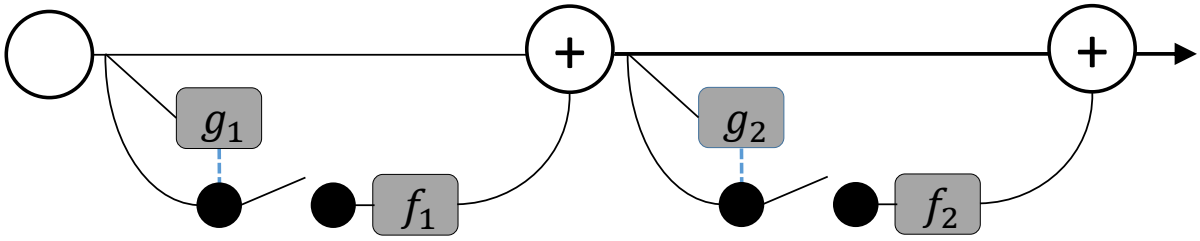


Figure 1: Architecture for ResNet with probabilistic bypass. The gate determines whether or not f_1 is bypassed, depending on the output of g_1. Dashed blue lines indicate control of a gate.

sense, probabilistic bypass serves as an additional regularizer for the network. This is the approach taken by AIG [43], where the bypass decision is data-dependent and the loss is per-layer.

We propose a per-batch loss function, which allows the network to more flexibly distribute bypass among different layers, compared to AIG's per-layer loss. This in turn leads to more advantageous tradeoffs between accuracy and inference speed. When our per-batch loss is applied with data-independent bypass, we observe a form of mode collapse where individual layers are either nearly always bypassed or nearly never bypassed. This effectively prunes layers, and again results in advantageous tradeoffs between accuracy and inference speed.

Whether a network uses data-dependent or data-independent probabilistic bypass, there remains a question of how to perform inference. We explore several alternative inference strategies, and provide evidence that the natural MAP approach gives good performance.

This paper is organized as follows. We begin by introducing notation and briefly reviewing related work. Section 3 introduces our per-batch loss function and our inference strategies. Experimental results on ImageNet and CIFAR are presented in section 4, followed by a discussion of some natural extensions of our work. Additional experiments and more details are included in the supplemental material.

2 Background

2.1 Notation

We first introduce some notation that allows us to more precisely discuss probabilistic bypass. Following [43], we write the output of layer $l \in \{0, 1, \ldots, L\}$ as x_l, with the input image being x_0. We can express the effect of a residual layer $\mathcal{F}_l$ in a feed-forward neural network as

$$x_l = x_{l-1} + \mathcal{F}_l(x_{l-1}).$$

Then a probabilistic bypass gate $z_l \in \{0, 1\}$ for this layer modifies the forward pass to either run or skip the layer:

$$x_l = x_{l-1} + z_l \mathcal{F}_l(x_{l-1}).$$

There are many different gating computations to determine the value of z_l, which can be set at training time, at inference time, or both. The degenerate case $z_l = 1$ corresponds to the original ResNet architecture.

Stochastic depth (SD) [18] can be formalized as setting

$$z_l = 1 - \frac{l}{L}(1 - p_L)$$

during training, where p_L is a hyperparameter set to 0.5 in the SD experiments. During inference, SD sets $z_l = 1$ but uses information gleaned during training to adjust the weights of each layer, where layers that were frequently bypassed are down-weighted.

We are particularly interested in AIG [43], which uses probabilistic bypass during both training and inference, along with a gating computation that depends on the input data (i.e., z_l is a function of x_{l-1}). Let $\mathcal{G}$ be the set of gates in the network and $\mathcal{B}$ be the set of instances in some mini-batch. AIG uses a target loss rate during training, and computes this on a per-gate basis. Given a target rate $t \in [0, 1]$ this is

$$\mathcal{L}_{\text{G}} = \frac{1}{|\mathcal{G}|} \sum_{g \in \mathcal{G}} \left(t - \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} z_{g,i} \right)^2$$

This loss function encourages each layer to be bypassed at the target rate. Note that this penalty is symmetric, so bypassing more layers is as expensive as bypassing fewer. The overall loss is the sum of the target loss (denoted as $\mathcal{L}_{\text{target}}$) and the standard multi-class logistic loss $\mathcal{L}_{\text{MC}}$,

$$\mathcal{L} = \mathcal{L}_{\text{target}} + \mathcal{L}_{\text{MC}}.$$

For AIG, the target loss is $\mathcal{L}_{\text{G}}$. AIG uses the straight through trick – the z's are categorical during the forward pass but treated as a Gumbel softmax during the backwards pass. At inference time AIG is stochastic, since a given layer l might be bypassed depending on its input x_{l-1} and the learned bypass probability for that layer.

2.2 Related work

Conditional computation has been well studied in computer vision. Cascaded classifiers [44] shorten computation by identifying easy negatives and have recently been adapted to deep learning [26, 45]. More directly, [14] and [31] both propose a cascading architecture which computes features at multiple scales and allows for dynamic evaluation, where at inference time the user can trade off speed for accuracy. Similarly, [42] adds intermediate classifiers and returns a label once the network reaches a specified confidence. [4, 6] both use the state of the network to adaptively decrease the number of computational steps during inference. [6] uses an intermediate state sequence and a halting unit to limit the number of blocks that can be executed in an RNN; [4] learns an image dependent stopping condition for each ResNet block that conditionally bypasses the rest of the layers in the block.

Another approach to decreasing the computation time is network pruning. The earliest works attempted to determine the importance of specific weights [10, 24] or hidden units [34] and remove those which are unimportant or redundant. Weight-based pruning on CNNs follows the same fundamental approach; [9] prunes weights with small magnitude and [8] incorporates these into a pipeline which also includes quantization and Huffman coding. Numerous techniques prune at the channel level, whether through heuristics [13, 25] or approximations to importance [12, 33, 41]. [29] prunes at the filter level using statistics from the following layer. [47] applies binary mask variables to a layer's weight tensors, sorts the weights during train time, and then sends the lowest to zero.

[19] is the most related to our data-independent bypass. They add a sparsity regularization and then modifies stochastic Accelerated Proximal Gradient to prune the network in an end-to-end fashion. Our work

differs from [19] by using GS to integrate the sparsity constraint into an additive loss which can be trained by any optimization technique; we use unmodified stochastic gradient descent with momentum (SGD), the typical technique for training classification. Recently, [27] suggests that the main benefits of pruning come primarily from the identified architecture.

Our work is also related to regularization techniques such as Dropout [40] and Stochastic Depth [18]. Both techniques try to induce redundancy through stochastically removing parts of the network during training time. Dropout ignores individual units and Stochastic Depth (as described above) skips entire layers. Both provide evidence that the increased redundancy improves helps to prevent overfitting. These techniques can be seen as applying stochastic gates to units or layers, respectively, where the gate probabilities are hyperparameters.

In the Bayesian machine learning community, data-independent gating is used as form of regularization. This line of work is cast as generalizing hyperparameter-per-weight dropout by learning individual dropout weights. [38] performs pruning by learning multipliers for weights, which are incentivized to be $0 - 1$ by a sparsity-encouraging loss $w(1 - w)$. [5] proposes per-weight regularization, using the straight-through Gumbel-Softmax trick. [37] uses a form of trainable dropout, learning a per-neuron gating probability. These are regularized by their likelihood against a beta distribution, and training is done with the straight-through trick. [39] learns sparsity at the weight level using a binary mask. They adopt a complexity loss which is L_0 on weights, plus a sparsification loss similar to [38]. This is similar to a per-batch loss. [28] extends the straight-through trick with a hard sigmoid to obtain less biased estimates of the gradient. They use a loss equal to the sum of Bernoulli weights, which is similar to a per-batch loss. [32] extends the variational dropout in [21] to allow dropout probabilities greater than a half. Training with the straight-through trick and placing a log-scale uniform prior on the dropout probabilities, they find substantial sparsification with minimal change in change in accuracy, including on some vision problems.

3 Using probabilistic bypass in deep networks

In this section we investigate ways to use probabilistic bypass in deep networks. We propose a per-batch loss function, which we use during training with Gumbel softmax following AIG. This frequently leads to mode collapse (not dissimilar to that encouraged by the sparsity-encouraging loss in [39]), which effectively prunes network layers. At inference time we take a deterministic approach, and observe that a simple MAP approach gives strong experimental results.

3.1 Batch loss during training

We note that $z_{g,i}$ can be viewed as a random variable depending on the instance, whose expectation is

$$\mathbb{E}_{\mathcal{B}}[z_g] = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} z_{g,i}.$$

Seen this way, AIG's per-gate loss is

$$\mathcal{L}_{\mathrm{G}} = \mathbb{E}_{\mathcal{G}} \left[\left(t - \mathbb{E}_{\mathcal{B}} \left[z \right] \right)^2 \right],$$

that is, a squared L_2 loss on $(t - z)$. The intuition for per-gate target loss is that each gate should be on expectation open around target t amount of time. When the gates are data-independent, this loss encourages

each layer to execute with probability t. When the gates are data-dependent, this loss encourages each layer to learn to execute on a fraction t of the training instances.

This per-gate target loss was intended to caused the layers of the network to specialize [43]. However, it is not a-priori obvious that specialization is the best architecture from a performance perspective. Instead, the most natural approach is to allow optimizer to select the network configuration which performs the best given a target activation rate. With this intuition, we propose per-batch target loss, which can be trained against using Gumbel softmax following AIG. For a target rate $t \in [0, 1]$

$$\mathcal{L}_{\text{B}} = \left(t - \frac{1}{|\mathcal{G}||\mathcal{B}|} \sum_{g \in \mathcal{G}} \sum_{i \in \mathcal{B}} z_{g,i} \right)^2 .$$

This can be interpreted as:

$$(t - \mathbb{E}_{\mathcal{G},\mathcal{B}}[z])^2$$

that is, a squared L_1 loss on $(t - z)$.

This loss only induces the network to have an activation of t. The intuition is that each batch is given t capacity and distributes the capacity among the instances and gates however it chooses. For example, if there exists a per-gate configuration with zero training loss that was easily found by optimization techniques, then per-batch would converge to it.

3.1.1 Mode collapse

With AIG's per-gate loss, each gate independently tries to hit its target rate, which means that the bypass rates will in general be fairly similar among gates. Our per-batch loss, however, allows different layers to have very different bypass rates, a network configuration would be heavily penalized by AIG.

In our experiments we frequently observe a form of mode collapse, where layers are nearly always bypassed or nearly never bypassed. In this situation, our loss function encourages a form of network pruning, where we start with an overcapacitated network and then determine which layers to remove during training. Surprisingly, our experiments demonstrate that we end up with improved accuracy.

3.2 Inference strategies

Once training has produced a deep network with stochastic gates, it is necessary to decide how to perform inference. The simplest approach is to leave the gates in the network and allow them to be stochastic during inference time. This is the technique that AIG uses. Experimentally, we observe a small variance so this may be sufficient for most use cases.

In addition, one way to take advantage of the stochasticity is to create an ensemble composed of multiple runs with the same network. Then any kind of ensemble technique can be used to combine the different runs: voting, weighing, boosting, etc. In practice, we observe a small bump in accuracy from this ensemble technique, though there is obviously a computational penalty.

However, stochasticity has the awkward consequence that multiple classification runs on the same image will often return different results. There are several techniques to remove stochasticity. The gates can be removed, setting $z_l = 1$ at test time. This is natural when viewing these gates as a regularization technique, and is the technique used by Stochastic Depth.

Alternately, inference can be made deterministic by using a threshold τ instead of sampling. Thresholding with value τ means that a layer will be executed if the learned probability is greater than τ. This also

allows the user some small degree of dynamic control over the computational cost of inference. If the user passes in a very high τ, then fewer layers will activate and inference will be faster. In our experiments, we set $\tau = \frac{1}{2}$

Note that we observe mode collapse for a large number of our per-batch experiments (particularly with data-independent gates). In this situation, for a wide range of τ thresholding can be interpreted as a pruning technique, where layers below a certain probability τ are pruned.

4 Experiments

Our primary experiments centered around probabilistic bypass to ResNet [11] and running the resulting network on ImageNet [3]. Our main finding is that our techniques improve both accuracy and inference speed. We also perform an empirical investigation into our networks in order to better understand their performance. Additional experiments, including CIFAR [22] as well as ImageNet, as well as more details are included in the supplemental material.

4.1 Improving speed and accuracy on ImageNet

We have implemented several probabilistic bypass techniques on the ResNet-50 and ResNet-101, and explored their performance on ImageNet. Since ResNet-101 is so computationally demanding, we have done more experiments on ResNet-50. Our techniques demonstrate improvements in accuracy and inference time on both ResNet-50 and ResNet-101.

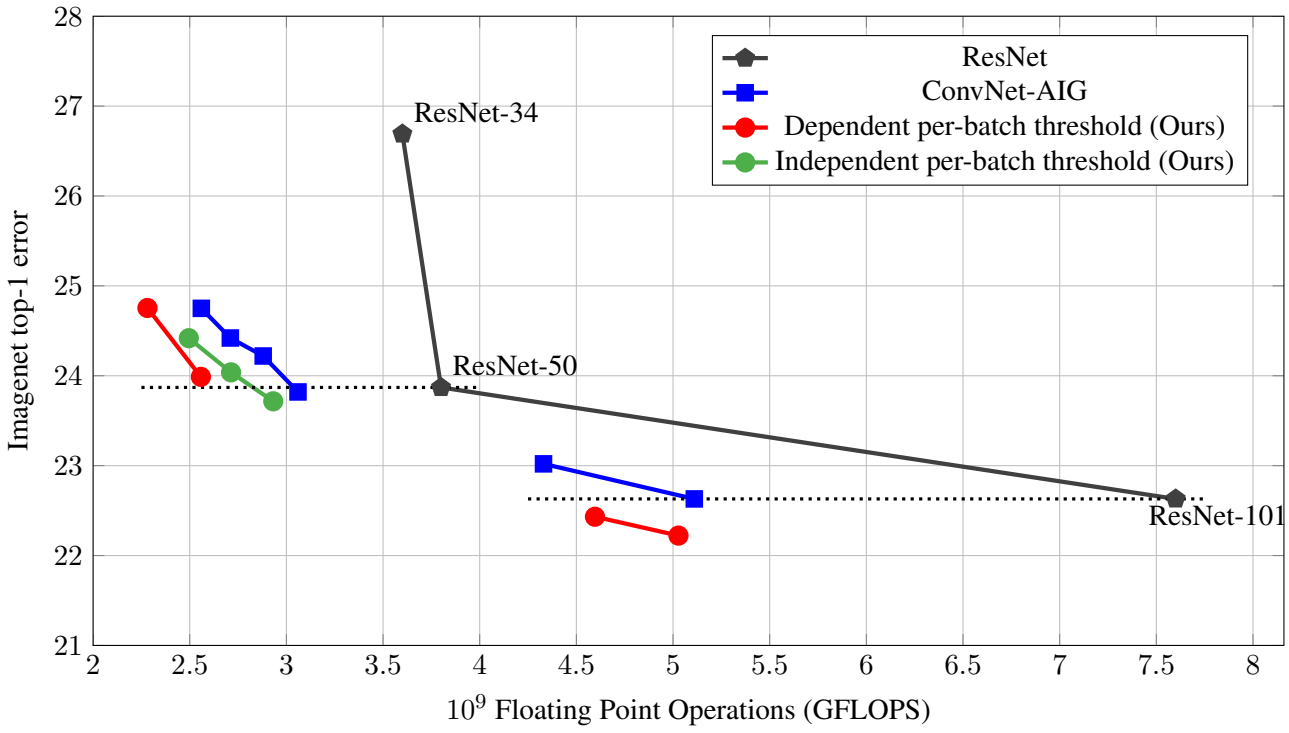


Figure 2: Selected results on ResNet (-50 and -101) and ImageNet. We improve on both accuracy and runtime over ConvNet-AIG [43] and plain ResNet.

Variant	Precision @1	Average Activation
Dep PB	**93.38**	**0.509**
Ind PB	92.9	0.517
Dep PG	92.5	0.54
Ind PG	92.85	0.568

Figure 3: CIFAR-10 results. Dependent Per-Gate is our implementation of [43]. Note that Dependent Per-Batch has both higher accuracy and lower activation than any of the other combinations.

4.1.1 Architecture and training details

We use the baseline architecture of ResNet-50 and ResNet-101 [11] and place gates at the start of each residual layer. We adopt the AIG [43] gate architecture. During training we explore different combinations of data-dependent or -independent gates. We used our per-batch loss, as well as AIG's per-gate loss, with target rates $t = \{0.4, 0.5, 0.6\}$. We kept the same training schedule as AIG, and followed the standard ResNet training procedure: mini-batch size of 256, momentum of 0.9, and weight decay of 10^{-4}. We train for 100 epochs from a pretrained model of the appropriate architecture with step-wise learning rate starting at 0.1, and after every 30 epochs decay by 10^{-1}.

We use standard training data-augmentation, and rescale the images to 256×256 followed by a 224×224 center crop. We observe that configurations with low gate activations cause the batch norm estimates of mean and variance to be slightly unstable. Therefore before final evaluation, we run training with a learning rate of zero and a large batch size for 200 batches in order to improve the stability and performance of the BatchNorm layers. This general technique was also utilized by [43].

4.1.2 Experimental results

Our results are shown in figures 2 and 4. The most interesting experimental results are obtained with data-dependent gates and our per-batch loss function $\mathcal{L}_{\text{B}}$, along with thresholding at inference time. This combination gives a 0.41 improvement in top 1 error over ResNet-101 while using 30% less computation. It also gives the same improvement in top 1 error over AIG. On ResNet-50, this technique saves significant computation compared to AIG or the baseline ResNet, albeit with a small loss of accuracy.

On the ResNet-50 architecture, we also investigated data-independent gates with per-batch loss, with thresholding at inference time. This produces an improvement in accuracy over our data-dependent architecture, as well as over AIG and vanilla ResNet-50. It saves significant computation (33% fewer gFLOPs) over ResNet-50, and is slightly faster than AIG but slower than the data-dependent architecture.

Figure 4 also shows the impact of the thresholding inference strategy, which is used for the 3 columns at right. We found that thresholding at inference time often gives the best performance, leading to roughly $.1 - .2$ percentage points improvement in top-1 accuracy.

We report the result on CIFAR10 and ResNet-101 in table 3. We note, for target rate of 0.5, dependent per-batch has the best performance in both accuracy and average activation. Figures for the remaining target rates can be found in the supplemental materials.

4.2 Empirical investigations

We performed a number of experiments to try to better understand the performance of our architecture. In particular, examining our learned bypass probability provides some interesting insights into how the network

Model	Top-1 Stochastic	Top-5 Stochastic	GFLOPs Stochastic	Top-1 Det.	Top-5 Det.	# GFLOPs Det.
ResNet-34	-	-	-	26.69	8.58	3.6
ResNet-50	-	-	-	23.87	7.12	3.8
AIG 50 [t=0.4]	24.75	7.61	2.56	-	-	2.56
AIG 50 [t=0.5]	24.42	7.42	2.71	-	-	2.71
AIG 50 [t=0.6]	24.22	7.21	2.88	-	-	2.88
AIG 50 [t=0.7]	23.82	7.08	3.06	-	-	3.06
Ind PG* [t=0.5]	24.99 (0.05)	7.71 (0.04)	3.81	24.23	7.17	3.04
Dep PG* [t=0.4]	25.25 (0.08)	7.81 (0.05)	2.51	24.78	7.57	2.52
Dep PG* [t=0.5]	24.92 (0.05)	7.50 (0.02)	2.73	24.52	7.27	2.79
Dep PG* [t=0.6]	24.47 (0.07)	7.36 (0.05)	2.99	24.07	7.16	3.03
Ind PB* [t=0.4]	24.70 (0.08)	7.63 (0.03)	2.43	24.42	7.48	2.49
Ind PB* [t=0.5]	24.39 (0.05)	7.46 (0.03)	2.65	24.04	7.29	2.71
Ind PB* [t=0.6]	**24.04 (0.03)**	7.11 (0.03)	2.93	**23.72**	**6.93**	2.93
Dep PB* [t=0.4]	24.98 (0.02)	7.63 (0.10)	2.27	24.75	7.56	2.28
Dep PB* [t=0.5]	24.22 (0.04)	7.18 (0.03)	2.52	23.99	7.06	2.55
Dep PB* [t=0.6]	24.16 (0.05)	7.24 (0.01)	2.71	23.99	7.14	2.73
ResNet-101	-	-	-	22.63	6.45	7.6
AIG 101 [t=0.3]	23.02	6.58	4.33	-	-	-
AIG 101 [t=0.4]	22.63	6.26	5.11	-	-	-
Dep PB* [t=0.5]	22.73 (0.02)	6.46 (0.02)	4.48	22.43	6.34	4.60
Dep PB* [t=0.6]	**22.45 (0.08)**	6.28 (0.04)	5.11	**22.22**	6.28	5.28

Figure 4: Error and GFLOPs for our method (marked with asterisks) compared to ConvNet-AIG. For stochastic errors we run the test set 5 times and report mean and, in parenthesis, standard deviation. For deterministic error, we use the thresholding inference technique.

behaves.

4.2.1 Pruning

With the per-batch loss, we often observe mode collapse, where some layers are nearly always on and some nearly always off. In the case of data-dependent bypass, we can measure the observed activation of a gate during training. For example, on a per-batch run on ResNet-50 (16 gates) on ImageNet, nearly all of the 16 gates mode collapse, as shown in figure 4.2.1: four gates collapsed to a mode of zero or one exactly; more than half were at their mode more than 99.9% of the time. Interestingly, we observe different activation behavior on different datasets. ImageNet leads to frequent and aggressive mode collapse, all networks exhibited some degree of mode collapse; CIFAR10 can induce mode collapse but does so much less frequently, approximately less than 40% of our runs.

Mode collapse can effectively perform end-to-end network pruning. At inference time, layers with near zero activation can be permanently skipped and even removed from the network entirely, decreasing the number of parameters in the network.

In the data-independent per-batch case, the threshold inference technique will permanently skip all layers with probability lower than threshold value τ, essentially pruning them from the network. Thus, we propose this combination as a pruning technique and report an experimental comparison[1] with other modern pruning techniques, shown in figure 6.

[1]We note a discrepancy between the GFlops reported by the baseline ResNet-50 between [43] and [16]. We calculate our

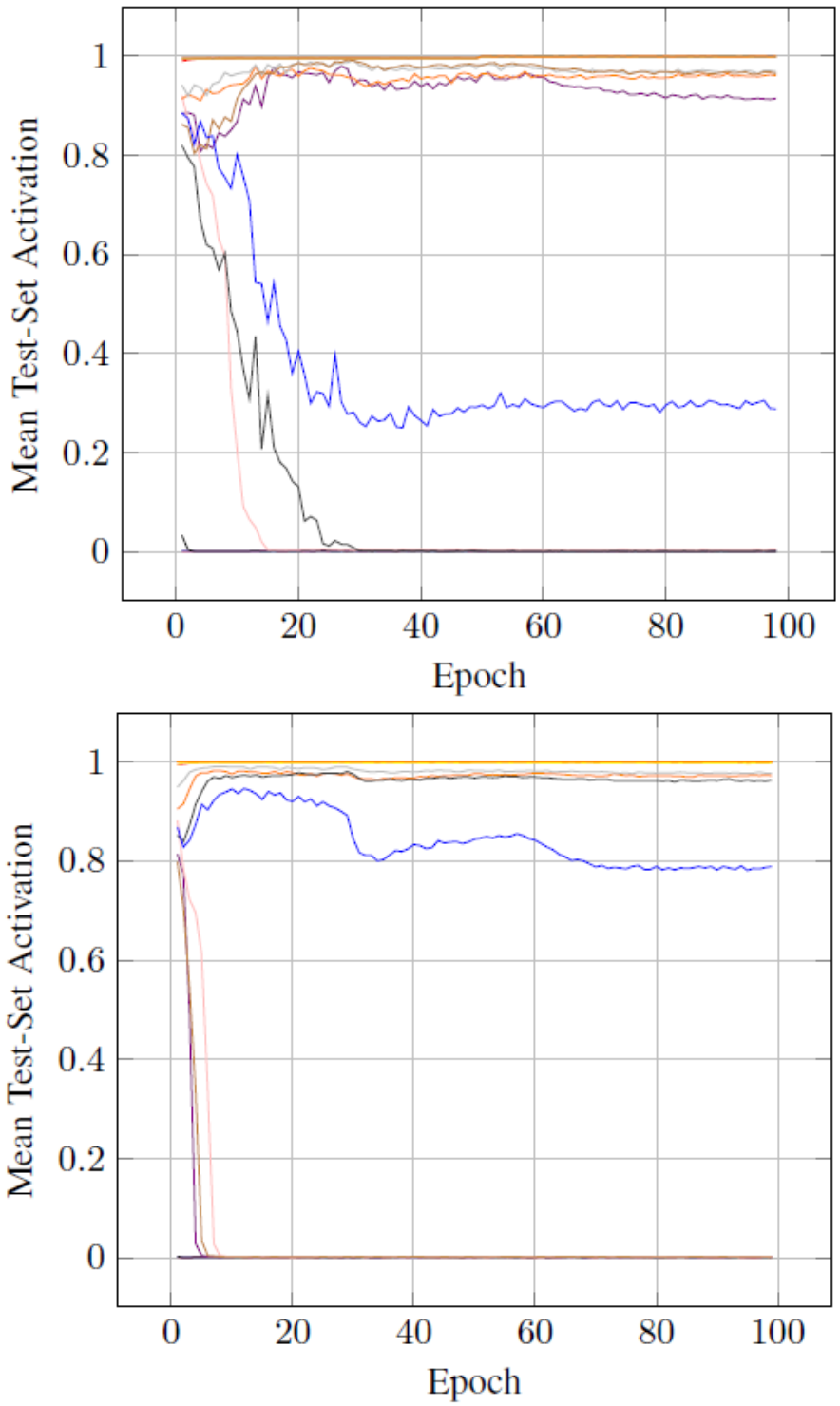


Figure 5: Demonstration of mode collapse on (left) data-dependent, per-batch ResNet-50 on ImageNet with target rate of .5, and (right) data-independent per-batch with target rate of .4 (right). Nearly all of the 16 gates collapse. Note that full mode collapse is discouraged by the quadratic loss whenever the target rate t is not equal to an integer over the number of gates g? Even if the layers try to mode collapse, the network will either be penalized by $gt \mod 1$ or learn activations that utilize the extra amount of target rate.

Model	Top-1	Top-5	# GFLOPs
ResNet-34-pruned [25]	27.44	-	3.080
ResNet-50-pruned (2×) [12]	27.70	9.20	**2.726**
ResNet-32 [19]	25.82	8.09	2.818
Res50 Ind PB* tr=.4	24.418	7.478	2.784
Res50 Ind PB* tr=.5	24.038	7.294	3.002
Res50 Ind PB* tr=.6	**23.716**	**6.934**	3.221

Figure 6: Error and GFLOPs for our method (marked with asterisks) compared with selected layer-based pruning techniques.

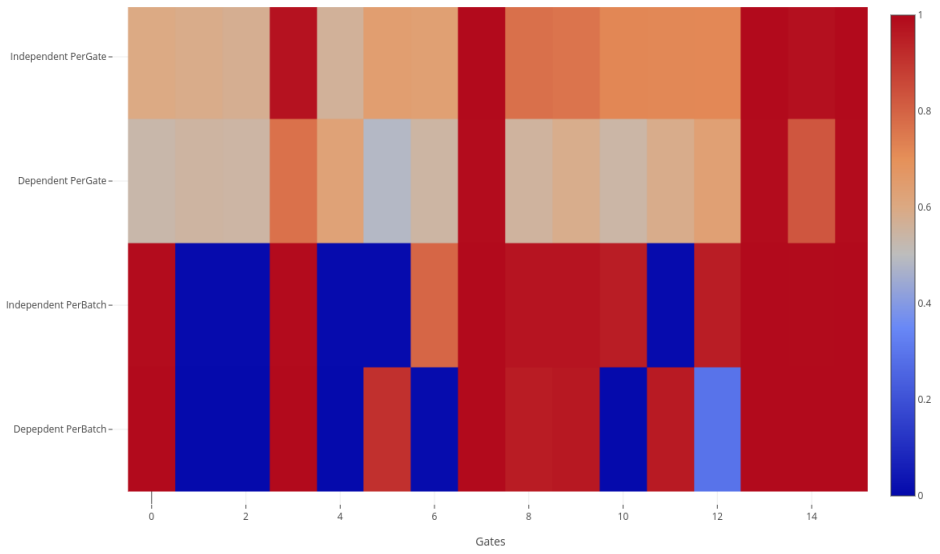


Figure 7: Mode collapse on a ResNet-50 run with (left) target rate 0.5 and (right) target rate 0.6. Counter-intuitively, we see that the network chooses to prune the early layers, corresponding to low-level features.

4.2.2 Understanding networks

The learned activation rates for various gates can be used to explore the relative importance of the gated layers. If the average activation for a layer in the dependent or independent case is low, this suggests the network has learned that layer is not very important. Counter-intuitively, our experiments show that early layers are not particularly important in both ResNet-50 and -101. As seen in figure 7 and figure 8, for low-level features, the network only keeps one layer out of the three available.

This suggests that fewer low-level features are needed for classification than generally thought. For example, on the ResNet-101 architecture, AIG constrains the three coarsest layers to have a target rate of 1, which indicates that these layers are essential for the rest of the network and must be on.

We can also experimentally investigate the extent to which layers specialize. AIG [43] uses their per-gate loss to encourage specialization, hoping to reduce overall inference time by letting the network restrict certain layers be used on specific classes. Although we find that per-batch generally outperforms per-gate in terms of overall activation, we note that in layers which are not mode collapsed, we do observe this kind of specialization even with a per-batch loss.

An interesting example of specialization is shown in figure 8. The figure shows activation rates for dependent per-batch ResNet-101 with a target rate of 0.5 using thresholding at inference time. The network has mostly mode collapsed – most layers' activations are either 1 or 0. However, the layers that did not mode collapse show an interesting specialization, similar to what AIG reported.

5 Extensions

There are a number of natural extensions to our work that we have started to explore. We have focused on the use of probabilistic bypass gates to provide an early exit, when the network is sufficiently certain of the answer. We are motivated by MSDNet [14], which investigated early exit for both ResNet [11] and DenseNet [17]. We tested the usage of probabilistic bypass gates for early exit on both ResNet and DenseNet. Consistent with [14], we found that ResNet tended to degrade with intermediate classifiers while DenseNet did not.

An immediate challenge is that in DenseNet, unlike ResNet, there is no natural interpretation of skipping a layer. Instead, we simply use the gate as a masking term. When the layer computation is skipped, the layer's output is set to zero and then, as per the architecture's design, is passed to later layers.

For early exit in DenseNet, we follow [42] and place gates and intermediate classifiers at the end of each dense block. At the gate, the network makes a discrete decision as to whether the the instance can be successfully classified at that stage. If the gate returns true, then the instance is run through the classifier and the answer is returned; if the gate returns false, then the instance continues throughout the network. The advantage of using GS here is that the early exit can be trained in an end-to-end fashion unlike [42] which uses reinforcement learning.

In our experiment, we implemented both early exit and probabilistic bypass on a per-layer basis. We set a per-gate target for layers of .9 and a per-gate target for both early exit points of .3 using a piece-wise function with a quadratic before the target rate and a constant after. This function matches the intuition that we should not penalize the network if it can increase the number of early exits without affecting accuracy.

We observe that these early exit gates can make good decisions regarding which instances to classify early; more specifically, the first classifier has a much higher accuracy on the instances chosen by the gate than on the entire test set. The network had an overall error of 5.61 while utilizing on average only 68.4% of

numbers the same way as [43]. To compare fairly to [16], we do the most conservative thing and add back the discrepancy.

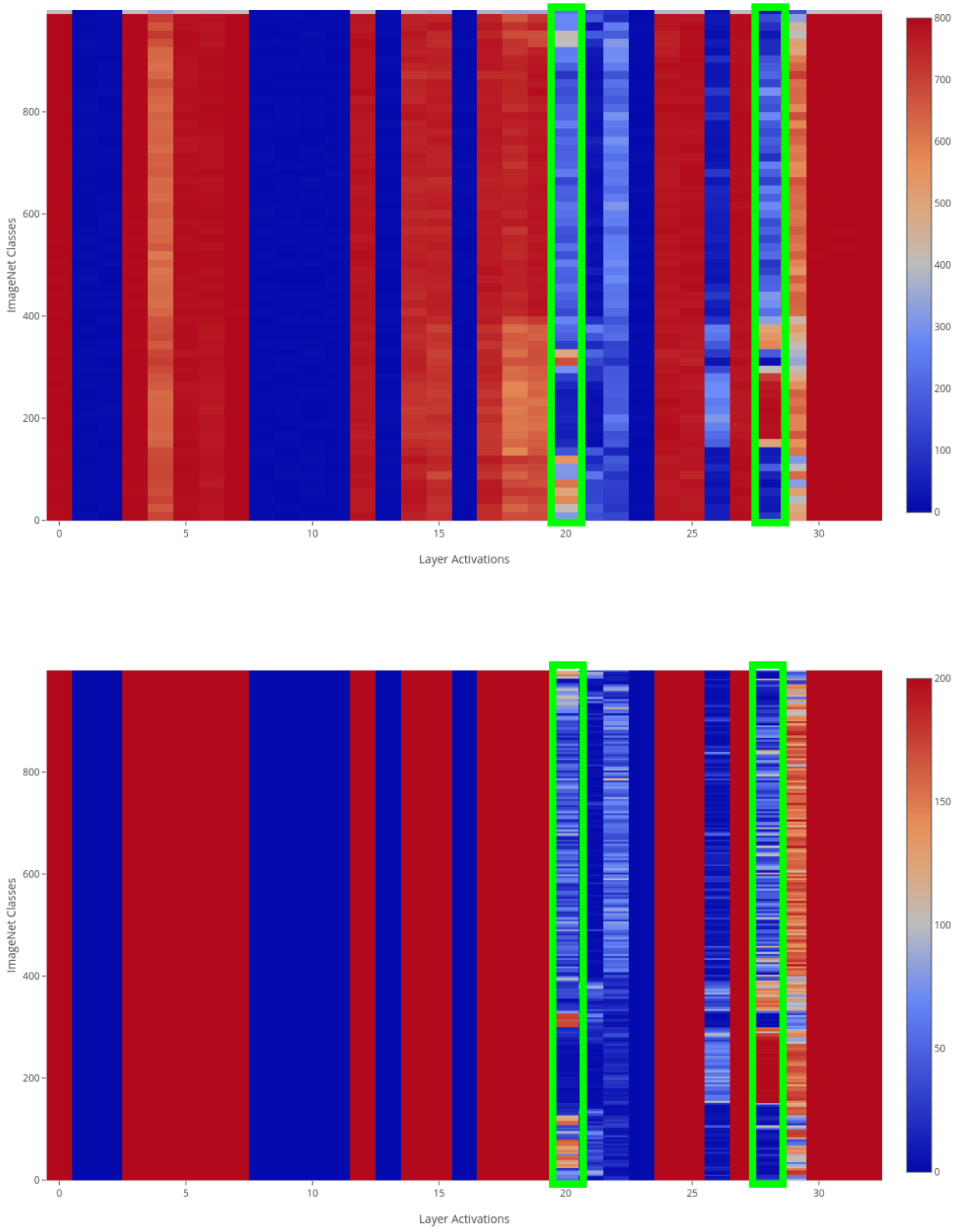


Figure 8: Specialization on ResNet-101 with data-dependent per-batch at target rate of 0.5. The left heatmap uses the stochastic strategy technique and the right heatmap uses the threshold inference strategy. Each vertical stripe is one layer; each row is an ImageNet class. While most layers have mode collapsed, the ones that have not mode collapsed show similar specializations as seen in [43]. For example, layer 24 runs mostly on fish and lizards, while layer 28 runs specifically on cats and dogs. These layers are highlighted in green.

	Block 1	Block 2	Final Block
Images classified	28.71%	11.56%	59.73%
Error on all images	18.64	6.65	5.81
Error on chosen images	3.63	1.47	7.37

Figure 9: DenseNet on CIFAR-10 with early exit via probabilistic bypass. 3906 out of 10000 test examples exited early (more than a third), with low error. Note that for the early classifiers, the error on chosen images is significantly lower than the error on all images. This suggests that the gates are successfully learning to recognize "easy" examples in the first block and "intermediate" examples in the second block.

the layers; our implementation of the original DenseNet architecture achieves an error of 4.76 ([17] reports an error of 4.51). The results for each block classifier are seen in Figure 9. More than a third of examples exited early, while overall error was still low. This demonstrates the potential of early exit with probabilistic bypass for DenseNet.

6 Conclusion and future work

One intriguing direction to explore is to remove the notion of a target activation rate completely, since it is not obvious what a good target would be for a particular use case. In general a user would prefer an accurate network with fast inference. The exact tradeoff between speed and accuracy will in general vary between applications, but there is no natural way for a user to express such a preference in terms of a target activation rate. It might be possible to automatically chose a target rate that optimizes a particular combination of inference speed and accuracy.

Another promising avenue is the idea of annealing the target rate down from 1. This makes the adjustment to the target loss more gradual and may encourage more possible configurations. Intuitively, this could gives the network a greater chance to 'change its mind' regarding a layer and alter the layer's representation instead of always skipping it.

We have demonstrated that probabilistic bypass is a powerful tool for optimizing and understanding neural networks. The per-batch loss function that we have proposed, together with thresholding at inference time, has produced strong experimental results both in terms of speed and accuracy.

Acknowledgements

We are especially grateful to Andreas Veit who spent considerable time and effort helping us understand his work on AIG. We also thank Serge Belongie for helpful conversations.

This work was generously supported by Google Cloud, without whose help it could not have been completed. It was funded by NSF grant IIS-1447473, by a gift from Sensetime and by a Google Faculty Research Award.

References

[1] Y. Bengio. Deep learning of representations: Looking forward. *CoRR*, abs/1305.0445, 2013. 1

[2] Y. Bengio, N. Léonard, and A. C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *CoRR*, abs/1308.3432, 2013. 1

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. Ieee, 2009. 1, 6

[4] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. P. Vetrov, and R. Salakhutdinov. Spatially adaptive computation time for residual networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1, 3

[5] Y. Gal, J. Hron, and A. Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3581–3590, 2017. 4

[6] A. Graves. Adaptive computation time for recurrent neural networks. *CoRR*, abs/1603.08983, 2016. 3

[7] E. J. Gumbel. Statistical theory of extreme values and some practical applications. *NBS Applied Mathematics Series*, 33, 1954. 1

[8] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 3

[9] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015. 3

[10] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993. 3

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 6, 7, 11

[12] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *International Conference on Computer Vision (ICCV)*, 2017. 3, 10

[13] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016. 3

[14] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. *CoRR, abs/1703.09844*, 2, 2017. 3, 11, 19

[15] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get m for free. *arXiv preprint arXiv:1704.00109*, 2017. 20, 25

[16] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 8, 11

[17] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 11, 13

[18] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, pages 646–661. Springer, 2016. 1, 3, 4, 26

[19] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. *European Conference on Computer Vision (ECCV)*, 2018. 3, 4, 10

[20] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *International Conference on Learning Representations (ICLR)*, 2017. arXiv preprint arXiv:1611.01144. 1

[21] D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2575–2583. Curran Associates, Inc., 2015. 4

[22] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009. 6

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. 1

[24] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990. 3

[25] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 3, 10

[26] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua. A convolutional neural network cascade for face detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5325–5334, 2015. 3

[27] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018. NIPS Workshop CDNNRIA. 4

[28] C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. *International Conference on Learning Representations (ICLR)*, 2017. arXiv preprint arXiv:1712.01312. 4

[29] J.-H. Luo, J. Wu, and W. Lin. Thinet: A filter level pruning method for deep neural network compression. *International Conference on Computer Vision (ICCV)*, 2017. arXiv preprint arXiv:1707.06342. 3

[30] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016. 1

[31] M. McGill and P. Perona. Deciding how to decide: Dynamic routing in artificial neural networks. In *ICML*, 2017. arXiv preprint arXiv:1703.06217. 3

[32] D. Molchanov, A. Ashukha, and D. Vetrov. Variational dropout sparsifies deep neural networks. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2498–2507, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. 4

[33] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *International Conference on Learning Representations (ICLR)*, 2016. arXiv preprint arXiv:1611.06440. 3

[34] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems*, pages 107–115, 1989. 3

[35] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018. 17

[36] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1

[37] S. Srinivas and R. V. Babu. Generalized dropout. *arXiv preprint arXiv:1611.06791*, 2016. 4

[38] S. Srinivas and V. Babu. Learning neural network architectures using backpropagation. In E. R. H. Richard C. Wilson and W. A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 104.1–104.11. BMVA Press, September 2016. 4

[39] S. Srinivas, A. Subramanya, and R. Venkatesh Babu. Training sparse neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017. 4

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 4

[41] X. Suau, L. Zappella, V. Palakkode, and N. Apostoloff. Principal filter analysis for guided network compression. *arXiv preprint arXiv:1807.10585*, 2018. 3

[42] S. Teerapittayanon, B. McDanel, and H. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 2464–2469. IEEE, 2016. 3, 11

[43] A. Veit and S. Belongie. Convolutional networks with adaptive inference graphs. In *European Conference on Computer Vision (ECCV)*, 2017. 1, 2, 3, 5, 6, 7, 8, 11, 12, 19, 26

[44] P. Viola and M. J. Jones. Robust real-time face detection. *International Journal of Computer Vision*, 57(2):137–154, 2004. 3

[45] F. Yang, W. Choi, and Y. Lin. Exploit all the layers: Fast and accurate CNN object detector with scale dependent pooling and cascaded rejection classifiers. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2129–2137, 2016. 3

[46] A. R. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722, 2018. 17

[47] M. Zhu and S. Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017. 3

Appendix: Additional extensions and experimental results

S1 Ongoing Work

Currently we are pursuing several different avenues of extending this project.

S1.1 Taskonomy

We are working to apply this work to Taskonomy [46], which tries to identify the most important features for a given task. One of the problems this paper faces is the combinatoral explosion in the number of higher-order transfers; specifically, to find the best k features for a given task, they need to exhaustively try $\binom{|S|}{k}$. In the paper, they rely on a beam search using the performance of a single feature by itself as a proxy for how well the feature will do in a subset. However, this seems highly suboptimal since it is plausible that some feature will perform poorly on its own but perform well when matched with a complement feature.

Instead, we propose to use all features as input and place probabilistic gates on the input. If the behavior of our data-independent gates remains the same (namely we observe mode collapse), then we can use our PerBatch training schedule to figure out the best subset of features. More specifically, we would restrict the number of features used through the target rate. For example, Taskonomy uses 26 features, so we expect that a target rate of $\frac{k}{26}$ will give the k best features for the task.

S1.2 Multi-task Learning

More generally, we plan to observe how probabilistic gates affects the common architecture for multi-task learning. We plan to apply data-dependent gates to the different feature representations and allow the network to either use or ignore the representation depending on the input value. The main motivation for this line of research is that for some inputs a given feature representation may not be useful and in fact using it may lead to worse results. Therefore the network should be allowed to ignore it depending on the input.

S1.3 MobileNet

We are actively exploring applying these techniques to MobileNet [35] and have some initial results. Applying this technique as is gives results that roughly work as well as changing the expansion factor; more specifically, our results are approximately on the line for MobileNetV2 224×224 in Figure 5. We are now working on improving on the line given by the expansion factor. Specifically, we are exploring two directions: (1) increasing the granularity with which we apply the gates and (2) creating an over-capacitated version of MobileNet and then using our techniques to prune it to be the correct size.

S2 Additional techniques for training

S2.1 Annealing

Additionally in the training stage, we propose annealing the target rate. In particular, we use a step-wise annealing target rate which decreases a amount after k epochs. Typical values are $a = .05$ and $k = 5$. The intuition behind annealing is that with per-batch activation loss, annealing prevents the network from greedily killing off the worst starting layers. Instead, layers which perform worse in the beginning have a chance to change their representation. In practice, we have observed that over time layer activations are not always monotonic and some layers will initially start to be less active but will recover. We observe this behavior more with an annealing schedule than for a fixed target rate.

S2.2 Variable target rate

As far as we are aware, previous work has used the L_2 term exclusively. This has the affect of forcing activations towards a specific target rate and letting them vary only if it leads to improvements in accuracy. However, this also prevents a situation where the network can decrease activation while retaining the same accuracy - a scenario which is clearly desirable.

As a result, we propose a piece-wise activation loss composed of a constant and then quadratic which indicates that there is no penalty for decreasing activation. Let $B_i = \sum_{i \in \mathcal{B}} z_{g,i}$ and t be the target activation rate. For the per-batch setup, this loss is as follows.

$$\mathcal{L}_{CQ,B} = \begin{cases} 0, \text{for } B_i \leq t \\ \left(t - \frac{1}{|\mathcal{G}||\mathcal{B}|} \sum_{g \in \mathcal{G}} B_i\right)^2, \text{for } B_i > t \end{cases}$$

Additionally, there are many cases where a target activation is not clear and the user simply wants an accurate network with reduced train time. For this training schedule, we propose Variable Target Rates, which treat the target rate as a moving average of the network utilization.

For each epoch, the target rate starts at a specific hyperparameter (to prevent collapse to 1) and then is allowed to change according to the batch's activation rate. The two simplest possibilities for the update step are: 1) moving average, and 2) exponential moving average.

S2.3 Granularity

For more granularity, we consider gating blocks of filters separately. In this case, we assume that $\mathcal{F}$ has N filters, i.e., $\mathcal{F}(x_{l-1})$ has dimension $(W \times H \times N)$. Then we write

$$x_l = x_{l-1} + \begin{bmatrix} z_{l,1}\mathcal{F}_{l,1}(x_{l-1}) \\ z_{l,2}\mathcal{F}_{l,2}(x_{l-1}) \\ \vdots \\ z_{l,n}\mathcal{F}_{l,n}(x_{l-1}) \end{bmatrix}$$

where $\mathcal{F}_{l,i}$ has dimension $(W \times H \times (N/n))$.

We note that it is not essential that the layers $\mathcal{F}$ are residual; we consider also dropping the added x_{l-1} on the right-hand side of these equations.

S2.4 Additional Early Exit Details

For our early exit classifiers, we use the same classifiers as [14]. For the gate structure, we use a stronger version of the gate described by [43]. The gates are comprised of the following: a 3×3 convolutional layer with stride of 1 and padding of 1 which takes the current state of the model and outputs 128 channels, a BatchNorm, another 3×3 convolutional layer with stride of 1 and padding of 1 which outputs 128 channels, a BatchNorm, a 4×4 average pool, a linear layer, and then finally a GumbleSoftmax.

S3 Observations

Observation S3.1 *A layer with activation of p can only affect the final accuracy by p.*

Observation S3.2 *Given a set of layers with activation $\mathbf{p}$, we can apply the Inclusion-Exclusion principle to get an upper bound for the amount these layers can affect the final accuracy of the network.*

So for example, if Layer 1 and Layer 2 both run with probability p but always co-occur (activate at the same time), then the set of Layer 1 and Layer 2 can only affect final accuracy by p.

We use these observations to motivate an explanation for mode collapse in the data-independent per-batch case. Consider a network with only two layers and the restriction that, on expectation, only one layer should be on. Then let p be the probability that layer 1 is on. Intuitively, if $p \notin \{0, 1\}$, then we are in a high entropy state where the network must deal with a large amount of uncertainty regarding which layers will be active. Furthermore, some of the work training each individual layer will be wasted during inference some percentage of the time since that layer will be skipped with non-zero probability. More precisely:

Observation S3.3 *Consider a network with two layers with data-independent probability p_1 and p_2 of being on. The network is then given a hard capacity of 1 (ie. one layer can be on, or each layer can be on half the time). Let a_1 be the expected accuracy of a one-layer network, a_2 be the expected accuracy of a two-layer network. Then if the network is given a hard capacity of 1, in order for $p_1 \notin \{0, 1\}$, we need that $\frac{a_2}{a_1} \geq 2$.*

Since the network is given a hard capacity of 1, then we are only learning a single parameter p_1 since $p_2 = 1 - p_1$. Let $p = p_1$. Then $p(1-p)$ is the probability that both layers will be on and also the probability that both layers will be off. Note that the network has a strict upper bound on accuracy of $1 - p(1-p)$ since with probability $p(1-p)$ none of the layers will activate and no output will be given.

Then the expected accuracy of the network for any probability $p \in [0, 1]$ is $(1-2p+2p^2)a_1+p(1-p)a_2$, note that for $p = 0, 1$ the accuracy is simply a_1. For a value $p \in (0, 1)$ to be better than $p \in \{0, 1\}$, we need

$$\begin{aligned} a_1 &< (1 - 2p + 2p^2)a_1 + p(1-p)a_2 \\ \frac{-2p^2 + 2p}{p(1-p)} &< \frac{a_2}{a_1} \\ 2 &< \frac{a_2}{a_1} \end{aligned}$$

S4 ImageNet Results

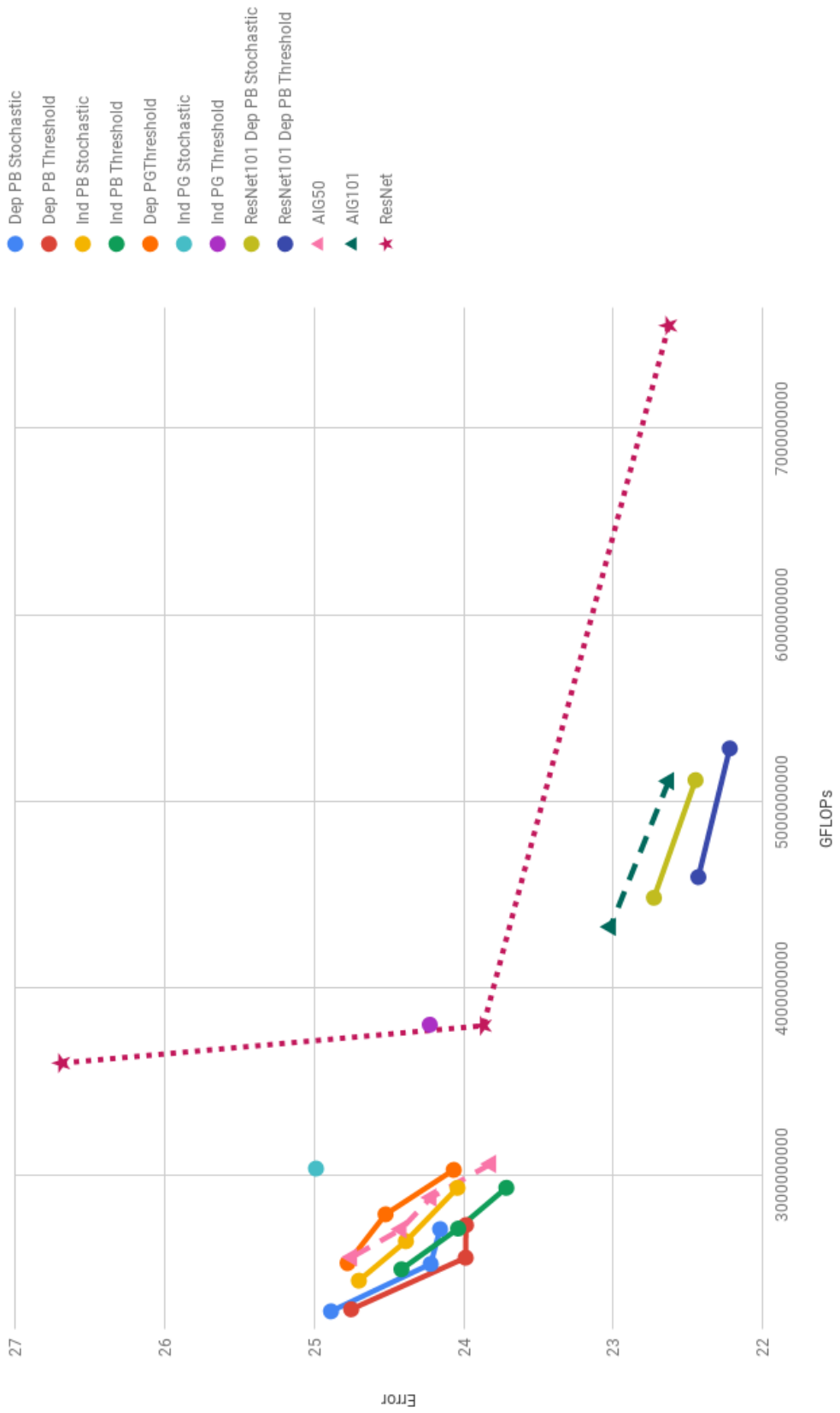
We report all the data collected on ImageNet using the different gate strategies (independent, dependent), target loss strategies (per-batch, per-gate), and inference time strategies (threshold, always-on, stochastic, ensemble). Note that we try to include AIG for reference whenever it's a fair comparison.

Also of note, for the ensemble technique we also include data from Snapshot Ensembles: Train 1, Get M For Free [15]. Note that using the stochastic networks, we outperform their ensemble technique. Also note that their technique is orthogonal to ours, so both could be utilized to identify an even better ensemble.

In general, we observe that unsurprisingly, ensemble has the highest performance in terms of error; however, this requires multiple forward passes through the network, so the performance gain is somewhat offset by the inference time required. We also observe that threshold generally outperforms stochastic. This roughly makes sense if you consider stochastic inference as drawing a sample from a inference distribution - in this interpretation, threshold at .5 basically acts as an argmax. In addition to the improvement in performance, for the per-batch cases, threshold also tends to increase the number of activations.

For all ImageNet results, we used the pretrained models provided by TorchVision[1].

[1]The model links are provided here: https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py


Dep PB Stochastic
Dep PB Threshold
Ind PB Stochastic
Ind PB Threshold
Dep PGThreshold
Ind PG Stochastic
Ind PG Threshold
ResNet101 Dep PB Stochastic
ResNet101 Dep PB Threshold
AIG50
AIG101
ResNet
Error
27
26
25
24
23
22
GFLOPs
3000000000
4000000000
5000000000
6000000000
7000000000

Technique	Threshold GFlops	Threshold Prec@1	Threshold Prec@5	Threshold Acts
Dep PB @ tr=.6	2732360786	23.986	7.14	0.6929
Dep PB @ tr=.5	2557646188	23.988	7.058	0.6429
Dep PB @ tr=.4	2281151219	24.754	7.562	0.5637
Ind PB @ tr=.6	2932012776	**23.716**	**6.934**	0.75
Ind PB @ tr=.5	2713646824	24.038	7.294	0.6875
Ind PB @ tr=.4	2495280872	24.418	7.478	0.625
Dep PB ConstQuad Anneal	2717097006	23.98	6.982	0.6885
Dep PG @ tr=.6	3027899835	24.068	7.16	0.7726
Dep PG @ tr=.5	2789914344	24.524	7.272	0.69
Dep PG @ tr=.4	2528475066	24.778	7.57	0.6098
Ind PG @ tr=.5	3805476584	24.228	7.168	1
AIG @ tr=.7	3060000000	23.82	7.08	-
AIG @ tr=.6	2880000000	24.22	7.21	-
AIG @ tr=.5	2710000000	24.42	7.41	-
AIG @ tr=.4	2560000000	24.75	7.61	-
Res101 Dep PB @ tr=.6	5284996031	**22.222**	6.28	0.69
Res101 Dep PB @ tr=.5	4596138799	22.432	6.336	0.594
AIG101 @ tr=.5	5110000000	22.62	**6.26**	-
AIG101 @ tr=.3	4330000000	23.02	6.58	-

Technique	AlwaysOn Prec@1	AlwaysOn Prec@5	AlwaysOn Acts
Dep PB @ tr=.6	23.968	7.134	1
Dep PB @ tr=.5	24.362	7.248	1
Dep PB @ tr=.4	25.25	7.826	1
Ind PB @ tr=.6	**23.75**	**6.944**	1
Ind PB @ tr=.5	24.08	7.362	1
Ind PB @ tr=.4	24.46	7.448	1
Dep PB ConstQuad Anneal	23.932	6.994	1
Dep PG @ tr=.6	23.864	6.968	1
Dep PG @ tr=.5	24.406	7.19	1
Dep PG @ tr=.4	24.734	7.536	1
Ind PG @ tr=.5	24.228	7.168	1
ResNet 50	23.87	7.12	1
Res101 Dep PB @ tr=.6	**22.42**	**6.222**	1
Res101 Dep PB @ tr=.5	23.276	6.708	1
ResNet 101	22.63	6.45	

Stochastic Technique	GFlops	Prec@1 Mean	Prec@1 StdDev	Prec@5 Mean	Prec@5 StdDev	Acts Mean	Acts StdDev
Dep PB @ tr=.6	2710541198	24.1592	0.047	7.24	0.0055	0.6862	0.0002
Dep PB @ tr=.5	2524273294	24.222	0.0358	7.18	0.0278	0.6321	0.0002
Dep PB @ tr=.4	2270347346	24.8892	0.0224	7.6284	0.096	0.5596	0.0001
Ind PB @ tr=.6	2932012776	24.0428	0.0246	7.1056	0.0282	0.7222	0.0001
Ind PB @ tr=.5	2646139684	24.3876	0.0496	7.46	0.0299	0.6676	0.0001
Ind PB @ tr=.4	2434652927	24.7024	0.0753	7.634	0.0291	0.6068	0.0002
Dep PB ConstQuad Anneal	2638223224	24.4576	0.0753	7.2944	0.0403	0.66631	0.0003
Dep PG @ tr=.6	2988683827	24.472	0.0741	7.3616	0.0508	0.7549	0.0002
Dep PG @ tr=.5	2734859869	24.9176	0.052	7.4972	0.0231	0.6797	0.0002
Dep PG @ tr=.4	2510905649	25.2476	0.0802	7.8084	0.047	0.6136	0.0002
Ind PG @ tr=.5	3035128056	24.9892	0.05	7.7064	0.0345	0.7681	0.0005
AIG @ tr=.7	3060000000	23.82	-	7.08	-	-	-
AIG @ tr=.6	2880000000	24.22	-	7.21	-	-	-
AIG @ tr=.5	2710000000	24.42	-	7.41	-	-	-
AIG @ tr=.4	2560000000	24.75	-	7.61	-	-	-
Res101 Dep PB @ tr=.6	5115483937	**22.4508**	0.077	6.2768	0.0346	0.6665	0
Res101 Dep PB @ tr=.5	4485480286	22.7288	0.0229	6.4552	0.0206	0.5791	0.0001
AIG101 @ tr=.5	5110000000	22.62		**6.26**			
AIG101 @ tr=.3	4330000000	23.02		6.58			

Technique	Ensemble Prec@1	Ensemble Prec@5
Dep PB @ tr=.6	23.952	7.14
Dep PB @ tr=.5	24.014	7.076
Dep PB @ tr=.4	24.76	7.558
Ind PB @ tr=.6	**23.814**	**6.97**
Ind PB @ tr=.5	24.116	7.32
Ind PB @ tr=.4	24.448	7.506
Dep PB ConstQuad Anneal	23.952	7.09
Dep PG @ tr=.6	23.844	7.1
Dep PG @ tr=.5	24.342	7.188
Dep PG @ tr=.4	24.62	7.486
Ind PG @ tr=.5	24.376	7.328
ResNet 50	23.87	7.12
Snapshot [15]	23.96[2]	
Res101 Dep PB @ tr=.6	**22.186**	**6.162**
Res101 Dep PB @ tr=.5	22.42	6.31
ResNet 101	22.63	6.45

S5 CIFAR10 Results

S5.1 CIFAR10 Performance

We report all the data collected on CIFAR10 using the different gate strategies (independent, dependent), target loss strategies (per-batch, per-gate). We report only the numbers for the stochastic inference time technique. We used CIFAR10 as a faster way to explore the space of parameters and combinations and as such have a more dense sweep of the combination and parameters. Note that for CIFAR10, we did not use a pretrained model; the entire model is trained from scratch.

In general, we found that for a wide set of parameters per-batch outperforms per-gate. This includes independent per-batch outperforming dependent per-gate.

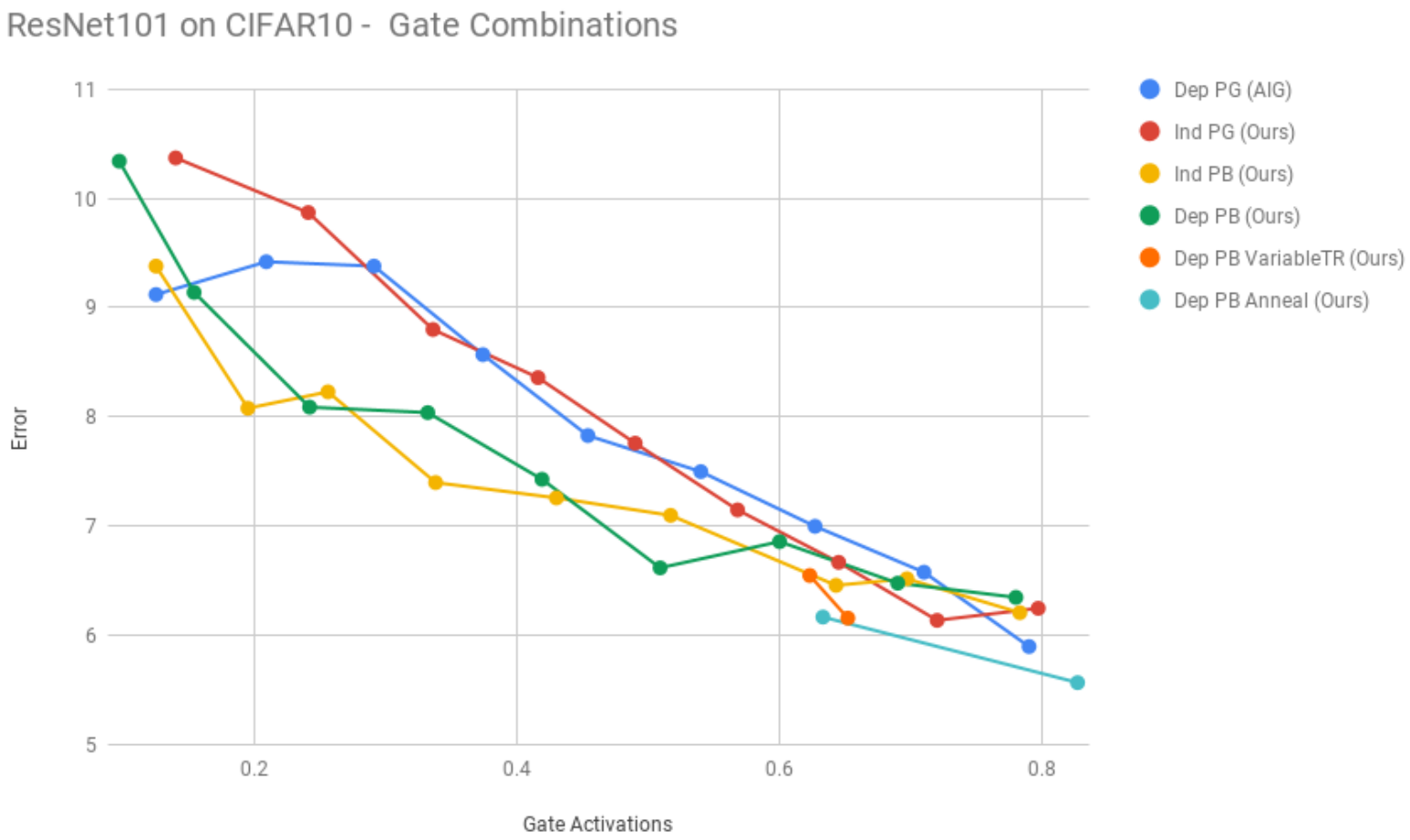
The only exception to this is very high and very low target rates. However we note that at very high target rates, the accuracy of per-batch can be recovered through annealing. We attribute this to the fact that for CIFAR10 we train from scratch. Since the model is completely blank for the first several epochs, the per-batch loss can lower activations for any layers while still improving accuracy. In other words, at the beginning, the model is so inaccurate that any training on any subset of the model will result in a gain of accuracy; so when training from scratch, the per-batch loss will choose the layers to decrease activations for greedily and sub-optimally.

One surprising result is that independent per-gate works at all for a wide range of target rates. This suggests that the redundancy effect described in [18] is so strong that the gates can be kept during inference time. This also suggests that at least for CIFAR10, most of the gains described in [43] were from regularization and not from specialization.

We also report some variable rate target rates. We note that these tend to outperform the quadratic loss on a constant target rate. We believe that this is because variable target rate allows the optimizer to take the easiest and farther path down the manifold. We note that some of the variable target rates that worked on CIFAR10 did not work on ImageNet; namely, variable target rates which updated the target to previous mean quickly (within 5 epochs) lead to mode collapse to 1 for all gates. We attribute this to the much larger amount of training data for ImageNet and increased complexity of the task. However, both annealing and variable target rates merit further experimentation and research to truly understand how they perform on different datasets and with different training setups (from scratch vs from pretrained).

[2]Their ResNet50 baseline had Prec@1 of 24.01. Note that their improvement over the baseline (0.07) is on the same order as ours (0.03). Also these methods are orthogonal and can be used together.

Technique	Best Error	Final Error	Activations
Dep PG @tr=.0	9.12	9.26	0.125
Dep PG @tr=.1	9.42	9.8	0.209
Dep PG @tr=.2	9.38	9.4	0.291
Dep PG @tr=.3	8.57	8.88	0.374
Dep PG @tr=.4	7.83	8.3	0.454
Dep PG @tr=.5	7.5	7.5	0.54
Dep PG @tr=.6	7	7.14	0.627
Dep PG @tr=.7	6.58	6.82	0.71
Dep PG @tr=.8	5.9	6.08	0.79
Dep PB @tr=.0	10.34	10.5	0.097
Dep PB @tr=.1	9.14	9.36	0.154
Dep PB @tr=.2	8.09	8.45	0.242
Dep PB @tr=.3	8.04	8.48	0.332
Dep PB @tr=.4	7.43	7.8	0.419
Dep PB @tr=.5	6.62	7.04	0.509
Dep PB @tr=.6	6.86	7.86	0.6
Dep PB @tr=.7	6.48	6.83	0.69
Dep PB @tr=.8	6.35	6.35	0.78
Ind PG @tr=.0	10.37	10.75	0.14
Ind PG @tr=.1	9.87	10.27	0.241
Ind PG @tr=.2	8.8	9.15	0.336
Ind PG @tr=.3	8.36	8.75	0.416
Ind PG @tr=.4	7.76	7.95	0.49
Ind PG @tr=.5	7.15	7.43	0.568
Ind PG @tr=.6	6.67	6.98	0.645
Ind PG @tr=.7	6.14	6.54	0.72
Ind PG @tr=.8	6.25	6.82	0.797
Ind PB @tr=.0	9.38	9.52	0.125
Ind PB @tr=.1	8.08	8.43	0.195
Ind PB @tr=.2	8.23	8.41	0.256
Ind PB @tr=.3	7.4	7.65	0.338
Ind PB @tr=.4	7.26	7.71	0.43
Ind PB @tr=.5	7.1	7.39	0.517
Ind PB @tr=.6	6.46	6.92	0.643
Ind PB @tr=.7	6.52	6.86	0.697
Ind PB @tr=.8	6.21	6.23	0.783
Dep PB VariableTarget Quad @ tr=.8	6.16	6.43	0.652
Dep PB VariableTarget ConstQuad @ tr=.8	6.55	6.72	0.623
Dep PB Anneal $\Rightarrow$ 1.0; 0.95; 0.9; 0.85; 0.8	5.57	6	0.827
Dep PB Anneal $\Rightarrow$ 0.8; 0.7; 0.6	6.17	6.66	0.633


ResNet101 on CIFAR10 - Gate Combinations
Error
Gate Activations
11
10
9
8
7
6
5
0.2
0.4
0.6
0.8
Dep PG (AIG)
Ind PG (Ours)
Ind PB (Ours)
Dep PB (Ours)
Dep PB VariableTR (Ours)
Dep PB Anneal (Ours)

S5.2 CIFAR10 Activation Graphs

Here we provide graphs of the activation rates for each layer over time. This demonstrates that on CIFAR10, each layer does not instantly mode collapse to its eventual activation rate; the activation rates can change throughout training time.

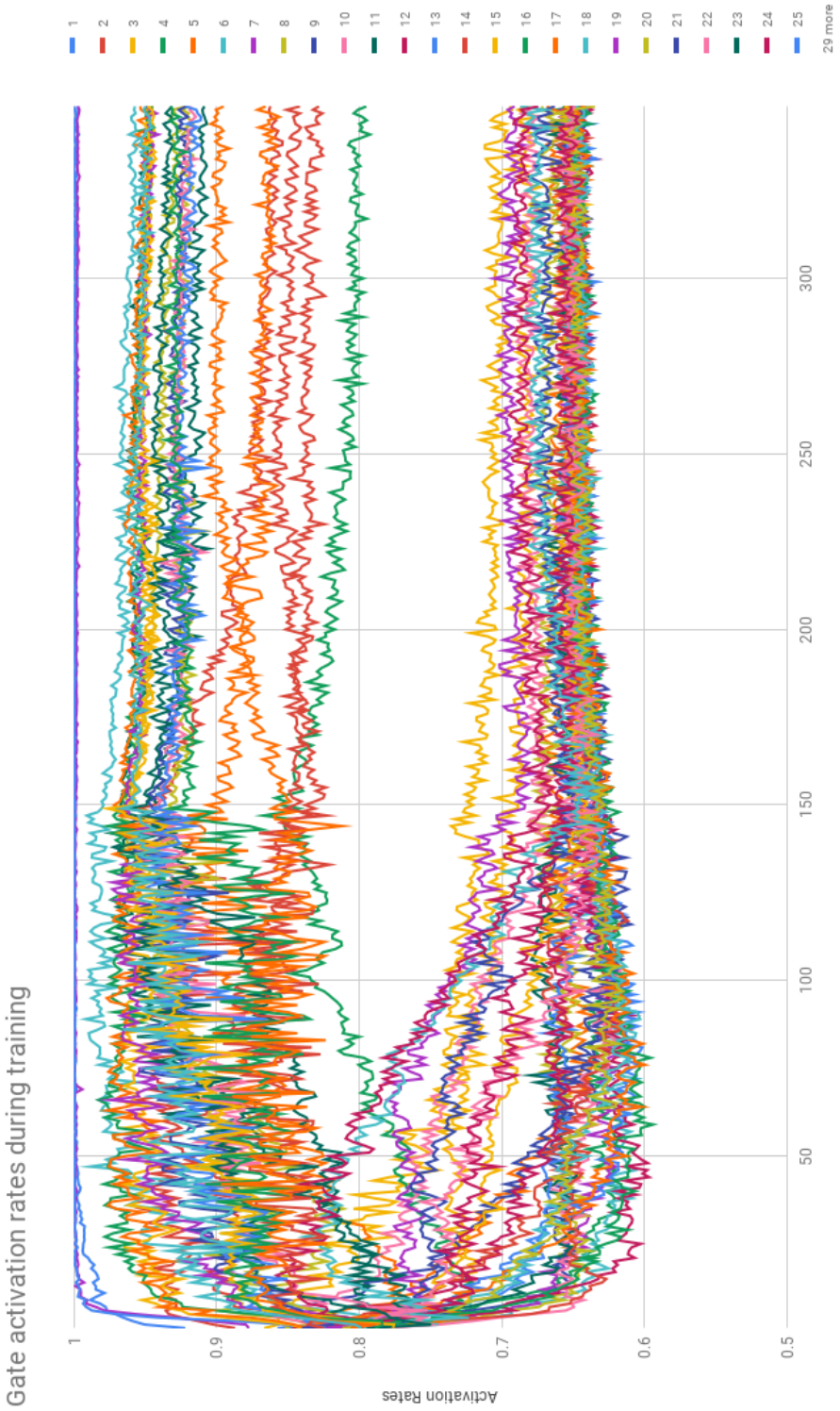


Figure 10: Data-dependent per-batch with target rate of 0.8

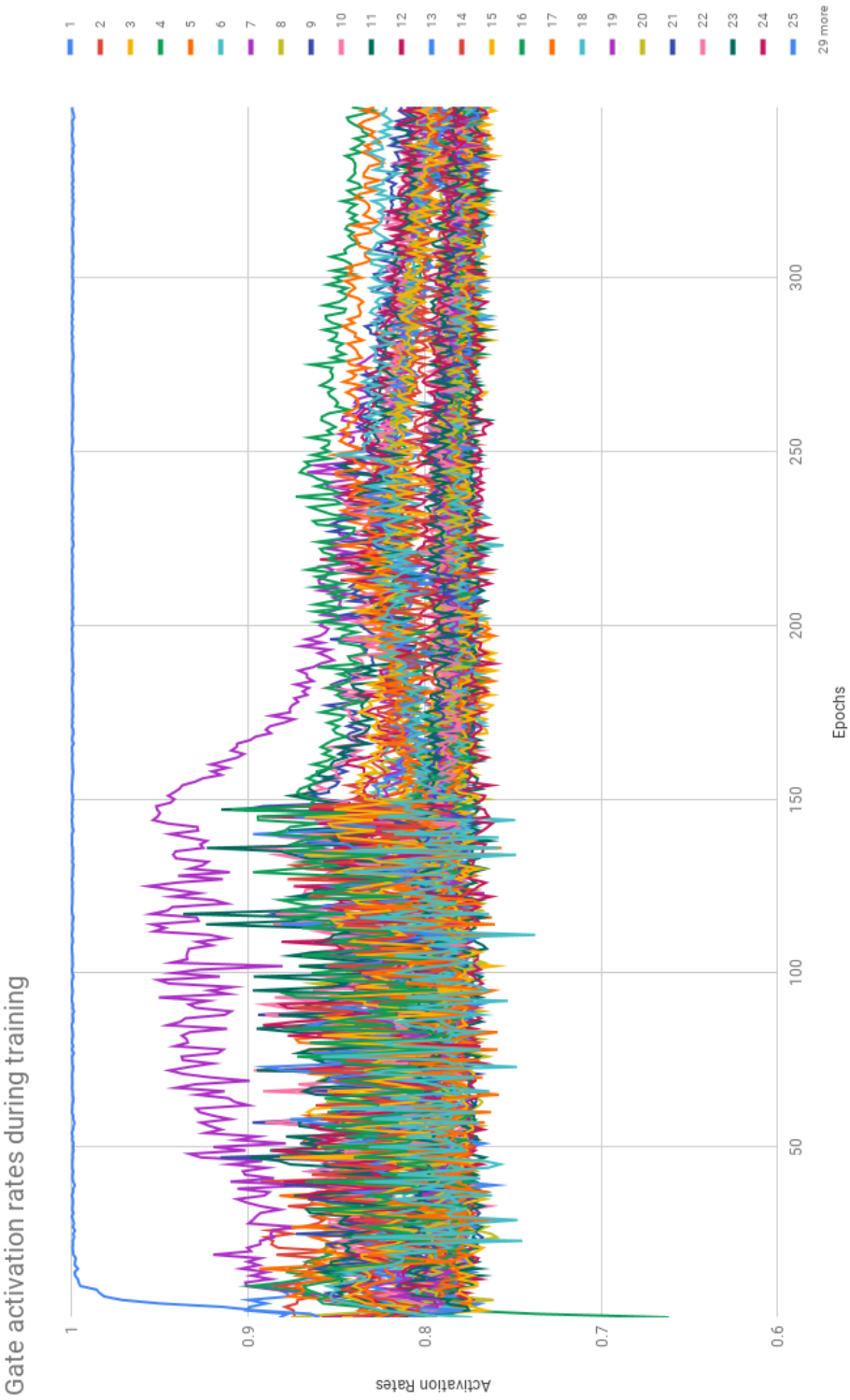


Figure 11: Data-dependent per-gate with target rate of 0.8

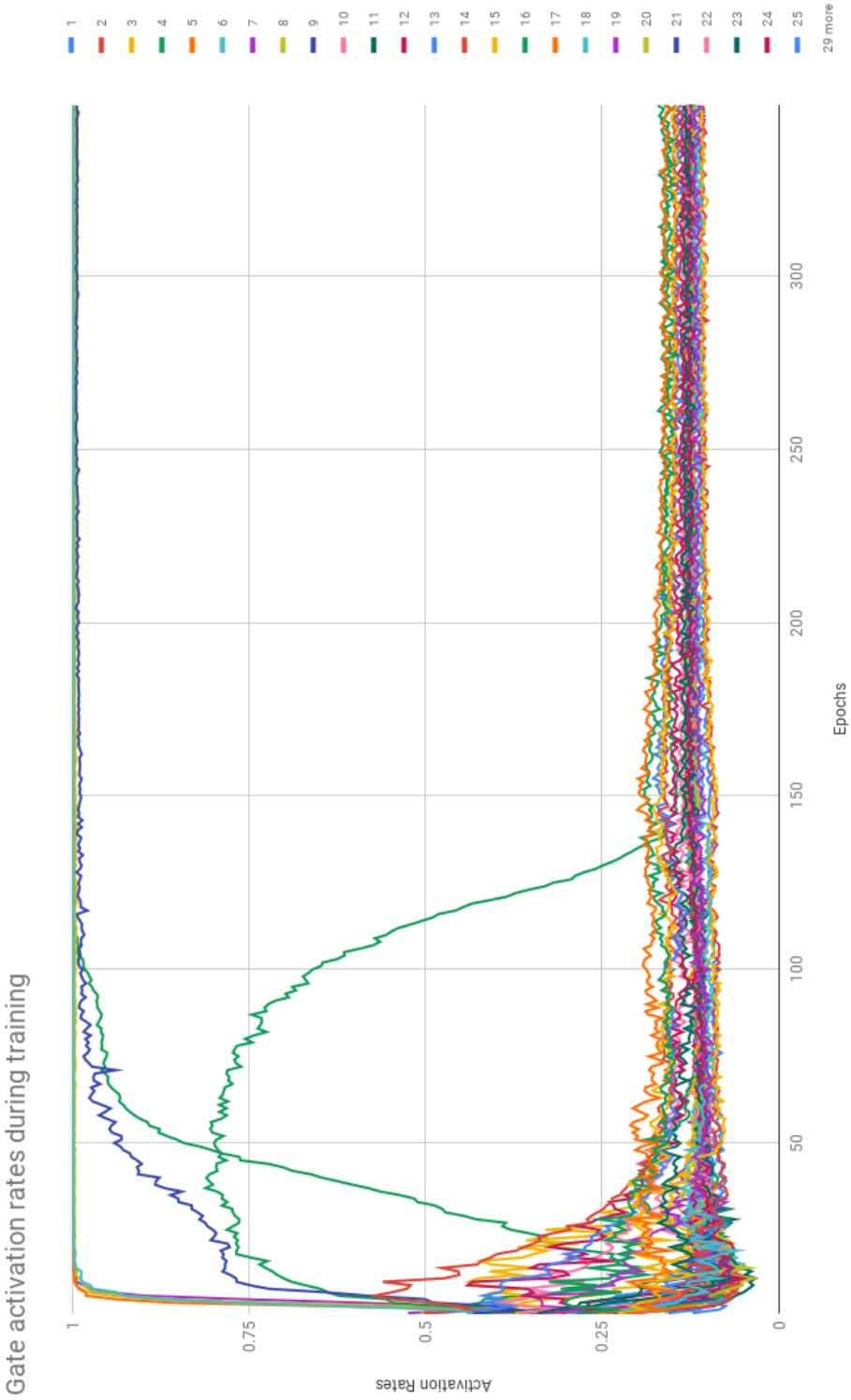


Figure 12: Data-dependent per-batch with target rate of 0.2

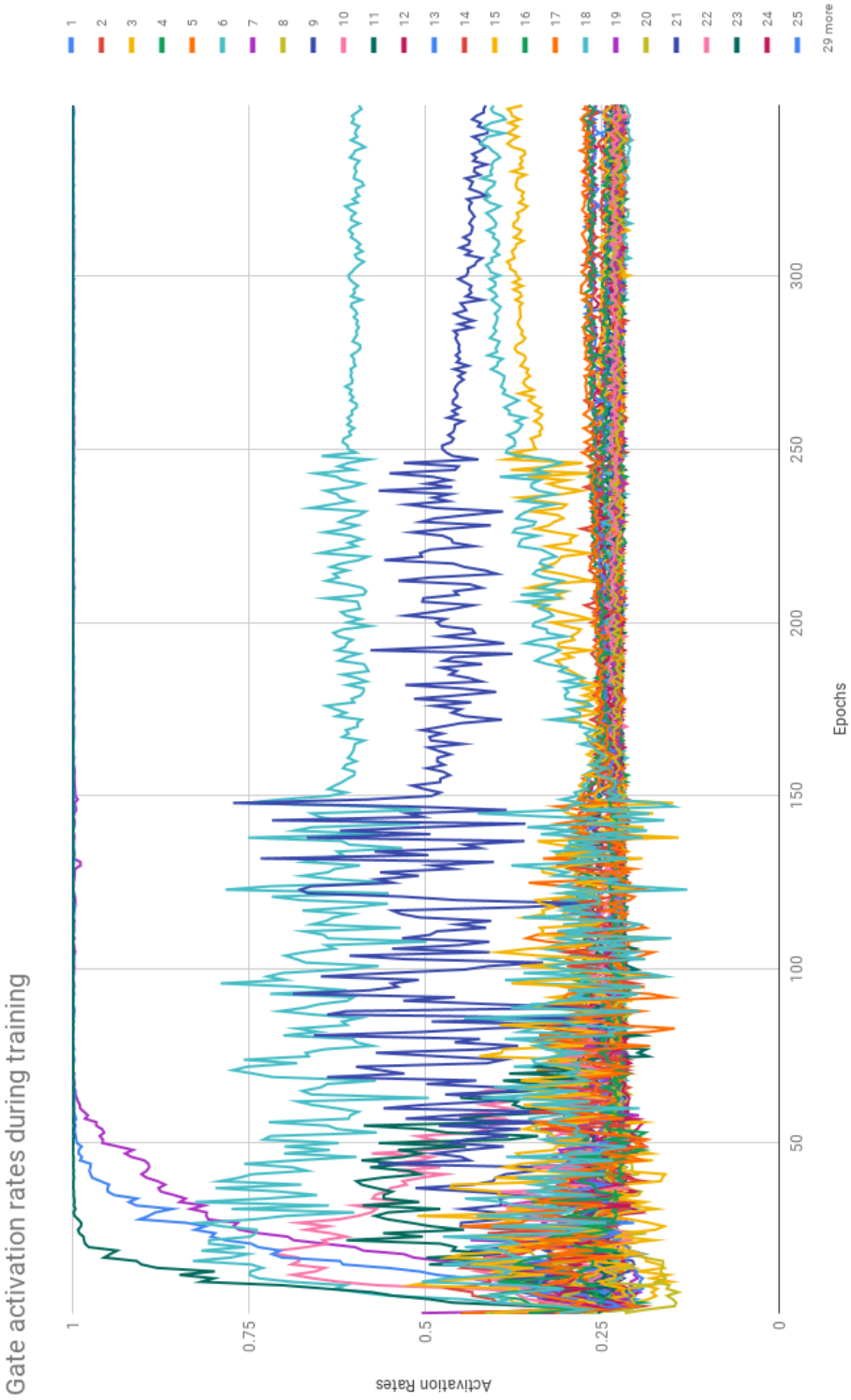


Figure 13: Data-dependent per-gate with target rate of 0.2