TRIANGULAR DROPOUT: VARIABLE NETWORK WIDTH WITHOUT RETRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

One of the most fundamental design choices in neural networks is layer width: it affects the capacity of what a network can learn and determines the complexity of the solution. This latter property is often exploited when introducing information bottlenecks, forcing a network to learn compressed representations. However, such an architecture decision is typically immutable once training begins; switching to a more compressed architecture requires retraining. In this paper we present a new layer design, called Triangular Dropout, which does not have this limitation. After training, the layer can be arbitrarily reduced in width to exchange performance for narrowness. We demonstrate the construction and potential use cases of such a mechanism in three areas. Firstly, we describe the formulation of Triangular Dropout in autoencoders, creating an MNIST autoencoder with selectable compression after training. Secondly, we add Triangular Dropout to VGG19 on ImageNet, creating a powerful network which, without retraining, can be significantly reduced in parameters with only small changes to classification accuracy. Lastly, we explore the application of Triangular Dropout to reinforcement learning (RL) policies on selected control problems, showing that it can be used to characterize the complexity of RL tasks, a critical measurement in multitask learning and lifelong-learning domains.

1 INTRODUCTION

In this paper we describe Triangular Dropout, a novel dropout (Srivastava et al., 2014) technique that imparts a powerful property to fully-connected layers: the layer can be reduced in width after training to select a desired size of representation. This exposes a trade-off between performance and network width that can be varied without retraining the network. Additionally, constructing and applying Triangular Dropout is straightforward. Triangular Dropout amounts to masking minibatches during training with a binary lower triangular matrix with zeros above the diagonal. After training, the desired layer size can be specified or relevant weights copied into a physically smaller network.

In cases for which the size of representation affects the purpose of the network, Triangular Dropout can be used to approximately train a superset of these representations, such that the exact architecture can be decided after training. For example, we demonstrate autoencoders trained with Triangular Dropout that allow for the desired level of compression to be decided after training.

Architecturally, Triangular Dropout represents a shift towards models that can be downsized to fit the needs of a particular application. For example, a wide model could be trained once on a computer cluster, and narrower versions would be immediately available for use on personal machines, mobile devices, IoT devices, or other embedded platforms. An image processing pipeline could have its frame-rate or feature descriptor length altered as needed, perhaps to process more samples in exchange for lower performance per sample.

We discuss related works to Triangular Dropout in Section 2, and formally define our mechanism in Section 3. We then investigate some applications of Triangular Dropout in unsupervised learning of latent encodings (Section 4), supervised learning in larger models (Section 5), and reinforcement learning for locomotion control problems (Section 6). Sections 7 and 8 further discuss the characteristics of Triangular Dropout and suggest potential future research directions.

2 BACKGROUND

Network width is perhaps most critical in the domain of autoencoding (Rumelhart et al., 1986), in which the width of the encoding layer (along with its activation) describes the amount of information retained in an encoded sample. The smaller the latent space (narrower the encoding layer), the more compressed the learned representation, and therefore, the more lossy the compression performed by the network. There are countless variations of autoencoders which seek to improve on this compression ratio (Theis et al., 2017) or introduce additional properties to the latent variables (Vincent et al., 2008; Rifai et al., 2011).

The degree to which the latent variables are independent is the subject of disentanglement research, which introduces further properties into the latent variables. β-VAE (Higgins et al., 2017) demonstrates that the gaussian distributions in a VAE (Kingma & Welling, 2014) can be made increasingly orthogonal in exchange for reconstruction accuracy. The independence and ordering of learned features can also be enforced manually by methods such as PCAAE (Pham et al., 2020), which learns one latent variable at a time, progressively widening the latent layer to learn additional features via additional training. A comparison to PCAAE is discussed in detail in Section 4.

Autoencoding makes use of a well-known tradeoff between network size and capability to encourage compression, a relationship that holds in other domains (Amodei & Hernandez, 2019; Kaplan et al., 2020; McCandlish et al., 2018). This is perhaps most well known with regards to ImageNet (Deng et al., 2009), in which historically a clear correlation is seen between network size and performance (summarized nicely in Figure 1 of Tan & Le (2020)). More recently, incredibly large NLP models have been created to serve as general language tools (Devlin et al., 2019; Brown et al., 2020). A clear downside of increasingly large models is that they are also increasingly inaccessible and difficult to deploy: They are often followed by work looking to distill these models into smaller footprints (less parameters), for example SqueezeNet (Iandola et al., 2016) and EfficientNet (Tan & Le, 2020) in the image processing domain. This is especially needed if the network will be deployed on a mobile device, embedded platform, or simply needs high data throughput (Howard et al., 2017). Section 5 examines this trade-off in more detail.

Network size selection has additional repercussions in the deep reinforcement learning (DRL) domain. Works like amplified control (Paulhamus et al., 2019) and robust predictable control (Eysenbach et al., 2021) seek compressed policies to perform tasks, using information bottlenecking (Tishby et al., 2001) to enforce this compression. Robust predictable control demonstrates that compression creates policies that are more risk-averse and potentially more transferable. In DRL for multitask learning and lifelong-learning, and the ability to measure the complexity of a policy or task is an open problem (see the implications of task complexity discussed in Srivastava et al. (2019)). In Section 6, we explore applications of Triangular Dropout to this area.

Finally, Triangular Dropout has many similarities to network pruning techniques (Blalock et al., 2020; Frankle & Carbin, 2018). However, the objective of Triangular Dropout is to create adjustable networks, not necessarily those that are as small as possible or even as performant as possible. It further differs from most pruning techniques in that it enables parameter removal by organizing learned dense features, not by searching for removable parameters. We refer to removing layer nodes as "ablation," although this can also be thought of as "pruning" parameters relevant to a given layer output.

3 TRIANGULAR DROPOUT

Triangular Dropout is simple to implement in practice. For a fully-connected layer of width n, processing a minibatch x of B datapoints, the output y has size (B, n). During training, standard dropout (with probability p) multiplies the output by a random binary mask $\boldsymbol{M}$ of the same size as the output:

$$y = \boldsymbol{M}(\boldsymbol{x}^T \boldsymbol{W} + \boldsymbol{b}) \quad \text{where} \quad \boldsymbol{M} \sim Bernoulli(p)$$

where $\boldsymbol{W}$ and $\boldsymbol{b}$ are the layer weights and biases, respectively. Triangular Dropout has the same form, but the mask $\boldsymbol{M}$ is populated as a binary lower triangular matrix instead of a randomly sampled one:

$$y = \boldsymbol{M}_{Tri}(\boldsymbol{x}^T\boldsymbol{W} + \boldsymbol{b}) \quad \text{where} \quad \boldsymbol{M}_{Tri} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix}$$

This is visualized in Figure 1. In the case where $B > n$, the rows of $\boldsymbol{M}_{Tri}$ are simply repeated to reach the desired size. We explicitly limit our experiments to cases where B is a multiple of n, which does not particularly hinder training. For example, if $n = 2$ and $B = 4$, $\boldsymbol{M}_{Tri}$ would have the following form:

$$n = 2, \quad B = 4, \quad \boldsymbol{M}_{Tri} = \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}$$

The implications of such a mask can be viewed in different ways. Considering the rows of the output y, the mask creates a set of datapoints which have been processed with the full range of layer widths from 1 to n. If the batch size is equal to n, the ith row represents a training sample in which the effective layer width is i. Training now amounts to something like a multitask learning problem, in which the "tasks" are processing data with varying network widths. We have an even distribution of data from all these tasks, and the loss function reflects training over this set of architectures (widths):

$$y = \begin{pmatrix} y_{1,1} & 0 & 0 & \dots \\ y_{1,2} & y_{2,2} & 0 & \dots \\ y_{1,3} & y_{2,3} & y_{3,3} & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \begin{array}{l} \rightarrow \text{ effective width 1} \\ \rightarrow \text{ effective width 2} \\ \rightarrow \text{ effective width 3} \\ \\ \end{array}$$

Taking this multitask view, we do not re-scale the layer outputs as in normal dropout; we want the downstream network to be able to handle any of the "tasks" (any of the potential widths of this layer).

However, the "tasks" in this view are not independent. Examining the columns of the output, the jth column represents the output of the jth layer node, masked out for the first j rows. Because the mask is triangular, all of the samples which are non-zero at column j are necessarily also non-zero at columns 1 through $j - 1$. That is, parameters relevant to the jth node are only updated for samples in which the previous $j - 1$ nodes (smaller widths) are also updated. This is not true for subsequent nodes: nodes $j + 1$ and onwards may or may not be masked when node j is utilized.

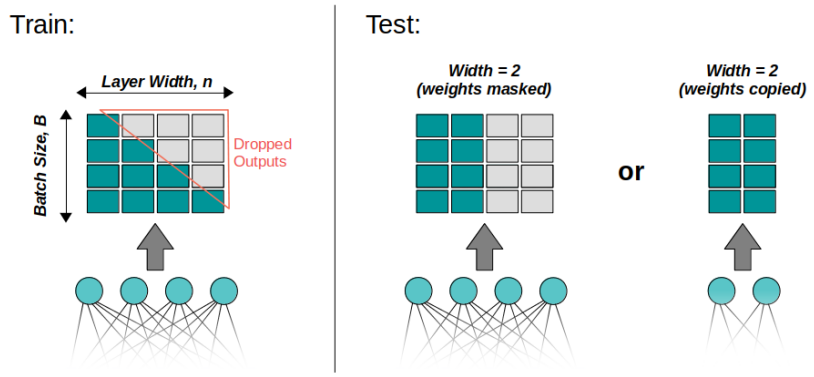

Figure 1: Diagram of Triangular Dropout application in training (Left) and used during test or deployment (Right). During test, a layer width can be simulated by masking the layer output, or realized by copying relevant weights into a narrower architecture.

This cascading dependence is what masks Triangular Dropout work. The parameters related to node j can be learned with a guarantee that the proceeding $j-1$ outputs in terms of width are also present. However, they cannot rely as heavily on outputs $j+1$ and beyond, and are essentially restricted from fully co-optimizing with these outputs.

During test time, it is up to the user to decide on the masking strategy. Providing a mask filled with ones makes the layer approximately equivalent to a typical fully-connected layer, or masking out the last $N-j$ nodes causes the layer to behave as though it has width j. To permanently ablate the layer down to some desired width, the relevant model parameters (portions of $\boldsymbol{W}$ and b from this layer and the next) simply need to be copied into a smaller model, as shown in Figure 1.

4 Variable Compression in Autoencoders

To further detail the properties of Triangular Dropout and compare it to alternative methods, we demonstrate the construction of an autoencoder with a selectable latent size. We train an autoencoder on MNIST (Deng, 2012) with a latent size n, and use Triangular Dropout on the latent layer. After training, we can select any level of compression $\leq n$ and receive both a valid encoding and reconstruction.

4.1 Baselines

We compare results quantitatively (reconstruction loss) and qualitatively (visualization of reconstruction) to two baselines: n autoencoders that are trained for specific latent sizes 1 through n, and PCAAE, which trains an autoencoder of size n progressively. PCAAE trains one latent variable at a time and includes a correlation minimization loss term. PCAAE also trains a separate encoder for each latent variable, and a new decoder for each width. This means that, like the set of standard autoencoders, PCAAE requires n training runs.

Compared to standard autoencoders, our expectation is that Triangular Dropout must sacrifice some compression in exchange for the introduction of some one-way dependence among features (the first i features are not very dependent on the $i+1th$ feature). Conversely, we would not expect a typical autoencoder to have this property. The encoded features from a typical autoencoder, having no encouragement to be independent or disentangled, should only be meaningful when the full feature descriptor is retained.

We would expect that PCAAE further sacrifices reconstruction loss for an even more stringent independence among learned features. However, the features are independent, the reconstructions may still be meaningful when less important features are removed from the final model.

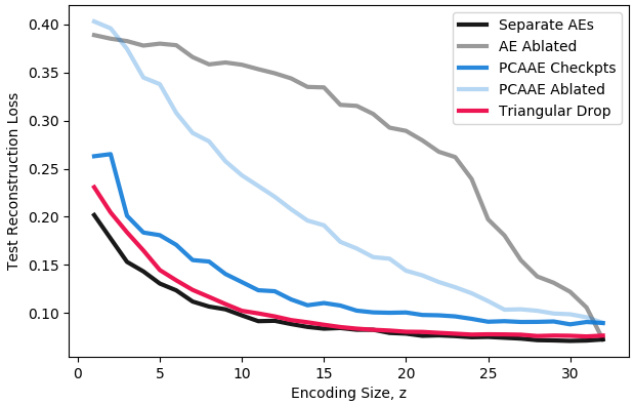

Figure 2: Reconstruction loss versus encoding size for a variety of methods. Triangular Dropout (red) is most similar to independent autoencoders trained at specific encoding sizes (black). PCAAE models produced during progressive training are shown in blue. Light grey and light blue show the reconstruction loss of the full-size standard autoencoder and PCAAE models, respectively, when the latent space is ablated down to a given size.

4.2 Experiments and Results

A convolutional autoencoder with latent size 32 was trained with PCAAE and Triangular Dropout to reconstruct MNIST digits. The smaller widths of PCAAE were saved off during training. Additionally, standard autoencoders were trained for sizes 1 through 32. We examine the reconstruction loss of the models as the latent width is increased during training and decreased via ablation of the final 32-width model. Triangular Dropout falls into the latter category by design, but has performance similar to the checkpoint models trained at specific sizes. These results are shown in Figure 2. Full architecture and training details are available in the Appendix.

Notably, our hypotheses regarding feature independence vs reconstruction loss hold true: for a given latent size z, the most performant model (without feature ablation) is a standard autoencoder trained at that width, the least performant is PCAAE which introduces a correlation loss, and somewhere in-between is Triangular Dropout. The baseline models with ablated encodings do not have comparable reconstruction losses unless the ablation ois very minor (most latent features retained). Triangular Dropout, trained only at size 32, can be ablated down to very small sizes while only performing slightly worse than individual autoencoders at those sizes. Triangular Dropout roughly encompasses the superset of these autoencoders, while only requiring a single training run.

Figure 3 shows the qualitative reconstruction in each of these cases, for selected latent widths. The standard autoencoders trained at increasing sizes (Figure 3, A) show increasing reconstruction fidelity, with recognizable digits appearing around 4-6 latent features. When removing features from the size 32 autoencoder (Figure 3, B), the encoding are clearly corrupted unless most features are present. PCAAE (Figure 3, C and D) has only similar reconstructions to the standard autoencoders once sufficient features are present, and can also content with significant ablations. Finally, Triangular Dropout (Figure 3, E) has similar reconstructions to the individual autoencoders, even though its latent size is determined by ablation. A user of this model could select some latent size approximately between 6 and 32 and expect it to yield recognizable digits.

5 Parameter Reduction in Large Networks

The results of our MNIST autoencoding experiments demonstrate the primary property of a Triangular Dropout layer: it can be reduced in width after training and still provide meaningful output. We now consider the ramifications of such a mechanism in the supervised learning domain.

Many powerful models for image feature extraction and natural language are so large that they cannot be easily utilized by the average practitioner at their full sizes. To make them more accessible and study the effects of scale, multiple specific sizes are often trained. For example, VGG (Simonyan & Zisserman, 2015) is often used at one of 4 sizes (VGG11, VGG13, VGG16, and VGG19), and so is GPT-2 (Radford et al., 2019) (sizes small, medium, large, and extra large). We envision Triangular Dropout as a stepping stone towards single master models that can be reduced in size arbitrarily to fit the needs of any given user. For example, a practitioner may decide that they only need k features to describe their data, or that their hardware limitations can only accommodate y parameters. In embedded or real-time systems, perhaps a much smaller version of a model is preferred in exchange for high data throughput.

To motivate this view, we re-train the fully connected classification portion of VGG19 on ImageNet using Triangular Dropout for the two hidden layers of size 4096. We then investigate the fidelity of the model when run at reduced widths. We concentrate on this portion of the model because the classifier head contains the majority of VGG's parameters (roughly 123 million out of 143 million in total).

5.1 Model Accuracy Findings

VGG19 retrained with Triangular Dropout does not achieve quite the accuracy of the original VGG19, but like out autoencoding experiments it is not far behind the standard model. VGG19 reported an ImageNet classification accuracy of 71.3 percent, while our model achieves 69.7 percent when not reduced in width. This small drop in performance is to be expected, and we do not believe it is large enough to significantly impact the usefulness of the model.

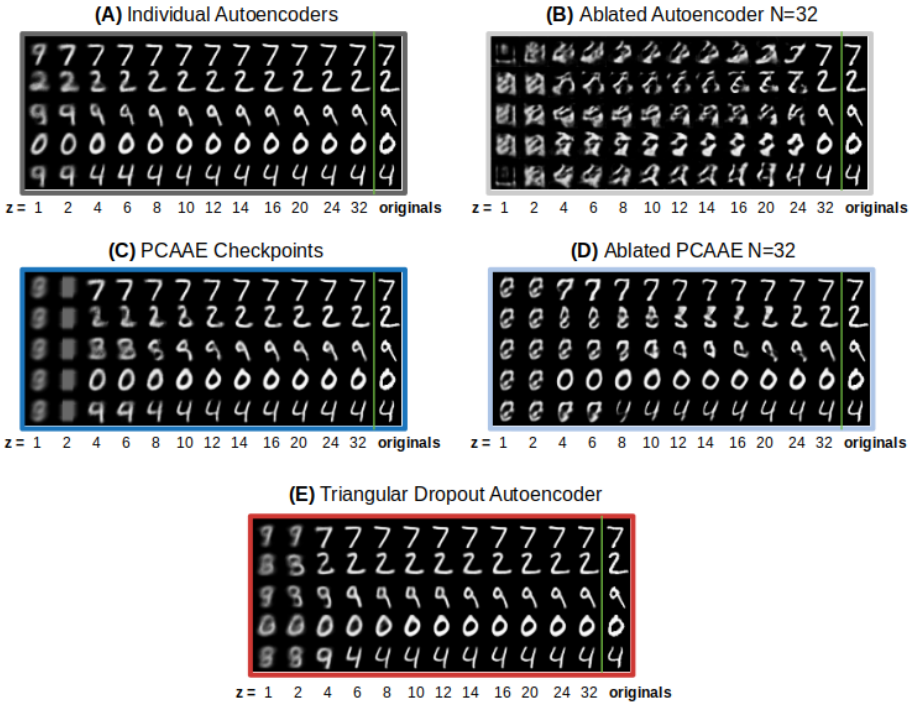

Figure 3: Qualitative analysis of MNIST reconstructions for all methods. Colored borders correspond to line color in Figure 2. **A:** Reconstructions from individual autoencoders at selected sizes, with original inputs on the right. **B:** Width 32 autoencoder with latent features ablated to desired size. **C:** Reconstructions from PCAAE checkpoint models produced by progressively training latent channels towards a full width of 32. **D:** Width 32 PCAAE with latent features ablated to desired size. **E:** Reconstructions from Triangular Dropout trained once at size 32, with smaller size encodings created via ablation.

Additionally, the success of the model with two layers of Triangular Dropout indicates that Triangular Dropout layers can be stacked in sequence to create a multi-layer MLP with adjustable width. During training, all Triangular Dropout layers experience the same batch size and therefore have identical masks. This means that any one sample in the batch will be processed identically by all Triangular Dropout layers of a given size, simulating the whole stack of layers having reduced width.

5.2 Reduced Width

Our primary finding is that not only does the introduction of Triangular Dropout allow the classification portion of the VGG19 to be reduced in width after training, but that it can in fact be significantly reduced with only minor decreases in accuracy. This is especially noteworthy when considering that the number of parameters in a fully-connected layer is proportional to the square of the width. Our experiments with ablated width are summarized in Table 1 and graphically in Figure 4.

The overall trend is that there are significant diminishing returns on classifier accuracy with respect to number of parameters. As seen in Figure 4, an accuracy of around 50 percent can be achieved with only 1 percent of the original classifier parameters. This corresponds to hidden layers that are roughly 10 percent as wide as in the full model. In fact, most of the parameters contribute very little to the model's performance. The hidden layers can be reduced in size by half and only incur a 0.1 percent loss in classification accuracy. This corresponds to ablating more than half of the classifier parameters, and roughly 46 percent of all the parameters in VGG19.

5.3 Architecture Details and Training

Our version of VGG19 with Triangular Dropout was created by retraining the classification portion of VGG19. The convolutional layers were frozen and not retrained. The convolutional layers output 25088 features for a given input, which then pass through two hidden layers of size 4096, and finally output as 1000 features to be interpreted as classification scores for the image. Our model simply replaced the two hidden layers with Triangular Dropout layers of the same size.

We trained our model for 100 epochs using a batch size of 4096 in order to apply a full triangular mask to the batch. We used SGD with an initial learning rate of 0.01, reducing the learning rate by a factor of 10 every 30 epochs.

It is unclear if our model would have behaved differently if the entire architecture were retrained, which may have enabled Triangular Dropout to be applied to the feature descriptors of length 25088. This is of interested because VGG19's 25088 features are often used as a starting point for other image processing tasks. This is discussed further in the Section 8.

Ablated Width	Accuracy	vs. VGG19	vs. Full Width	Classif. Params	Param. Reduction
4096	69.7%	-1.6%	-0.0%	123,642,856	0.0%
2048	69.6%	-1.7%	-0.1%	57,627,624	53.4%
1024	69.2%	-2.1%	-0.5%	27,765,736	77.5%
512	67.8%	-3.5%	-1.9%	13,621,224	89.0%
256	65.3%	-6.0%	-4.4%	6,745,576	94.5%
128	61.7%	-9.6%	-8.0%	3,356,904	97.3%
64	54.9%	-16.4%	-14.8%	1,674,856	98.6%
32	38.8%	-32.5%	-30.9%	836,904	99.3%

Table 1: Accuracy and parameter numbers for VGG19 trained with Triangular Dropout, with various layer widths after training. The first row (width 4096) is the unreduced width of the network identical to VGG19's original architecture.

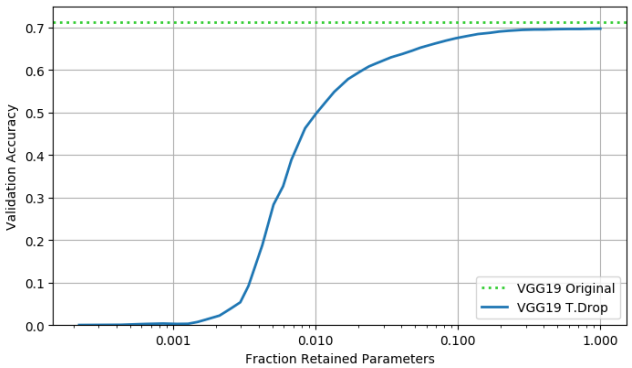

Figure 4: Validation accuracy of VGG19 with Triangular Dropout as the two hidden layers in the classification MLP are modified in width after training (blue). For comparison, the reported testing accuracy of the original VGG19 is also shown (green).

6 Characterizing Task Complexity

The results from Section 5 indicate that VGG19 could use a more compressed architecture and still perform well. In this section, we demonstrate the potential uses of Triangular Dropout for data analysis, utilizing network width as a measurement of task complexity. Specifically, we train RL policies on four locomotion control tasks from the MuJoCo (Todorov et al., 2012) environment suite within OpenAI Gym (Brockman et al., 2016), and then distill these policies into secondary networks that use Triangular Dropout in a hidden layer. We then examine the performance of these distilled policies as they are reduced in width at this middle layer, recovering the compressability of these policies, and, in turn, describing the complexity of the tasks that they solve.

6.1 TASK COMPLEXITY MEASUREMENT

Measuring task complexity is an important and open problem in reinforcement learning, multitask learning, and meta-learning. In single-task cases, the complexity of a problem impacts the size of the network needed to produce a good policy and the amount of data required to train such a network (McCandlish et al., 2018). In multitask and meta-learning, in which multiple tasks may be in competition for limited network parameters, characterizing the complexity of tasks can help determine how they should be prioritized during training or to what extent they may have overlapping representations (for example, this serves as the basis of methods such as elastic weight consolidation Kirkpatrick et al. (2016)).

In our experiments, we characterize the complexity of a task by first training an expert RL policy to perform that task, and then determining how much the policy can be compressed before performance is lost. We make this determination by first imitating the policy in a network that has a hidden layer using Triangular Dropout, and then ablating nodes from this layer until it affects the performance of this imitated policy. We do not use Triangular Dropout in RL directly because the masking significantly impacts the policy's behavior during data collection. Additionally, it is unclear how to manage Triangular Dropout when processing single samples, since a full batch is needed to apply the triangular mask.

We define the complexity of a policy as the smallest hidden width that can be retained without affecting performance. Other choices could be used to ease comparison amongst tasks, such as "smallest width that enables performance above human-level" or "smallest width that achieves a score of x". Since the policies are trained to perform well on specific tasks, the policy complexity should also roughly measure the complexity of the underlying task.

6.2 EXPERIMENT DETAILS

We apply our complexity technique to four environments: Hopper-v3, HalfCheetah-v3, Walker2d-v3, and Ant-v3, all of which come from OpenAI Gym's MuJoCo environment collection. We selected these tasks because they have similar objectives (locomote forward at the highest speed possible), but differ in complexity. For example, Hopper-v3 represents its observations as a vector of 11 values, while Ant-v3 has observations of size 111, indicating the Any-v3 is probably more complex.

For each environment, we first train an expert policy using proximal policy optimization (PPO) (Schulman et al., 2017), and then imitate them through supervised learning. To imitate these expert policies, we first gather (state, action) pairs from several thousand evaluation runs with occasional random actions taken to prevent over-sampling of the most likely trajectories. We randomly down-select to 100,000 data points and train a new model to imitate this policy in a supervised fashion. The new model consists of 3 fully-connected layers of size 128, with the middle layer using Triangular Dropout. After training, we can reduce the width of this layer to enforce a bottleneck in the mapping of states to actions. Training details of both the experts and imitation networks are available in the Appendix.

6.3 COMPLEXITY RESULTS

We analyze our imitated policies by comparing their average top speed over 100 evaluation episodes to the retained internal width of the middle layer. For each policy, we find that there is a clear separation between internal widths that are too small, and some ideal width which encompasses the full policy. The full results are shown in Figure 5.

Examining the curves from Figure 5, the policies for Hopper, HalfCheetah, Walker2d, and Ant seem to require widths of 4, 2, 3, and 7 respectively to reach full performance compared to the trained experts. This suggests that HalfCheetach is perhaps the least complex task of the four, while Ant is (predictably) the most complicated. Or perhaps more accurately, the behaviors required to solve these tasks are most simplistic for HalfCheetah, and most intricate for Ant. Notably, the imitated policy for Ant actually out-performed the PPO expert considerably, while the other imitations levelled off at similar performance to the original experts (noted by the dotted horizontal lines in Figure 5).

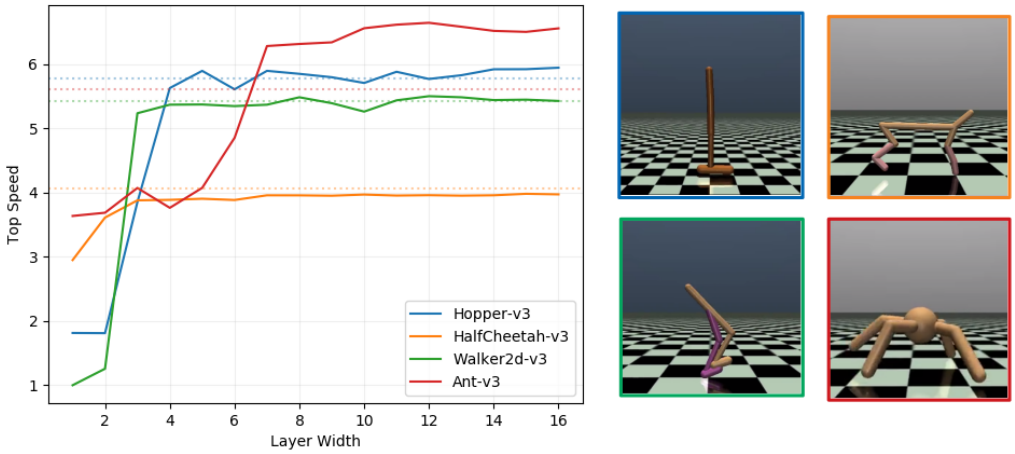

Figure 5: Imitator performance as the Triangular Dropout layers are reduced in width after training. We only show widths 1 through 16 (out of 128 available) because larger widths are more or less identical in performance. On the right, we show visuals from the corresponding MuJoCo environments.

7 Discussion

Throughout our experimentation, we found several characteristics of Triangular Dropout that were surprising. Firstly, the results indicate that features learned with Triangular Dropout are indeed one-way dependent. That is, the $n + 1th$ feature relies on the nth to be meaningful, but not the other way around. It is not immediately clear that this property should form inherently, since for a given training batch there are many examples for which adjacent features are both optimized.

Perhaps our most significant finding was that Triangular Dropout seems to encourage significant compression in learned features without strongly impacting performance. In many cases the network used far less of its available width than we had allocated, and to our surprise did not spread representation over all available resources. This was most evident in the RL experiments, in which we gave the Triangular Dropout layers a maximum width of 128. However, the imitation networks only seemed to require the first $\sim$10 features, indicating that these networks not only form compressed representations but do so in a very organized way: they have diminishing reliance on features as the effective layer size increases.

We believe Triangular Dropout to be an interesting mechanism that could lead to networks which are self-organizing and variable in size. In addition to the useful properties discussed above, a network that can be resized without retraining has many practical benefits, especially as AI models get larger and mobile and embedded platforms become increasingly widespread.

8 Future Directions

There are several promising directions in which to continue this work. Triangular Dropout has only been designed and tested for fully-connected layers, and immediate future work could examine reforming Triangular Dropout for convolutions or transformers. Additionally, Triangular Dropout allows a network layer to be resized in width, but not depth. A corresponding mechanism which would allow for variable-depth networks post-training would be similarly useful, and very powerful when combined with variable width.

Another potential direction would be to examine variations on Triangular Dropout which are closer in performance to the corresponding architecture without this mechanism. While we find Triangular Dropout to perform very similarly to standard networks, we do realize that in some domains a small change of performance may have very significant real-world consequences.

REFERENCES

Dario Amodei and Danny Hernandez. Ai and compute. 2019. URL https://openai.com/blog/ai-and-compute/.

Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. What is the state of neural network pruning? *CoRR*, abs/2003.03033, 2020. URL https://arxiv.org/abs/2003.03033.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Robust predictable control, 2021.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Training pruned neural networks. *CoRR*, abs/1803.03635, 2018. URL http://arxiv.org/abs/1803.03635.

Irina Higgins, Loïc Matthey, Arka Pal, Christopher P. Burgess, Xavier Glorot, Matthew M. Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size, 2016.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

James Kirkpatrick, Razvan Pascanu, Neil C. Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *CoRR*, abs/1612.00796, 2016. URL http://arxiv.org/abs/1612.00796.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training, 2018.

B. L. Paulhamus, Edward W. Staley, Corban G. Rivera, Kapil D. Katyal, and Chien-Ming Huang. Amplified control for robotic teleoperation. 2019.

Chi-Hieu Pham, Saïd Ladjal, and Alasdair Newson. Pcaae: Principal component analysis autoencoder for organising the latent space of generative networks, 2020.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. https://github.com/DLR-RM/stable-baselines3, 2019.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pp. 833–840, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. *Learning Internal Representations by Error Propagation*, pp. 318–362. MIT Press, Cambridge, MA, USA, 1986. ISBN 026268053X.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Shivangi Srivastava, Maxim Berman, Matthew B. Blaschko, and Devis Tuia. Adaptive compression-based lifelong learning. *CoRR*, abs/1907.09695, 2019. URL http://arxiv.org/abs/1907.09695.

Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2020.

Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders, 2017.

Naftali Tishby, Fernando Pereira, and William Bialek. The information bottleneck method. *Proceedings of the 37th Allerton Conference on Communication, Control and Computation*, 49, 07 2001.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. ICML '08, pp. 1096–1103, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605582054. doi: 10.1145/1390156.1390294. URL https://doi.org/10.1145/1390156.1390294.

A APPENDIX

A.1 AUTOENCODER EXPERIMENTS

In this section we detail architectures, training routines, and additional experiments concerning Triangular Dropout applied to MNIST autoencoders.

A.1.1 ARCHITECTURE AND TRAINING DETAILS

In all cases the network architectures were as follows, given some encoder output size z:

Model	Type	Activation	Parameters
Encoder	Convolution	Relu	32 Filters, 9x9 Kernel, Stride 1
	Convolution	Relu	32 Filters, 7x7 Kernel, Stride 1
	Convolution	Relu	32 Filters, 3x3 Kernel, Stride 1
	Fully-Connected	linear	4608 Inputs, z Outputs
Decoder	Fully-Connected	Relu	z Inputs, 4608 Outputs
	Transp. Conv.	Relu	32 Filters, 3x3 Kernel, Stride 1
	Transp. Conv.	Relu	32 Filters, 7x7 Kernel, Stride 1
	Transp. Conv.	Relu	32 Filters, 9x9 Kernel, Stride 1
	Convolution	sigmoid	1 Filter, 3x3 Kernel, Stride 1, Padding 1

Models are trained with batch size 1024 using the Adam optimizer, with a learning rate of 0.005 that decreases by a factor of 10 if the average epoch loss does not decrease by at least 2 percent over the last 15 epochs. This is repeated for 5 decreases of the learning rate. MNIST images were normalized to values between 0 and 1, and binary cross entropy was used as the loss for reconstruction.

A.1.2 AVAILABLE WIDTH VERSUS COMPRESSION

We additionally performed experiments to test if the available encoding size affected the learned representations in Triangular Dropout. If extra width is available, does the model utilize that width or is it ignored? In Figure 6 we show a plot of autoencoders trained using Triangular Dropout with encoding widths up to 32, however they were trained with available widths of 4, 8, 16, 32, 64, 128, 256, and 512.

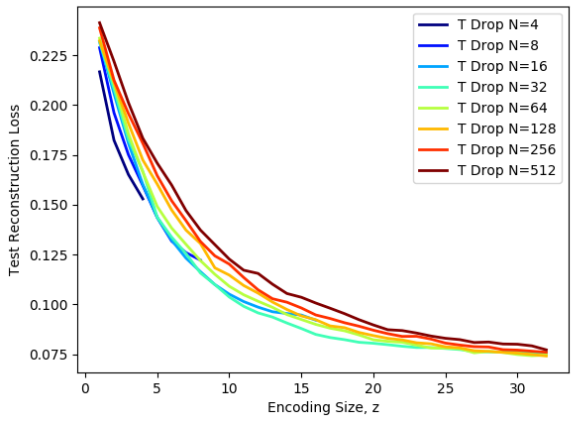

Figure 6: MNIST reconstruction loss for the widths 1 through 32 of autoencoders trained with Triangular Dropout and a variety of available latent widths.

We find that there is a clear pattern of models with more available width being less compressed, but the overall difference in performance diminishes as available width increases.

A.2 RL Training Schemes

In this section we detail hyperparamters and training routines needed to reproduce our experiments from Section 6.

A.2.1 PPO Hyperparameters and Training Details

The PPO expert networks discussed in Section 6 consisted of separate policy and value networks. The networks were MLPs with two hidden layers of size 128, with input and output sizes determined by the environment. All layers (besides value output) used hyperbolic tangent as the activation function. We used the implementation of PPO from Stable Baselines 3 (Raffin et al., 2019).

Hyperparameter	**Value**	**Hyperparameter**	**Value**
Training Steps	10 million	Entropy Coefficient	0.01
Parallel Actors	20	Clip Range	0.2
Rollout Length	256	Adam Learning Rate	0.0003
Training Epochs	3	Gamma, γ	0.99
Value Coefficient	1.0	Lambda, λ	0.95

Table 2: PPO Hyperparameters

A.2.2 PPO Learning Curves and Statistics

For reference, we provide the learning curves and final performance of our PPO experts.

Environment	**Top Speed**	**Final Distance**	**Reward**	**Episode Length**	**State Variables**	**Actions**
Hopper-v3	5.77 ± 0.69	7.65 ± 2.78	1286 ± 457	330 ± 111	11	3
HalfCheetah-v3	4.06 ± 0.10	88.87 ± 6.35	1272 ± 124	1000 ± 0	17	6
Walker2d-v3	5.43 ± 1.08	16.07 ± 8.15	2690 ± 1309	685 ± 294	17	6
Ant-v3	5.61 ± 0.86	57.15 ± 41.4	715 ± 602	527 ± 351	111	8

Table 3: Summary statistics for RL policies trained on selected MuJoCo locomotion environments.

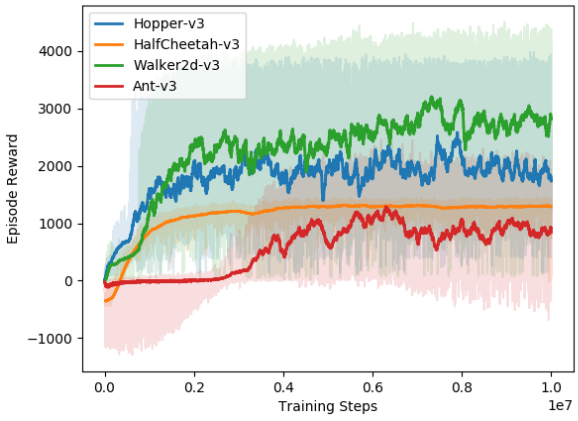

Figure 7: PPO Training Curves

A.2.3 Imitation Network Training Details

The imitation networks consisted of MLPs with 3 hidden layers of width 128, the middle layer using Triangular Dropout. Internal layers used the Relu activation function while the final layer used hyperbolic tangent. The networks were trained in a supervised fashion on 100,000 (state,action) pairs

collected from running the PPO-trained policies on their respective environments. The networks were trained for 240 epochs with a batch size of 1024, using the Adam optimizer with a learning rate of 0.01. Every 80 epochs, the learning rate was decreased by a factor of 10.