Just-in-Time Sparsity: Learning Dynamic Sparsity Schedules

Kale-ab Tessera [1] **Chiratidzo Matowe** [1] **Arnu Pretorius** [1] **Benjamin Rosman** [2] **Sara Hooker** [3]

Abstract

Sparse neural networks have various computational benefits while often being able to maintain or improve the generalization performance of their dense counterparts. Popular sparsification methods have focused on *what to sparsify*, i.e. which redundant components to remove from neural networks, while *when to sparsify*, has received less attention and is usually handled using heuristics or simple schedules. In this work, we focus on learning sparsity schedules from scratch using reinforcement learning. In simple CNNs and ResNet-18, we show that our learned schedules are diverse across layers and training steps, while achieving competitive performance when compared to naive handcrafted schedules. Our methodology is general-purpose and can be applied to learning effective sparsity schedules across any pruning implementation.

1. Introduction

Overparameterized models have led to many breakthroughs in machine learning (Chowdhery et al., 2022; Brown et al., 2020; Zhai et al., 2021; Reed et al., 2022). However, even with this success, the growth in parameter counts – at times exceeding billions of parameters – present many challenges. These challenges include efficient storage, inference and training of large models. One of the most common methods to handle some the challenges of these models is pruning.

Pruning aims to remove unnecessary components (weights, neurons, whole layers etc.) in neural networks (LeCun et al., 1989). When designing a pruning algorithm, two important questions arise: (1) *what to prune* (the pruning criteria) and (2) *when to prune* (the pruning schedule). Pruning criteria focuses on identifying which redundant elements to remove from a network, while pruning schedules focus on what stages of a network's lifecycle to introduce and remove components.

Pruning methods can introduce sparsity at various periods. Some methods prune once, such as SNIP (Lee et al., 2018) which prunes once at initialization (pruning from scratch or static sparse training), other methods follow a training, pruning, and fine-tuning cycle (Liu et al., 2018) (post-training pruning), while other methods prune and possibly regrow weights during training (dynamic sparsity during training), such as Sparse Evolutionary Training (SET) (Mocanu et al., 2018). Post-training pruning methods have a high computational cost because they require training an overparameterized dense model first, while pruning from scratch methods have shown promise, but have been outperformed by dynamic sparsity methods such as Rigging the Lottery (RigL) (Evci et al., 2020).

Even with the potential for better performance, dynamic sparsity methods pose some challenges. Since these methods grow and prune during training, as opposed to just before or after training, their pruning schedules become difficult to calibrate. These methods require decisions about when to start and stop pruning, the number of iterations between pruning, and the rate or function at which pruning changes over time. Furthermore, all of these hyperparameter choices have to be made at a layerwise level. The vast possibilities for these hyperparameters have resulted in most dynamic sparsity methods using simple schedules uniformly across all layers (Zhu & Gupta, 2017). This is not ideal since the dynamics across layers could be different and this choice of uniform schedules could be limiting the method's performance.

Not all dynamic sparsity methods apply sparsity uniformly across layers, various heuristics have been developed. SET (Mocanu et al., 2018) uses an Erdős Rènyi (ER) random graph, which scales the number of active weights according to the number of neurons. ER was adapted to Erdős Rènyi-Kernel (ERK) ratios by (Evci et al., 2020), to work better with Convolutional Neural Networks (CNNs). Su et al. (2020) used predefined heuristics and schedules such as decaying density ratios as a function of network depth, while Gale et al. (2019) leave the first and final layer dense.

For determining the pruning schedule (change in sparsity), most dynamic sparsity methods use a function parameter-

[1]InstaDeep [2]School of Computer Science and Applied Mathematics, University of the Witwatersrand [3]Cohere AI. Correspondence to: Kale-ab Tessera <kaleabtessera@gmail.com>.

DyNN workshop at the 39^{th} International Conference on Machine Learning, Baltimore, Maryland, USA, 2022.

ized by training steps. SET (Mocanu et al., 2018), Deep Rewiring (DeepR) (Bellec et al., 2017) and Neural Network Synthesis Tool (NEST) (Dai et al., 2019) use a constant sparsity schedule. RigL (Evci et al., 2020) and Sparse Network From Scratch (SNFS) (Dettmers & Zettlemoyer, 2019) use a cosine annealing schedule, while Zhu & Gupta (2017); Mostafa & Wang (2019) use a cubic schedule.

There are also methods which aim to learn the sparsity schedule as opposed to using heuristics or a predefined function. Dynamic Sparse Training (DST) (Liu et al., 2020) proposes adapting the layerwise pruning schedule by using a notion of parameter importance and a hyperparameter α, which is a scaling coefficient for a regularization term. Kusupati et al. (2020) use the soft-threshold version of a weight tensor and construct the loss as a function of the layer sparsity and learn this sparsity via backpropagation. Although these methods have been able to learn layerwise pruning schedules, these schedules are not general. They can only be learned for the method they were designed for since they rely on modified formulations of weights or handcrafted notions of parameter importance.

In this work, we focus on learning these sparsity schedules from scratch using reinforcement learning (RL). Our agent learns to take actions that alter the sparsity level of layers in a network, while observing the changes in the dynamics of the networks and leveraging holdout accuracy as a reward signal. Contrary to other learned approaches, such as DST (Liu et al., 2020) and (Kusupati et al., 2020), our methodology is general-purpose and can be applied to any pruning method.

Our results shows that our RL agent is able to learn diverse and well-performing learning schedules in a flexible manner. These schedules are diverse across training steps and layers in a network, and often leads to better test accuracy than commonly used handcrafted schedules like linear, quadratic, cubic, cosine, and constant schedules.

2. Learning Dynamic Sparsity Schedules

Reinforcement learning (RL) has been shown to be promising at solving complex tasks, in a wide variety of domains. The goal of an RL agent is to learn from interactions with an environment, by using trial-and-error. The agent takes in *observations* from its environment and uses these observations to select *actions* in order to maximize a numerical *reward* signal.

In our application, the agent learns the sparsity schedules for the different layers in a network, across timesteps. This is shown in Figure 1. This formulation is convenient because our agent can learn sparsity schedules across any network and pruning method.

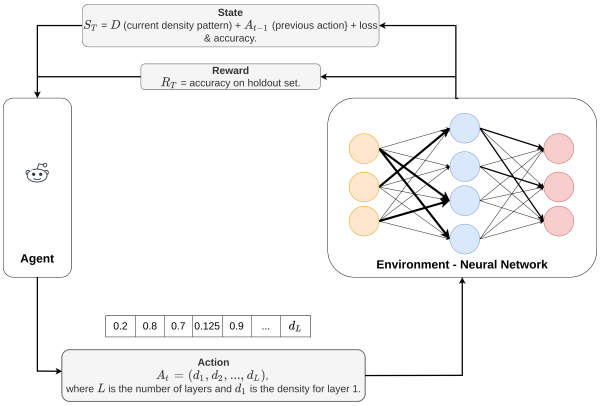


Figure 1: The Reinforcement Learning (RL) flow for Learning Sparsity Schedules

2.1. Observations, Rewards and Actions

Observation For our observations, we use the previous action A_{t-1} (previous sparsity schedule), the density for each layer D, the loss and accuracy (validation loss and accuracy during training, test loss and accuracy during evaluation). We use both A_{t-1} and D because $A_{t-1} \neq D$, since even if our agent allows a certain density for a layer by defining the possible active weights through a mask, the density in that layer could be lower because of the possible propagation of dead gradients. We also explored other observations related to network dynamics such as the mean and variance of the weights and their gradients in each layer, but these provided no performance benefit, while at times making it harder for our agents to learn.

Reward For our reward, during training, we use validation accuracy. This is computed every ΔT steps (e.g. every 100 steps) and provides dense rewards to our agent after an action has been taken. During evaluation, where we simply measure the performance of our agent, we record test accuracy.

Action At each pruning step (every ΔT steps), our agent takes actions that updates the sparsity schedule of each layer in a network. The actions at step t are described as follows:

$$A_t = (d_1, d_2, ..., d_L), \tag{1}$$

where L is the number of layers. For every layer l, our density for that layer, d_l, corresponds to the percentage of possible active weights in layer l, that is to say, the mask applied to layer l, m_l has d_l percentage of active weights compared to a dense/fully-connected layer. Every layer's density is bound by the minimum (min_d) and maximum density (max_d) in that layer — $d_l \in [min\_d_l, max\_d_l]$. This formulation allows for the learning of non-uniform layerwise sparsity schedules.

Table 1: Test Accuracy (mean and standard deviation) of different schedules on CIFAR-10, using Simple-CNN.

Target Density (%)	Schedule	Random Pruning with Random Regrowth (RP-RR)	Magnitude Pruning with Random Regrowth (MP-RR)
10	Linear	20.365 +- 17.952	60.418 +- 1.362
	Quadratic	23.15 +- 20.043	61.259 +- 1.485
	Cubic	33.721 +- 21.693	60.396 +- 0.832
	Cosine	18.302 +- 14.379	59.807 +- 0.384
	Constant	**61.475 +- 0.731**	62.536 +- 0.314
	Learned (Ours)	61.071 +- 1.574	**63.191 +- 0.810**
50	Linear	64.54 +- 0.477	64.78 +- 0.464
	Quadratic	64.987 +- 0.86	63.933 +- 0.431
	Cubic	65.31 +- 0.49	64.315 +- 0.437
	Cosine	64.672 +- 0.771	64.737 +- 0.345
	Constant	65.1 +- 0.283	65.388 +- 0.375
	Learned (Ours)	**65.655 +- 0.515**	**65.686 +- 0.284**
100	Linear	66.228 +- 0.691	66.711 +- 0.423
	Quadratic	66.947 +- 0.749	67.25 +- 0.578
	Cubic	66.857 +- 0.627	67.395 +- 0.547
	Cosine	66.074 +- 0.282	66.18 +- 1.027
	Full Dense	**67.815 +- 0.146**	67.878 +- 0.482
	Learned (Ours)	67.534 +- 0.174	**67.908 +- 0.162**

Table 2: Test Accuracy (mean and standard deviation) of different schedules on CIFAR-10, using Resnet-18 with 100% Target Density.

Schedule	Test Accuracy
Linear	75.611 +- 0.588
Quadratic	76.445 +- 0.248
Cubic	76.588 +- 0.51
Cosine	75.593 +- 0.457
Constant (Fully Dense)	77.631 +- 0.346
Learned (Ours)	**78.398 +- 0.164**

2.2. PPO Agent

We use Proximal Policy Optimization (PPO) (Schulman et al., 2017) to learn these sparsity schedules. PPO is a sample-efficient policy gradient algorithm, that aims to stabilize training by ensuring policy (agent controller) updates are not too large. Our agent is implemented using acme (Hoffman et al., 2020), a distributed reinforcement learning framework.

3. Experimental Results

3.1. Experimental Setting

We evaluate our proposed method on CIFAR-10 (Krizhevsky et al., 2009). We use a train-val-test split of 45 000, 5 000 and 10 000 images respectively. During training, our agent uses the validation accuracy as a reward and never sees the test set (which we only use to evaluate how well our model is performing). We evaluate our approach on two network architectures – a Simple-CNN, with three convolutional layers (each followed by a max pooling layer) and two dense layers; and ResNet18 (He et al., 2016). All hyperparameters are presented in Appendix A.

We learn schedules for two pruning methods:

1. Random Pruning, with Random Regrowth (RP-RR) : A naive, yet effective pruning method (Gale et al., 2019; Lee et al., 2018; Frankle et al., 2020; Su et al., 2020; Liu et al., 2022), where weights are randomly pruned and added.

2. Magnitude Pruning, with Random Regrowth (MP-RR) : A pruning method where during pruning, the smallest weights are removed, and during regrowth, new weights are randomly added. This can be thought of as a generalized version of SET (Mocanu et al., 2018) that doesn't rely on an Erdős–Rényi random graph initialization and with a custom or learned pruning schedule as opposed to a constant schedule.

These simple pruning methods are used because they are closely related to a variety of common pruning implementations and can be used to learn about these implementations in their simplest form.

3.2. Learning Schedules in Shallow Networks

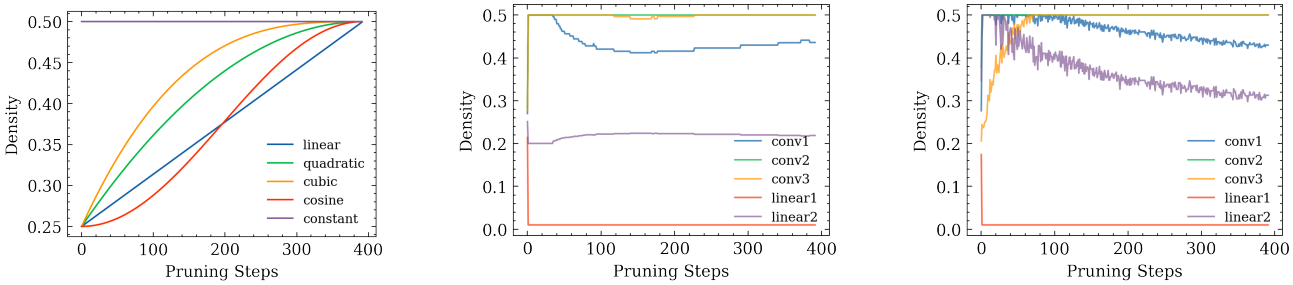
Using our Simple-CNN network, we compare the schedules learned by our PPO agent against common adaptive sparsity schedules, namely linear, quadratic, cubic, and cosine, while also comparing our schedules to a constant schedule. For the common schedules, we use uniform sparsity across the layers.

We compare these schedules at different target densities – 10% dense, 50% dense and 100% (fully) dense. For the predefined schedules, the target density is the final density and we use an initial density equal to half of the final density (not too low as to hurt performance - see illustration in Figure 2a). For our learned schedule, the maximum density for each layer is equal to the target density and the minimum density is equal to 1% dense.

3.2.1. Learned schedules are Competitive

From Table 1, we see that across various densities, in Random Pruning with Random Regrowth (RP-RR) and Magnitude Pruning with Random Regrowth (MP-RR), our learned schedules are competitive.

Figure 2: Different Schedules at 50% Target Density



(a) Common Sparsity Schedules (b) Learned - RP-RR (c) Learned - MP-RR

3.2.2. Random Pruning is More Sensitive to Pruning Schedule than Magnitude Pruning

In RP-RR, a constant schedule is a strong baseline, often outperforming all adaptive schedules. This is especially apparent at a density of 10% (high sparsity). This suggests that at low parameter counts and naive pruning (RR-RR), that it is better to keep a consistent pruning schedule. Of the adaptive schedules, our learned schedule performs better than other adaptive schedules (linear, quadratic, cubic and cosine), while being comparable or slightly worse than a constant schedule.

For MP-RR, we see that our learned schedules consistantly outperforms all other schedules. We also see that in MP-RR, even at low densities, adaptive schedules are competitive to a constant schedule, unlike RP-RR. This hints at MP-RR being less sensitive to pruning schedules than RP-RR.

3.2.3. The Learned Schedules are Diverse

From Figures 2b and 2c, we see that the learned schedules are diverse across layers and different pruning methods.

In Naive Pruning, a Piecewise Constant Schedule is Competitive When random pruning is used, in Figure 2b, we see that most layers (`conv1`, `conv2` and `linear1`) follow a piecewise constant function, where the density level is chosen early and kept constant, with layer `linear1` effectively being removed completely. This learned schedule agrees with (Gale et al., 2019), who note that for random pruning, it is better to start and end the pruning early and train with the fixed sparse mask.

When magnitude pruning is used, Figure 2c, we see a wide variety of layer schedules, with some layers growing connections (layer `conv2` and `conv3`), some layers pruning over time (layer `conv1` and `linear2`), and some layers being made redundant such as layer `linear1` (similarly to RP-RR).

Schedules Should be Learned and Non-Uniform The diversity of these schedules imply that they should rather be learned as these would be difficult and costly to handcraft per layer, timestep and pruning method.

3.3. Learning Schedules in ResNet-18

We also explore learning sparsity schedules in larger networks, specifically ResNet-18. We compare our learned schedules to common sparsity schedules with a target density of 100%.

From Table 2, we see that our methodology can still learn well-performing schedules in deeper networks. Furthermore, the learned schedules outperform other schedules and even outperform a fully dense network, with only 67% of its weights. The performance benefit of learning a custom schedule is more apparent in ResNet-18 than in Simple-CNN, implying that custom schedules are more relevant for deeper, more complicated networks.

4. Conclusions, Limitations and Future Directions

In this work, we demonstrate that it is possible to learn well-performing dynamic sparsity schedules using reinforcement learning. The schedules learned are not arbitrary and are distinct per layer and pruning method. The results show that appropriate pruning schedules can help improve the performance of pruning algorithms and thereby motivates learning optimal schedules per method using a similar approach to the one presented in this paper.

Even with the success of these results, there are certain limitations to our approach. Our method is computationally intensive, at times taking 25-50 episodes (25-50 times the standard amount of training) to converge to optimal sparsity schedules. We envision future work being able to improve the speed of learning by finding better representations of the observations and exploring different policy architectures and reward formulations. Finally, we also believe being able to generalize across architectures and datasets would vastly improve the efficiency of our approach (e.g. we train on CIFAR-10 and generalize to ImageNet (Deng et al., 2009)).

References

Bellec, G., Kappel, D., Maass, W., and Legenstein, R. Deep rewiring: Training very sparse deep networks. *arXiv preprint arXiv:1711.05136*, 2017.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

Dai, X., Yin, H., and Jha, N. K. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 68(10):1487–1497, 2019.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Dettmers, T. and Zettlemoyer, L. Sparse networks from scratch: Faster training without losing performance. *arXiv preprint arXiv:1907.04840*, 2019.

Evci, U., Gale, T., Menick, J., Castro, P. S., and Elsen, E. Rigging the lottery: Making all tickets winners. In *International Conference on Machine Learning*, pp. 2943–2952. PMLR, 2020.

Frankle, J., Dziugaite, G. K., Roy, D. M., and Carbin, M. Pruning neural networks at initialization: Why are we missing the mark? *arXiv preprint arXiv:2009.08576*, 2020.

Gale, T., Elsen, E., and Hooker, S. The state of sparsity in deep neural networks, 2019. URL https://arxiv.org/abs/1902.09574.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hoffman, M., Shahriari, B., Aslanides, J., Barth-Maron, G., Behbahani, F., Norman, T., Abdolmaleki, A., Cassirer, A., Yang, F., Baumli, K., et al. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

Krizhevsky, A., Nair, V., and Hinton, G. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, 6:1, 2009.

Kusupati, A., Ramanujan, V., Somani, R., Wortsman, M., Jain, P., Kakade, S., and Farhadi, A. Soft threshold weight reparameterization for learnable sparsity. In *International Conference on Machine Learning*, pp. 5544–5555. PMLR, 2020.

LeCun, Y., Denker, J., and Solla, S. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989.

Lee, N., Ajanthan, T., and Torr, P. H. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

Liu, J., Xu, Z., Shi, R., Cheung, R. C., and So, H. K. Dynamic sparse training: Find efficient sparse network from scratch with trainable masked layers. *arXiv preprint arXiv:2005.06870*, 2020.

Liu, S., Chen, T., Chen, X., Shen, L., Mocanu, D. C., Wang, Z., and Pechenizkiy, M. The unreasonable effectiveness of random pruning: Return of the most naive baseline for sparse training. *arXiv preprint arXiv:2202.02643*, 2022.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Mocanu, D. C., Mocanu, E., Stone, P., Nguyen, P. H., Gibescu, M., and Liotta, A. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9 (1):1–12, 2018.

Mostafa, H. and Wang, X. Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization. In *International Conference on Machine Learning*, pp. 4646–4655. PMLR, 2019.

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Su, J., Chen, Y., Cai, T., Wu, T., Gao, R., Wang, L., and Lee, J. D. Sanity-checking pruning methods: Random tickets can win the jackpot. *Advances in Neural Information Processing Systems*, 33:20390–20401, 2020.

Zhai, X., Kolesnikov, A., Houlsby, N., and Beyer, L. Scaling vision transformers. arxiv. *arXiv preprint arXiv:2106.04560*, 2021.

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

A. Hyperparameters

Table 3: Pruning

Hyperparamter	Configuration
ΔT	100
min_density:	0.01
max_density:	1.0
num_epochs:	100

Table 4: Optimizer

Hyperparamter	Configuration
Optimizer	Stochastic Gradient Descent (SGD)
Learning Rate	0.01
Momentum	0.9

Table 5: Dataset

Hyperparamter	Configuration
Train Batch Size	128
Test Batch Size	250
Data Augmentation	None (Simple-CNN), Random Crop & Random Flip (ResNet-18)

Table 6: PPO Agent

Hyperparamter	Configuration
unroll_length:	1
num_minibatches:	1
num_epochs:	1
batch_size:	64
clip_value:	False
ppo_clipping_epsilon:	0.2
gae_lambda:	0.95
discount:	0.99
learning_rate:	1e-3
adam_epsilon:	1e-5
entropy_cost:	0.01
value_cost:	1.
max_gradient_norm:	0.5
prefetch_size:	4
variable_update_period:	1