Pruning Transformer via Sparsity

Haowen Shen
Department of Physics
Fudan University

Abstract

While Vision Transformers (ViTs) have gained immense popularity, their large model sizes and high training costs remain challenging. In this project, I extend the method of SViTE Chen et al. [2021] and propose a novel approach to address this issue. Instead of training full ViTs, I dynamically extract and train sparse subnetworks within a fixed, compact parameter budget. The method not only optimizes the model parameters but also explores connectivity throughout the training process. As a result, I obtain a promising output with slight penalized accuracy based on a significant reduction in non-zero parameters. Detailed experiment results can be found in Table 2.

1 The Task for Network Sparsity: Pruning Transformer via Sparsity

1.1 Background Introduction

In recent years, Transformer has shown remarkable results in various applications such as NLP and computer vision . However, such results rely on millions or even billions of parameters, which limits its deployment in limited storage platforms. Yet, many domains would benefit from neural networks, hence the need to reduce their cost while maintaining their performance.
Neural network pruning is a method that revolves around the intuitive idea of removing superfluous parts of a network that performs well but costs a lot of resources. Indeed, even though large neural networks have proven countless times how well they could learn, it turns out that not all of their parts are still useful after the training process is over. The idea is to eliminate these parts without impacting the network's performance.

1.2 Vision Transformer and Sparsity

Transformer stems from natural language processing (NLP) applications. The Vision Transformer (ViT) Dosovitskiy et al. [2021] pioneered to leverage a pure transformer, to encode an image by splitting it into a sequence of patches, projecting them into token embeddings, and feeding them to transformer encoders. With sufficient training data, ViT is able to outperform convolution neural networks on various image classification benchmarks.

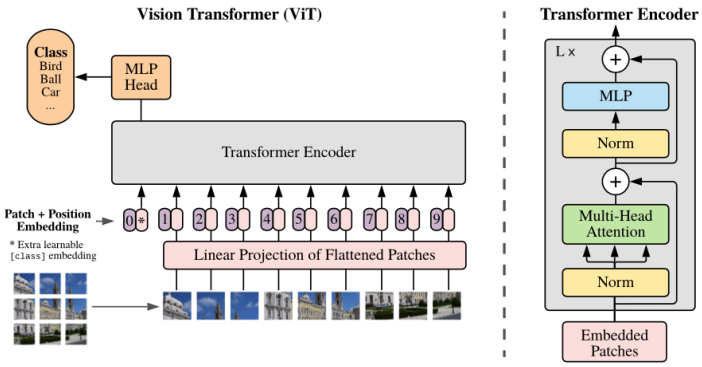


Figure 1: Vision Transformer (ViT) Structure

Despite the impressive empirical performance, ViTs are generally heavy to train, and the trained models remain massive. That naturally motivates the study to reduce ViT inference and training costs. As a result, chasing sparsity in Vision Transformers was crucial. The most common type of sparse regularization is l1 sparsity , which has been incorporated into neural networks and applied to network pruning . Besides, structural sparsity (including group sparsity, and layer sparsity) was also presented. Recently, it has been applied to prune the transformer, by exploiting different candidates for pruning (including pruning multiple heads, and pruning the tokens).

1.3 Review of the state-of-the-arts

Pruning is well-known to effectively reduce deep network inference costs. It can be roughly categorized into two groups:
(i) unstructured pruning by removing insignificant weight elements per certain criterion, such as weight magnitude Han et al. [2016], gradientMolchanov et al. [2019];
(ii) structured pruning He et al. [2017] by remove model sub-structures, e.g., channels Liu et al. [2017] and attention heads Michel et al. [2019], which are often more aligned with hardware efficiency. All above require training the full dense model first, usually for several train-prune-retrain rounds.

In the case of pruning within Vision Transformer (ViT), two concurrent works [Zhu et al. [2021], Tang et al. [2022]] made initial attempts towards ViT post-training compression by pruning the intermediate features and tokens respectively, but did not jointly consider weight pruning nor efficient training. Another loosely related field is the study of efficient attention mechanisms Roy et al. [2020]. They mainly reduce the calculation complexity for self-attention modules via various approximations such as low-rank decomposition.

1.4 Algorithms and critical codes

1.4.1 Sparse Vision Transformer Exploration (SViTE)

Our proposed techniques was mainly inspired by [Zhu et al. [2021], Chen et al. [2021]]. It starts from a randomly sparsified model; after optimizing several iterations, it shrinks a portion of parameters based on pre-defined pruning criterion(e.g. weight magnitude), and activates new connections. After upgrading the sparse topology, it trains the new subnetwork until the next update of the connectivity. An illustration of the overall procedure is shown in Figure 2. The key factors of sparse training are ①.sparsity distribution, ②.update schedule, ③.pruning and ④.grow criterion.

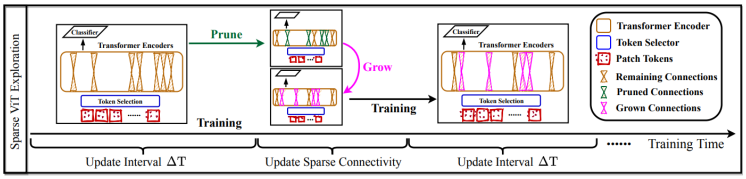


Figure 2: Basic Sparse Training Procedure

To be specific, in the Chen et al. [2021], the authors propose the Sparse Vision Transformer Exploration (SViTE) model. SViTE explores the unstructured sparse topology in vision transformers. Based on the above procedure, SViTE first adopts Erdo¨s-Re´nyiMocanu et al. [2018] as the ①.sparsity distribution. The number of parameters in the sparse layer is scaled by $1 - \frac{n_{l-1}+n_l}{n_{l-1}*n_l}$, where n_l is the number of neurons at transformer layer l. This distribution assigns higher sparsities to layers that have more parameters by adjusting the fraction of remaining weights based on the sum of the number of output and input neurons or channels. For the ②.update schedule, it contains: (i) the update interval ΔT, which is the number of training iterations between two sparse topology updates; (ii) the end iteration T_{end}, indicating when to stop updating the sparsity connectivity, and we set T_{end} to 80% of total training iterations in our experiments;

```
torch.cuda.synchronize()
if model_ema is not None:
    model_ema.update(model)
metric_logger.update(loss=loss_value)
metric_logger.update(lr=optimizer.param_groups[0]["lr"])
# update sparse topology

ite_step = mask.steps
if ite_step % args.update_frequency == 0 and ite_step < args.t_end * total_iteration:
    mask.at_end_of_epoch(pruning_type=args.pruning_type,
                         indicator_list=atten_pruning_indicator)
```

Figure 3: T_{end} indicating when to stop updating in ②.update schedule(ii)

(iii) the initial fraction α of connections that can be pruned or grow, which is 50% in our case; (iv) a decay schedule of the fraction of changeable connections $f_{decay}(t, \alpha, T_{end}) = \frac{\alpha}{2}(1 + cos(\frac{t*\pi}{T_{end}}))$, where a cosine annealing is used.

```
class CosineDecay(object):
    def __init__(self, death_rate, T_max, eta_min=0.005, last_epoch=-1):
        self.sgd = optim.SGD(torch.nn.ParameterList([torch.nn.Parameter(torch.zeros(1))]), lr=death_rate)
        self.cosine_stepper = torch.optim.lr_scheduler.CosineAnnealingLR(self.sgd, T_max, eta_min, last_epoch)

    def step(self):
        self.cosine_stepper.step()

    def get_dr(self, death_rate):
        return self.sgd.param_groups[0]['lr']
```

Figure 4: Function of CosineDecay in ②.update schedule(iv)

During each connectivity update, we choose the weight magnitude as ③.pruning indicator, and gradient magnitude as ④.the grow indicator. Specifically, SViTE eliminates the parameters with the layer-wise smallest weight values by applying a binary mask m_{prune}, then grow new connections with the highest magnitude gradients by generating a new binary mask m_{grow}. Both masks are employed to $W^{(l)}$ via the element-wise dot product, and note that the number of non-zero elements in m_{prune} and m_{grow} are equal and fixed across the overall procedure, making the fixed sparsity level in the network.

```python
self.gather_statistics() # count each of module's zeros and non-zeros
#prune
index = 0
for module in self.modules:
    for name, weight in module.named_parameters():
        name_cur = name + '_' + str(index)
        index += 1
        if name_cur not in self.masks: continue
        mask = self.masks[name_cur]

        # death
        if self.death_mode == 'magnitude':
            new_mask = self.magnitude_death(mask, weight, name_cur)
        elif self.death_mode == 'SET':
            new_mask = self.magnitude_and_negativity_death(mask, weight, name_cur)
        elif self.death_mode == 'threshold':
            new_mask = self.threshold_death(mask, weight, name_cur)
        # record pruning numbers
        self.pruned_number[name_cur] = int(self.name2nonzeros[name_cur] - new_mask.sum().item())
        self.masks[name_cur][:] = new_mask  # update new mask
```

(a) code for m_{prune}

```python
#grow
index = 0
for module in self.modules:
    for name, weight in module.named_parameters():
        name_cur = name +'_' + str(index)
        index += 1
        if name_cur not in self.masks: continue
        new_mask = self.masks[name_cur].data.byte()

        if self.growth_mode == 'random':
            new_mask = self.random_growth(name_cur, new_mask, self.pruned_number[name_cur], weight)

        elif self.growth_mode == 'momentum':
            new_mask = self.momentum_growth(name_cur, new_mask, self.pruned_number[name_cur], weight)

        elif self.growth_mode == 'gradient':
            # implementation for Rigging Ticket
            new_mask = self.gradient_growth(name_cur, new_mask, self.pruned_number[name_cur], weight)

        # exchanging masks
        self.masks.pop(name_cur)
        self.masks[name_cur] = new_mask.float()
```

(b) code for m_{grow}

Figure 5: Masks of prune and grow (in the code we choose weight magnitude and gradient magnitude respectively)

Also, apart from SViTE, we compare several effective pruning methods from CNN compression to help explore the sparse ViT performance including:

1.4.2 One-shot weight Magnitude Pruning (OMP)

In Han et al. [2016], the authors start by learning the connectivity via normal network training. Next, they prune the small-weight connections: all connections with weights below a threshold are removed from the network. Finally, they retrain the network to learn the final weights for the remaining sparse connections. Also, as 6 illustrates, network quantization and weight sharing further compresses the pruned network by reducing the number of bits required to represent each weight. It limits the number of effective weights we need to store by having multiple connections share the same weight, and then fine-tune those shared weights.

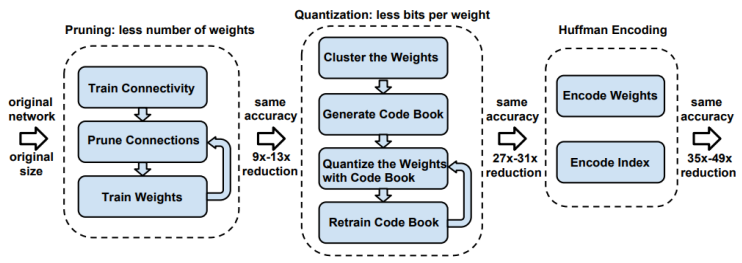


Figure 6: Structure of One-shot weight Magnitude Pruning

1.4.3 Gradually Magnitude Pruning (GMP)

In Zhu and Gupta [2017], the authors introduce a new automated gradual pruning algorithm in which the sparsity(represents proportion of zero parameters in this case) is increased from an initial sparsity value s_i (usually 0) to a final sparsity value s_f over a span of n pruning steps, starting at training step t_0 and with pruning frequency Δt:

$$s_t = s_f + (s_i - s_f)(1 - \frac{t - t_0}{n\Delta t})^3 \qquad for \quad t \in \{t_0, t_0 + \Delta t, ..., t_0 + n\Delta t\}$$

The binary weight masks are updated every t steps as the network is trained to gradually increase the sparsity of the network while allowing the network training steps to recover from any pruning-induced loss in accuracy. The intuition behind this sparsity function is to prune the network rapidly in the initial phase when the redundant connections are abundant and gradually reduce the number of weights being pruned each time as there are fewer and fewer weights remaining in the network.

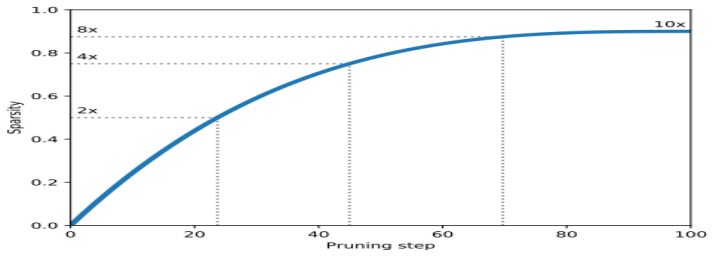


Figure 7: Sparsity function used for gradual pruning

1.4.4 Taylor Pruning (TP)

In Molchanov et al. [2019], the authors propose a method that iteratively removes the least important set of neurons (typically filters) from the trained model. It defines the importance in the form of the squared change in loss induced by removing a specific filter from the network. Since computing the exact importance is extremely expensive for large networks, they approximate it with a Taylor expansion, resulting in a criterion computed from parameter gradients readily available during standard training.
During each epoch, the following steps are repeated: 1. For each minibatch, we compute parameter gradients and update network weights by gradient descent. We also compute the importance of each neuron (or filter) using the gradient averaged over the minibatch. 2. After a predefined number of minibatches, we average the importance score of each neuron (or filter) over the minibatches, and remove the N neurons with the smallest importance scores. Fine-tuning and pruning continue until the target number of neurons is pruned, or the maximum tolerable loss can no longer be achieved.

1.5 Experiment and discussion of proposed methodology

1.5.1 Baseline pruning methods

Based on the pretrained ViT model "vit_base_patch16_224" from timm Wightman [2019], we conduct network pruning using the SViTE method as mentioned above. As we know, ViT is a large deep neural network containing approximately 85000000 parameters. Due to the algorithm limitation since we only have one GeForce GTX 1080 Ti, we set the batch_size to 48. The original paper of SViTE conducts 600 epochs for each method. Unfortunately, we can only run about 30 epochs for each sparsity level owing to the time and GPU limitation. As a result, the accuracy of pruning ViT on CIFAR10 may be penalized. Table of detailed hyperparameters for training is listed below.

Here we compare the top 1 and top 5 accuracies of ViT with different sparsity level (0.1-0.9) on the CIFAR10 test data. We prune the pretrained ViT model with the method of SViTE and compare the accuracy on CIFAR10 with methods of OMP, GMP and TP. The detailed training configurations are listed in Table 2.

Table 1: Train with SViTE: Hyperparameters

Hyperparameters	values
model	vit_base_patch16_224
batchsize	48
learning rate	5e-4
criterion	LabelSmoothingCrossEntropy
optimizer	AdamW
GPU	GeForce GTX 1080 Ti
growth_mode	gradient
death_mode	magnitude
t_end	0.8
death_rate α	0.5

Table 2: Results of different methods on CIFAR10 with different sparsity level

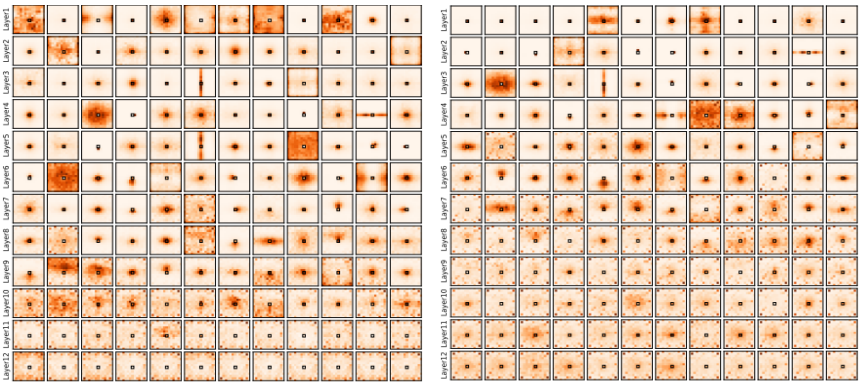
Method	Sparsity Level	Parameters	Top1 Accuracy(%)	Top5 Accuracy(%)
SViTE	0.1 (35 epochs)	8.6M	56.08	94.39
	0.2 (30 epochs)	17.2M	57.43	94.97
	0.3 (35 epochs)	25.8M	58.08	95.18
	0.4 (30 epochs)	34.4M	71.90	97.85
	0.5 (30 epochs)	43.0M	72.84	98.05
	0.6 (30 epochs)	51.6M	84.98	98.80
	0.7 (30 epochs)	60.2M	93.83	99.62
	0.8 (30 epochs)	68.8M	97.30	99.94
	0.9 (20 epochs)	77.4M	97.89	99.96
OMP	0.5 (paper)	43.0M	80.26	-
GMP	0.5 (paper)	43.0M	80.79	-
TP	0.5 (paper)	43.0M	80.55	-

The table provides an insightful analysis of the SViTE-base model's performance at varying sparsity levels. For sparsity levels ranging from 0.1 to 0.3, the model exhibits relatively lower top1 accuracy, hovering around 57%. However, as the sparsity level increases from 0.4 to 0.6, the pruned model demonstrates substantial improvement, achieving a top1 accuracy of approximately 80%. Impressively, with sparsity levels of 0.7 to 0.9, the pruned model showcases performance comparable to the original ViT model. Notably, by utilizing only 70% of the original ViT parameters, we achieve a remarkable accuracy of 94%. Furthermore, since the CIFAR10 dataset only has 10 categories, the top5 accuracy remains consistently high across different sparsity levels, which contains slight comparative value in this case.
It is worth mentioning that the accuracy obtained in the table may not be the best accuracy of the model. Due to the limit of time and arithmetic, we can only run about 30 epochs for each sparsity level. There is still much room for improvement. Nevertheless, this observation further highlights the effectiveness of our pruning approach in preserving model performance while reducing computational requirements.

1.5.2 Comments on SViTe

With the SViTE pruning method, we reach a promising accuracy of sparse ViT model on the CIFAR10 test dataset, especially models with sparsity level above 0.6. With only 60% parameters of the original pretrained ViT model, our sparse model can also achieve high accuracy with only few minor penalties. With such big proportion of saved parameters, we can save lots of time and algorithm power, making it possible to deploy ViT on mobile devices.

Inspired by SViTE, we utilize a pruning approach that dynamically extracts and trains sparse sub-networks instead of training complete models. By adhering to a predetermined, limited parameter budget, our technique simultaneously optimizes model parameters and explores connectivity during the entire training process.
In Chen et al. [2021], the authors implemented a visualization that shows the SViTE principle very well. They compared the attention probabilities for DeiT-Base(an enhanced ViT model) and SViTE-Base with 40% Sparsity over 100 test samples from ImageNet-1K. The results are showed in the form of heatmap where darker colors mean larger values. With regard to SViTE-Base's visual results, it seems to activate fewer attention heads for predictions, compared to the ones of DeiT-Base. They also observe that in the bottom layers, the attention probabilities are more centered at several heads; while in the top layers, the attention probabilities are more uniformly distributed. This visualization explains well that SViTE-Base reduces the probability weight of the non-essential attention head by pruning and retains the critical part.



(a) Attention probabilities for DeiT-Base (b) Attention probabilities for SViTE-Base with 40% Sparsity

Figure 8: Heatmap of Attention probabilities for DeiT-Base and SViTE-Base with 40% Sparsity

References

T. Chen, Y. Cheng, Z. Gan, L. Yuan, L. Zhang, and Z. Wang. Chasing sparsity in vision transformers: An end-to-end exploration, 2021.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1398–1406, 2017. doi: 10.1109/ICCV.2017.155.

Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming, 2017.

P. Michel, O. Levy, and G. Neubig. Are sixteen heads really better than one?, 2019.

D. C. Mocanu, E. Mocanu, P. Stone, P. H. Nguyen, M. Gibescu, and A. Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications*, 9(1), jun 2018. doi: 10.1038/s41467-018-04316-3. URL `https://doi.org/10.1038%2Fs41467-018-04316-3`.

P. Molchanov, A. Mallya, S. Tyree, I. Frosio, and J. Kautz. Importance estimation for neural network pruning, 2019.

A. Roy, M. Saffar, A. Vaswani, and D. Grangier. Efficient content-based sparse attention with routing transformers, 2020.

Y. Tang, K. Han, Y. Wang, C. Xu, J. Guo, C. Xu, and D. Tao. Patch slimming for efficient vision transformers, 2022.

R. Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019.

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

M. Zhu, Y. Tang, and K. Han. Vision transformer pruning, 2021.