ICLR

Github Repo

Dynamic Layer Tying for Parameter- Efficient Transformers

Tamir David-Hay, Lior Wolf

School of Computer Science, Tel Aviv University

Motivation

The motivation behind "Dynamic Layer Tying for Parameter-Efficient Transformers" lies in the quest to reduce the number of trainable parameters in deep transformer networks.

Pruning can be used to reduce the number of FLOPs of transformers during inference time at least by half, with little effect on accuracy (Kurtic et al., 2022; Kwon et al., 2022).

Attention heads can be removed post-training with little effect on performance (Michel et al., 2019; Voita et al., 2019).

Layers can be dropped altogether during inference (Fan et al., 2019; Sajjad et al., 2020).

Attention scores can be reused (Bhojanapalli et al., 2021).

Method

Our method employs Reinforcement Learning to dynamically tie transformer layers during training, significantly reducing trainable parameters while maintaining or enhancing model performance.

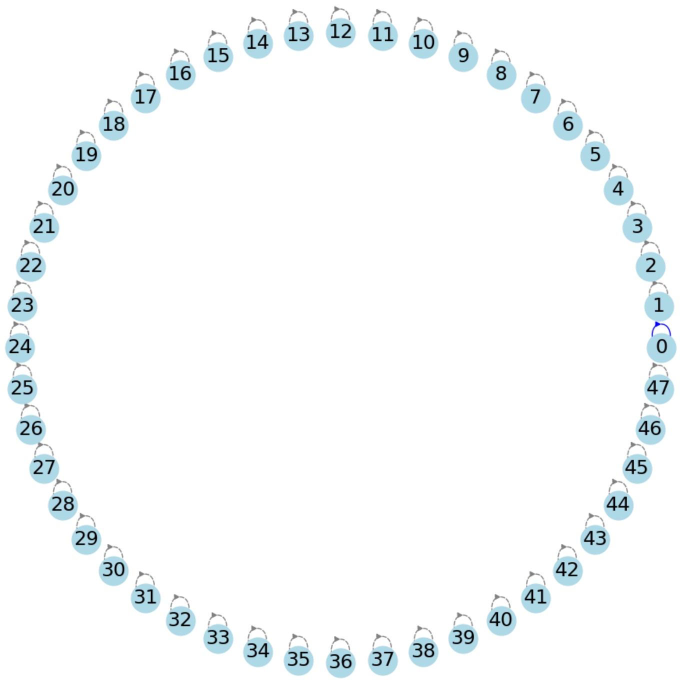
Algorithm 1 Q-learning driven dynamic layer tying

Require: L the number of layers, K the number of training steps of $\mathcal{T}$, k the number of training steps between the update and evaluation of $\mathcal{Q}$, γ the discount factor, and ϵ initial exploration probability

Initialize the primary model $\mathcal{T}$ and the Q-network $\mathcal{Q}$
Freeze layers 1 to $L-1$ in $\mathcal{T}$, such that only layer 0 trains at initialization.
Initialize $\boldsymbol{s} = \boldsymbol{a} = 0$ ▷ An all zero vector
for step $= 0$ to $K-1$ **do**
    Sample a mini-batch B from the dataset
    Perform a training step with $\mathcal{T}$ on B
    if mod(step,k) == 0 **then** ▷ Every k steps
        Obtain an action vector $\boldsymbol{a} = \pi(\boldsymbol{s})$
        Compute $\boldsymbol{s}'$ based on $\boldsymbol{a}$ ▷ Eq. 1
        for $i = 0$ to $L-1$ **do**
            if $\boldsymbol{s}'_i \neq \boldsymbol{s}_i$ **then**
                if $\boldsymbol{s}'_i == i$ **then**
                    Untie layer i of $\mathcal{T}$ ▷ Copy its weights and update it independently of layer $\boldsymbol{s}_i$
                else
                    Replicate all weights of layer $\boldsymbol{s}'_i$ of $\mathcal{T}$ to layer i of $\mathcal{T}$
                    Tie the weights of layer i to layer $\boldsymbol{s}'_i$
                end if
            end if
        end for
        Sample a mini-batch B from the data-set
        r_{step} = Compute negative PPL score based on $\mathcal{T}$ on B
        $r_{predicated}$ = $\mathcal{Q}(\boldsymbol{s}, \boldsymbol{a})$ ▷ Eq. 3
        $r = r_{step} + \gamma * \max_a \mathcal{Q}(\boldsymbol{s}')_{\boldsymbol{a}}$
        $L = MSE(r_{predicted}, r)$
        update $\mathcal{Q}$ using L
        $\boldsymbol{s} = \boldsymbol{s}'$
        $\epsilon = \max\{\epsilon * 0.95, 0.1\}$
    end if
end for

The state space is defined by a vector where each element represents the lowest-index layer whose weights are tied to the layer.
The action space is described by a vector, parallel to the state vector, where each element determines from which previous layer the weights should be copied.

The initial state has all layers except the first one frozen.

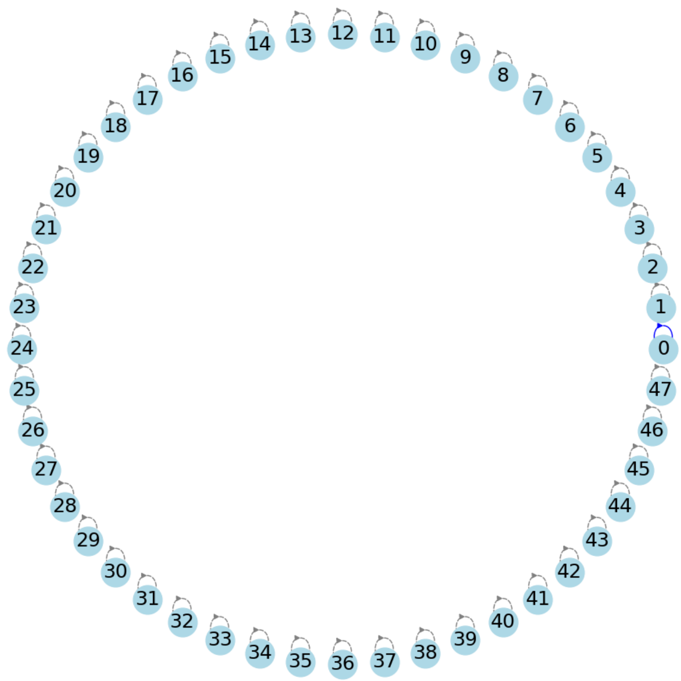


Each action represents the probability of tying a layer with a previous one using a lower triangle matrix of probabilities generated by the Q network.

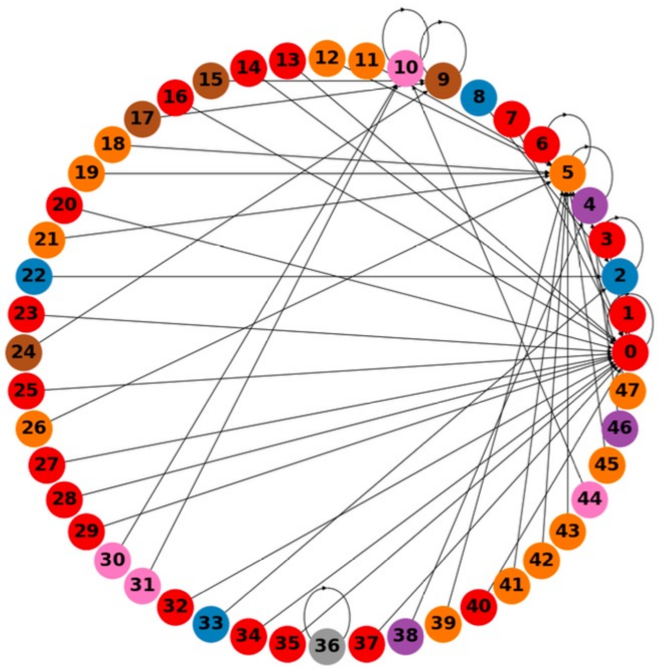
A layer pointing to itself means that the layer is trainable.

Q- network

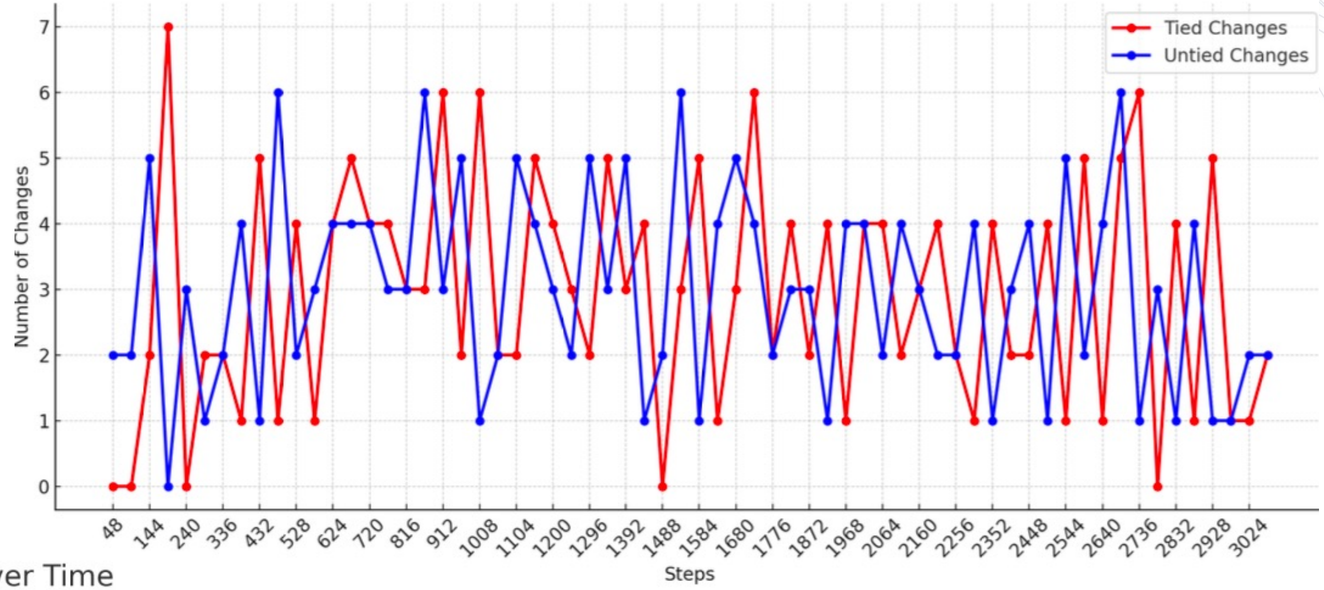
$$action = \begin{pmatrix} 1.0 & 0.0 & \cdots & 0.0 \\ 0.3 & 0.7 & \cdots & 0.0 \\ \vdots & \vdots & \ddots & \vdots \\ 0.03 & 0.1 & \cdots & 0.05 \end{pmatrix}$$

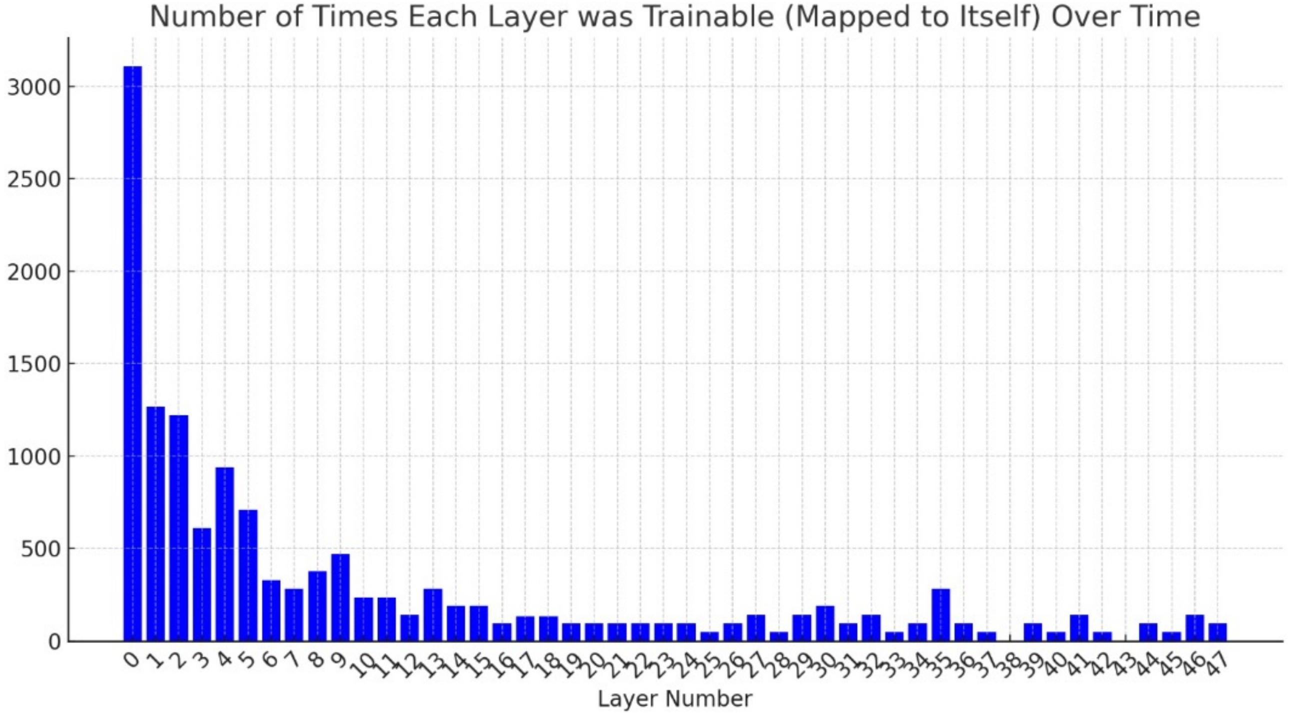


Apply action and tie weights



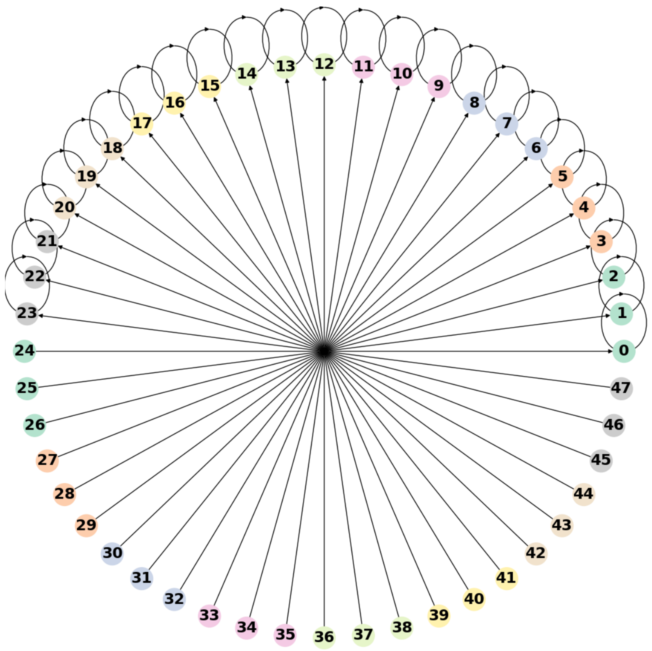
Throughout training, layers frequently changed states between being tied or untied, with every layer experiencing being trainable at different points, ensuring dynamic adaptability without diminishing frequency over time.

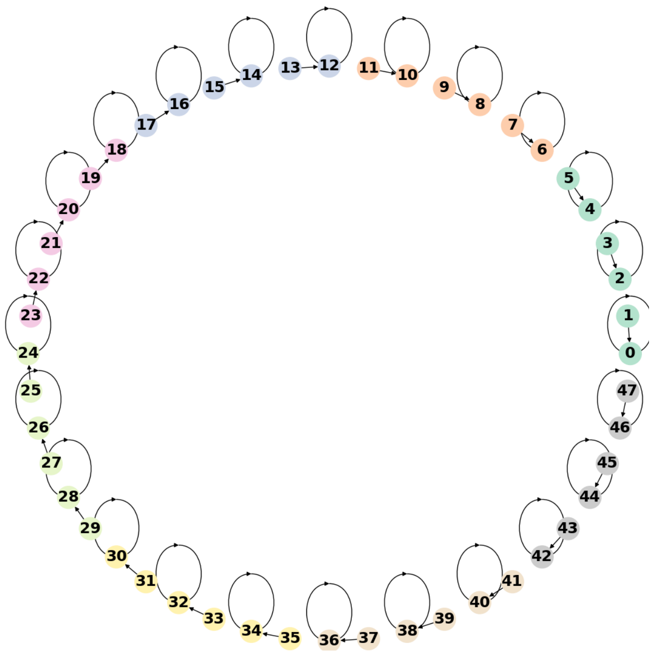


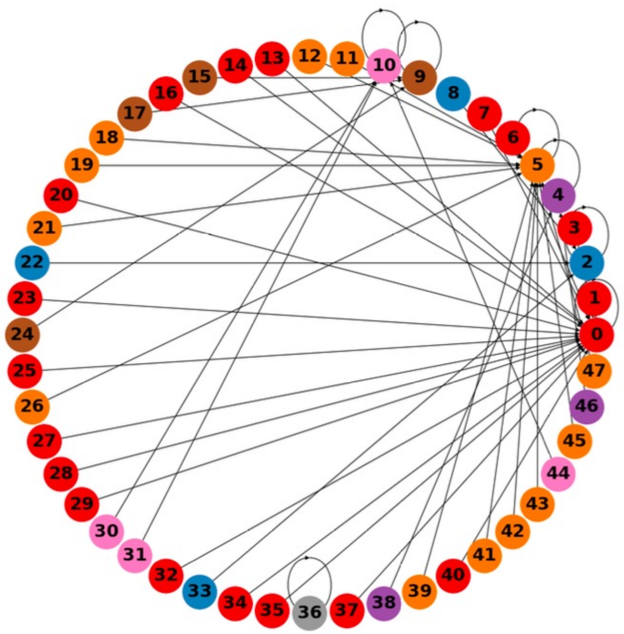


Experiments

we compared the full method with variations, such as training all epochs using the final architecture, applying recorded dynamics to different layers, and training without weight tying, demonstrating the critical role of the proposed dynamic architecture changes.







Results

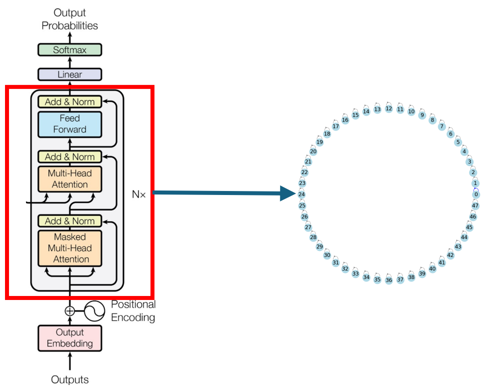
The results show that our method achieves up to a tenfold reduction in memory consumption and maintains or improves model accuracy with at least 75% replication of transformer layers and carries over to downstream tasks as well as image classification tasks.

Metric	Method	Training set			
		Wiki-2	Wiki-103	Lambada	1-billion
Perplexity	Conventional training	53.57	**22.32**	94.96	88.35
	Our method	**49.37**	22.35	**93.84**	**72.35**
Number of trainable parameters	Conventional training	1.6B	1.6B	1.6B	1.6B
	Our method mean over training	171M	151M	166M	218M
	Our method at end of training	264M	142M	326M	203M
Number of independent layers	Conventional training	48	48	48	48
	Our method mean over training	4.395	2.309	3.547	4.486
	Our method at end of training	7	6	9	10

Metric	Conventional	Our
SST-2 (Accuracy)	0.811	0.799
Cola (Accuracy)	0.691	0.691
QNLI (Accuracy)	0.608	0.599
MRPC (Accuracy)	0.697	0.697
RTE (Accuracy)	0.527	0.541
# trainable params	1.5B	235M
# trainable layers	48	5

Statistics	Conventional training	Our method
Peak memory	12,566.66 MB	4,514.31 MB
Average memory consumption	10,223.08 MB	3,395.16 MB

Metric	ViT	Our
Accuracy	0.999	0.995
# trainable params (mean)	630M	80M
# trainable params (end of training)	630M	139M
# trainable layers (mean)	32	5.5
# trainable layers (end of training)	32	7



Q-
network

$$action = \begin{pmatrix} 1.0 & 0.0 & \cdots & 0.0 \\ 0.3 & 0.7 & \cdots & 0.0 \\ \vdots & \vdots & \ddots & \vdots \\ 0.03 & 0.1 & \cdots & 0.05 \end{pmatrix}$$