Module-wise Training of Neural Networks via the Minimizing Movement Scheme

Skander Karkar, Ibrahim Ayed, Emmanuel de Bézenac, Patrick Gallinari
Sorbonne University, Criteo, ETH Zurich

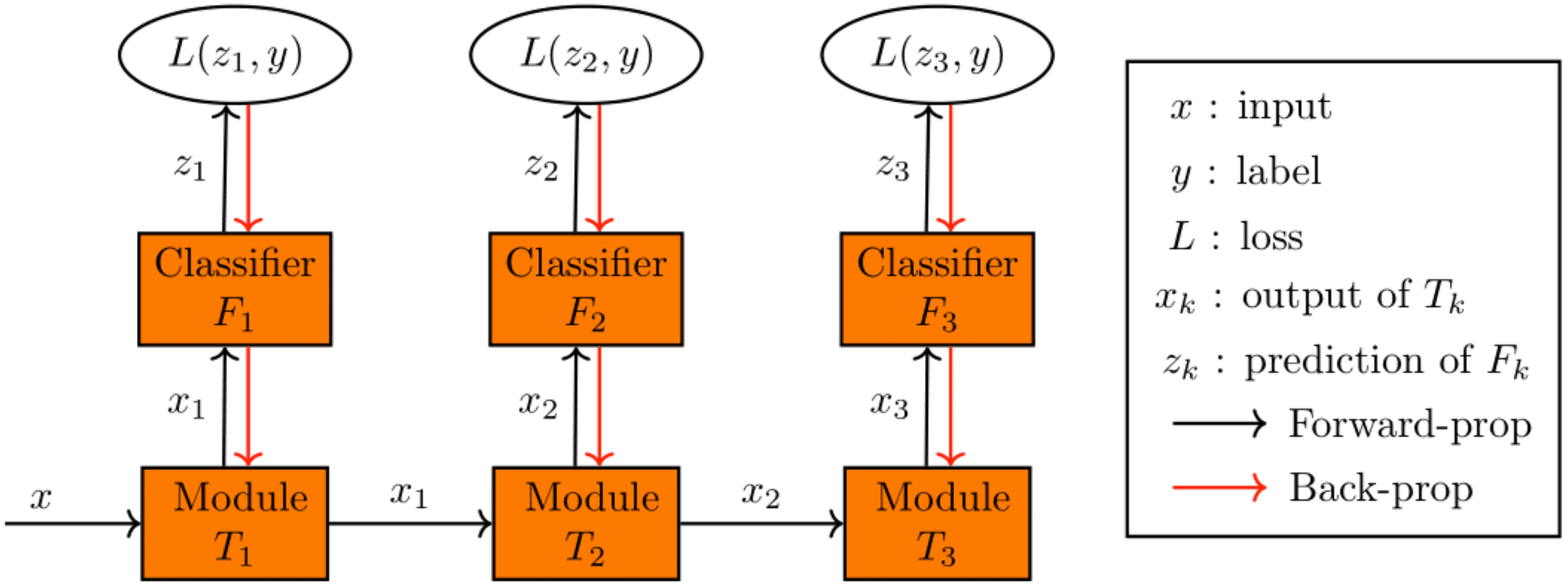
NeurIPS 2023

Introduction

- End-to-end back-propagation requires storing the whole model and computational graph during training, which requires large memory consumption.
- It also prohibits training the layers in parallel because of its locking problems:
  - Forward locking: each layer must wait for the previous layers to process its input.
  - Update locking: each layer must wait for the end of the forward pass to be updated.
  - Backward locking: each layer must wait for errors to back-propagate from the last layer to be updated.

Introduction

- Module-wise training splits the network into successive modules, a module being made up of one or more layers. Each module takes as input the output of the previous module.
- Each module has an auxiliary classifier so that a local loss can be computed, with back-propagation happening only inside the modules and not between them.

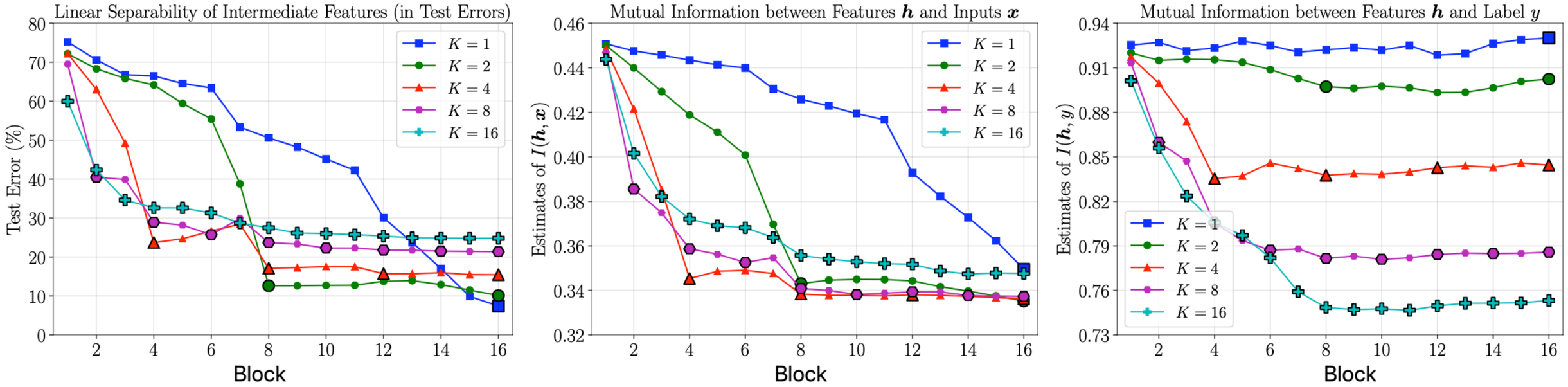


Introduction

- Module-wise training consumes less memory than end-to-end training as it stores less activations.
- It has therefore been used in constrained setting for training on mobile devices and dealing with very large whole slide images.
- It also solves update locking (and therefore also backward locking). When combined with batch buffers, module-wise training solves all three problems and allows parallel training of the modules.
- Despite its simplicity, it outperforms more complicated approaches to solve the locking problems such as delayed or synthetic gradients.

Introduction

- Module-wise training suffers however from a stagnation problem, whereby early modules overfit and learn more discriminative features than end-to-end training, destroying task-relevant information, and deeper modules don't improve the test accuracy significantly, or even degrade it.



Linear separability (left), mutual information with input (center), and with output (right) of the features at different depths learned with module-wise training with K modules. The experiment is training a ResNet-32 on CIFAR10. K=1 is end-to-end training. K=2 means training two modules with 16 blocks each. From [13].

Introduction

- To tackle this issue, InfoPro [13] propose to maximize the mutual information that each module keeps with the input, in addition to minimizing the loss, while Sedona [12] make the first module deeper.
- Most methods add a second auxiliary network besides the classifier to each module to compute a second term, which limits the memory savings of module-wise training.
- We use the transport regularization from the previous section to regularize the modules, and do not need to implement another auxiliary network.
- We show that the regularization makes every module act as proximal optimization step in the Wasserstein distance for maximizing the separability of the data embedding, therefore explaining why it avoids the stagnation or collapse in accuracy.

Method

Vanilla module-wise training

- We denote the modules T_k and the classifiers F_k,
- $G_k = T_k \circ \ldots \circ T_1$ is the composition of all the modules until depth k for $1 \leq k \leq K$, and $G_0 = \text{id}$.
- The typical setting of module-wise training for minimizing a loss L, is, given a dataset $\mathscr{D}$, to solve one after the other, for $1 \leq k \leq K$:

$$(T_k, F_k) \in \arg\min_{T,F} \sum_{x \in \mathscr{D}} L(F, T(G_{k-1}(x)) \tag{1}$$

- The final network trained this way is $F_K \circ G_K$, but we can stop and use $F_k \circ G_k$ for $1 \leq k < K$ if it performs better.

Method

Transport-regularized module-wise training

- We add the kinetic energy to the loss in the target of problems (1)

$$(T_k^\tau, F_k^\tau) \in \arg\min_{T,F} \sum_{x \in \mathscr{D}} L(F, T(G_{k-1}^\tau(x)) + \frac{1}{2\tau} \|T(G_{k-1}^\tau(x)) - G_{k-1}^\tau(x)\|^2 \tag{2}$$

- This problem is a discretization of

$$(T_k^\tau, F_k^\tau) \in \arg\min_{T,F} \ \mathscr{L}(F, T_\sharp \rho_k^\tau) + \frac{1}{2\tau} \int_\Omega \|T(x) - x\|^2 \mathrm{d}\rho_k^\tau(x) \tag{3}$$

where $\rho_{k+1}^\tau = (T_k^\tau)_\sharp \rho_k^\tau$ and $\rho_1^\tau = \rho$ is the input distribution of which $\mathscr{D}$ is a sample.

Method

Theoretical results

- Consider the Wasserstein space $\mathbb{W}_2(\Omega)$, i.e. the space of probability densities over $\Omega \subset \mathbb{R}^d$ equipped with the Wasserstein distance W_2.
- **Definition.** Given $\mathscr{Z} : \mathbb{W}_2(\Omega) \to \mathbb{R}$ the minimizing movement scheme is a discretized gradient flow that is well-defined in metric spaces and minimizes under some conditions $\mathscr{Z}$ starting from $\rho_1^\tau \in \mathscr{P}(\Omega)$. It is given by

$$\rho_{k+1}^\tau \in \arg\min_{\rho \in \mathscr{P}(\Omega)} \mathscr{Z}(\rho) + \frac{1}{2\tau} W_2^2(\rho, \rho_k^\tau) \tag{4}$$

- **Proposition.** The distributions $\rho_{k+1}^\tau = (T_k^\tau)_\sharp \rho_k^\tau$ where functions T_k^τ are found by solving problems (3) coincide with the minimizing movement scheme (4) for $\mathscr{Z} = \min_F \mathscr{L}(F, .)$.
- In other terms, we are following the minimizing movement scheme which is a discrete optimisation algorithm for following the gradient flow of functionals on probability distributions, and which converges under some conditions to a solution as $\tau \to 0$ and $k \to \infty$.
- In other terms, this can be seen as taking Wasserstein proximal steps to minimize $\mathscr{Z}$, which is the property we want in module-wise training (modules building upon each other to minimize the loss).

Method

Theoretical results

- We also show existence and regularity results for the modules solving problems (3).
- **Theorem.** Problem (3) has a minimizer (T_k^τ, F_k^τ) such that T_k^τ is an optimal transport map. And for any minimizer (T_k^τ, F_k^τ), T_k^τ is an optimal transport map.
- **Corollary.** T_k^τ is η-Hölder continuous a.e. and if the optimization algorithm returns an approximate solution pair $(\tilde{F}_k^\tau, \tilde{T}_k^\tau)$ such that $\tilde{T}_k^\tau$ is an ϵ-optimal transport map, i.e. $\|\tilde{T}_k^\tau - T_k^\tau\|_\infty \leq \epsilon$, then for almost every $x, y \in \text{supp}(\rho_k^\tau)$ and a constant $C > 0$, we have

$$\|\tilde{T}_k^\tau(x) - \tilde{T}_k^\tau(y)\| \leq 2\epsilon + C\|x - y\|^\eta$$

Method

Multi-block modules

- For simplicity, we presented in (2) the case where each module is a single residual block. However, in practice, we often split the network into modules that contain many residual blocks each.
- If each module T_k is made up of M residual blocks, and we denote ϕ_m^x the position of a point x after m blocks, then regularizing the kinetic energy of multi-block modules means solving

$$(T_k^\tau, F_k^\tau) \in \arg\min_{T,F} \sum_{x \in \mathcal{D}} (L(F, T(G_{k-1}^\tau(x)) + \frac{1}{2\tau} \sum_{m=0}^{M-1} \|r_m(\phi_m^x)\|^2) \tag{4}$$

$$\text{s.t. } T = (\text{id} + r_{M-1}) \circ \ldots \circ (\text{id} + r_0),\ \phi_0^x = G_{k-1}^\tau(x)$$

$$\phi_{m+1}^x = \phi_m^x + r_m(\phi_m^x)$$

Method

Multi-block modules

- Problems (4) are the discretization of

$$(T_k^\tau, F_k^\tau) \in \arg\min_{T,F} \ \mathscr{L}(F, T_\sharp \rho_k^\tau) + \frac{1}{2\tau} \int_0^1 \|v_t\|^2_{L^2((\phi_t^\cdot)_\sharp \rho_k^\tau)} \mathrm{d}t \tag{5}$$

$$\text{s.t. } T = \phi_1^\cdot,\ \partial_t \phi_t^x = v_t(\phi_t^x),\ \phi_0^\cdot = \mathrm{id}$$

- We recognize in the second term in the target of (5) the optimal transport problem in its dynamical form.
- Therefore the previous theoretical results still apply.

Method

Varying the regularization weight

- The theoretical discussion in suggests taking τ as small as possible. However, instead of using a fixed τ, we might want to vary it along the depth k to further constrain with a smaller τ_k the earlier modules to avoid that they overfit or the later modules to maintain the accuracy of earlier modules.
- We might also want to regularize the networks further in earlier epochs when the data is more entangled as in the previous section.
- To unify and formalize this varying weight $\tau_{k,i}$ across modules k and SGD iterations i, we use a scheme inspired by the method of multipliers as in the previous section.
- We instead consider the weight $\lambda_{k,i} = 2\tau_{k,i}$ given to the loss. We denote $\theta_{k,i}$ the parameters of both T_k and F_k at SGD iteration i. We denote $L(\theta, x)$ and $C(\theta, x)$ respectively the loss and the transport cost as functions of parameters θ and data x.

$$\begin{cases} \theta_{k,i+1} & = \theta_{k,i} - \eta_i \nabla_\theta (\lambda_{k,i}\ L(\theta_{k,i}, x_i) + C(\theta_{k,i}, x_i)) \\ \lambda_{k,i+1} & = \lambda_{k,i} + hL(\theta_{k,i+1}, x_{i+1}) \text{ if } i \text{ mod } s = 0 \text{ else } \lambda_{k,i} \end{cases}$$

Method

Varying the regularization weight

- The weights $\lambda_{k,i}$ will vary along modules k even if we use the same initial weights $\lambda_{k,1} = \lambda_1$ because they will evolve differently with iterations i for each k. They will increase more slowly with i for larger k because deeper modules will have smaller loss.
- We use the same initial value $\lambda_1 = \lambda_{k,1}$ for all modules so this method requires choosing three hyper-parameters (h, s and λ_1).
- In practice, it works best in only one experiment. Its dynamics suggest manually finding a value of τ for the first half of the modules and multiplying it by 2 for the second half, which works best in all the other experiments.

Method

Solving the module-wise problems

- The module-wise problems can be solved in one of two ways. One can completely train each module with its auxiliary classifier for N epochs before training the next module, which receives as input the output of the previous frozen trained module. We call this sequential module-wise training.
- We can also do this batch-wise, i.e. do a complete forward pass on each batch but without a full backward pass, rather a backward pass that only updates the current module T_k^τ and its auxiliary classifier F_k^τ, meaning that T_k^τ forwards its output to T_{k+1}^τ immediately after it computes it. We call this parallel module-wise training.
- Combining it with batch buffers solves all locking problems and allows a linear training parallelization in the depth [14].
- We propose a variant of sequential module-wise training that we call multi-lap sequential module-wise training, in which instead of training each module for N epochs, we train each module from the first to the last sequentially for N/R epochs, then go back and train from the first module to the last for N/R epochs again, and we do this for R laps.
- For the same total number of epochs and time, and the same advantages (loading one module at a time) this provides a non-negligible improvement in accuracy over normal sequential module-wise training in many cases.
- Despite our theoretical framework being that of sequential module-wise training, our method improves the test accuracy of all three module-wise training regimes.

Experiments

- We focus first on parallel module-wise training as it performs better and has been more explored in recent works.
- The methods we compare to are:
  - InfoPro and its variant InfoProL, which maximize the mutual information each module keeps with the input, therefore requiring an additional auxiliary layer besides the classifier.
  - Sedona, which uses an architecture search phase to decide where to split the network into modules and which auxiliary classifier to use. This leads to larger early modules, reducing the memory savings of module-wise training.
  - DGL, which only focuses on the architecture of the auxiliary classifier, and whose auxiliary classifier we use.
  - PredSim, which adds a similarity matching loss to the loss of every module, also requiring an additional auxiliary layer.
  - DDG and FR, which are delayed gradient methods that aim to break the locking problems for parallelization and not for saving memory.
- We call our method TRGL for transport-regularized greedy learning and include the results of vanilla module-wise training without the regularization (called VanGL) for ablation study purposes.
- We have better test accuracy than the other methods, and except on Transformers, than end-to-end training, with as much as 60% less memory usage.
- We then run experiments on sequential module-wise training with each module being a single residual block, which allows for the largest memory savings, as only one block and its classifier have to be loaded at a time.

Experiments

Parallel module-wise training

Dataset	Architecture	K	Parallel VanGL	Parallel TRGL (ours)	PredSim	DGL	Sedona	E2E
TinyImageNet	VGG-19	4	56.17 ± 0.29 (↓ 27%)	**57.28** ± 0.20 (↓ 21%)	44.70	51.40	56.56	58.74
	ResNet-50	4	58.43 ± 0.45 (↓ 26%)	**60.30** ± 0.58 (↓ 20%)	47.48	53.96	54.40	58.10
	ResNet-101	4	63.64 ± 0.30 (↓ 24%)	**63.71** ± 0.40 (↓ 11%)	53.92	53.80	59.12	62.01
	ResNet-152	4	63.87 ± 0.16 (↓ 21%)	**64.23** ± 0.14 (↓ 10%)	51.76	57.64	64.10	62.32

Accuracy of vanilla and transport regularized module-wise training with 4 modules, and memory savings compared to end-to-end training in red.

Experiments

Parallel module-wise training

Architecture	Dataset	K	Parallel VanGL	Parallel TRGL (ours)	InfoPro	InfoProL	E2E
Swin-Tiny	STL10	4	67.00 ± 1.36 (↓ 55%)	**67.92** ± 1.12 (↓ 50%)	64.61 (↓ 38%)	66.89 (↓ 45%)	72.19
	CIFAR10	4	83.94 ± 0.42 (↓ 33%)	**86.48** ± 0.54 (↓ 29%)	83.38 (↓ 38%)	86.28 (↓ 45%)	91.37
	CIFAR100	4	69.34 ± 0.91 (↓ 33%)	**74.11** ± 0.31 (↓ 29%)	68.36 (↓ 38%)	73.00 (↓ 45%)	75.03

Accuracy of vanilla and transport regularized module-wise training with 4 modules, and memory savings compared to end-to-end training in red.

Experiments

Parallel module-wise training

Dataset	Architecture	K	Par VanGL	Par TRGL (ours)	DGL	InfoPro Softmax	InfoPro Contrast	E2E
STL10	ResNet-110	2	79.85 ± 0.93	**80.04** ± 0.85	75.03 ± 1.18	78.98 ± 0.51	79.01 ± 0.64	77.73 ± 1.61
		4	77.11 ± 2.31	**79.72** ± 0.81	73.23 ± 0.64	78.72 ± 0.27	77.27 ± 0.40	77.73 ± 1.61
		8	75.71 ± 0.55	**77.82** ± 0.73	72.67 ± 0.24	76.40 ± 0.49	74.85 ± 0.52	77.73 ± 1.61
		16	73.57 ± 0.95	**77.22** ± 1.20	72.27 ± 0.58	73.95 ± 0.71	73.73 ± 0.48	77.73 ± 1.61

Accuracy of vanilla and transport regularized module-wise training with K modules

Experiments

Memory savings

- As seen above, parallel TRGL is lighter than end-to-end training by up to almost 60%. The extra memory consumed by our regularization compared to parallel VanGL is between 2 and 13% of end-to-end memory. Memory savings depend then mainly on the size of the auxiliary classifier, which can easily be adjusted.
- So far, the modules were of equal depth, which makes the first module the heaviest. Dividing the network into modules that weight the same leads to much larger memory savings, and a performance that is still better than end-to-end training when using 4 modules.

	Equally deep modules			Equally heavy modules		
K	Par VanGL	Par TRGL (ours)	InfoPro	Par VanGL*	Par TRGL* (ours)	InfoPro*
4	27% (77.11)	24% (**79.72**)	18% (78.72)	41% (77.14)	39% (**78.94**)	33 % (78.78)
8	50% (75.71)	48% (**77.82**)	37% (76.40)			
16	61% (73.57)	58% (**77.22**)	59% (73.95)			

Memory savings of module-wise training on STL10 as a percentage of end-to-end training

Experiments

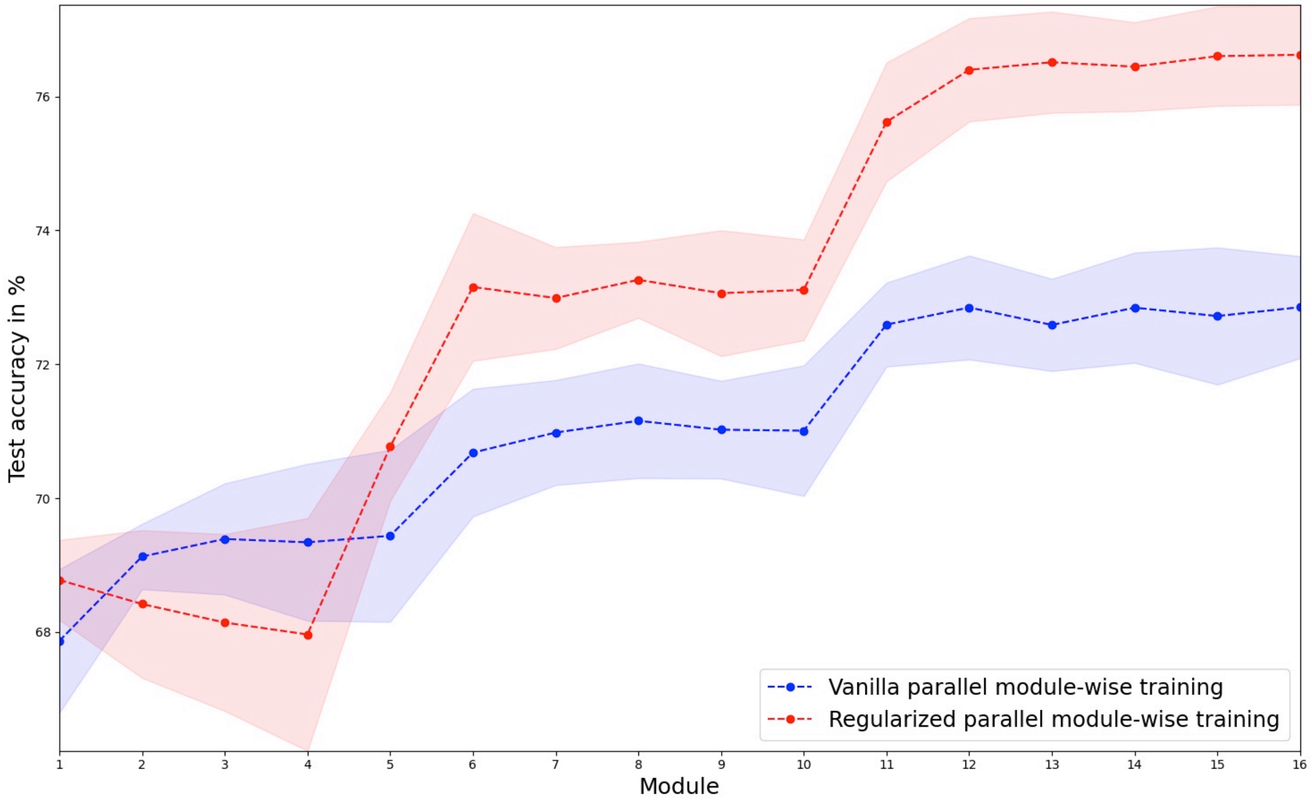
Sequential block-wise training

Train size	Seq VanGL	Seq TRGL	MLS VanGL	MLS TRGL	Par VanGL	Par TRGL	E2E
50000	68.74 ± 0.45	**68.79** ± 0.56	69.48 ± 0.53	**69.95** ± 0.50	72.59 ± 0.40	**72.63** ± 0.40	75.85 ± 0.70
25000	60.48 ± 0.15	**60.59** ± 0.14	61.33 ± 0.23	**61.71** ± 0.32	64.84 ± 0.19	**65.01** ± 0.27	65.36 ± 0.31
12500	51.64 ± 0.33	**51.74** ± 0.26	51.30 ± 0.22	**51.89** ± 0.30	55.13 ± 0.24	**55.40** ± 0.35	52.39 ± 0.97
5000	36.37 ± 0.33	**36.40** ± 0.40	33.68 ± 0.48	**34.61** ± 0.59	39.45 ± 0.23	**40.36** ± 0.23	36.38 ± 0.31

Accuracy of 10-1 ResNet with sequential, MLS and parallel TRGL on CIFAR100

Experiments

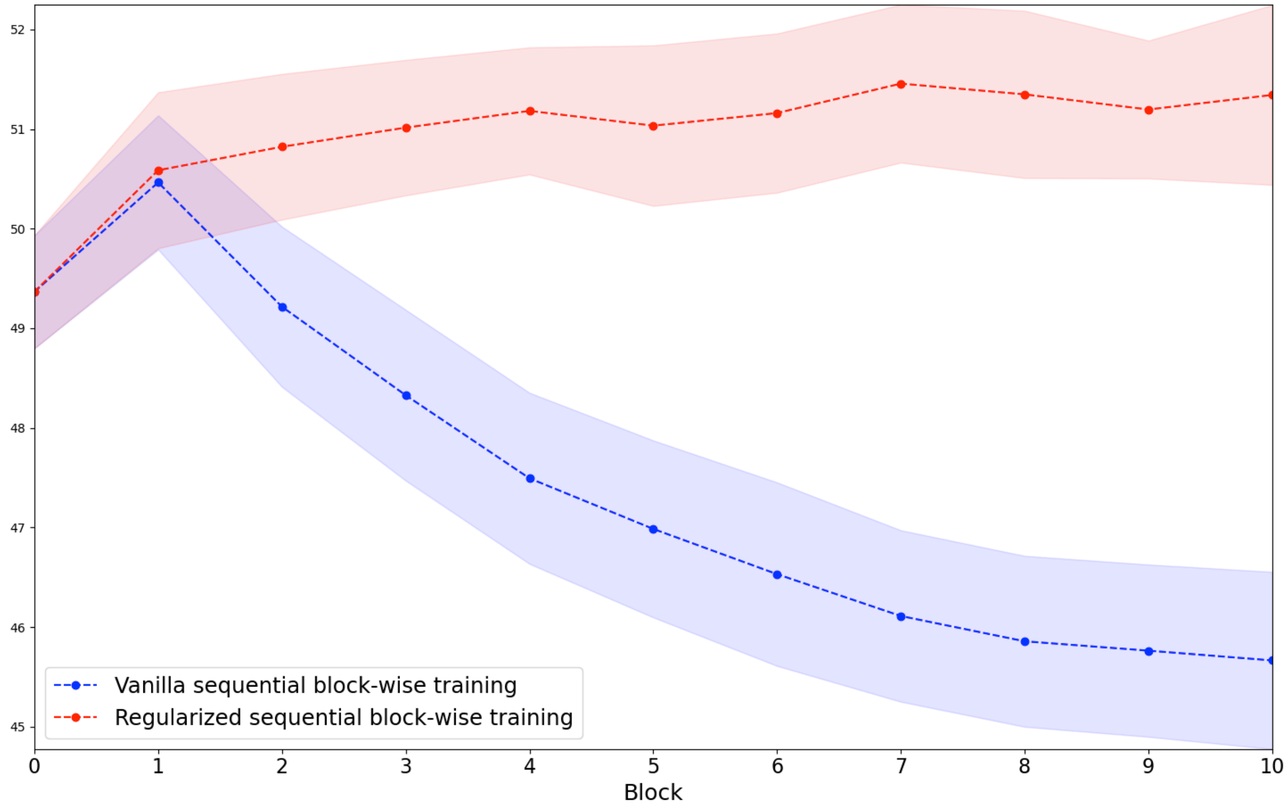
Avoiding early overfitting



Accuracy of each module after parallel module-wise training of a ResNet-110 with 16 modules on STL10

Experiments

Avoiding early overfitting



Accuracy of each module after sequential block-wise training of a 10-block ResNet on 2% of CIFAR10

Conclusion

- We introduced a transport regularization for module-wise training that theoretically links it to gradient flows of the loss in distribution space.
- Our method provably leads to more regular modules and experimentally consistently improves the test accuracy of module-wise and block-wise sequential, parallel and multi-lap sequential (a variant of sequential training that we introduce) training.
- For future work, one can ask how far the obtained composition network G_K is from being an optimal transport map itself, which could provide a better stability bound than the one obtained by naively chaining the stability bounds that follow from each module T_k being an optimal transport map.

References

- [1] Deep Residual Learning for Image Recognition, He et al., CVPR 2016
- [2] Identity Mappings in Deep Residual Networks, He et al. ECCV 2016
- [3] Residual Connections Encourage Iterative Inference, Jastrzebski et al., ICLR 2018
- [4] On Residual Networks Learning a Perturbation from Identity, Hauser, arXiv 2019
- [5] Multi-level Residual Networks from Dynamical Systems View, Chang et al., ICLR 2018
- [6] Highway and residual networks learn unrolled iterative estimation, Greff et al., arXiV 2016
- [7] Batch Normalization Biases Residual Blocks Towards the Identity Function in Deep Networks, De et al., Neurips 2020
- [8] Reversible Architectures for Arbitrarily Deep Residual Neural Networks, Chang et al., AAAI 2018
- [9] A Principle of Least Action for the Training of Neural Networks, Karkar et al., ECML 2020
- [10] Module-wise Training of Neural Networks via the Minimizing Movement Scheme, Karkar et al., AAAI 2022 workshop, submitted to Neurips 2023
- [11] Adversarial Sample Detection Through Neural Network Transport Dynamics, Karkar et al., submitted to ECML 2023
- [12] SEDONA: Search for Decoupled Neural Networks toward Greedy Block-wise Learning, Pyeon et al., ICLR 2021
- [13] Revisiting Locally Supervised Learning: an Alternative to End-to-end Training, Wang et al., ICLR 2021
- [14] Decoupled Greedy Learning of CNNs, Belilovsky et al., ICML, 2020
- [15] Deep neural networks are easily fooled: High confidence predictions for unrecognizable images, Nguyen et al., CVPR 2015
- [16] A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks, Lee et al., Neurips 2018
- [17] Characterizing Adversarial Subspaces Using Local Intrinsic Dimensionality, Xingjun Ma et al., ICLR 2018

References

- [18] PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples, Song et al., ICLR 2018
- [19] Adversarial Spheres: The Relationship Between High-Dimensional Geometry and Adversarial Examples, Gilmer et al., arXiv 2018
- [20] Adversarial Examples Detection in Features Distance Spaces, Carrara et al., ECCV 2018 Workshops
- [21] Towards Robust Detection of Adversarial Examples, Pang et al., Neurips 2018
- [22] Lipschitz regularity of deep neural networks: analysis and efficient estimation, Virmaux et al., Neurips 2018
- [23] Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation}, Hein et al., Neurips 2017
- [24] Parseval Networks: Improving Robustness to Adversarial Examples, Cisse et al., ICML 2017
- [25] A Boundary Tilting Perspective on the Phenomenon of Adversarial Examples, Tanay and Griffin, arXiv 2016
- [26] Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations, Lu et al., ICML 2018