Multirate Training of Neural Networks

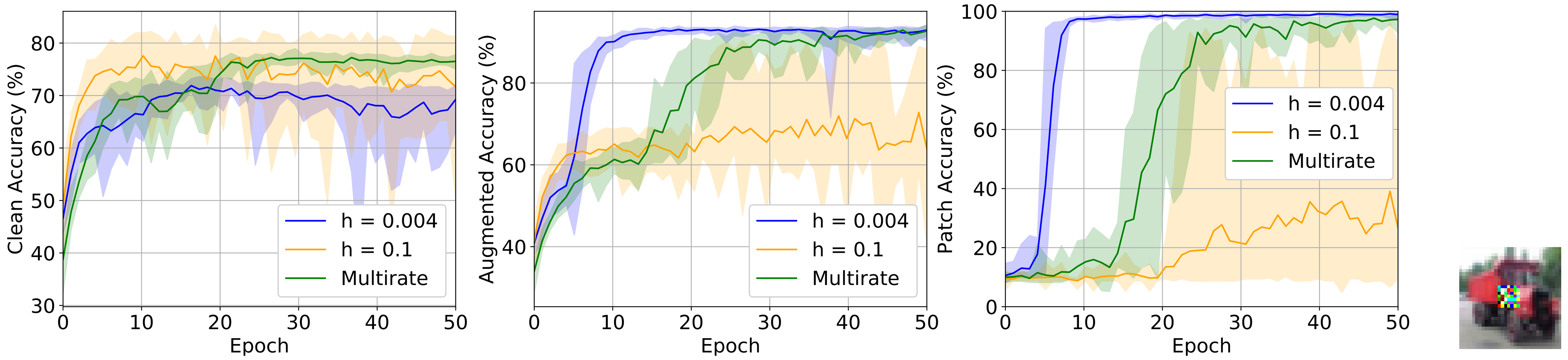
Tiffany Vlaar* and Ben Leimkuhler

ICML 2022

THE UNIVERSITY of EDINBURGH

Motivation

WideResNet-16, patch-augmented CIFAR-10 [training data from Li et al. 2019: 20% patch-free, 16% patch-only, 64% both]. Left: clean validation set. Middle: augmented data with patches. Right: patch-only data.



Multirate scheme (green) trained on both time scales ($h_F = 0.004$, $h_S = 0.1$) is able to memorize the patches **and** obtain high accuracy on the clean data, while fixed learning rate approaches (blue/orange) fail to do both.

Multirate SGD

Notation: parameters $\theta = (\theta_F, \theta_S) \in \mathbb{R}^n$, momenta $p = (p_F, p_S) \in \mathbb{R}^n$, slow learning rate $h_S = h$, fast learning rate $h_F = h/k$, momentum hyperparameter μ, and neural network loss $\mathcal{L}(\theta_S, \theta_F)$ evaluated on a minibatch.

No linear drift:

$$p_S = \mu p_S + \nabla_{\theta_S}\mathcal{L}(\theta_S, \theta_F)$$
$$\theta_S = \theta_S - h p_S$$
for $i = 1, 2, \ldots, k$
$$p_F = \mu p_F + \nabla_{\theta_F}\mathcal{L}(\theta_S, \theta_F)$$
$$\theta_F = \theta_F - \frac{h}{k} p_F$$

With linear drift:

$$p_S = \mu p_S + \nabla_{\theta_S}\mathcal{L}(\theta_S, \theta_F)$$
for $i = 1, 2, \ldots, k$
$$p_F = \mu p_F + \nabla_{\theta_F}\mathcal{L}(\theta_S, \theta_F)$$
$$\theta_F = \theta_F - \frac{h}{k} p_F$$
$$\Rightarrow \quad \theta_S = \theta_S - \frac{h}{k} p_S$$

We find that the use of linear drift further enhances performance for our applications.
In the paper we provide ablation studies and a **convergence analysis compared to vanilla SGD**.

How to Partition?

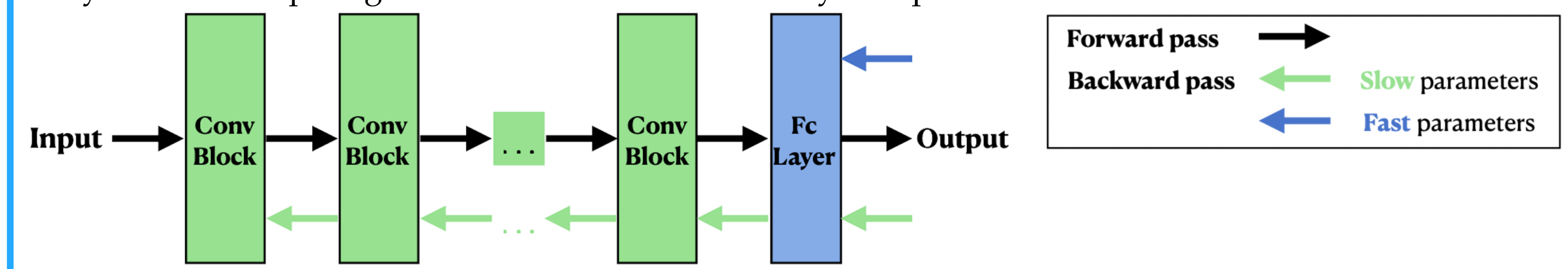
Need to partition neural network parameters into fast and slow parts. You have a choice!

Suggestions:	Inspiration:
• Layer-wise	Are all layers created equal? [Zhang et al., 2019]. Partitioned integrators for NN training [Leimkuhler, Matthews, TV, 2019]. Different layers are more data-hungry [Bansal et al., 2021]. Layerwise sensitivity to initialization & optimizer settings [TV & Frankle, 2021].
• Random subgroups	Dropout [Srivastava et al., 2014] and DropConnect [Wan et al., 2013].

Transfer Learning - Layerwise Partitioning

Inspiration: Early layers capture general features, later layers more task-specific knowledge [Yosinski et al., 2014; Hao et al., 2019; Raghu et al., 2019; Neyshabur et al., 2020]. MD: r-RESPA [Tuckerman et al. 1991, 1992].

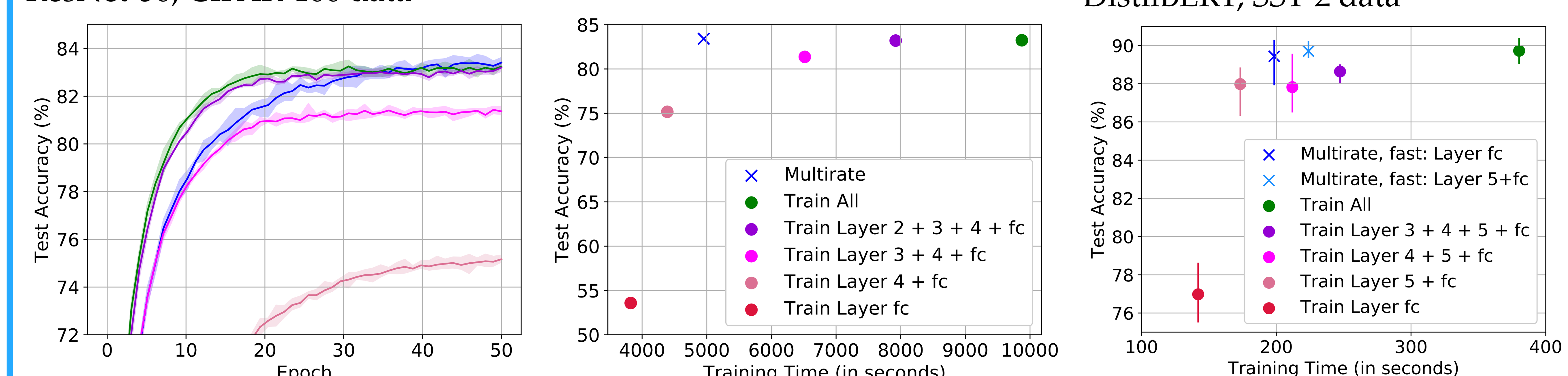
Idea: Get **computational speed-up** by splitting net in two parts: final layer(s) as the fast part, rest is slow part. Only need to compute gradients for full network every k steps!



For example, for a ResNet-34 architecture fast parameters are only 0.024% of total.

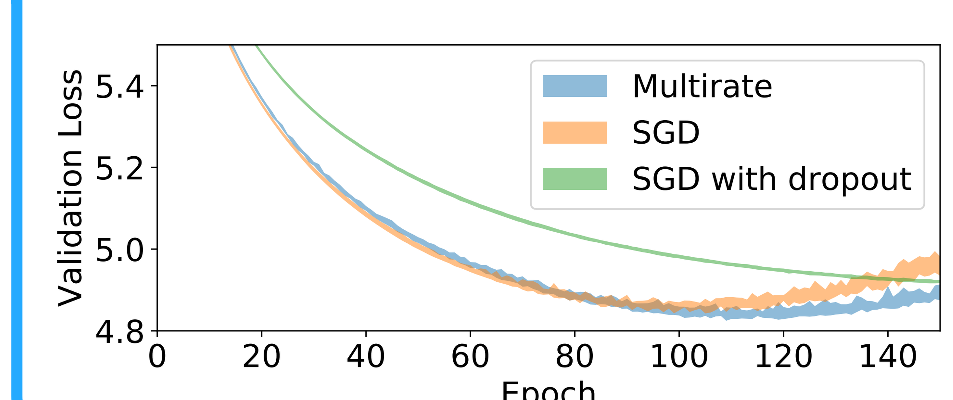
ResNet-50, CIFAR-100 data

DistilBERT, SST-2 data



Can train in **half** the time, while maintaining the same generalization performance!
Complexity analysis and ablation studies in paper.

Regularization - Random Subgroups Partitioning



Procedure for this application:
- θ_S are randomly selected subsets of NN weights.
- Set θ_S to zero during fast parameter update.
- Every k optimization steps randomly select new θ_S.

Small transformer trained on Penn Treebank data.

Future Work

- Other partitioning choices.
- Hybrid training schemes.
- Popular practices.
- Different base algorithm.
- Different data!
- *[Insert your application]*

*Tiffany.Vlaar@ed.ac.uk