THE UNIVERSITY *of* EDINBURGH

Multitrate Training of Neural Networks

Tiffany Vlaar and Ben Leimkuhler

Get in touch at: Tiffany.Vlaar@ed.ac.uk

ICML 2022

Latent Multiple Time Scales in Deep Learning

WideResNet-16 trained on patch-augmented [Li, Wei & Ma, NeurIPS 2019] CIFAR-10 data:

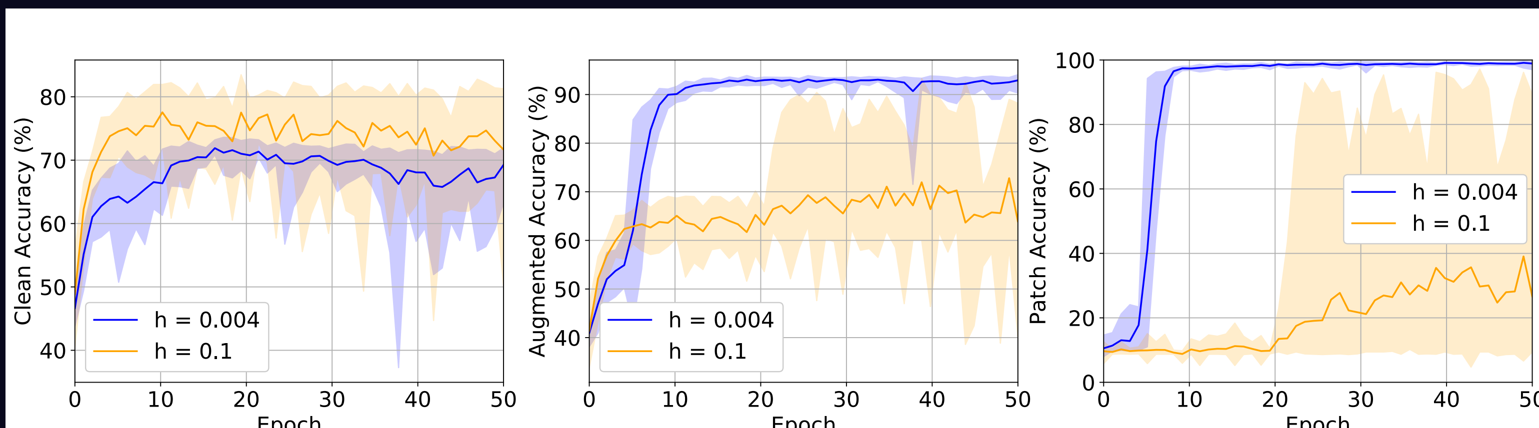
20% is patch-free, 16% has only the patch, and the rest has both data and patch.

Latent Multiple Time Scales in Deep Learning

WideResNet-16 trained on patch-augmented [Li, Wei & Ma, NeurIPS 2019] CIFAR-10 data:

20% is patch-free, 16% has only the patch, and the rest has both data and patch.

Optimizer: SGD with momentum with weight decay.



A net trained using a:

- Small learning rate (blue) memorizes patch.

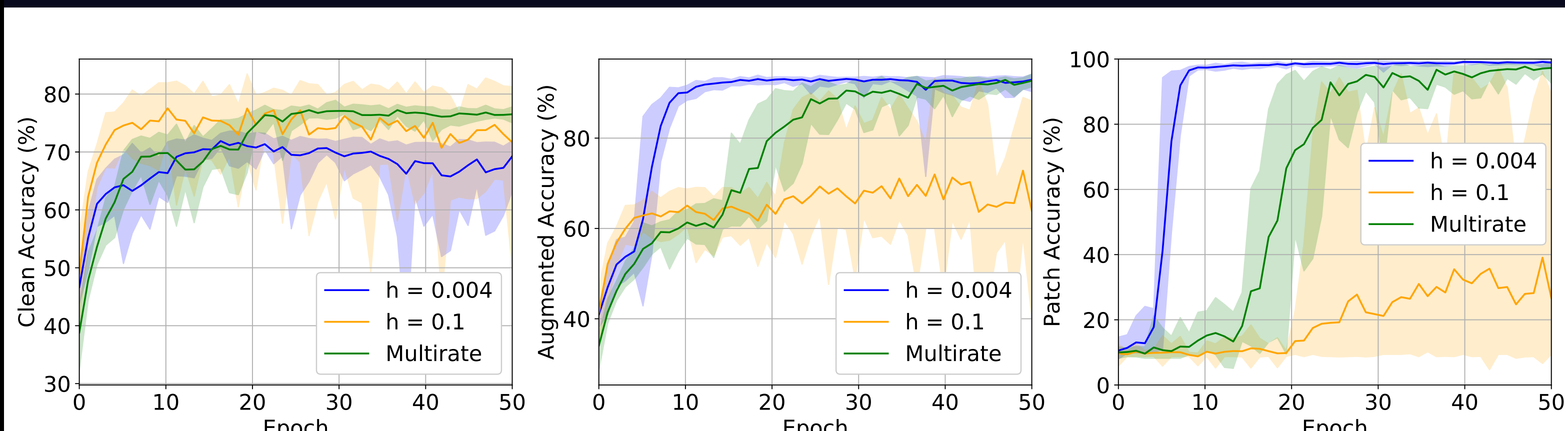
- Large learning rate (orange) gives higher accuracy on clean data.

Latent Multiple Time Scales in Deep Learning

WideResNet-16 trained on patch-augmented [Li, Wei & Ma, NeurIPS 2019] CIFAR-10 data:

20% is patch-free, 16% has only the patch, and the rest has both data and patch.

Optimizer: SGD with momentum with weight decay.



A net trained using a:

- Small learning rate (blue) memorizes patch.

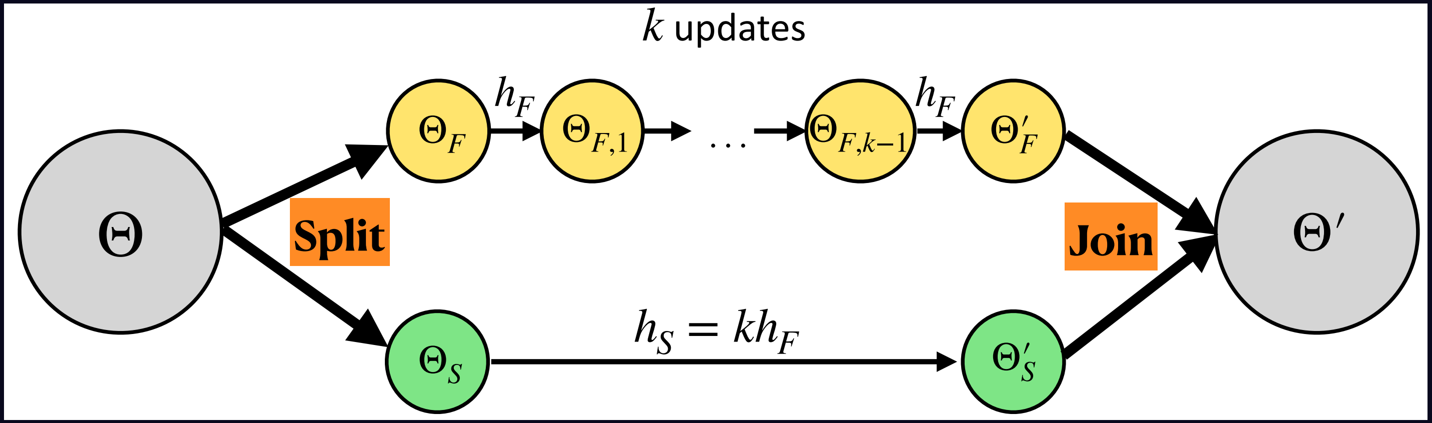
- Large learning rate (orange) gives higher accuracy on clean data.

- Our multirate approach (green) can perform well on both.

Multirate Methods

Two time-scales example:

Partition model (+ accompanying momentum) parameters $\Theta = (\Theta_F, \Theta_S)$.



Fast components Θ_F are updated every step with step size h_F
Slow components Θ_S are updated every k steps with step size $h_S = kh_F$

Partition-based Multirate Approach

Separate neural network parameters into different parts.
You have a choice!

Examples: layer-wise, weight vs. biases, or random subgroups.

Partition-based Multirate Approach

Separate neural network parameters into different parts.
You have a choice!

Examples: layer-wise, weight vs. biases, or random subgroups.

Act as an add-on to existing optimization schemes: they can be combined with any desired base algorithm.

If base algorithm SGD (as used in PyTorch code):

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

with momentum p and loss $\mathscr{L}$

Partition-based Multirate Approach

Separate neural network parameters into different parts.
You have a choice!

Examples: layer-wise, weight vs. biases, or random subgroups.

Act as an add-on to existing optimization schemes: they can be combined with any desired base algorithm.

If base algorithm SGD (as used in PyTorch code):

$p_S := \mu p_S + \nabla_{\theta_S} \mathcal{L}(\theta_S, \theta_F)$
$\theta_S := \theta_S - \boxed{h} p_S$
for $i = 1, 2, ..., k$ **do**
$\quad p_F := \mu p_F + \nabla_{\theta_F} \mathcal{L}(\theta_S, \theta_F)$
$\quad \theta_F := \theta_F - \boxed{\frac{h}{k}} p_F$
end

with momentum p and loss $\mathscr{L}$

Partition-based Multirate Approach

Separate neural network parameters into different parts.
You have a choice!

Examples: layer-wise, weight vs. biases, or random subgroups.

Act as an add-on to existing optimization schemes: they can be combined with any desired base algorithm.

If base algorithm SGD (as used in PyTorch code):

Without linear drift:

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

with momentum p and loss $\mathscr{L}$

With linear drift:

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

Partition-based Multirate Approach

Separate neural network parameters into different parts.
You have a choice!

Examples: layer-wise, weight vs. biases, or random subgroups.

Act as an add-on to existing optimization schemes: they can be combined with any desired base algorithm.

If base algorithm SGD (as used in PyTorch code):

Without linear drift:

$$
\begin{aligned}
&p_S := \mu p_S + \nabla_{\theta_S}\mathcal{L}(\theta_S, \theta_F)\\
&\theta_S := \theta_S - h p_S\\
&\textbf{for } i = 1, 2, \ldots, k \textbf{ do}\\
&\quad p_F := \mu p_F + \nabla_{\theta_F}\mathcal{L}(\theta_S, \theta_F)\\
&\quad \theta_F := \theta_F - \tfrac{h}{k} p_F\\
&\textbf{end}
\end{aligned}
$$

with momentum p and loss $\mathscr{L}$

With linear drift:

$$
\begin{aligned}
&p_S := \mu p_S + \nabla_{\theta_S}\mathcal{L}(\theta_S, \theta_F)\\
&\textbf{for } i = 1, 2, \ldots, k \textbf{ do}\\
&\quad p_F := \mu p_F + \nabla_{\theta_F}\mathcal{L}(\theta_S, \theta_F)\\
&\quad \theta_F := \theta_F - \tfrac{h}{k} p_F\\
&\quad \theta_S := \theta_S - \tfrac{h}{k} p_S\\
&\textbf{end}
\end{aligned}
$$

We compare convergence properties with vanilla SGD.

Application: Transfer Learning

Basics of transfer learning:

- Start with pre-trained model on large datasets, e.g., ImageNet.
- Remove task-specific layers and re-train (part of) network on new task.

Application: Transfer Learning

Basics of transfer learning:

- Start with pre-trained model on large datasets, e.g., ImageNet.
- Remove task-specific layers and re-train (part of) network on new task.

To obtain computational speed-up:

Split 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!

Application: Transfer Learning

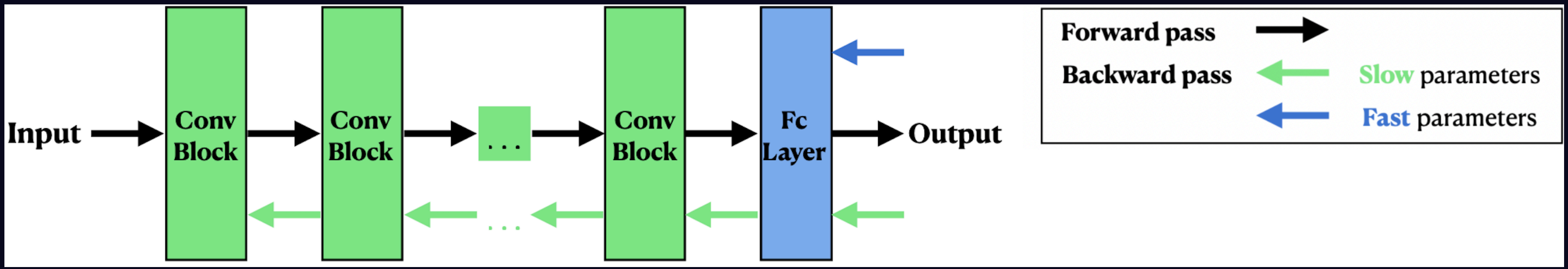
Basics of transfer learning:

- Start with pre-trained model on large datasets, e.g., ImageNet.
- Remove task-specific layers and re-train (part of) network on new task.

To obtain computational speed-up:

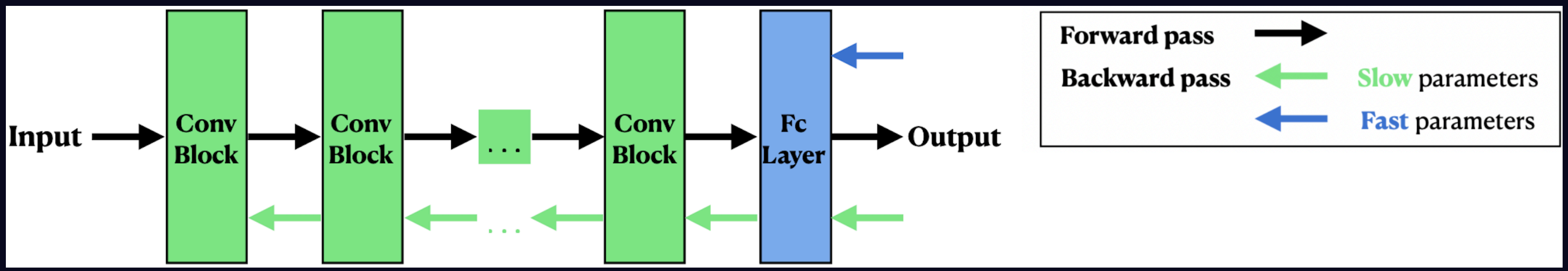
Split net in two parts: final layer(s) as the fast part, rest is slow part.
Only need to compute gradients for full network every k steps!

Example for a ResNet architecture:

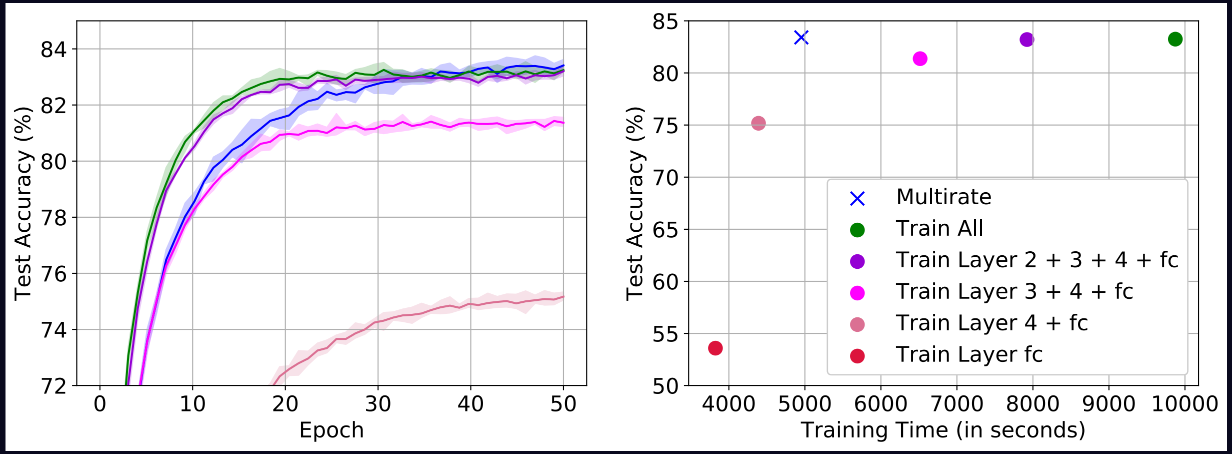


For a ResNet-34 architecture fast parameters are only 0.024% of total.

Application: Transfer Learning

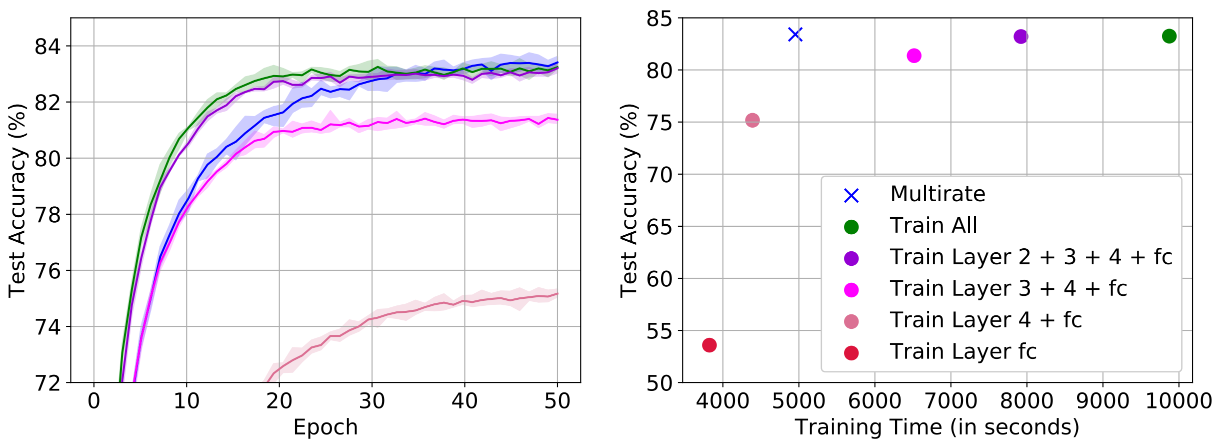


ResNet-50, CIFAR-100 data

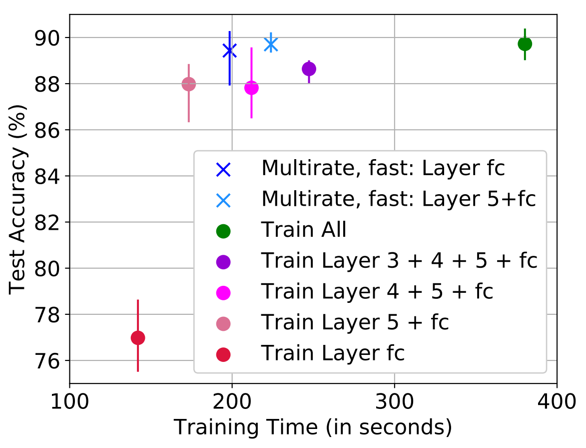


Application: Transfer Learning

ResNet-50, CIFAR-100 data

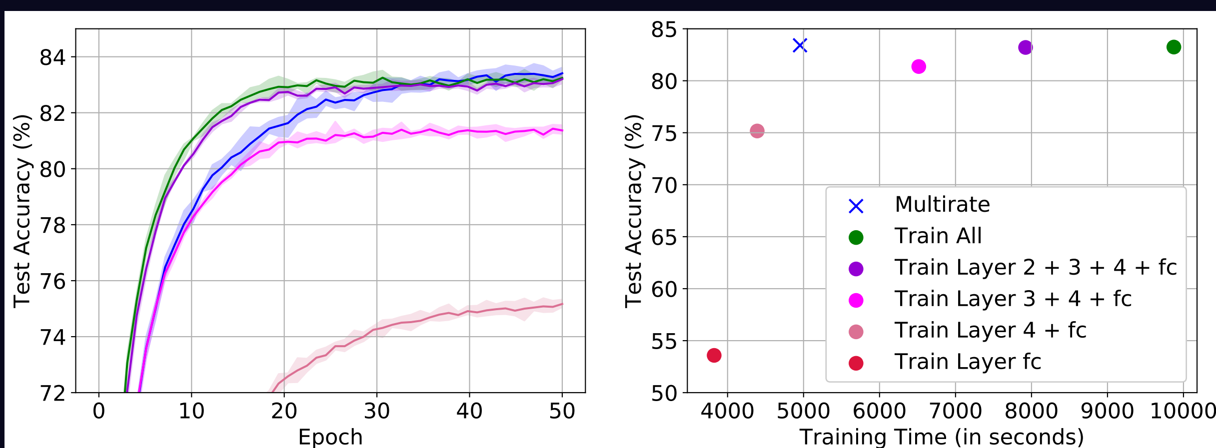


DistilBERT, SST-2 data

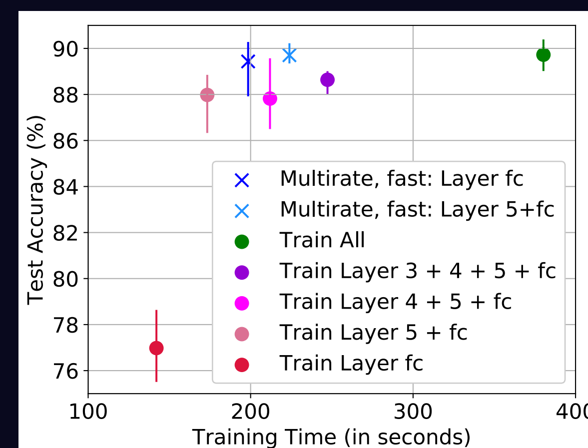


Application: Transfer Learning

ResNet-50, CIFAR-100 data



DistilBERT, SST-2 data



Can train in half the time, without losing performance!

Full ablation studies in paper.

Take-aways

Multirate methods can be used to enhance current neural network training techniques!

The proposed techniques:

- Act as add-on to existing optimizers.
- Can learn different features present in the data.
- Can train deep nets for transfer learning settings in half the time, without losing accuracy.

Take-aways

Multirate methods can be used to enhance current neural network training techniques!

The proposed techniques:

- Act as add-on to existing optimizers.
- Can learn different features present in the data.
- Can train deep nets for transfer learning settings in half the time, without losing accuracy.

Extensions

- Hybrid optimization schemes *[Leimkuhler, Matthews, Vlaar, 2019].*
- Combine with well-known machine learning techniques & apply to other settings.
- Different splitting choices of network parameters

Take-aways

Multirate methods can be used to enhance current neural network training techniques!

The proposed techniques:

- Act as add-on to existing optimizers.
- Can learn different features present in the data.
- Can train deep nets for transfer learning settings in half the time, without losing accuracy.

Extensions

- Hybrid optimization schemes *[Leimkuhler, Matthews, Vlaar, 2019]*.
- Combine with well-known machine learning techniques & apply to other settings.
- Different splitting choices of network parameters

Random Subgroups Partitioning

Transformer, Penn Treebank