THE UNIVERSITY of EDINBURGH

Multitrate Training of Neural Networks

arXiv preprint: 2106.10771

Tiffany Vlaar and Benedict Leimkuhler

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

Inspiration

- Neuroscience: synapses have dynamics at different time-scales.
- Fast/slow weights in machine learning: Hinton and Plaut (1987) set each connection to have fast changing weight *[temporary memory]* and slowly changing weight *[long-term knowledge]*.

Inspiration

- Neuroscience: synapses have dynamics at different time-scales.
- Fast/slow weights in machine learning: Hinton and Plaut (1987) set each connection to have fast changing weight *[temporary memory]* and slowly changing weight *[long-term knowledge]*.
- Different network layers play different roles, so train them differently:

  - Layer-wise adaptive learning rates [You et al., 2017].

  - Use partitioned integrators for neural network training [Leimkuhler, Matthews, TV, 2019].

Inspiration

- Neuroscience: synapses have dynamics at different time-scales.
- Fast/slow weights in machine learning: Hinton and Plaut (1987) set each connection to have fast changing weight *[temporary memory]* and slowly changing weight *[long-term knowledge]*.
- Different network layers play different roles, so train them differently:

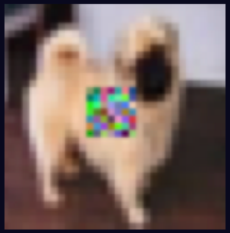
  - Layer-wise adaptive learning rates [You et al., 2017].

  - Use partitioned integrators for neural network training [Leimkuhler, Matthews, TV, 2019].
- Molecular dynamics: fast dynamics cheap to compute [e.g., r-RESPA Tuckerman et al. 1991, 1992] -can we obtain computational speed-up in neural network training applications?

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.

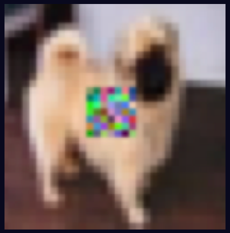


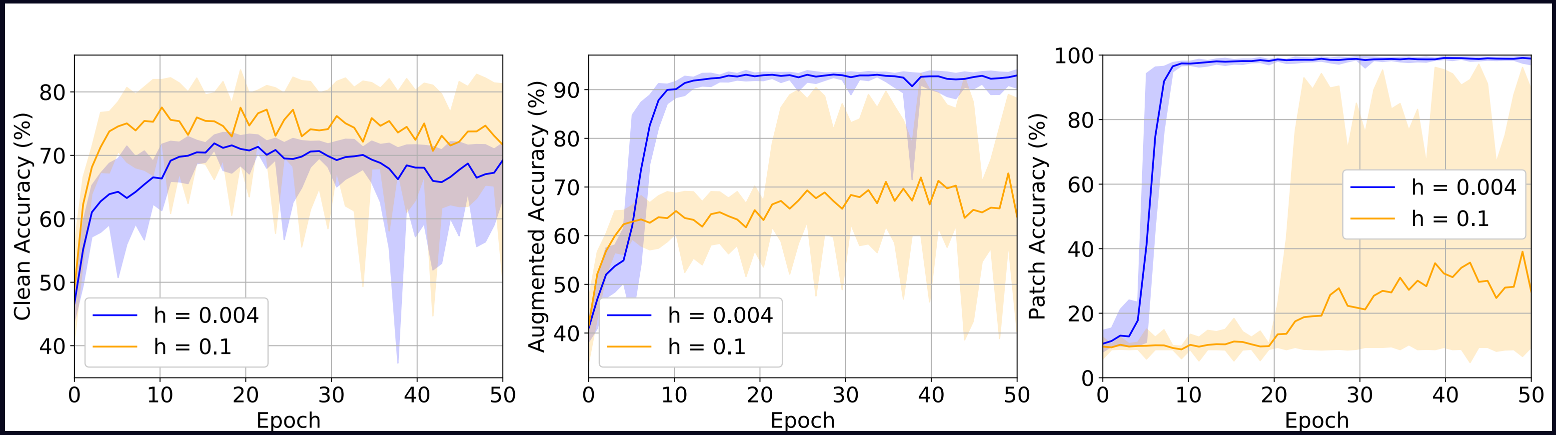
Latent multiple time scales in deep learning

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.



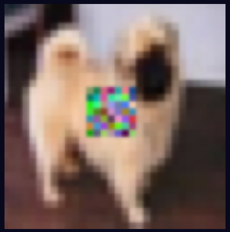


A net trained using a:

- Small learning rate (blue) memorizes patch.

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

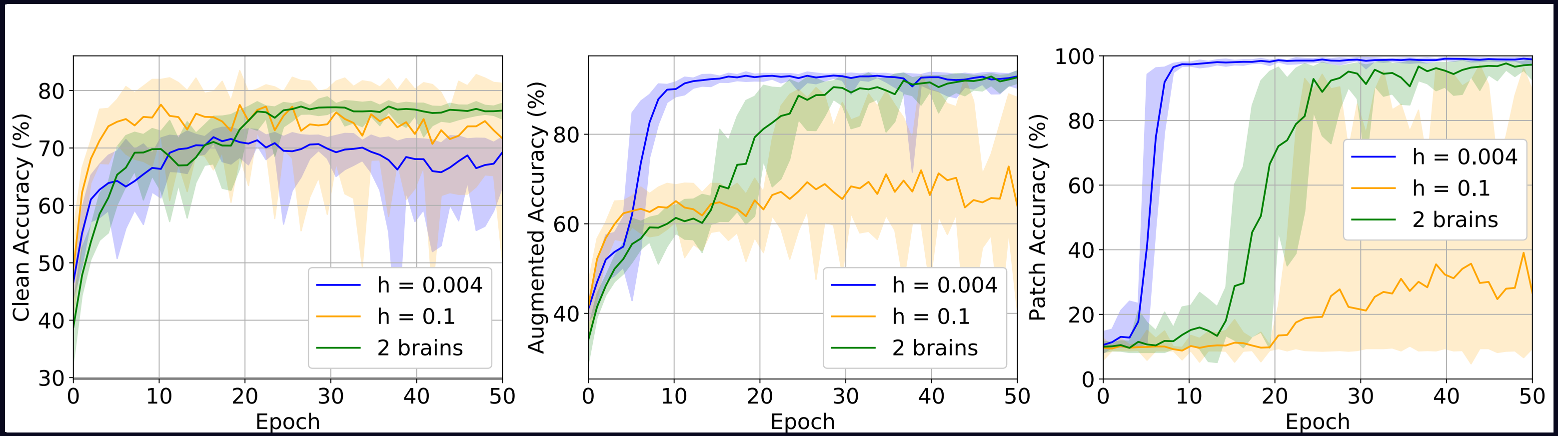
Latent multiple time scales in deep learning



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.

Neural network training

Basis: system of ODEs: $d\theta = G(\theta)\ dt$, with neural network parameters $\theta \in \mathbb{R}^n$

and G the negative gradient of the loss of entire dataset.

Neural network training

Basis: system of ODEs: $d\theta = G(\theta)\ dt$, with neural network parameters $\theta \in \mathbb{R}^n$

and G the negative gradient of the loss of entire dataset.

SGD: train on randomly selected subset of training data

$\Rightarrow$ replace pure gradient by noisy gradient $\tilde{G}(\theta)$.

SGLD: additionally drive with constant variance additive noise [Welling & Teh, ICML 2011].
Further extension: incorporate momentum p.

Neural network training

Basis: system of ODEs: $d\theta = G(\theta)\ dt$, with neural network parameters $\theta \in \mathbb{R}^n$

and G the negative gradient of the loss of entire dataset.

SGD: train on randomly selected subset of training data

$\Rightarrow$ replace pure gradient by noisy gradient $\tilde{G}(\theta)$.

SGLD: additionally drive with constant variance additive noise [Welling & Teh, ICML 2011].
Further extension: incorporate momentum p.

Underdamped Langevin Dynamics framework:

$$d\theta = p\ dt$$

$$dp = \tilde{G}(\theta)\ dt - \gamma p\ dt + \sqrt{2\gamma\tau}\ dW$$

Hyperparameter τ controls: pure optimization ($\tau = 0$) $\leftrightarrow$ pure sampling ($\tau = 1$).

Neural network training: Partitioning

Partition model parameters $\theta = (\theta^{(1)}, \ldots, \theta^{(N)})$.

Neural network training: Partitioning

Partition model parameters $\theta = (\theta^{(1)}, \ldots, \theta^{(N)})$.

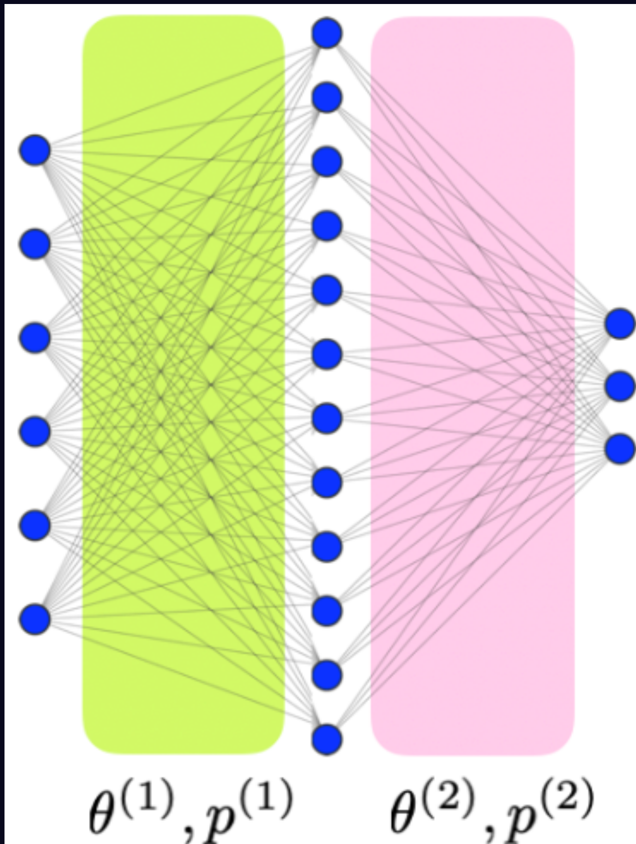
Partitioned integrators for thermodynamic parameterization of neural networks
[Leimkuhler, Matthews & TV, 2019].

$$d\theta^{(1)} = p^{(1)}\ dt$$

$$dp^{(1)} = \tilde{G}^{(1)}(\theta)\ dt - \gamma p^{(1)}\ dt + \sqrt{2\gamma\tau}\ dW^{(1)}$$

$$d\theta^{(2)} = p^{(2)}\ dt$$

$$dp^{(2)} = \tilde{G}^{(2)}(\theta)\ dt - \gamma p^{(2)}\ dt + \sqrt{2\gamma\tau}\ dW^{(2)}$$



Neural network training: Partitioning

Partition model parameters $\theta = (\theta^{(1)}, \ldots, \theta^{(N)})$.

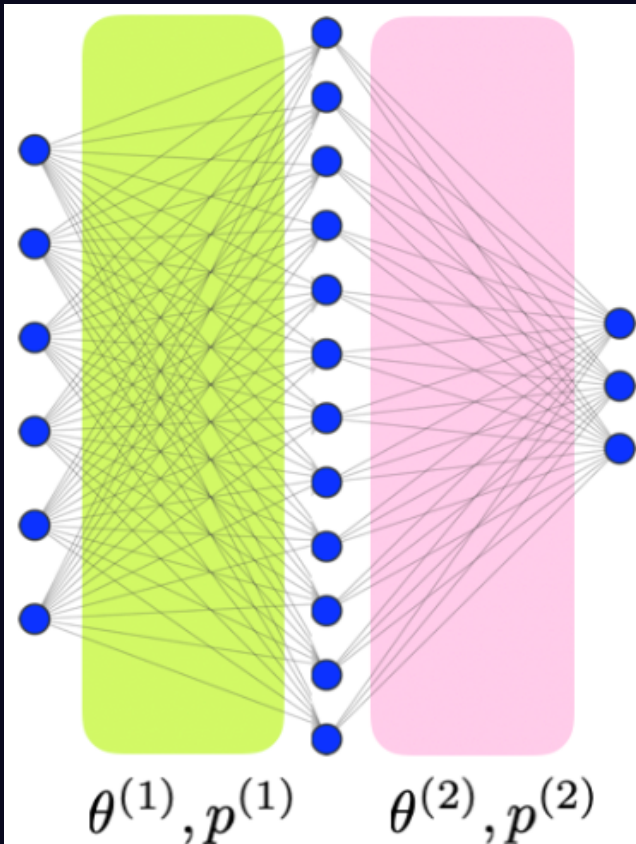
Partitioned integrators for thermodynamic parameterization of neural networks
[Leimkuhler, Matthews & TV, 2019].

$$d\theta^{(1)} = p^{(1)}\, dt$$

$$dp^{(1)} = \tilde{G}^{(1)}(\theta)\, dt - \gamma p^{(1)}\, dt + \sqrt{2\gamma\tau}\, dW^{(1)}$$

$$d\theta^{(2)} = p^{(2)}\, dt$$

$$dp^{(2)} = \tilde{G}^{(2)}(\theta)\, dt - \gamma p^{(2)}\, dt$$



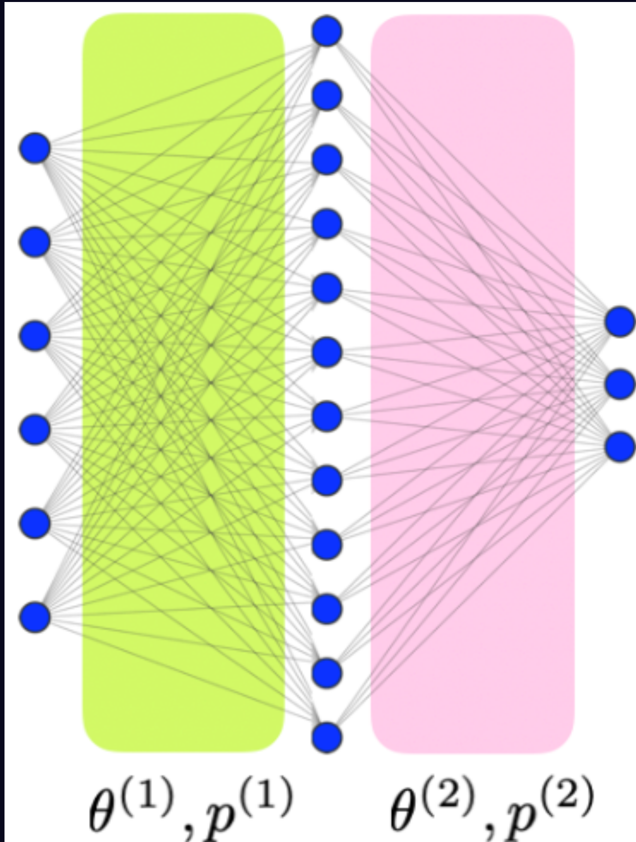
Neural network training: Partitioning

Partition model parameters $\theta = (\theta^{(1)}, \ldots, \theta^{(N)})$.

Partitioned integrators for thermodynamic parameterization of neural networks
[Leimkuhler, Matthews & TV, 2019].

$$d\theta^{(1)} = p^{(1)}\, dt$$
$$dp^{(1)} = \tilde{G^{(1)}}(\theta)\, dt - \gamma_1\, p^{(1)}\, dt + \sqrt{2\gamma_1\tau_1}\, dW^{(1)}$$

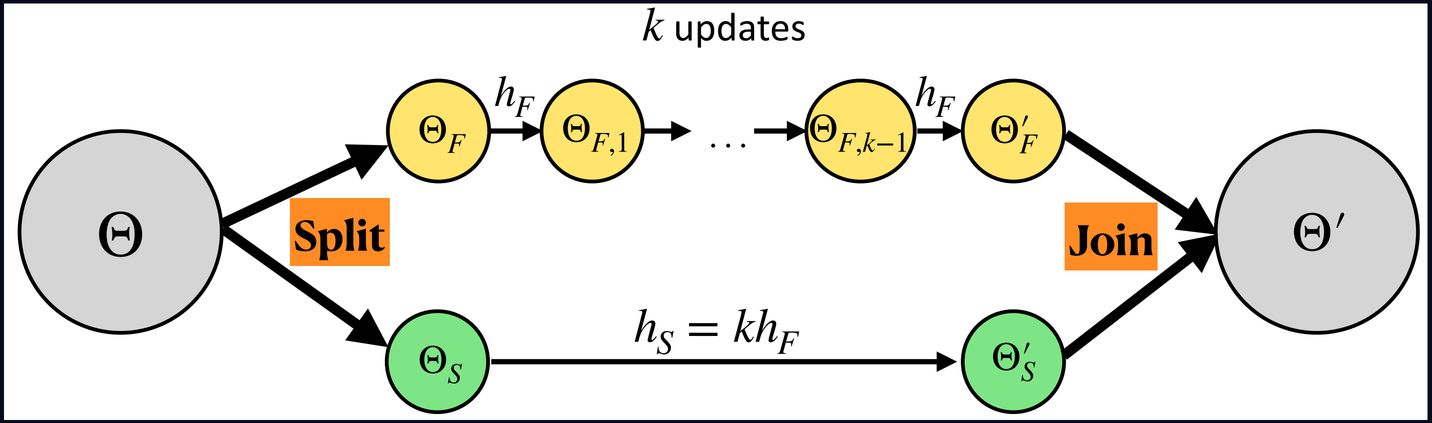
$$d\theta^{(2)} = p^{(2)}\, dt$$
$$dp^{(2)} = \tilde{G^{(2)}}(\theta)\, dt - \gamma_2\, p^{(2)}\, dt + \sqrt{2\gamma_2\tau_2}\, dW^{(2)}$$



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 $h_S = kh_F$

Multirate methods for neural network training

We consider multirate methods both *within* and *across* networks:

Within: Partition network parameters into fast and slow parts.

Across: Train multiple copies of a network on multiple time scales simultaneously.

Multirate methods for neural network training

We consider multirate methods both *within* and *across* networks:

Within: Partition network parameters into fast and slow parts.

Across: Train multiple copies of a network on multiple time scales simultaneously.

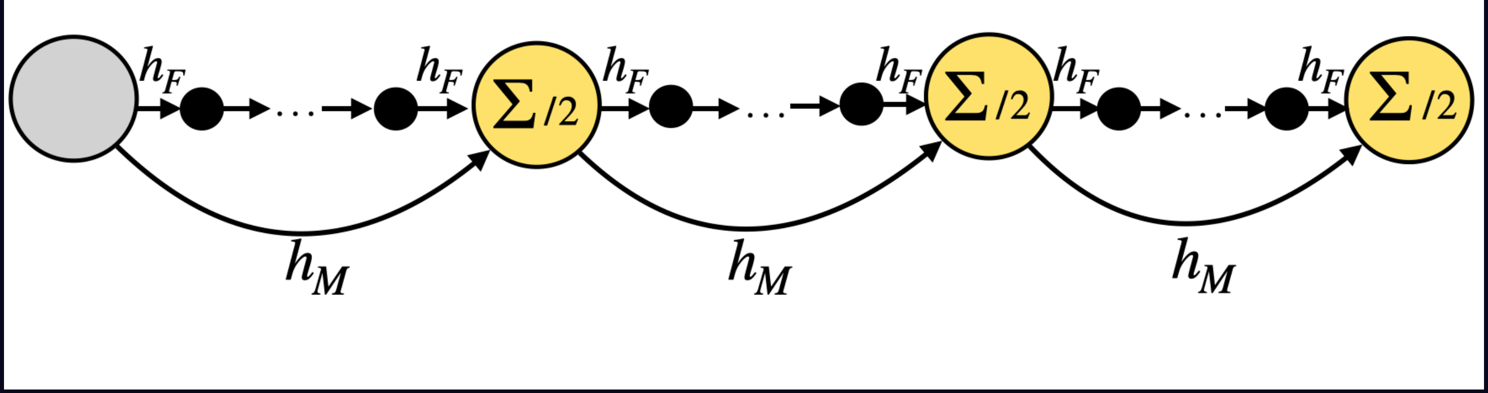
Act as an add-on to existing optimization schemes: they can be combined with any desired base algorithm.

Across networks: "multi-brain"

Train independent copies (brains) of the same network using different learning rates.
Merge parameters whenever two or more copies reach the same time threshold.

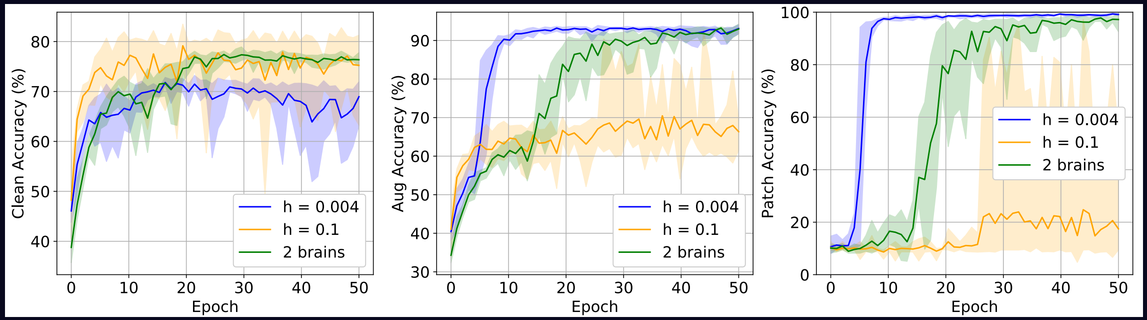
Can be used with optimizer of your choice.

Example:



Two brains with step sizes $h_M = kh_F$, $k \in \mathbb{Z}_{>0}$

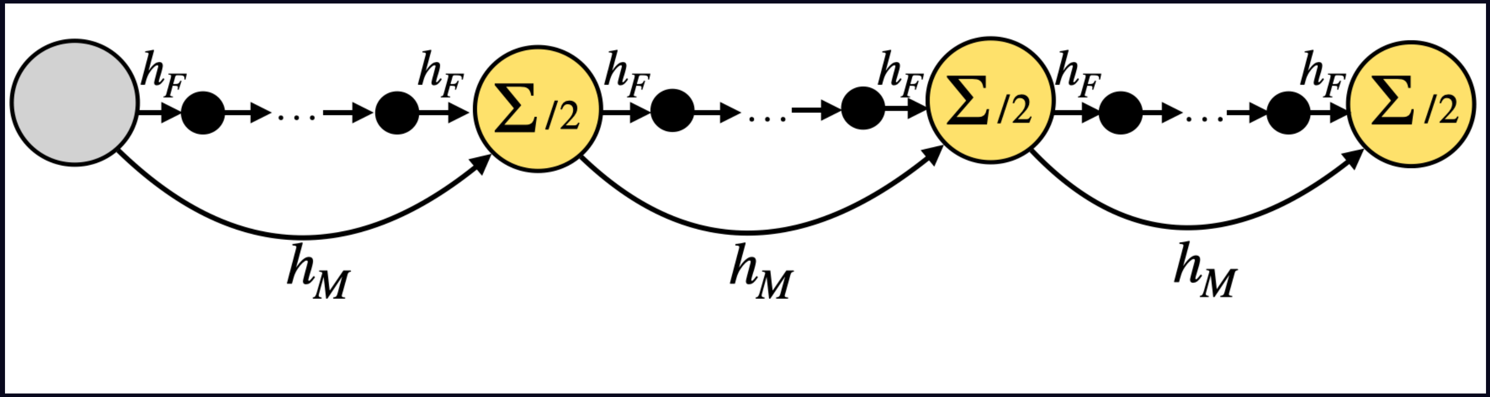
Across networks:



Train independent copies (brains) of the same network using different learning rates.
Merge parameters whenever two or more copies reach the same time threshold.

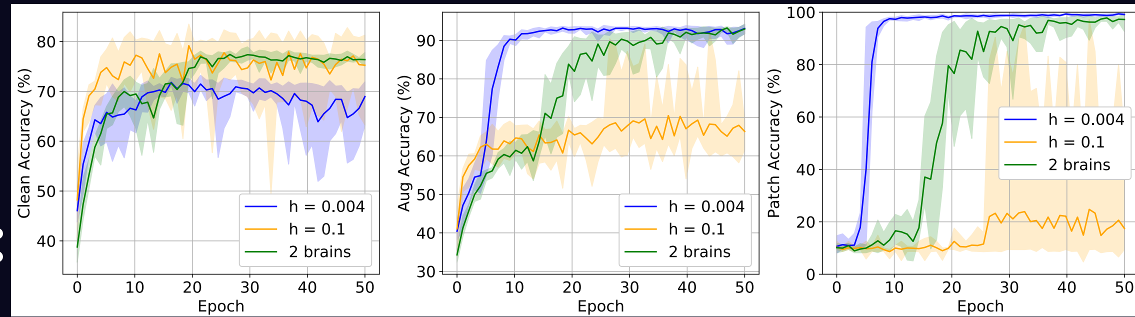
Can be used with optimizer of your choice.

Example:



Two brains with step sizes $h_M = kh_F,\ k \in \mathbb{Z}_{>0}$

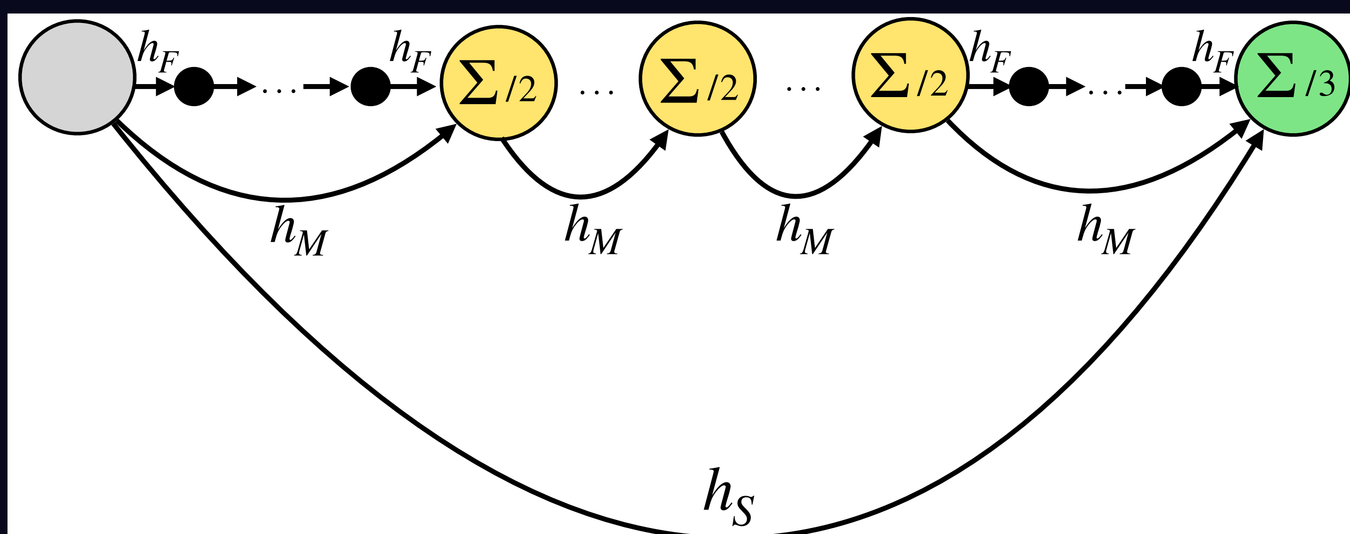
Across networks:



Train independent copies (brains) of the same network using different learning rates.
Merge parameters whenever two or more copies reach the same time threshold.

Can be used with optimizer of your choice.

Example: Training with *three* brains with stepsizes $h_S = k_2 h_M = k_1 h_F$



Within networks: partition-based multirate approach

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

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

Can be used with optimizer of your choice.

Within networks: partition-based multirate approach

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

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

Can be used with optimizer of your choice.
If base algorithm SGD (as used in PyTorch code):

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

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

Within networks: partition-based multirate approach

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

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

Can be used with optimizer of your choice.
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 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

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

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

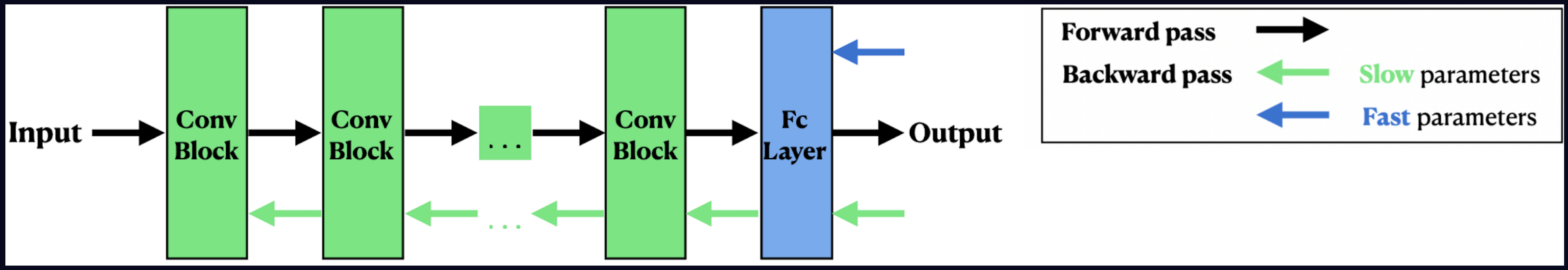
Basics of transfer learning:

- 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:

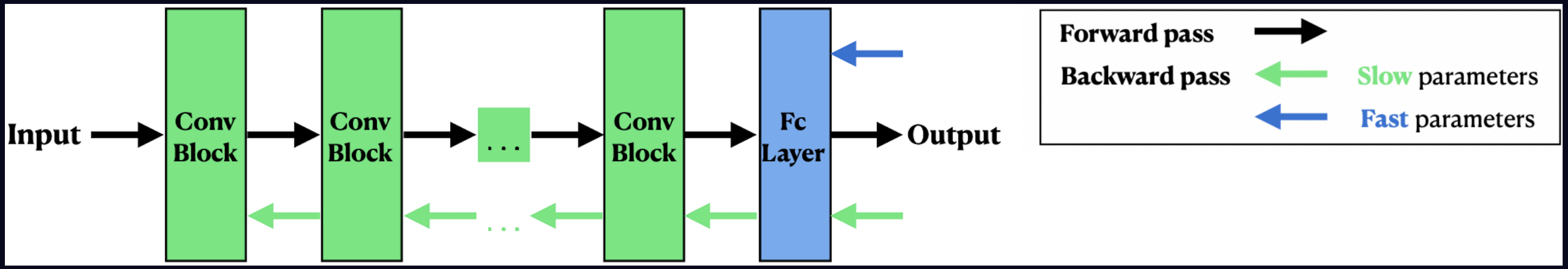
Split net in two parts: final layer(s) as the fast part, rest is slow part.
Only need to compute gradients for full network every k steps!

Example for a ResNet architecture:

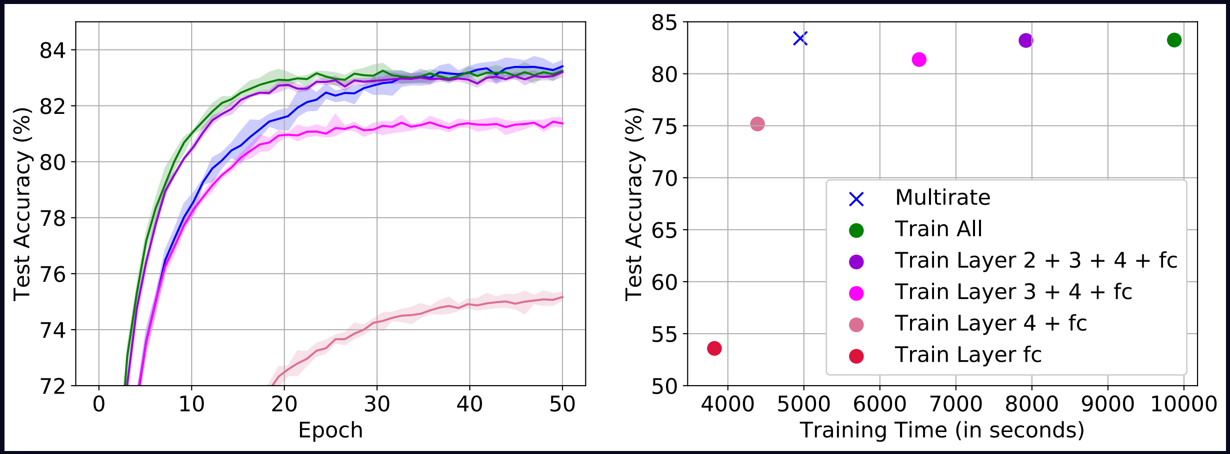


For a ResNet-34 architecture fast parameters are only 0.024% of total.

Application: Transfer Learning

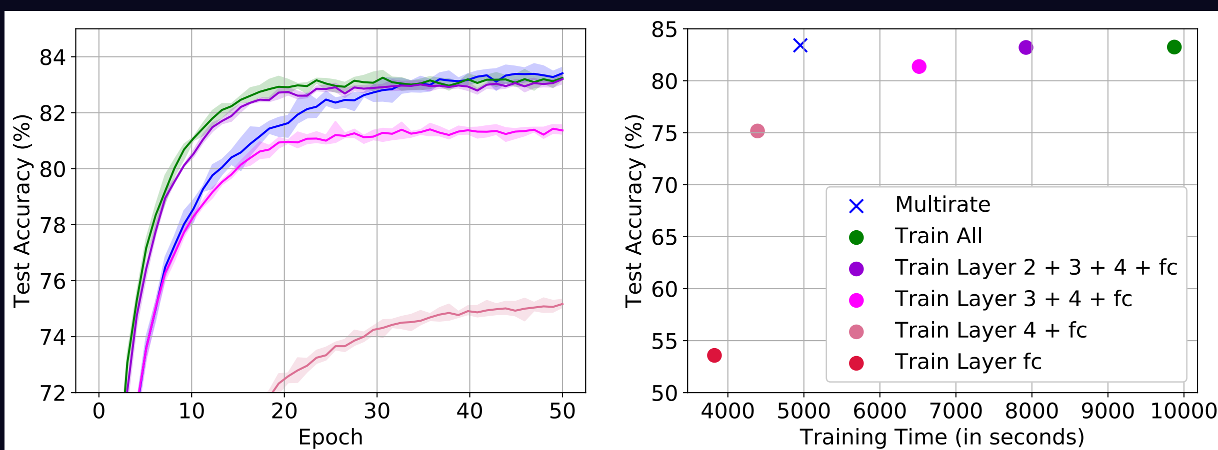


ResNet-50, CIFAR-100 data

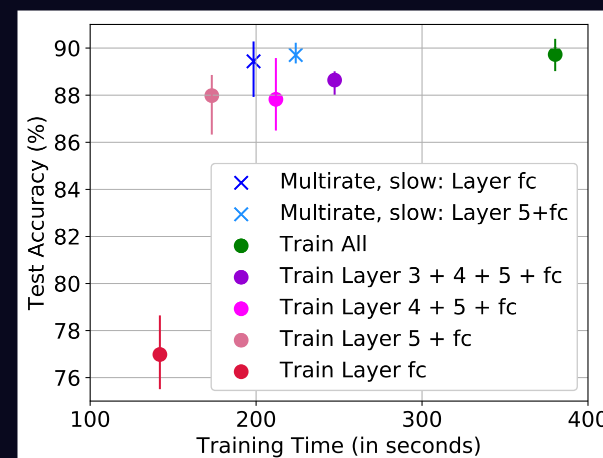


Application: Transfer Learning

ResNet-50, CIFAR-100 data

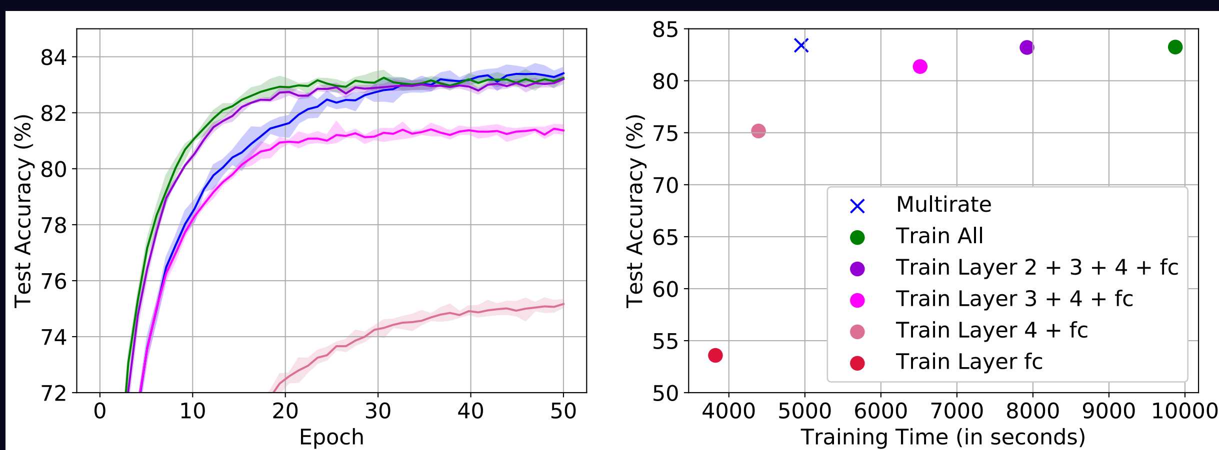


DistilBERT, SST-2 data

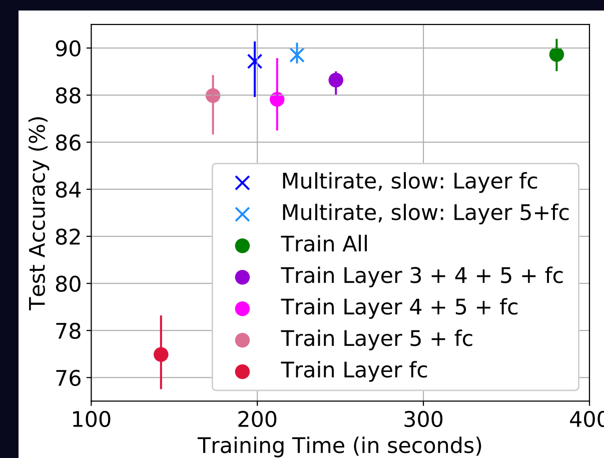


Application: Transfer Learning

ResNet-50, CIFAR-100 data



DistilBERT, SST-2 data



Can train in half the time, without losing much (if any) performance!

Full ablation studies in paper.

Application: Neural Network Regularization

Slow parameters θ_S : Randomly selected subset of network weights.

Application: Neural Network Regularization

Slow parameters θ_S : Randomly selected subset of network weights.

Training procedure:

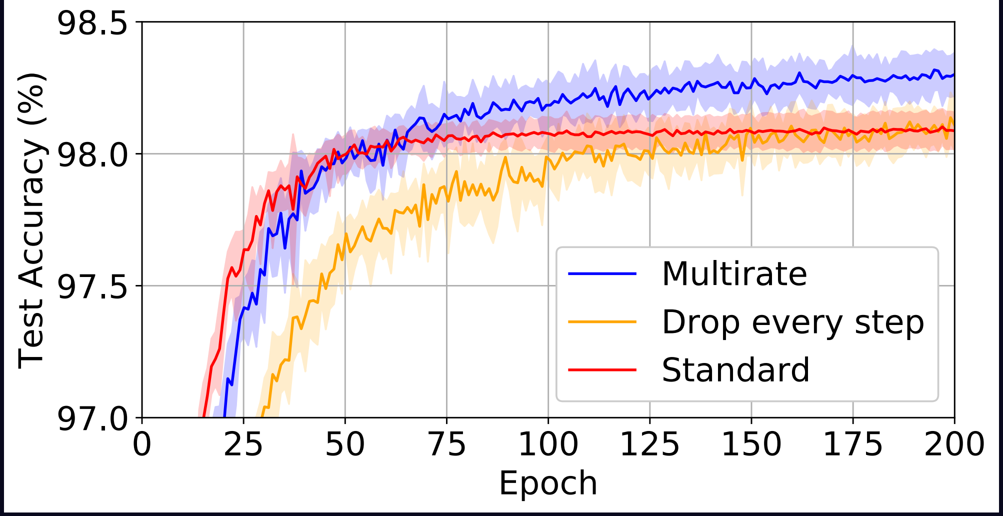
- Set θ_S to be a random subset and set all of them to zero.
- Update remaining parameters θ_F with stepsize h_F
- After k steps:
    - Re-activate slow parameters
    - Update θ_S with stepsize kh_F
    - Select new random subset to form θ_S

Application: Neural Network Regularization

Slow parameters θ_S : Randomly selected subset of network weights.

Training procedure:

- Set θ_S to be a random subset and set all of them to zero.
- Update remaining parameters θ_F with stepsize h_F
- After k steps:
  - Re-activate slow parameters
  - Update θ_S with stepsize kh_F
  - Select new random subset to form θ_S



Base algorithm: SGD
Model: SHLP
Data: MNIST

Take-aways

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

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 by training the net on different time scales simultaneously.
- Can train deep nets for transfer learning settings in half the time, without losing accuracy.

Extensions

- Different splitting choices of network parameters in fast/slow parts.

Extensions

- Different splitting choices of network parameters in fast/slow parts.
- Hybrid optimization schemes *[Partitioned Integrators for Thermodynamic Parameterization of NNs. Leimkuhler, Matthews, TV].*

Extensions

- Different splitting choices of network parameters in fast/slow parts.
- Hybrid optimization schemes *[Partitioned Integrators for Thermodynamic Parameterization of NNs. Leimkuhler, Matthews, TV].*
- Combine with well-known machine learning techniques.
- Apply to other settings: molecular dynamics.

Extensions

- Different splitting choices of network parameters in fast/slow parts.
- Hybrid optimization schemes *[Partitioned Integrators for Thermodynamic Parameterization of NNs. Leimkuhler, Matthews, TV].*
- Combine with well-known machine learning techniques.
- Apply to other settings: molecular dynamics.

We are eager to collaborate, so get in touch!

Also… looking for a postdoc.

THE UNIVERSITY *of* EDINBURGH

Multitrate Training of Neural Networks

arXiv preprint: 2106.10771

Tiffany Vlaar and Benedict Leimkuhler

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