Learning to Optimize Deep Neural Networks

Muneeb Shahid
University of Freiburg

November 13, 2017

Introduction

Given problem variables x^t and gradient $\nabla_x L^t$ with respect to the loss L^t, optimizers output a gradient step Δx^t.

Using Δx^t we update x^t, to get x^{t+1}.

$$x^{t+1} = x^t + \Delta x^t$$

Optimizers differ in how they compute Δx^t.

Handcrafted Optimizers

Gradient descent.

Gradient descent with momentum.

Nesterov’s Accelerated Gradient Descent.

RMSProp.

Adam.

Learned Optimizers

Learn to output Δx instead!

Optimizer $\mathbf{O}$, trains a meta optimizer $\mathbf{O}_m$ to output Δx.

Use the trained meta optimizer $\mathbf{O}_m$ to train neural networks.

Previous Work

Learning to Optimize.

Learning to Learn.

Learned optimizers that Scale and Generalize.

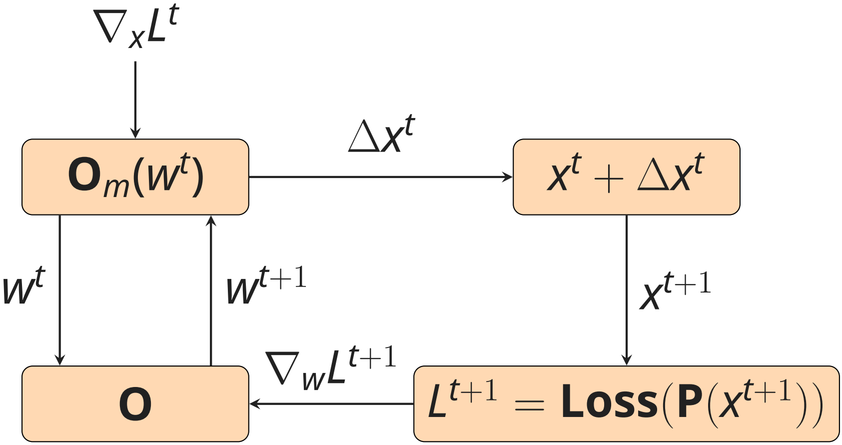
Learning to Optimize Deep Neural Networks

Learning to Optimize Deep Neural Networks

Learning to Optimize with Normalized Inputs.

Multiscale *Adam*.

The Learning Algorithm



Learning to Optimize with Normalized Inputs

Learning to Optimize with Normalized Inputs

Normalization provides invariance to gradient scaling.

Feed cooridnatewise normalized history of gradients to the meta optimizer.

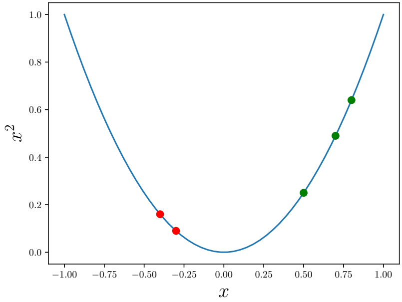
The meta optimizer then outputs $\Delta x^t \in [-1.0, 1.0]$.

Learning to Optimize with Normalized Inputs

First, let’s consider two different scenarios while traversing a noise free optimization landscape.

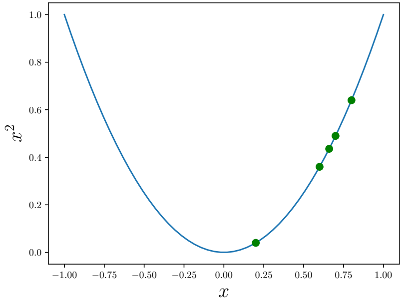
Scenario # 1

The gradient sign changes, a good indicator we just crossed a minima.

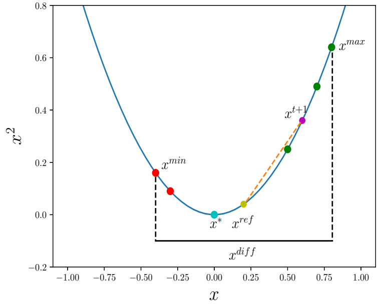


Scenario # 2

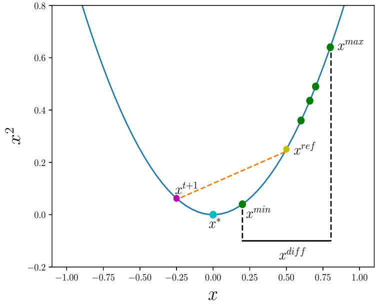
The gradient sign does not change - an indication that the minima lies outside the visible history.



Algorithm Overview



Minima lies with in the history.



Minima lies outside the history.

Algorithm Update Equations

We also introduce η_{min} as the minimum learning rate.

$$x^{t,ref} = (x^{t,max} + x^{t,min})/2$$

$$x^{t,diff} = x^{t,max} - x^{t,min}$$

$$x^{t+1} = x^{t,ref} + \Delta x^{t} \cdot (x^{t,diff} + \eta_{min})$$

Algorithm Variants

k Timesteps - use the last *k* timesteps.

k Momentums - use momentums at *k* different timescales.

Experiments

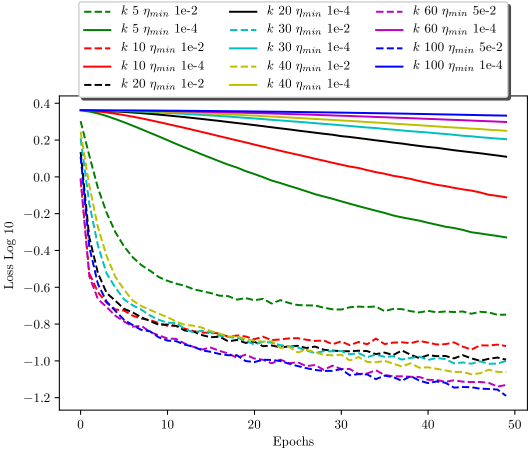
Meta optimizer $\mathbf{O}_m$ is trained for 200k meta iterations on *MlpMNIST*.

The trained meta optimizer is then evaluated on the same *MLP*.

Finally, we evaluate the best trained meta optimizer on a much larger network *ConvCIFAR*.

k Timesteps

Increasing k, results in better performance but η_{min} needs to be tuned as well.

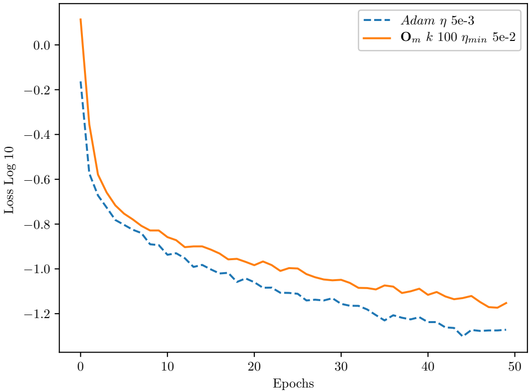


Learned $\Delta \boldsymbol{x}$, for $\boldsymbol{k} = 5$

$\nabla_x L^{t-4}$	$\nabla_x L^{t-3}$	$\nabla_x L^{t-2}$	$\nabla_x L^{t-1}$	$\nabla_x L^{t}$	Δx^t
1.0	1.0	1.0	1.0	1.0	0.99999
1.0	1.0	1.0	1.0	-1.0	0.99984
1.0	1.0	1.0	-1.0	-1.0	0.99128
1.0	1.0	-1.0	-1.0	-1.0	-0.99011
1.0	-1.0	-1.0	-1.0	-1.0	-0.99978
-1.0	-1.0	-1.0	-1.0	-1.0	-0.99999
-0.1	-0.1	-0.25	-0.25	1.0	0.84402
-0.1	-0.25	0	0.25	1.0	0.41817

k Timesteps vs *Adam*

Even with $k = 100$, the learned optimizer fails to outperform *Adam*.



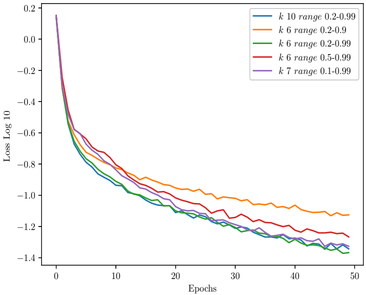
k Momentums

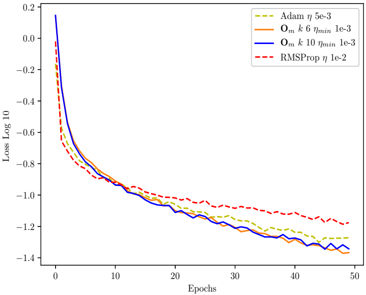
We can not just keep increasing number of timesteps. Thus, we turn to momentums instead.

But now $x_{ref} = (x_{max} + x_{min})/2$ can be very far back in history.

So, we set x_{ref} to the last value i.e. $x_{ref} = x_t$.

k Momentums



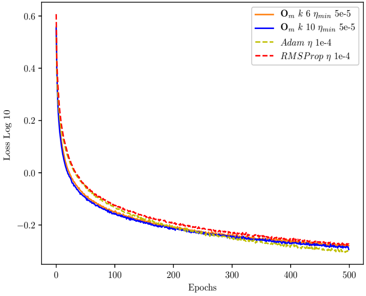


Performance comparison with different parameters.

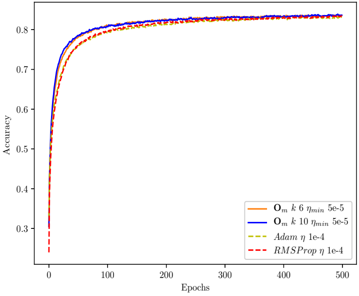
Performance against *Adam* and *RMSProp*.

Using momentums and the new update equation boosts the performance significantly.

k Momentums *ConvCIFAR*10



Loss



Accuracy

The learned optimizers outperform *RMSprop* but *Adam* manages to take a small lead towards the end.

Issues and Limitations

Up to two times slower than a simple *Adam* step.

One needs to store two history matrices of dimension $n \times k$ in memory, where n is the number of problem variables and k is the size of the history.

More thorough experimentation needs to be done.

Multiscale *Adam*

Multiscale *Adam*

Adam relies on normalized inputs.

Input *Adam* running at multiple timescales as inputs to the meta optimizer $\mathbf{O}_m$.

The meta optimizer then learns to output a weighted average over the inputs as the gradient step $\triangle x$

Experiments

We trained four different architectures for 50k meta iterations with varying number of input timescales *k*:

- Linear weighted Average (*LWA*).
- Single layered *MLP* with 50 neurons.
- RNN_a with a single hidden layer of 50 neurons.
- RNN_b with two hidden layers, each with 50 neurons in each layer.

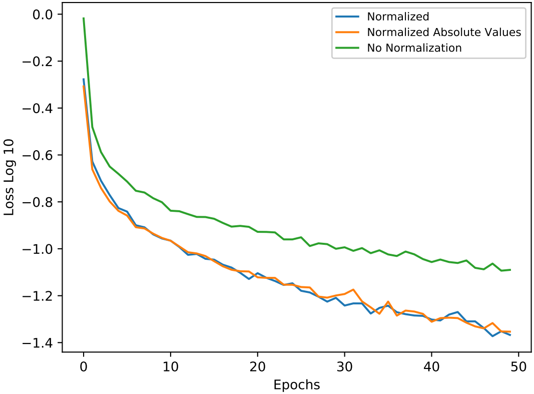
Linear Weighted Average

We experimented with three distinct architectures on *MlpMNIST*:

- Normalized weights constrained to sum up to one LWA_a.
- Normalized weights constrained to sum up to one and positive LWA_b.
- No constraints LWA_c.

LWA_a vs LWA_b vs LWA_c with $k = 6$

Normalization boosts performance by a huge margin.



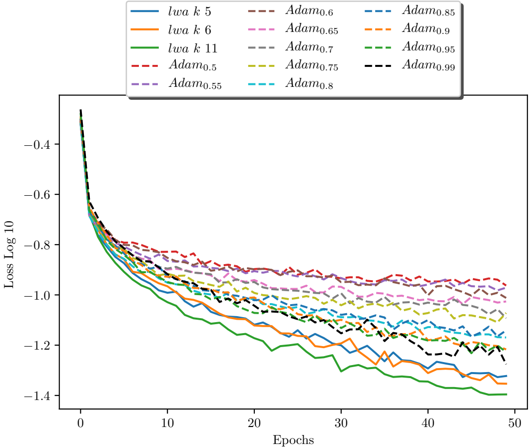
Learned Weights

Most recent and the most distant timescales, consistently have the highest weights.

Adam	LWA_a	LWA_b	LWA_c
$Adam_{.99}$	0.75668	0.66044	0.24911
$Adam_{.9}$	0.06197	0.00366	-0.08944
$Adam_{.8}$	0.68091	0.00065	-0.09293
$Adam_{.7}$	-1.21117	0.00064	0.00168
$Adam_{.6}$	-0.47439	0.00077	0.07846
$Adam_{.5}$	1.18599	0.33386	0.15519

LWA with different *k* vs *Adam*

LWA outperforms all of its individual input *Adam* timescales.



Learned Weights, $k = 5$ & $k = 11$

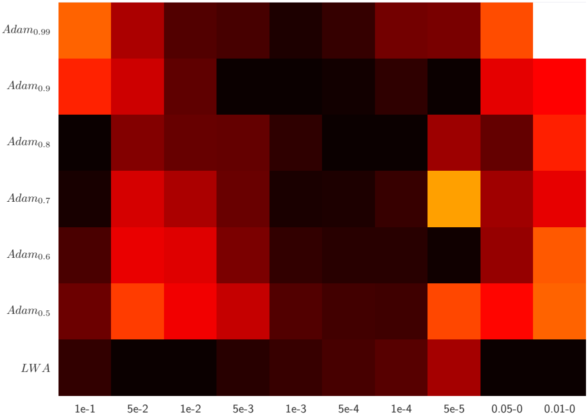
Adam	Weights
$Adam_{.99}$	0.60268
$Adam_{.9}$	0.00327
$Adam_{.8}$	0.00654
$Adam_{.7}$	0.00760
$Adam_{.6}$	0.37990

Adam	Weights
$Adam_{.99}$	0.66264
$Adam_{.95}$	0.00295
$Adam_{.9}$	0.00018
$Adam_{.85}$	0.00256
$Adam_{.8}$	0.00493
$Adam_{.7}$	0.00073
$Adam_{.75}$	0.00608
$Adam_{.65}$	0.00615
$Adam_{.6}$	0.00074
$Adam_{.55}$	0.00732
$Adam_{.5}$	0.29241

LWA with different learning rates.

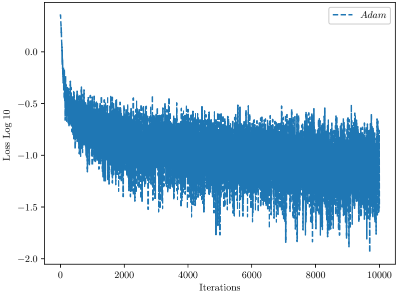
In order to further probe the performance of *LWA*, we tested against multiple learning rate schedules, including learning rate decay with $k = 6$.

LWA Performance Heatmap



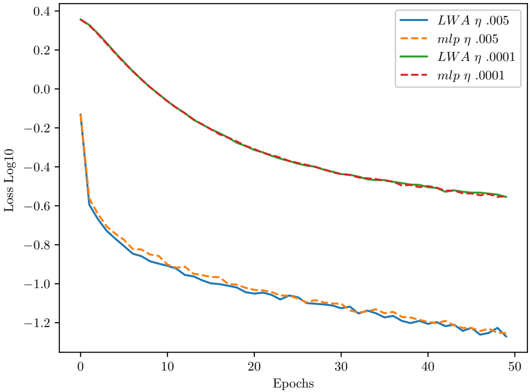
Why this drop in performance?

May be we need a more powerful model i.e. a *MLP*
May be we need to smooth the loss as it's very noisy.



LWA vs *MLP*

MLP showed no increase in performance.



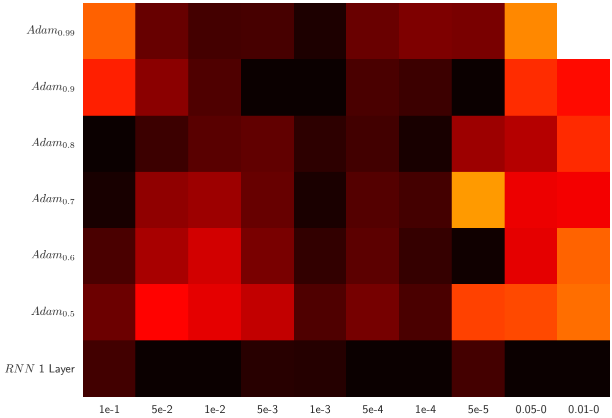
Enter *RNN*.

Using a *RNN* allows us to compute a smoothed loss by averaging over multiple timesteps i.e. the unroll length.

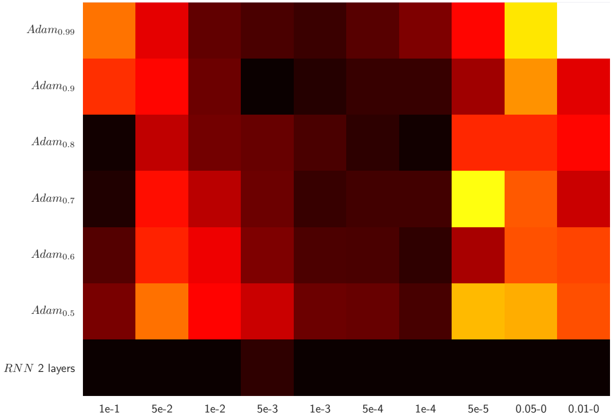
Optimizer **O**, uses this smoothed loss to update the parameters w^t of the meta optimizer $\mathbf{O}_m$.

RNN_a Performance Heatmap

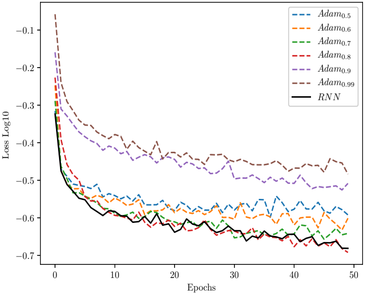
Smoothing the loss, increases the performance greatly.



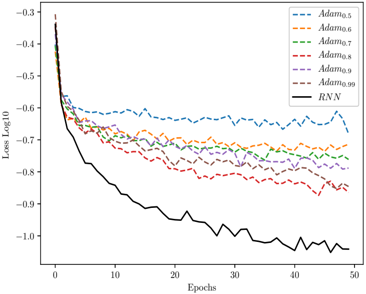
RNN_b Performance Heatmap



RNN_b Performance

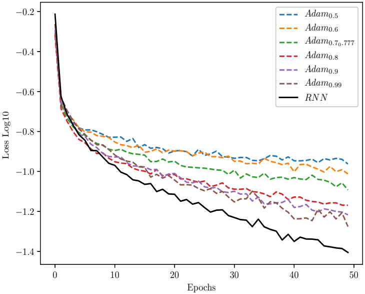


$$\eta = 1e - 1$$

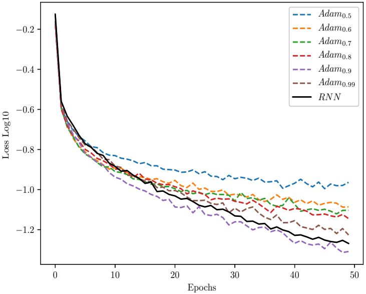


$$\eta = 5e - 2$$

RNN_b Performance

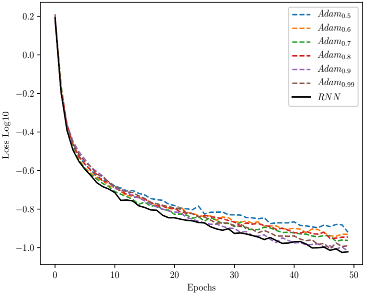


$\eta = 1e - 2$

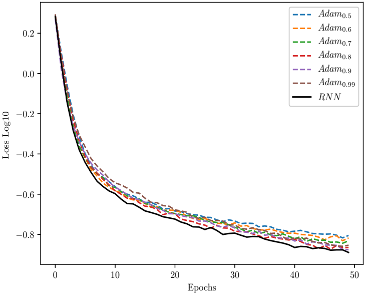


$\eta = 5e - 3$

RNN_b Performance

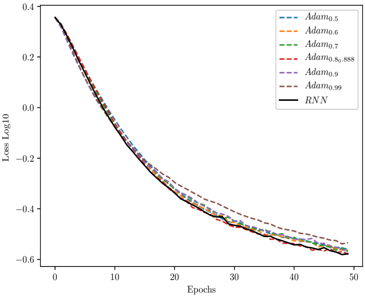


$$\eta = 1e - 3$$

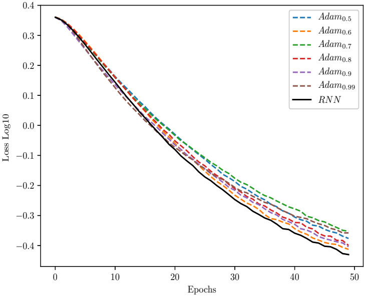


$$\eta = 5e - 4$$

RNN_b Performance

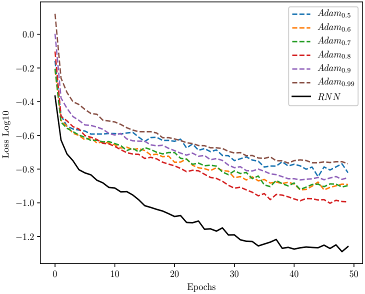


$\eta = 1e - 4$

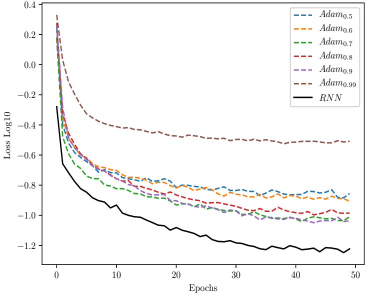


$\eta = 5e - 5$

RNN_b Performance



η decayed from $5e-2$ to 0.0



η decayed from $1e-2$ to 0.0

Multiscale *Adam*

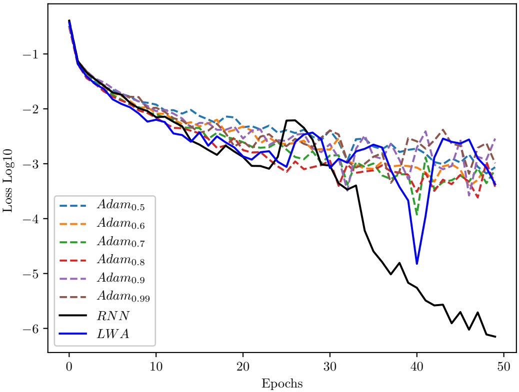
How well does the learned optimizer generalize?

Does it have any issues with *convolutional* Layers?

Does it work well with larger networks?

ConvMNISt

Trained *LWA* and RNN_b on a smaller Convnet with *Mnist* and evaluated on a larger Convnet. $\eta = 5e-4$

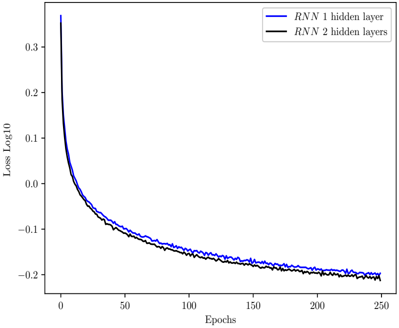


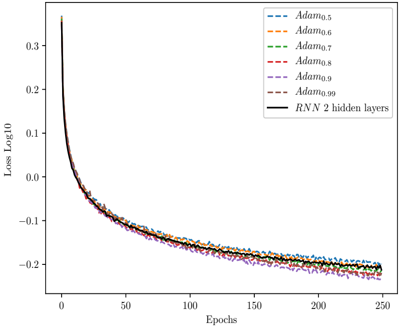
$ConvCIFAR10$

Trained RNN_a and RNN_b on a smaller Convnet for $Cifar10$ and evaluated on a larger Convnet (Million parameters). However, the trained optimizer failed to perform well.

Training directly on the larger network did not help either.

*ConvCIFAR*10





RNN_a vs RNN_b $\qquad$ RNN_b vs input *Adams*

Learning rate η was set to $5e-4$

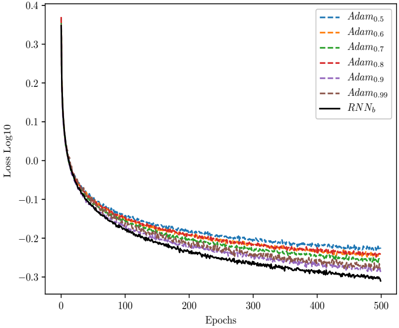
Training Directly on Large Networks

Training on networks with large amount of noise across batches is hard.

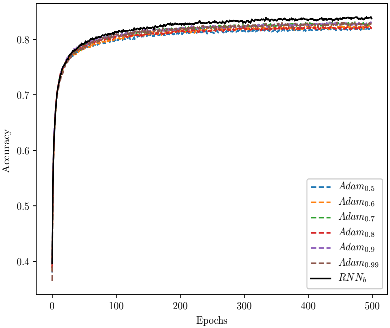
Moreover, training directly on larger networks is much slower compared.

Idea! Apply the optimizers trained on the small *MlpMNIST* to *ConvCIFAR*10.

*ConvCifar*10



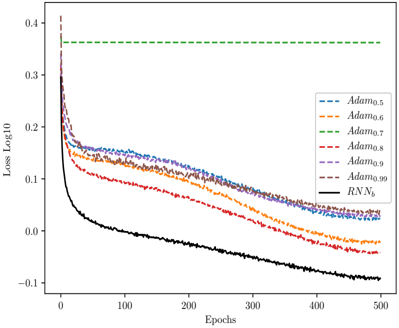
Loss

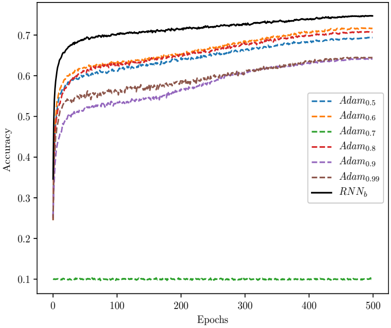


Accuracy

$$\eta = 5e - 4$$

$ConvCifar10$



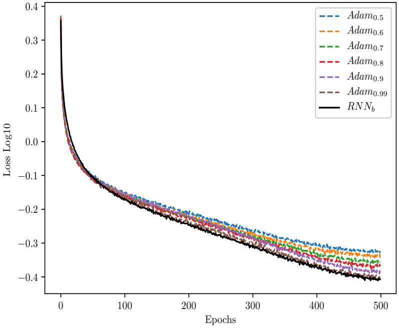


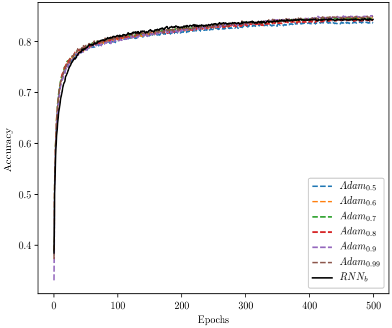
Loss

Accuracy

η decayed from $1e-2$ to $5e-4$

*ConvCifar*10





Loss

Accuracy

η decayed from $5e-4$ to $5e-5$

Issues and Limitations

Training on large networks takes hours.

Slower than individual input *Adam* timescales.

Training on $CIFAR10$ failed to yield good results.

Learned weighted average of timescales might not be the best choice for other networks.

Questions