Learning to learn by Gradient descent by Gradient descent

Shrey Gupta, Aarsh Chotalia, Pratyush Menon

Introduction to Optimisation Problems And Gradient descent

- Machine Learning ⇒ Optimization of some function f:

$$Find\ \theta^* \in argmin_{(\theta \in \Theta)} f(\theta)$$

- Most popular method: Gradient descent (Hand-designed learning rate)

$$\theta_{t+1} = \theta_t - \alpha \nabla(\theta_t)$$

- Better methods for some particular subclasses of problems available, but this works well enough for general problems

The Key Concept

(Optimiser and Optimisee)

- Use **learned** update rule instead:

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t)), \phi)$$

- g is the *optimiser* function parameterized by ϕ
- It is implemented using a RNN which maintains its own state h_t
- It outputs the update rule g_t to be used for the optimisee function f

Loss Function

- The loss function for g is defined as:

$$L(\phi) = E_f[\sum_{t=1}^{T} \omega_t f(\theta_t)]$$

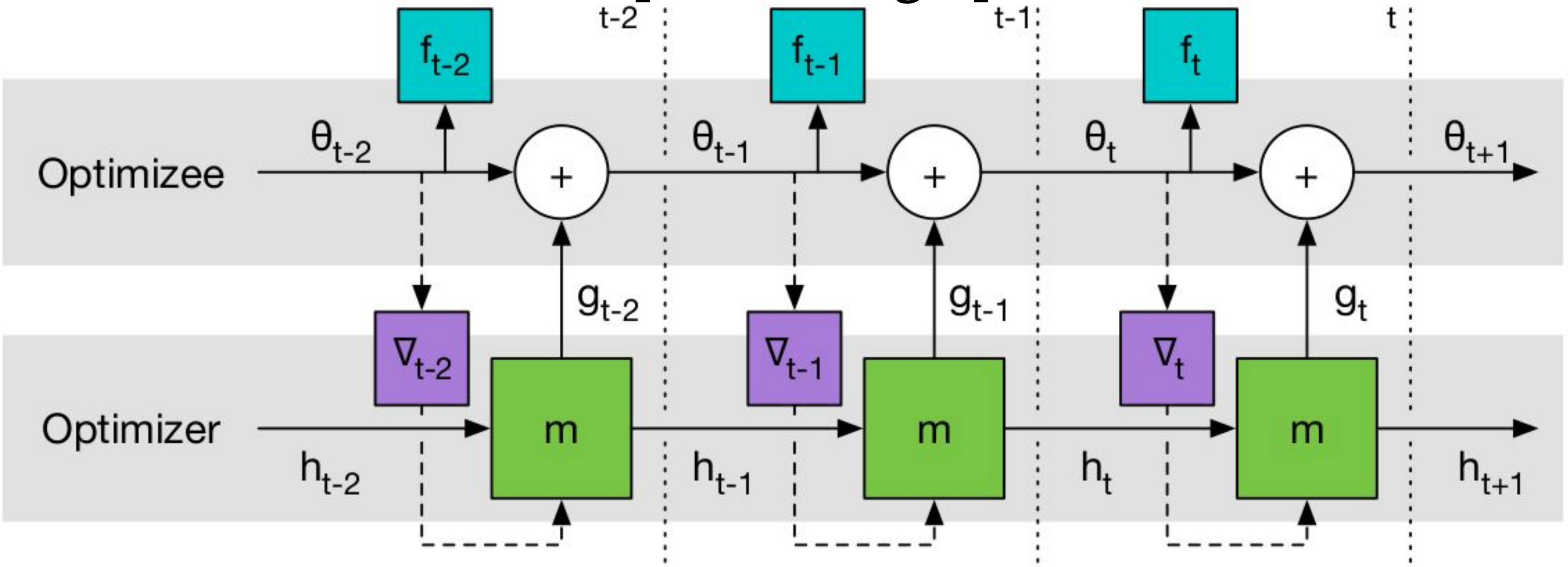
$$\theta_{t+1} = \theta_t + g_t;$$

$$where \begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} = m(\nabla_t, h_t, \phi)$$

- For this loss function, we train the optimiser for different datasets
- This is done using gradient descent

(Hence the title)

Computational graph



Assuming $\delta\nabla_t/\delta\phi = 0$, i.e. no need to compute second or higher derivatives of f. Hence gradients along the dashed lines neglected.

What we did

- Implemented optimiser function using LSTM (Long Short Term Memory) Architecture using the PyTorch library to utilise the .backward() function to conveniently calculate the gradients to be used in meta optimizer.

- Compared this with industry standard hand-designed techniques like ADAM, RMSprop, to do the same as the meta optimiser and compared their performance. We obtained similar results for the quadratic and MNIST databases

Challenges of implementation

- Needed to detach gradients from computational graph of pytorch to feed them to the meta optimizer
- Used CUDA to speed up the processing of the algorithm
- Had to make a new optimizee with the new parameters each time the meta optimizer is unrolled.
- Preprocess the gradients as the range can be quite large

$$\nabla^k \rightarrow \begin{cases} \left(\frac{\log(|\nabla|)}{p}, \operatorname{sgn}(\nabla)\right) & \text{if } |\nabla| \geq e^{-p} \\ (-1, e^p \nabla) & \text{otherwise} \end{cases}$$

Quadratic Loss function

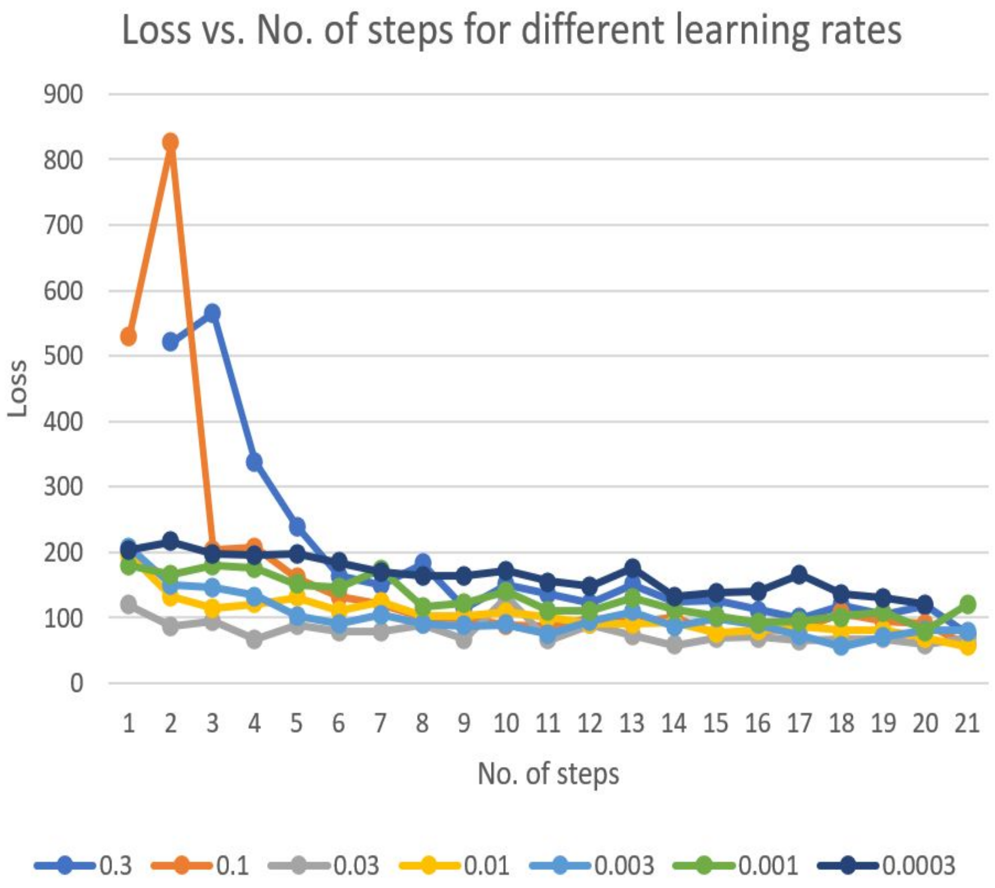
- Loss is defined simply as the quadratic loss:

$$L(x) = ||Wx - y||_2^2$$

- Objective: To find 10 dimensional vector x as close as possible to a given 10 dimensional vector y
- W is a 10*10 matrix

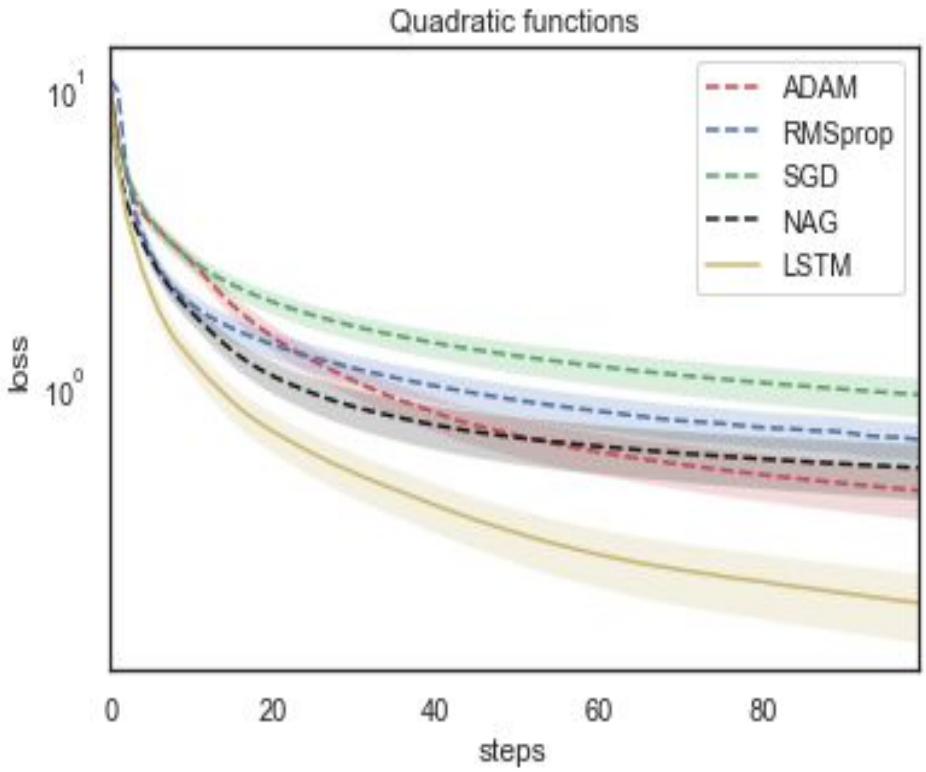
Best learning rate:

(For meta-optimiser)



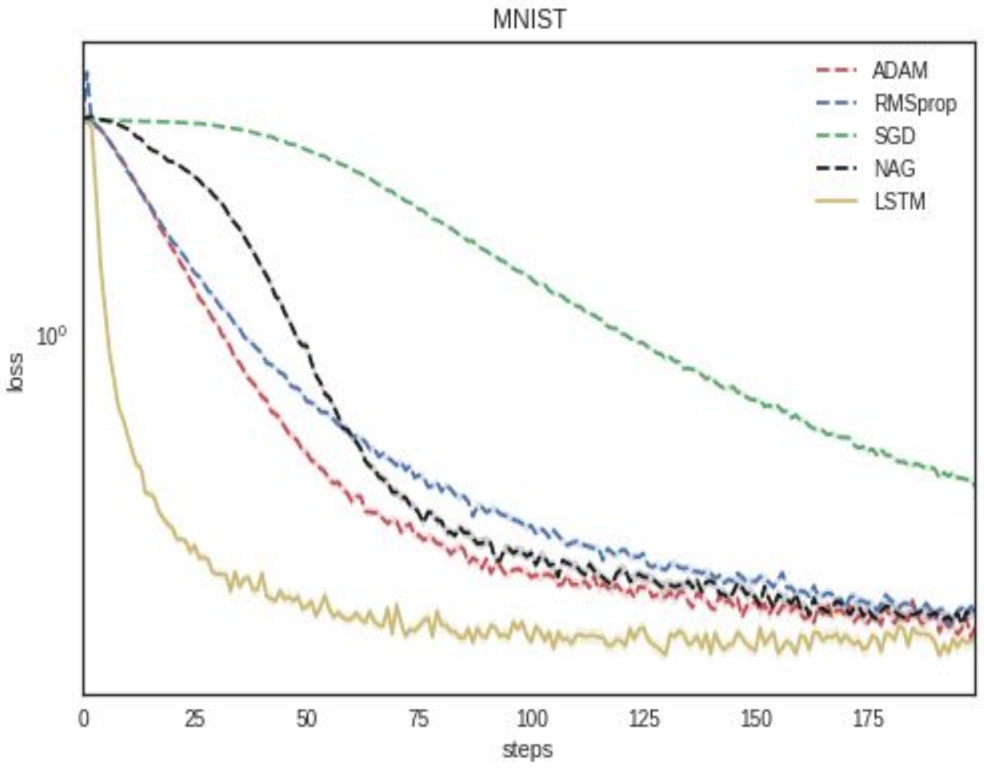
Hence the best learning rate= 0.003

Quadratic function: Comparison with hand-designed techniques



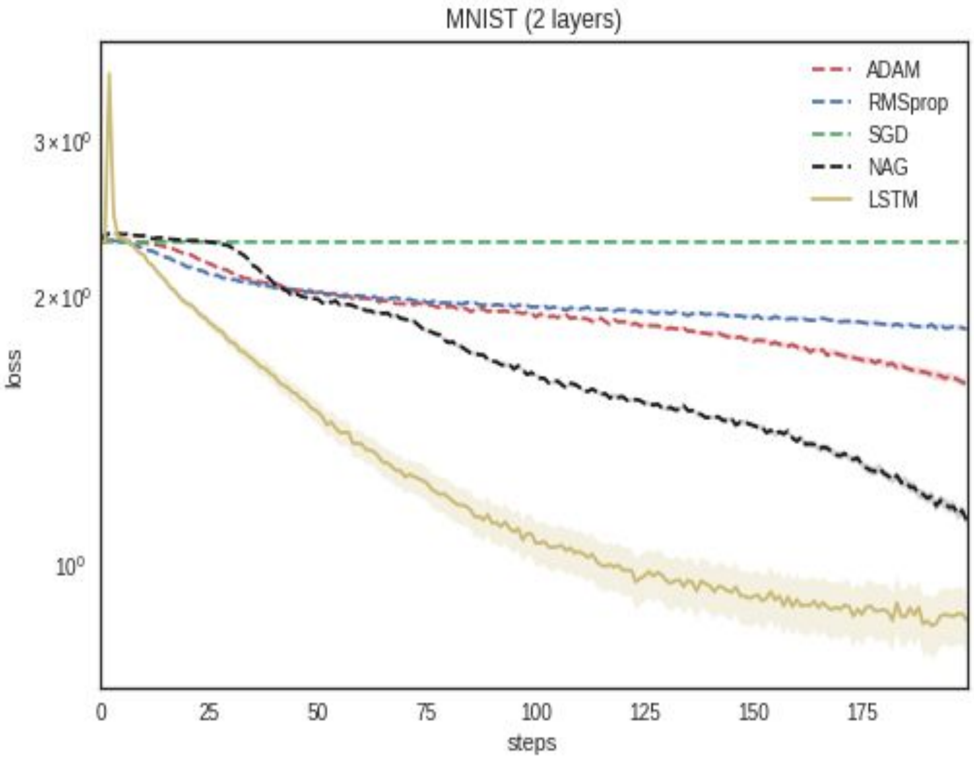
(W and y chosen randomly)

MNIST database: Comparison

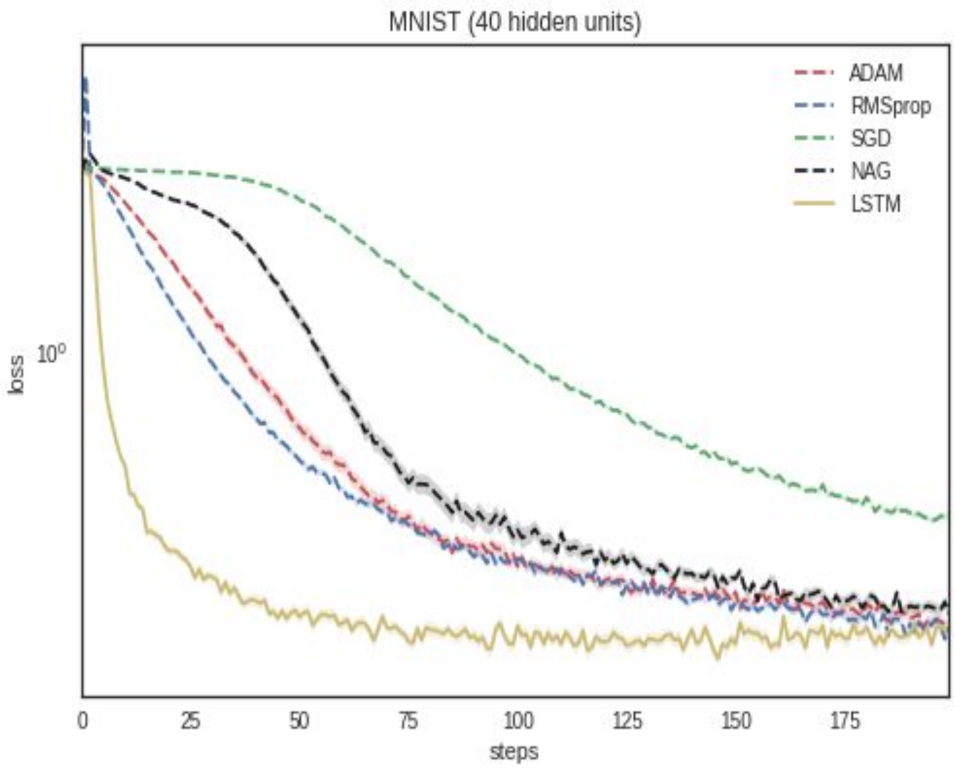


Sigmoid Function, 1 Hidden layer, 20 nodes

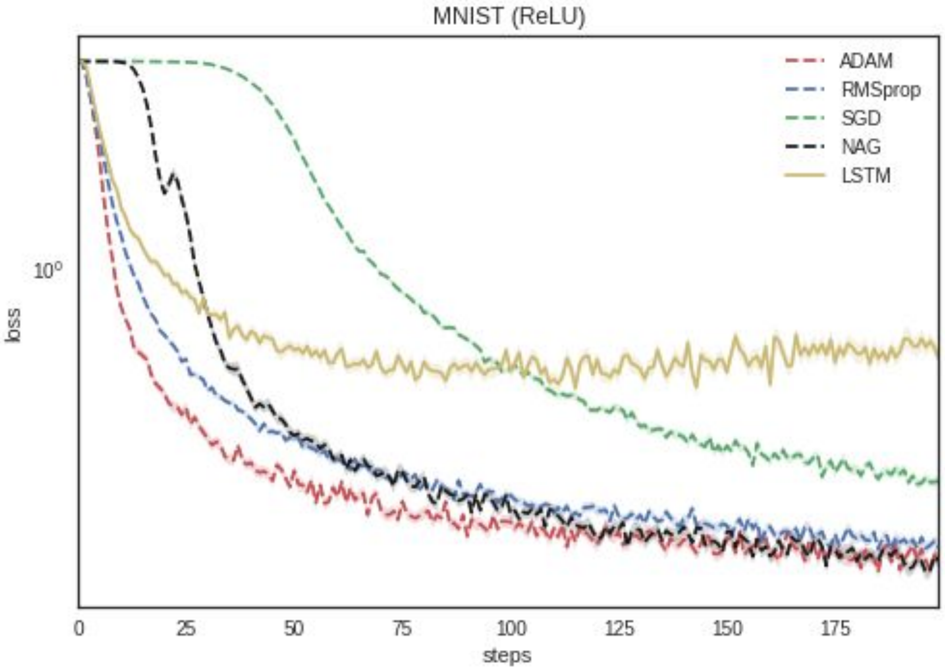
Variation:
2 Hidden Layers:



Variation: 40 hidden units



Variation: ReLU Activation Function



This is the only scenario where our LSTM meta-optimiser does not perform better than other standard methods

Proposal

- A three layer network in which the third network is used to decide the update rule for the optimiser which, in turns decides the update rule for the meta optimiser.
- The second network then will be a simple multi-perceptron network, and the third will be the LSTM cell used here.
- Since the challenge of programming just the meta optimizer is complicated enough for us, we have not tried to implement the structure here, but hope to work further.

Conclusion

- Meta-optimizer outperforms hand-designed algorithms in most of the cases
- But, this method is time-expensive so
  - It may not find use in live-time prediction algorithms (e.g. Automated driving)
  - But can certainly improve data analysis and prediction tasks (e.g. in fields of Astronomy and particle physics)
- Also emphasises the importance of what is called transfer learning where concepts learned in similar, simple settings can be used to predict outcomes in more complex scenarios.

References

1. Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pp. 3981–3989, 2016.
2. Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121– 2159, 2011.
3. Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv*:1412.6980, 2014.
4. Tieleman, T. and Hinton, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4 (2):26–31, 2012
5. Wolpert, D. H., Macready, W. G., et al. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.