Deep Learning [CSL4020] Major Project Report

Shalin Jain (B21CS070), Shashwat Roy (B21CS071)
Shubh Goyal (B21CS073), Sukriti Goyal (B21CS075)

The code can be found here: GitHub Repository

1 Related Work and Literature Review

The key idea behind Adam is to compute adaptive learning rates for each parameter by incorporating estimates of both the first and second moments of the gradients. This allows Adam to adapt the step size for each parameter based on the observed gradients, leading to faster convergence and better performance compared to traditional stochastic gradient descent (SGD) methods.

1.1 Advancements After Adam

- **AdamW[4]** It decouples weight decay from learning rate, thus improving Adam's generalization ability, enabling performance competitive with SGD momentum for image classification tasks where vanilla Adam underperforms.
- **RAdam[3]** Rectified Adam (RAdam) introduces a new term on the lines of learning rate warmup, rectifying the variance of the adaptive learning rate.
- **Lookahead[5]** In this algorithm, two sets of weights are updated iteratively. Intuitively, the algorithm chooses a search direction by looking ahead at the sequence of "fast weights" generated by another optimizer. It helps improve the learning stability and lowers the variance of its inner optimizer.
- **AdaBelief[6]** It introduces an exponentially moving average based 'belief' factor based on the magnitude and sign of the current gradient instead of directly averaging gradients for the updation of weights and biases. It provides increased stability and quality in GAN samples.
- **Dynamical Learning Rate Scheduling** Adaptively adjusts the learning rate during training, especially across different stages. YellowFin6, an automatic tuner for momentum and learning rate in SGD, is one such example. YellowFin optionally uses a negative-feedback loop to compensate for the momentum dynamics in asynchronous settings on the fly.

2 Problem Statement

Finding the perfect model parameters giving the minimum loss is like searching for a needle in the loss landscape. However, optimizers automate and fasten the searching process through various techniques like gradient descent, analyzing gradients (partial derivatives of the loss w.r.t. parameters) to steer updates towards lower loss regions. Popular optimizers such as SGD, RMSProp and Adam further introduce momentum and gradient accumulation techniques for better learning rate adaptation, and parameter adjustments. But still, there remains room for making them faster and more efficient. We focus on formulating, developing and analyzing a new optimizer, by introducing further adaptive techniques in the existing optimizers.

2.1 Focus Areas of our Approach

The existing optimizers, described previously (section 1.1), have many shortcomings, out of which we have focused on the following while implementing our approaches.

- Adaptive learning rate methods often use heuristics like moving averages or squared gradients, which might not accurately capture the true variance of the loss landscape.
- Techniques like momentum that sometimes help in escaping the local minima rely on specific parameter choices and might not be effective in all scenarios.

3 Experiments

To ease experimentation, we programmed our own Adam by taking references from the actual PyTorch implementation of Adam. The code for the same can be found in our GitHub repository[1].

3.1 Scaling β_1 and β_2

Optimization algorithms like Adam have tunable hyperparameters (coefficients in moments updates). We tried various scheduling algorithms for β_1, β_2 and compared the performance of a fully connected neural network model on MNIST and CIFAR-10 datasets against the Pytorch Adam optimizer. The performance was comparable to Vanilla Adam, RMSProp, but better than AdaGrad.
We tried to schedule β_1 and β_2 by replacing β_1 with $(\beta_1)e^{-t}$, and β_2 with $(\beta_2)e^{-t}$. We did this because as epochs keep on progressing, we have to reduce the weightage of past gradients and increase the weightage of current gradients. This was explicitly suggested in the Adam research paper.

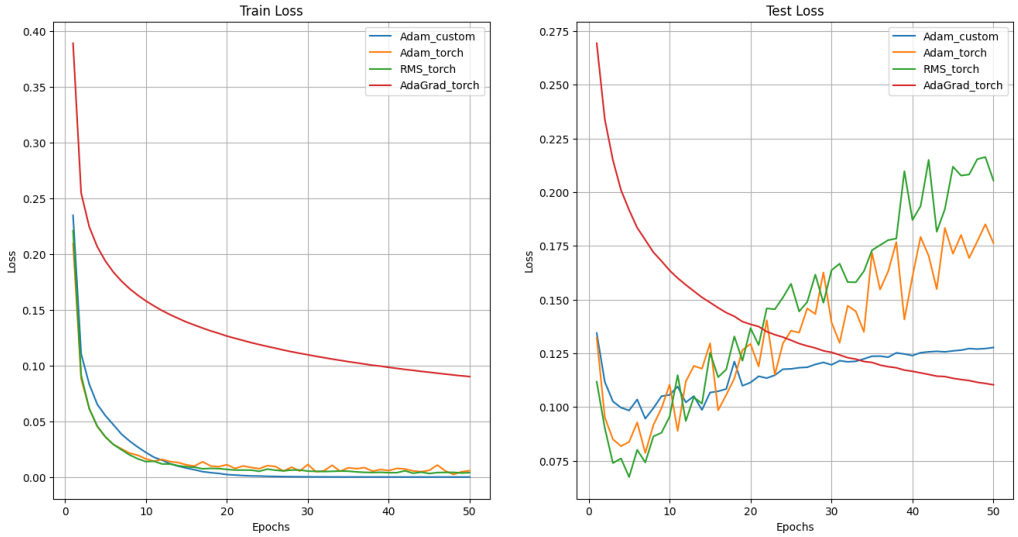

Fig. 1. Loss Curves showing performance of different optimizers on MNIST. Here Adam_custom refers to our implementation of Adam with the scheduled betas. Adam_torch refers to the Vanilla Adam. AdaGrad_torch is the Vanilla AdaGrad and RMS_torch refers to the Vanilla RMSProp.

[1] `https://github.com/Shubh-Goyal-07/DLProject`

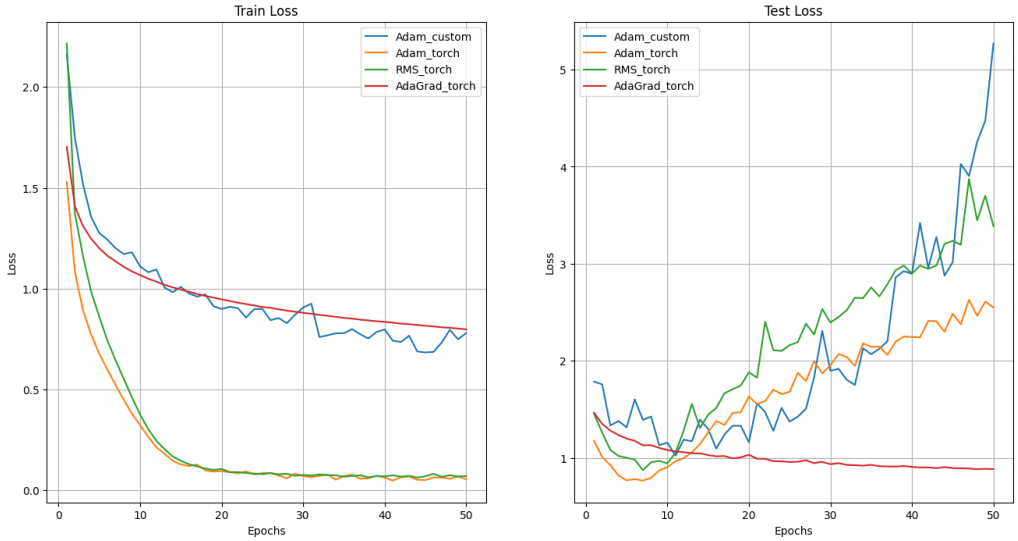

Fig. 2. Loss Curves showing performance of different optimizers on CIFAR-10. Here Adam_custom refers to our implementation of Adam with the scheduled betas. Adam_torch refers to the Vanilla Adam. AdaGrad_torch is the Vanilla AdaGrad and RMS_torch refers to the Vanilla RMSProp.

3.2 Higher Order Moments

The Adam optimizer incorporates the accumulated first and second-order moments. We can consider replacing a lower-order moment with a higher-order moment or incorporating the higher-order moments in the original equation.
A concept of skewness can be adopted in this context to help characterize the analysis. The skewness value of stochastic gradients can be negative, positive, or undefined, and based on the nature of distribution for each condition, we can incorporate skewness into Adam as the third raw moment. Skewness would help us with the non-normal distributions, enabling the machine learning models to perform better on datasets with a non-normal distribution.

Incorporating Third Order Moment We tried to incorporate the third-order raw moment along with the first and second-order moments to give some weight to the skewness. We used the following update rule:

$$
\begin{aligned}
s &\leftarrow \beta_1 s + (1-\beta_1) g \\
r &\leftarrow \beta_2 r + (1-\beta_2) g \odot g \\
q &\leftarrow \beta_3 q + (1-\beta_3) g \odot g \odot g \\
\hat{s} &\leftarrow \frac{s}{1-\beta_1^t} \\
\hat{r} &\leftarrow \left(\frac{r}{1-\beta_2^t}\right)^{\frac{1}{2}} \\
\hat{q} &\leftarrow \left(\frac{q}{1-\beta_3^t}\right)^{\frac{1}{3}} \\
w &\leftarrow w - \frac{\hat{s}}{\hat{r}+\hat{q}+\epsilon}
\end{aligned}
$$

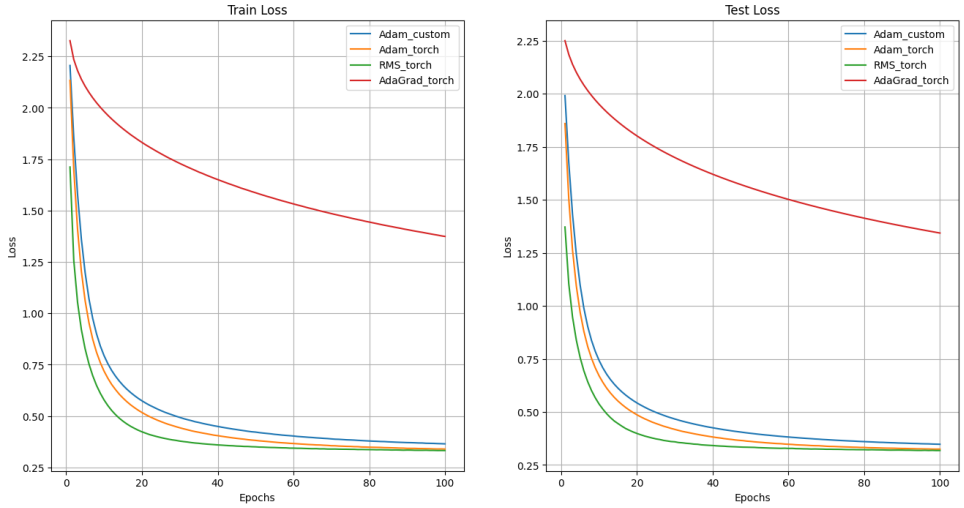

Fig. 3. Loss curves for fully-connected NN using different optimizers on MNIST. Here Adam_custom refers to our implementation of Adam with the scheduled betas. Adam_torch refers to the Vanilla Adam. AdaGrad_torch is the Vanilla AdaGrad and RMS_torch refers to the Vanilla RMSProp.

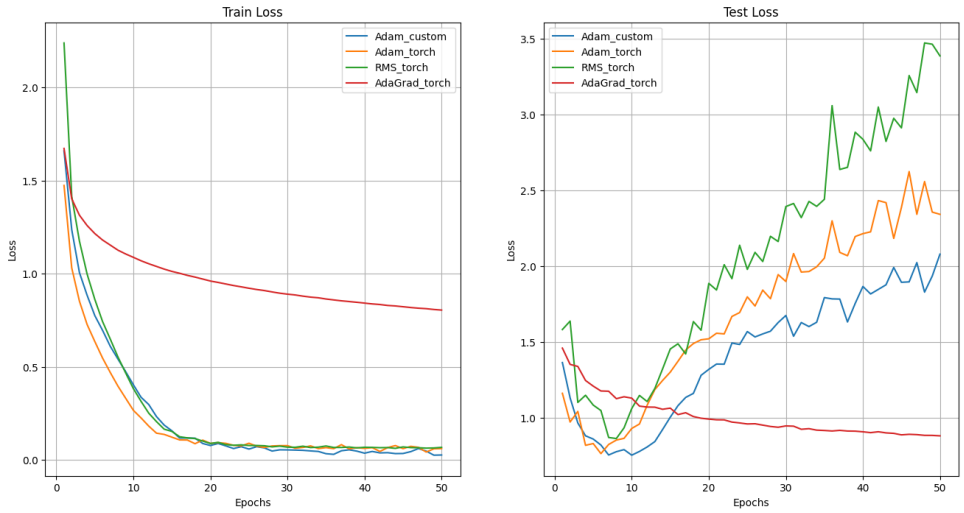

Fig. 4. Loss curves for fully-connected NN using different optimizers on CIFAR10. Here Adam_custom refers to our implementation of Adam with the scheduled betas. Adam_torch refers to the Vanilla Adam. AdaGrad_torch is the Vanilla AdaGrad and RMS_torch refers to the Vanilla RMSProp.

The results that we got with the custom Adam with higher moments were again comparable to Adam and other optimizers as in the case of the first experiment.

Replacing Second Order Moment with Third Order Moment Just as an additional experiment, we tried to replace the 2nd order moment with the 3rd order moment in the original Adam. We had to modify the update rule accordingly to incorporate the

first and third-order moments. The update rule was as follows:

$$
\begin{aligned}
s &\leftarrow \beta_1 s + (1-\beta_1) g \\
r &\leftarrow \beta_2 r + (1-\beta_2) g \odot g \odot g \\
\hat{s} &\leftarrow \left(\frac{s}{1-\beta_1^t} \right)^{\frac{3}{4}} \\
\hat{r} &\leftarrow \left(\frac{r}{1-\beta_2^t} \right)^{\frac{1}{4}} \\
w &\leftarrow w - \frac{\hat{s}}{\hat{r} + \epsilon}
\end{aligned}
$$

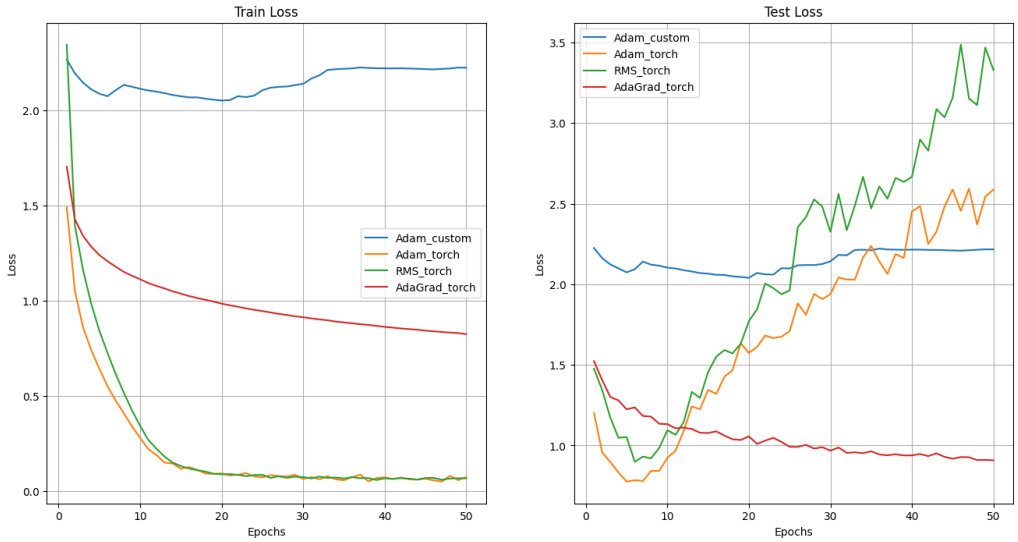

Fig. 5. Loss curves for the CNN models trained on the CIFAR10 dataset with different optimizers. The model using the custom optimizer does not show any remarkable improvement in training.

3.3 Learning Rate Scheduling

The number of saddle points in a larger network with millions of parameters is significantly high. So to make sure that the optimizer doesn't get stuck at any of these saddle points, we decided to schedule the learning rate. The algorithm for the same is described below as Triangular Learning Rate.

Triangular Learning Rate The motivation behind this idea is that during training, the model may encounter a saddle point due to which the gradient of the loss function becomes close to zero. This leads to insignificant updates in the trainable parameters. However, if we increase the learning rate linearly for a specific number of epochs and then decrease it in the same manner for the same number of epochs, the model might be able to escape the saddle point. The gradient may be zero when the model encounters a minima as well. However, scheduling the learning rate in this instance will most likely not lead to the model skipping past the minima since the gradient calculated after an update to the

parameters will be opposite to the direction of the minima. Thus, the gradient descent will always move the state closer to the minima.

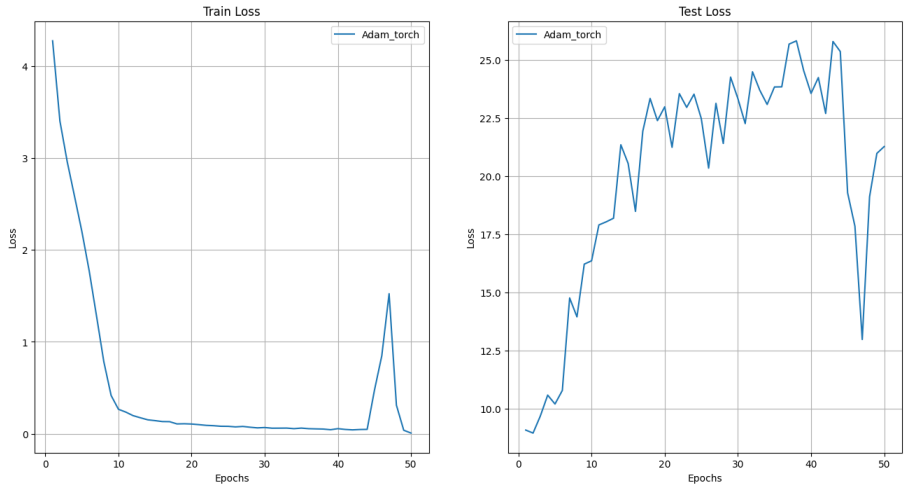

Fig. 6. Loss curves for the lr-scheduled Resnet18 model trained on the tiny ImageNet dataset. The sharp rise in the training loss curve corresponds to the learning rate being scheduled at the corresponding epochs. This scheduling also shows a marked drop in the test loss of the model.

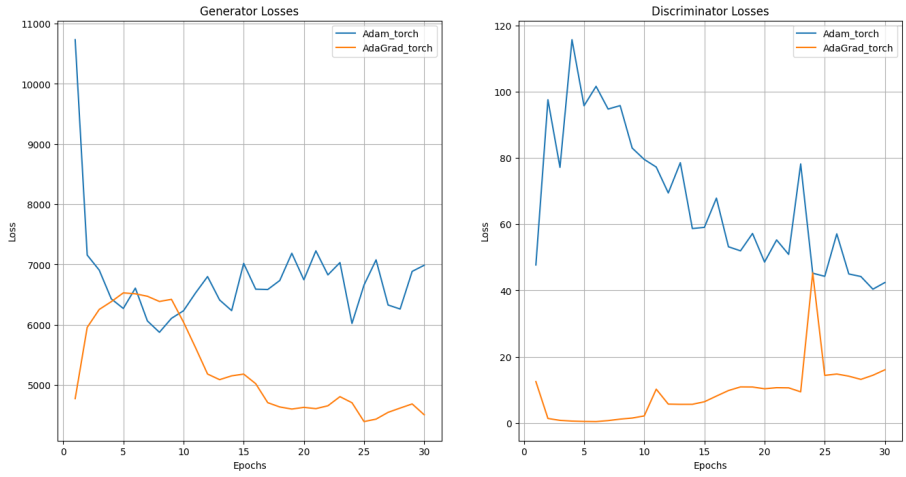

Fig. 7. Loss curves for the CNN models trained on the CIFAR10 dataset with different optimizers. The model using the custom optimizer does not show any remarkable improvement in training.

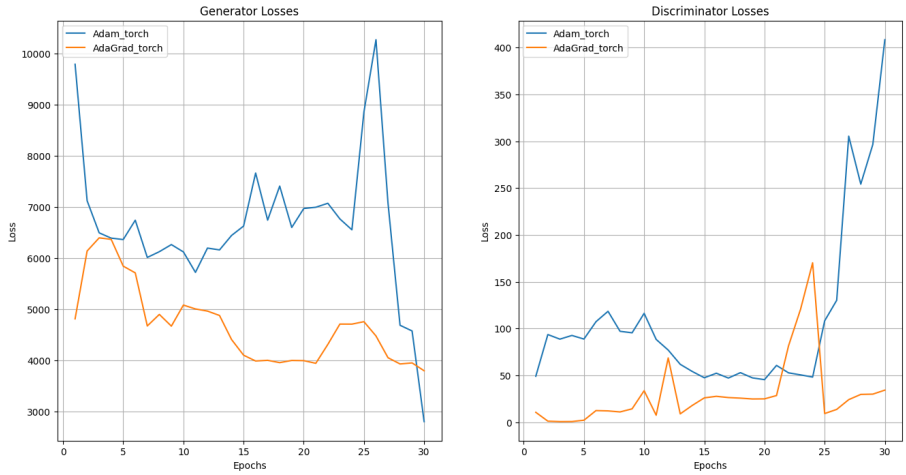

Fig. 8. Loss curves for the CNN models trained on the CIFAR10 dataset with different optimizers. The model using the custom optimizer does not show any remarkable improvement in training.

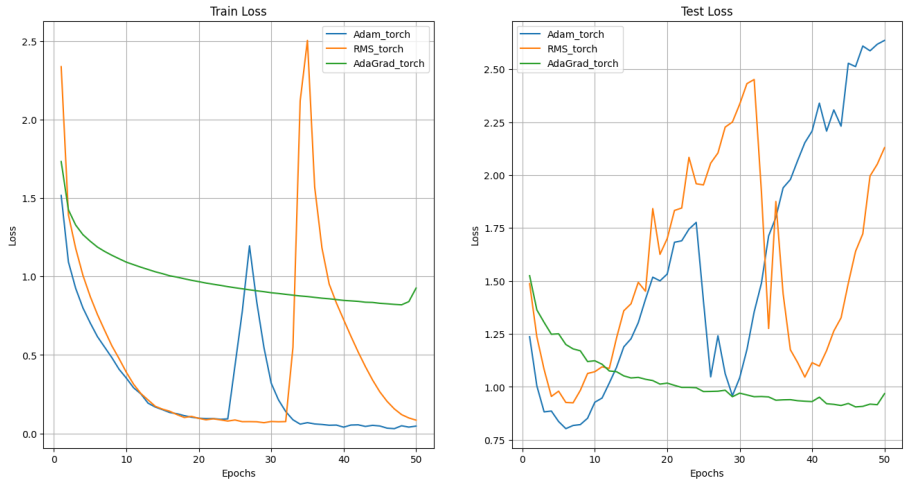

Fig. 9. Loss curves for the lr-scheduled Resnet18 models trained on the tiny Imagenet dataset with different optimizers. The peaks in the training loss curves the models also correspond with sharp drops in test losses in each model's respective loss curves.

4 Our Approach

After conducting all the experiments described above, we finally decided to incorporate the second-order derivative of loss with respect to the weights, or the gradient of the gradient.
As is stated in the deep learning book by Begnio, Goodfellow and Courville[2], the second derivative tells us how the first derivative will change as we vary the input. This is important because it tells us whether a gradient step will cause as much of an improvement as we would expect based on the gradient alone. We can think of the second derivative as measuring curvature.
We know that at saddle points, the value of both, the second and the first derivative is 0. At such a point, it is the past accumulated values of the gradients that would help the optimizer to propel itself out of the saddle point, thus preventing it from getting stuck in a saddle point. Thus to accelerate the convergence, we reasoned that the value of β_1 and β_2 in the vanilla Adam should tend to 1 so that the weight of the past accumulated gradients is significant and the convergence is accelerated.
Thus we developed an update rule to update β_1 and β_2 according to the logic defined above. Since only the magnitude of the second derivative matters, therefore, we have used the absolute value of the second derivative.
Furthermore, since we are scheduling the values of β_1 and β_2, therefore, we reasoned that there is no requirement of the bias correction term as in the beginning, the value of β_1 and β_2 is small in the beginning when the value of second derivative is non-zero.

$$\beta = \begin{cases} \beta_o, & \text{if } p < |g'| < 1 \\ \beta_o^{|g'|}, & \text{otherwise} \end{cases}$$

where, p refers to the precision of the hardware, beyond which the value is practically 0. We are defining this range because this update of β is particularly for the case when we encounter a saddle point or a shallow minima.

4.1 Proof of Concept

As we had to incorporate the second order gradient, which would have required modifying the backward function of PyTorch, therefore, we could not test our optimizer approach with actual datasets and models. Thus, to provide a proof of the concept, we applied our optimizer alongwith 3 others, vanilla Adam, AdaDelta, and RMS-Prop on the standard single-optimization objective test functions[1]. The visualisations of the results that we got can be seen below.

Inference It can be seen clearly that on many functions, like *Booth*, *Bukin*, *Himmelblau*, *Matyas*, *Sphere*, *Styblinski-Tang*, and the standard *Saddle function*. For some others like the *Beale*, *Easom*, *Goldstein-Price*, *Rosenbrock*, and the *Three-Hump Camel* functions, the convergence, however was slower than other optimizers. This indicates that with some more analysis and subsequent modification, the optimizer that we have proposed here can be improved significantly.

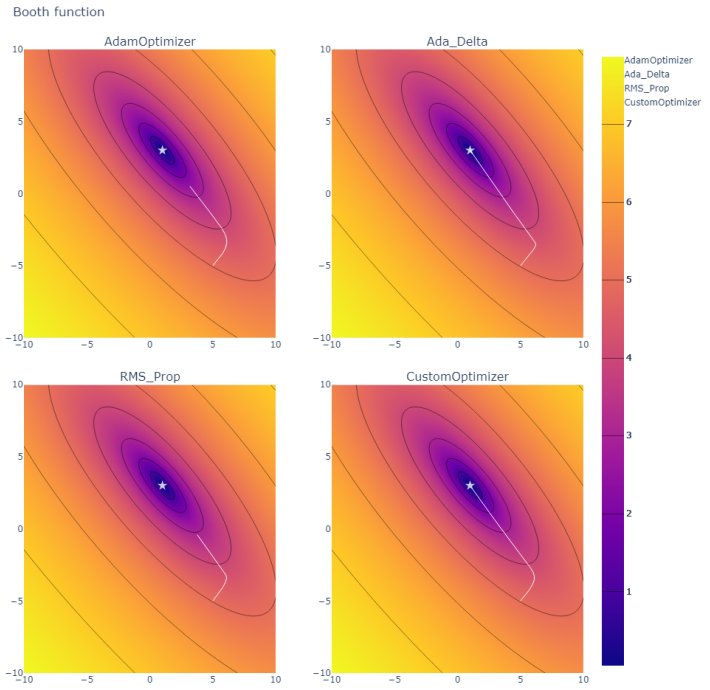

Fig. 10. Contour plots visualizing the convergence of optimization algorithms on Booth function. The custom optimizer converges relatively faster than Adam and RMS Prop.

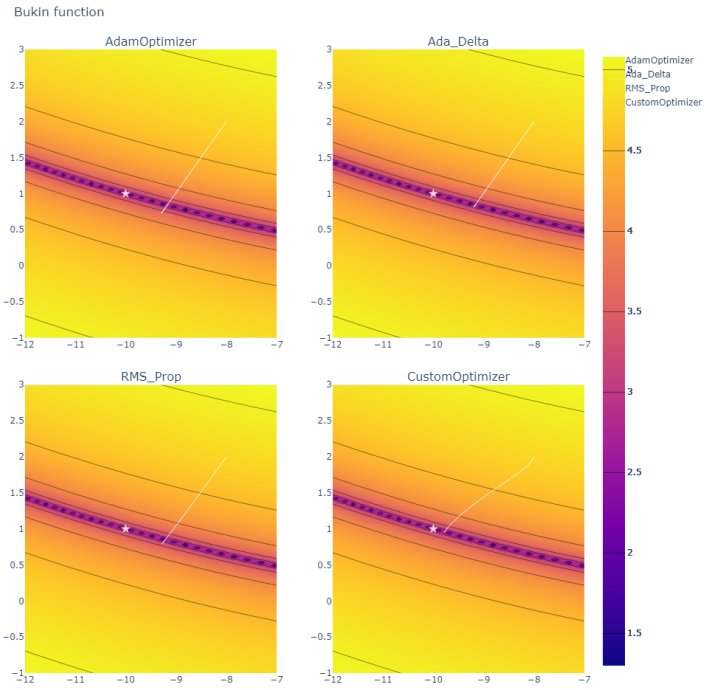

Fig. 11. Contour plots visualizing the convergence of optimization algorithms on Bukin function. The custom optimizer converges relatively faster than Adam and RMS Prop.

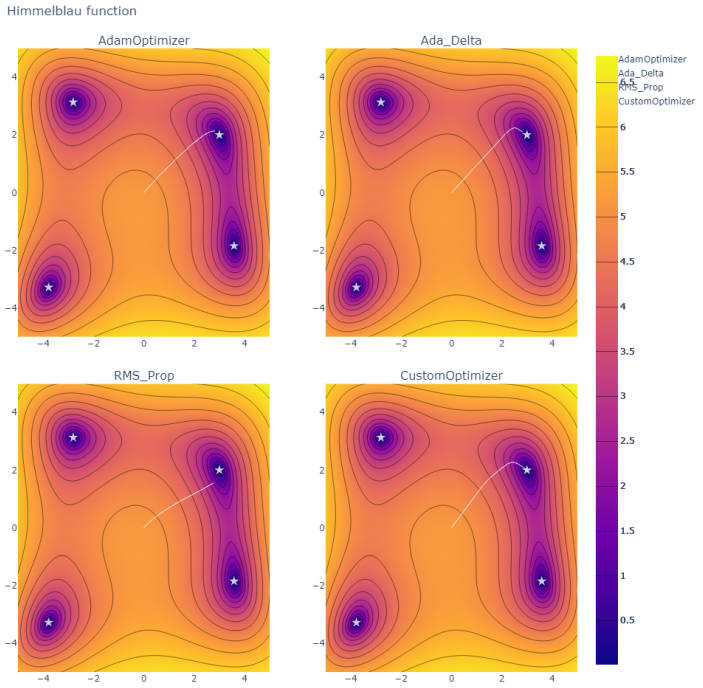

Fig. 12. Contour plots visualizing the convergence of optimization algorithms on Himmelblau function. The custom optimizer converges at a similar rate to the other optimizers.

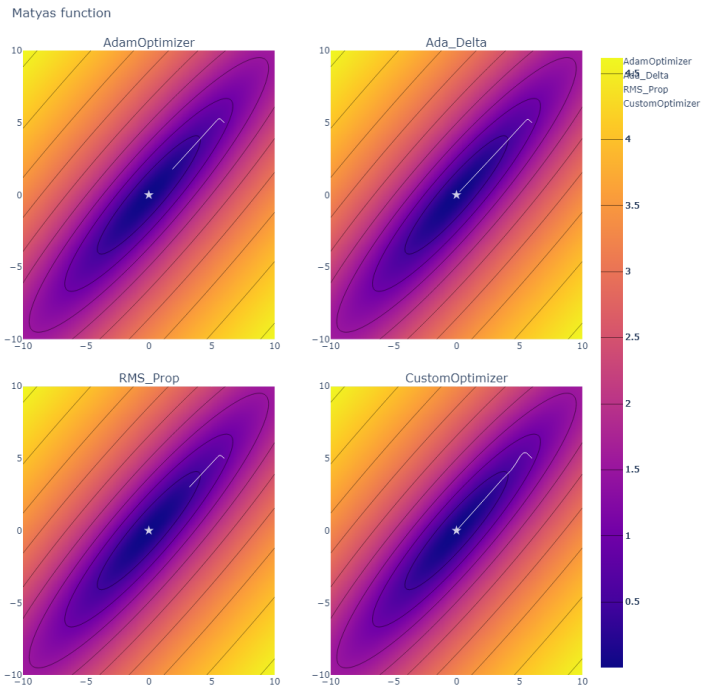

Fig. 13. Contour plots visualizing the convergence of optimization algorithms on Matyas function. The custom optimizer converges the fastest out of the four.

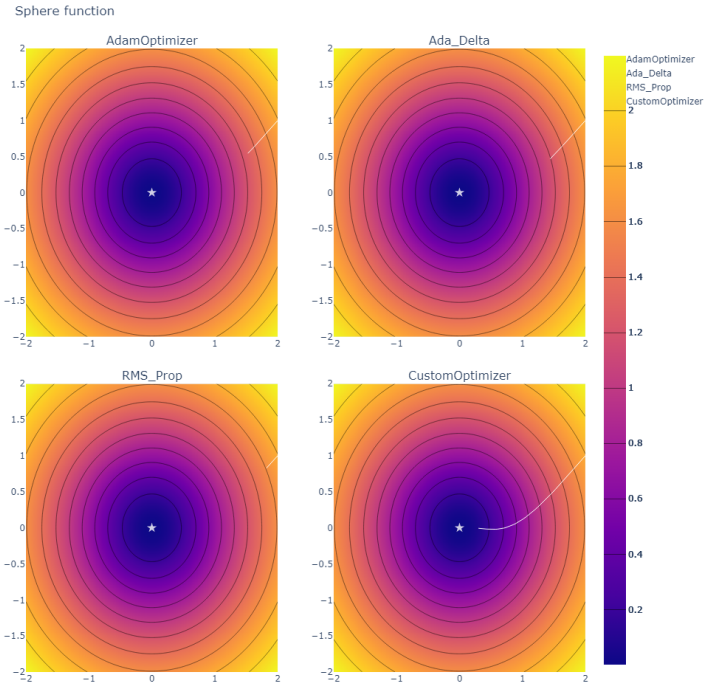

Fig. 14. Contour plots visualizing the convergence of optimization algorithms on Sphere function. The custom optimizer converges the fastest out of the four.

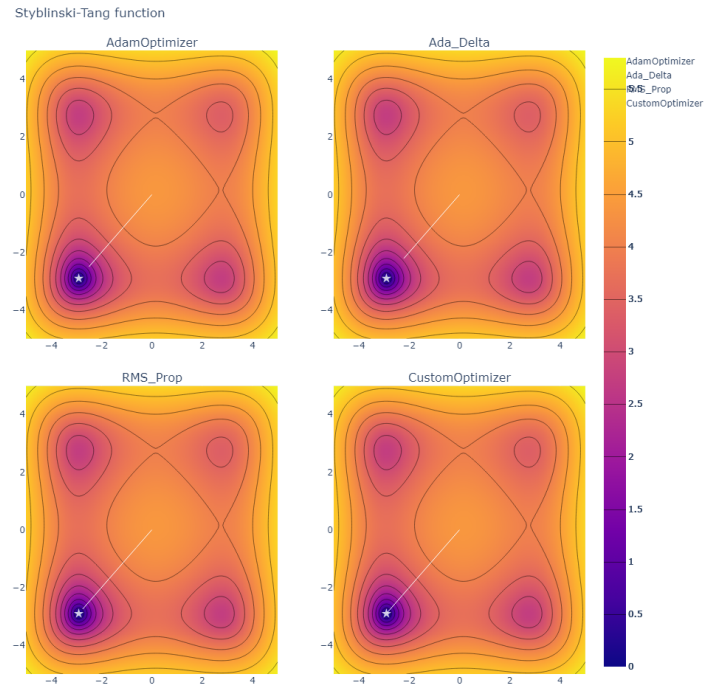

Fig. 15. Contour plots visualizing the convergence of optimization algorithms on Styblinski-Tang function. The custom optimizer converges the fastest out of the four.

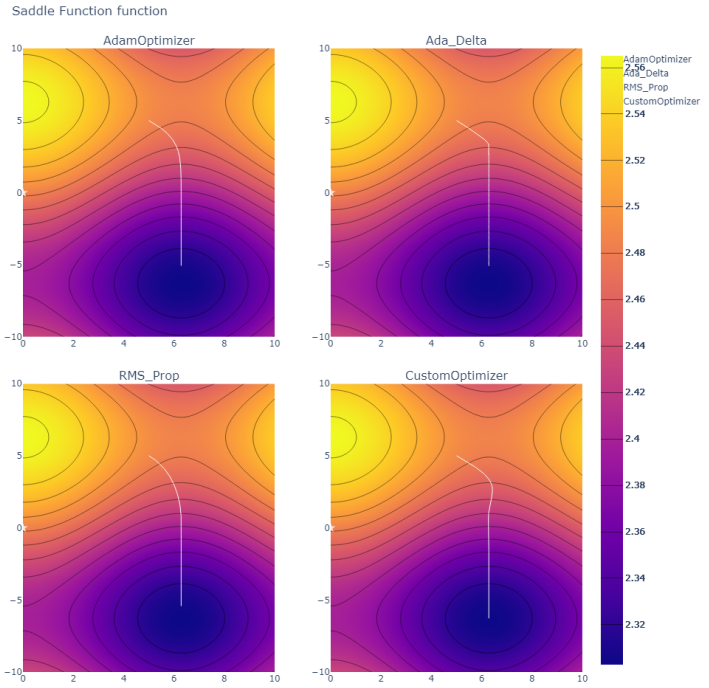

Fig. 16. Contour plots visualizing the convergence of optimization algorithms on Saddle function. The custom optimizer converges at a similar rate to Adam and AdaDelta.

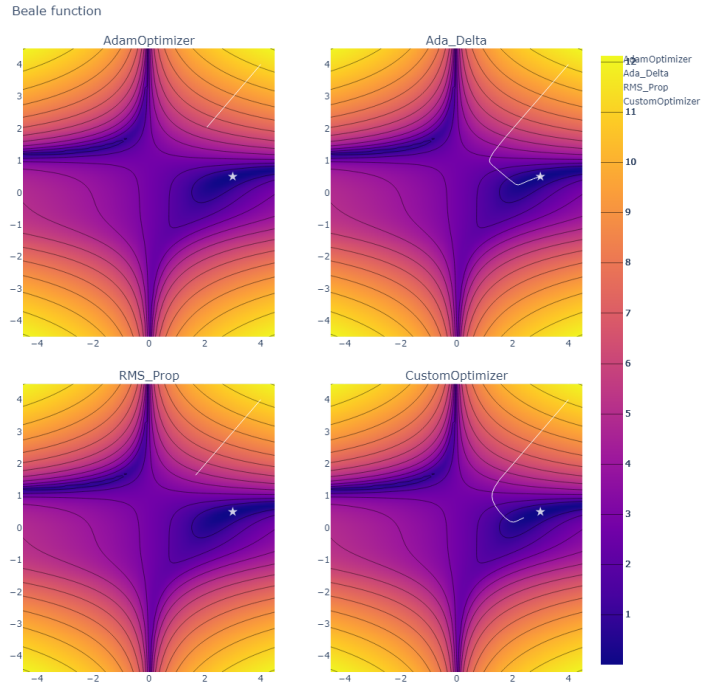

Fig. 17. Contour plots visualizing the convergence of optimization algorithms on Beale function. The custom optimizer converges significantly faster than Adam and RMS Prop.

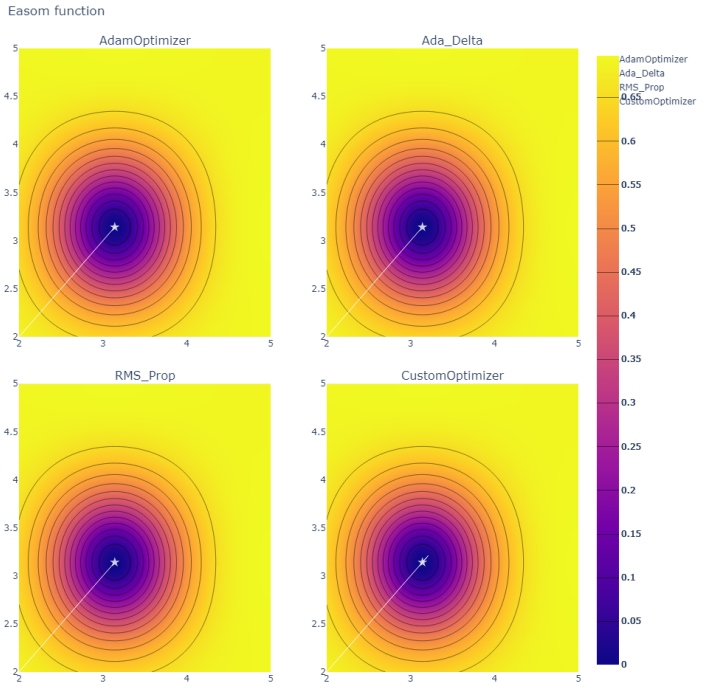

Fig. 18. Contour plots visualizing the convergence of optimization algorithms on Easom function. The custom optimizer slightly overshoots the minima.

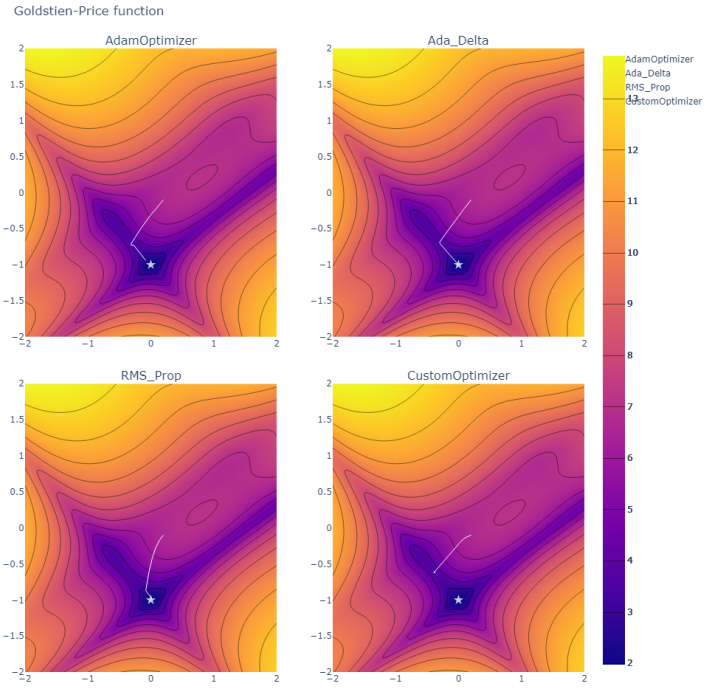

Fig. 19. Contour plots visualizing the convergence of optimization algorithms on Goldstein-Price function. The custom optimizer performs relatively slow against the other optimizers.

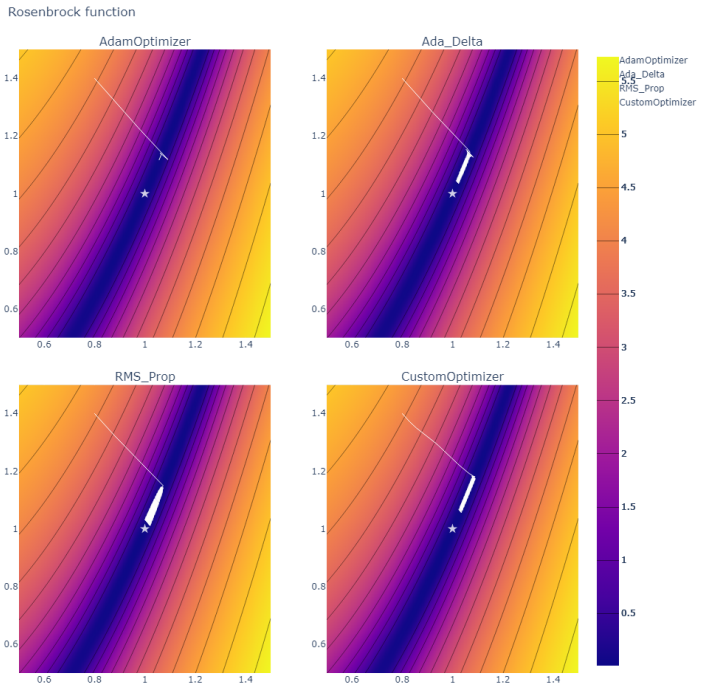

Fig. 20. Contour plots visualizing the convergence of optimization algorithms on Rosenbrock function. The custom optimizer performs comparably to AdaDelta and RMS Prop.

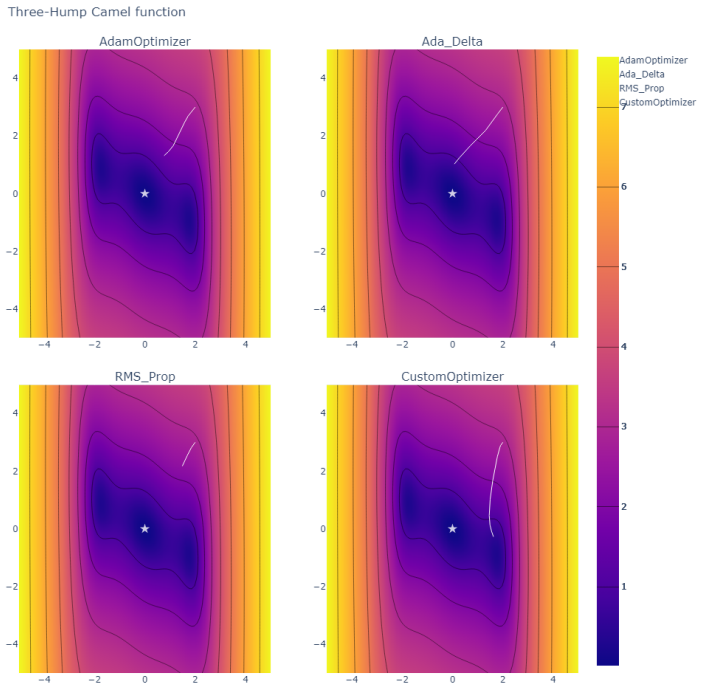

Fig. 21. Contour plots visualizing the convergence of optimization algorithms on Three Hump Camel function. The custom optimizer deviates from the minima during convergence.

5 Conclusion

Through this optimizer, we have successfully proven that inclusion of the second derivative in the optimization functions will significantly impact the performance of the optimizers for the better. In most of the graphs that we have added in this report, it is very clearly visible that the convergence speed has increased significantly. Currently the update rule is very standard and needs to be worked upon to make sure that it performs well on all the testing functions.
Furthermore, to test this approach with actual models and datasets, we have to develop a method to calculate the second derivative using the chain rule, quite like the calculation of the first derivative of loss, which is gradient.

References

1. Wikipedia contributors. Latex. Wikipedia, The Free Encyclopedia, 2022. [Online; accessed April 11, 2022].
2. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.
3. Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. 2021.
4. Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. 2019.
5. Michael R. Zhang, James Lucas, Geoffrey Hinton, and Jimmy Ba. Lookahead optimizer: k steps forward, 1 step back. 2019.
6. Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James S. Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients. 2020.

Team Contributions

Shalin Jain (B21CS070) developed the initial Adam overwriting codebase and made any further required changes to Adam for experimentation in the code. Also worked on the proof of concept.

Shashwat Roy (B21CS071) ideated and implemented the Triangular Learning Rate algorithm and the Higher Order Moments algorithm. Developed the codebase for scheduling the GAN training and testing.

Shubh Goyal (B21CS073) developed the repository code and the training loops for the various models and datasets. Worked on ideating and developing the proof of concept of the second derivative approach.

Sukriti Goyal (B21CS075) worked out the update rule of the second derivative approach and ideated the incorporation of skewness into vanilla Adam. Trained and tested the modified Adams on basic models and datasets.