Report

ADAM: A Method for Stochastic Optimization

Team name: Entropy Death

- **Vishal Reddy Mandadi** (2019101119)
- **Pranshul Chawla** (2019101057)
- **Kadari Ruthwik Reddy** (2019101121)
- **Naman Verma** (2019101066)
- TA advisor: **Abhinaba Bala**

GitHub Link: https://github.com/vishal-2000/SMAI-project (Restricted Access)

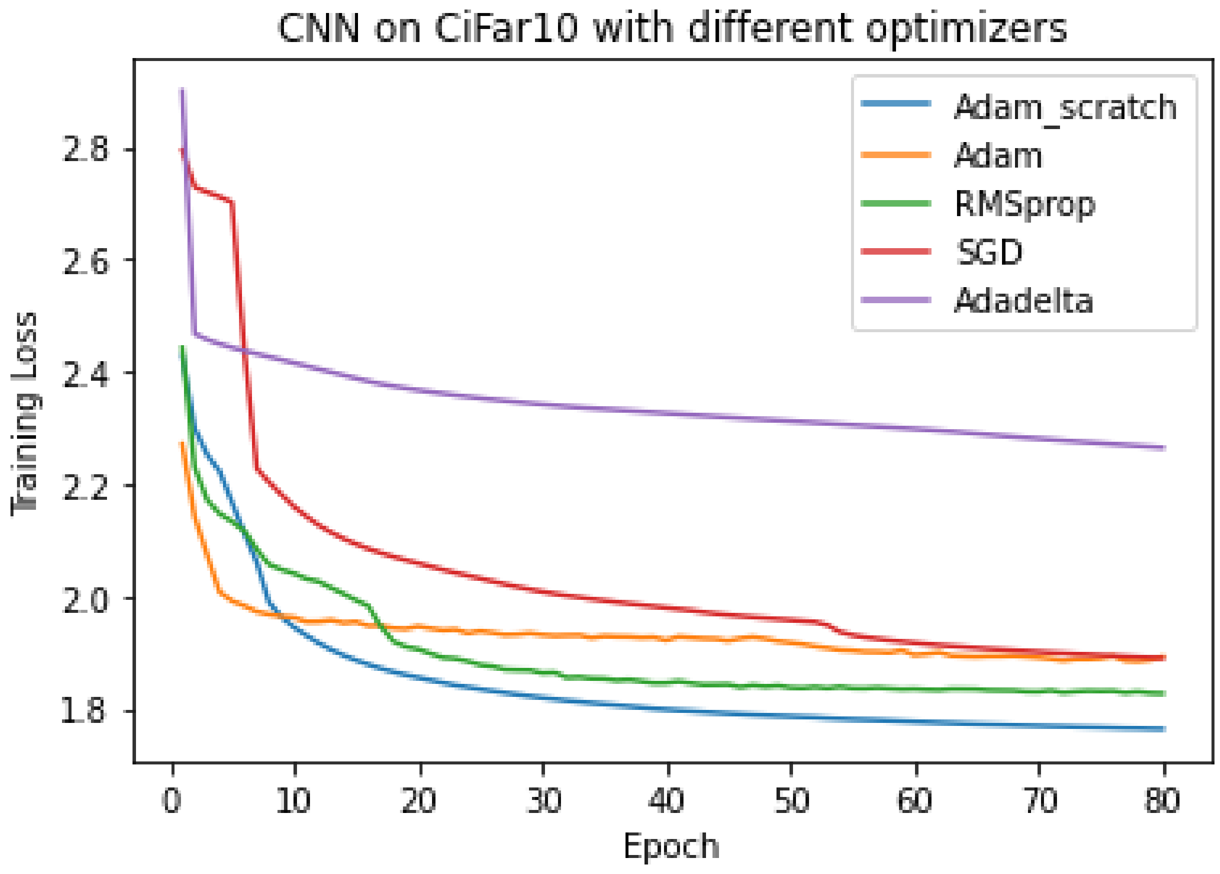
Submission contents:

Directory structure

```
.
├── ADAM.py
├── CNNCIFAR10_additional_results.ipynb
├── CNNCIFAR10.ipynb
├── graphs
│   ├── 50_epoch_cnn_seed8.png
│   ├── 80_epoch_cnn_seed17.png
│   ├── CNN_CIFAR10_50_epoch.png
│   ├── CNN_CIFAR10.png
│   ├── LR_IMDB.jpeg
│   ├── LR_MNIST.png
│   ├── MLP_MNIST.png
│   ├── MNIST_mideval_plot.png
│   ├── MNIST_Vishal_linear_activation.png
│   ├── MNIST_Vishal_linear_act_resize=14.png
│   └── Pranshul_linear_activation_resize=original=28.png
├── IMDB_LogisticRegression.ipynb
├── MNISTCNN.ipynb
├── MNIST_LogisticRegression.ipynb
├── MNIST_MLP.ipynb
├── README.md
├── report.pdf
├── results
├── vishal_cnn_saved_models-do-not-use.zip
└── visualizer.py

2 directories, 22 files
```

Introduction



Optimizers play a very important role in the world of AI. They are the backbone of the learning processes it is as important as the loss function itself. In the language of commons, they say the model what to learn from a given experience (experience being the loss). Gradient descent was the first-ever optimizer, introduced or rather suggested by Cauchy in 1847. Since the advent of backpropagation, AI picked up speed, and scientists have made attempts on a regular basis to solve and find the best optimizers for fast and robust training. This lead to the development of various optimizers as:

1. SGD

2. Momentum
3. Nesterov Accelerated GD
4. ADAGRAD
5. ADADELTA
6. RMSprop
7. ADAM

Each of them has a few variations too but are broadly classified above. All of these functions try to reach the minima of the loss function curve and how they do it is where they differ for example SGD has a constant learning rate and it tries to reach minima and it has a few issues too. Adam was proposed by considering the best of Adagrad and RMSprop and it solves issues raised because of them and is considered in general to be the best optimizer with minimal training loss and calculations.

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first-moment estimate) (Equation 1)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate) (Equation 2)

Equation 1 is actually from Adadelta which is the new learning parameter so the learning parameter is dependent on previous parameter and also current derivative.

Equation 2 is from RMSprop as we use a square of derivatives instead of derivatives for values to see the change in step.

The update step of Adam after scaling m_t and v_t appropriately using these equations

$m_t^1 \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$v_t^1 \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

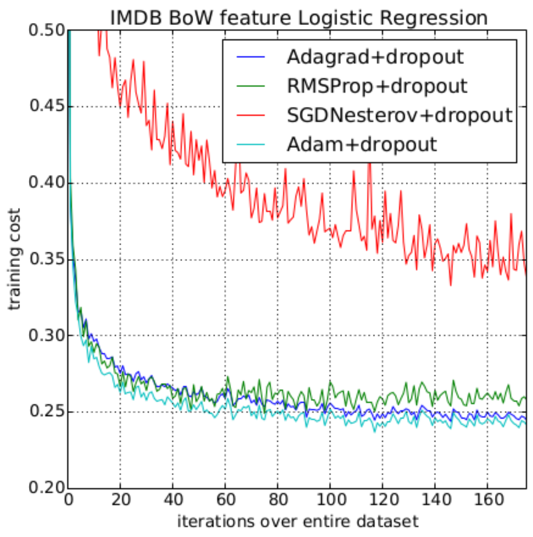
Update step:
√
θ t ← θ t−1 − α · m
b t /(v b t +) (Update parameters)

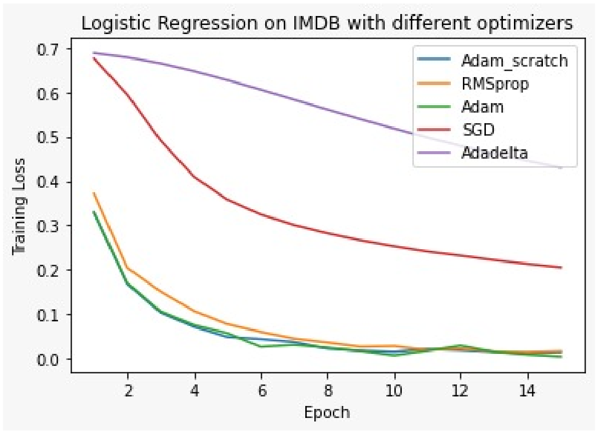
In this study, we try to prove the claims made by the 2015 paper which introduced Adam. In addition to the general results, we also went a bit further and compared various optimizers and their performances on 3D polynomial terrains.

Datasets and Models

We tried to prove the claims made in experiments showing how Adam performed better in a few use cases we recreated them using Adam_scratch (Which is Adam implemented from scratch).

- Logistic Regression over IMDB dataset (Section 6.1 in the paper): Implemented Logistic regression to classify IMDB dataset and these are the results were as follows

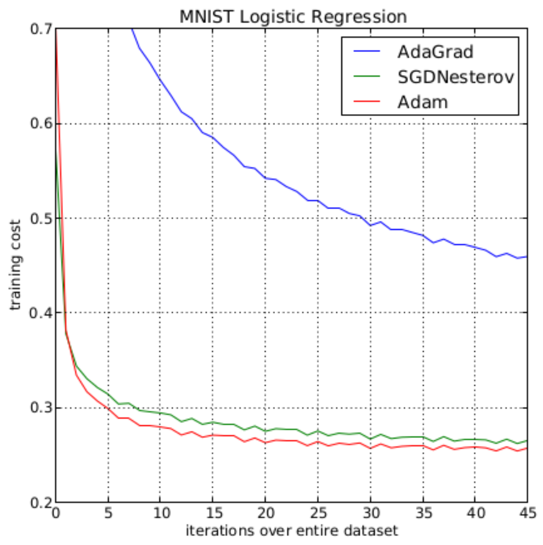


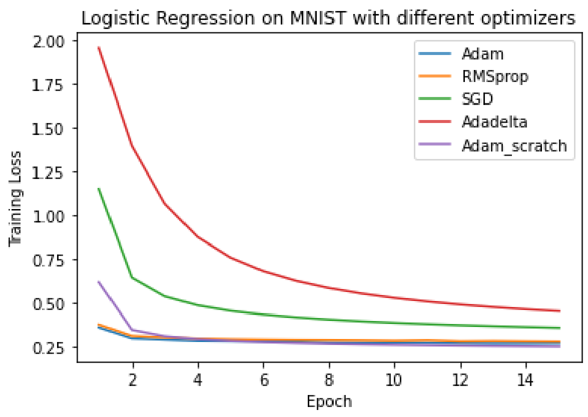


Our result

Expected Result according to the paper

- Logistic Regression over MNIST Dataset(Section 6.1 in the paper): Implemented logistic regression to classify MNIST dataset and these were the results.

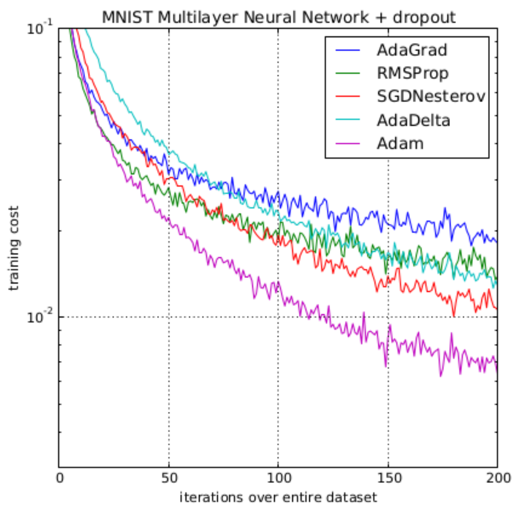




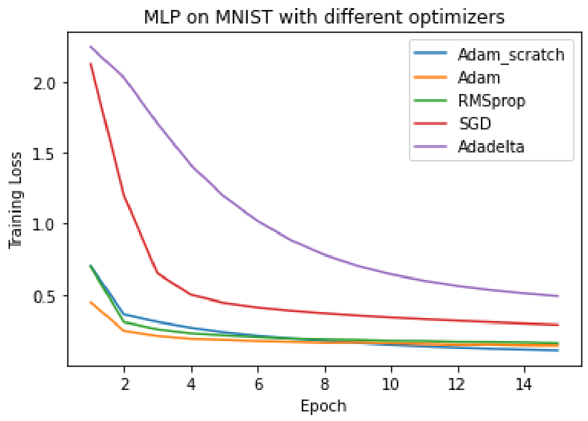
Our Result

Expected Result according to the paper.

- Multi-layer neural network with ReLU activation function (section 6.2 in the paper): Implemented a MLP with layers structure ***Enter the structure here***

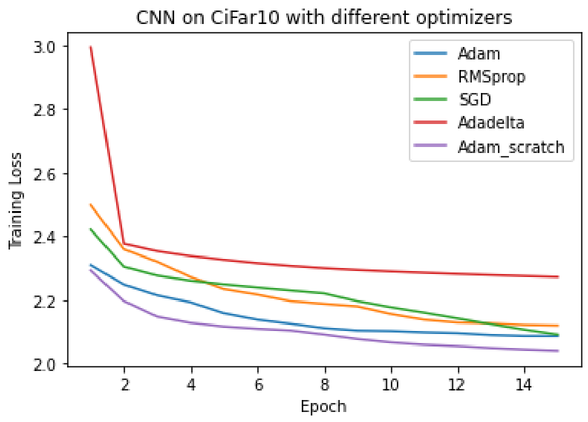


Expected Result according to the paper

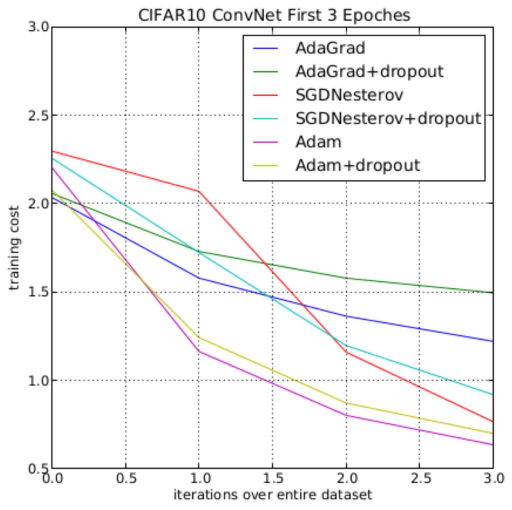


Our result

- CNN (Section 6.3 from the paper): Implemented a Convolutional neural network with three alternating stages of 5x5 convolution filters and 3x3 max pooling with a stride of 2 that are followed by a fully connected layer of 256 rectified linear hidden units (ReLU's). (Used 256 instead of 1000 for Computational reasons.) And fed CIFAR-10 to this model.

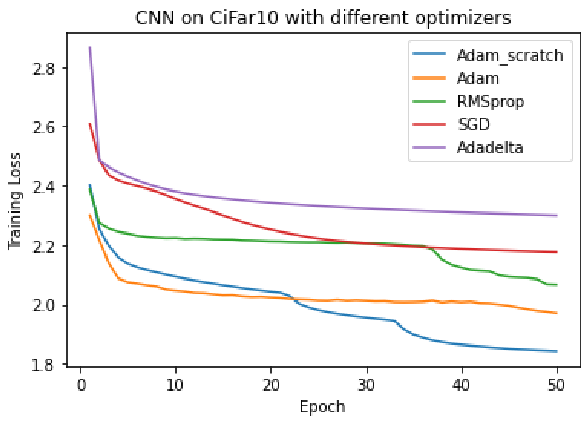


Actual result.

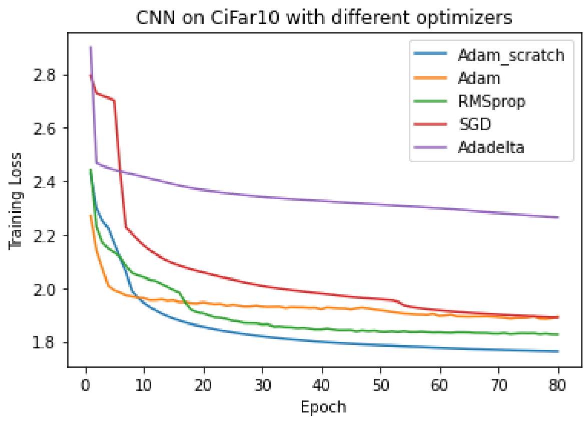


Expected result

Some additional results with longer training times are as follows:



Same CNN network trained on CIFAR 10 for 50 epochs (showing that our ADAM which had a bad start quickly started converging) (Seed = 8)



Same CNN network trained on CIFAR 10 for 80 epochs (showing that our ADAM which had a bad start quickly started converging) (Seed = 17)

Experimental conditions and constants

1. For repeatability, we used the same seed for all the optimizers while initializing the weights in PyTorch

2. Our implementation of Adam differed a bit from PyTorch's in-built implementation in almost every experiment. Although the implementations were similar (both based on the same paper), PyTorch's implementation had some additional features such as:
    a. L2 Regularization via weight decay option (which was never used so it shouldn't be a problem)
    b. AMSGRAD (offering guaranteed convergence)

    We did not implement these as they were not proposed in the original paper
3. We plotted PyTorch's in-built Adam too in all of the above graphs to compare our implementation with it
4. Due to limited computing capabilities and time, we discarded our planned experiments on heavy and deep networks to test the gradient flow in various optimizers (and the problems of exploding and vanishing gradients).

Visualizer

We built a visualizer in PyTorch which can be used to visualize the optimizer performances on various 3D terrains. Both PyTorch in-built and custom-built optimizers can be used. The main purpose of building this is to let us and the community have a deeper understanding of optimizers by allowing an easy way to create terrains using polynomials and standard gaussian functions.

Link to the results -
https://drive.google.com/drive/folders/1LuOS7AOZ5t_e7Uj8uGkgewJEz5NSwl6U?usp=sharing

Code can be found in the repository, the link to which is already shared in the first section.

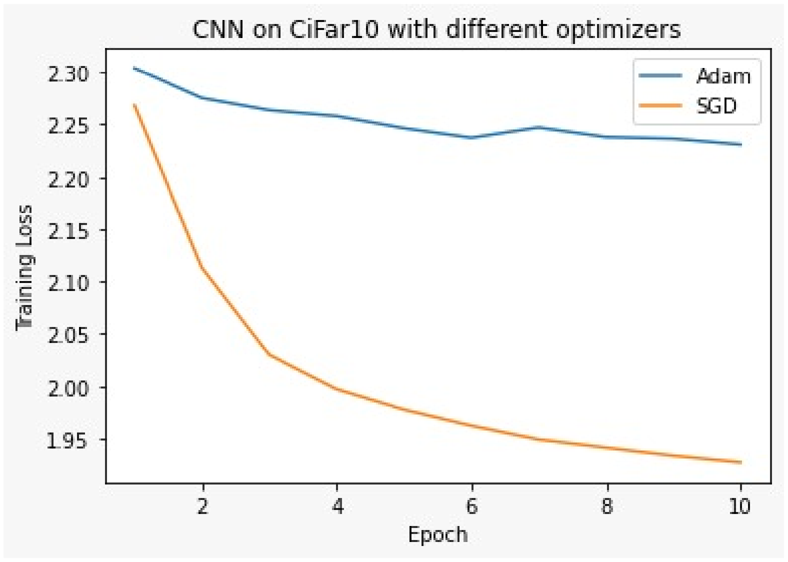
Observations and Analysis

1. Adam did perform better compared to other optimizers on these models and datasets as said in the paper which can be explained by the idea adam had and choosing the best things of other methods it was supposed to be better than both of them.

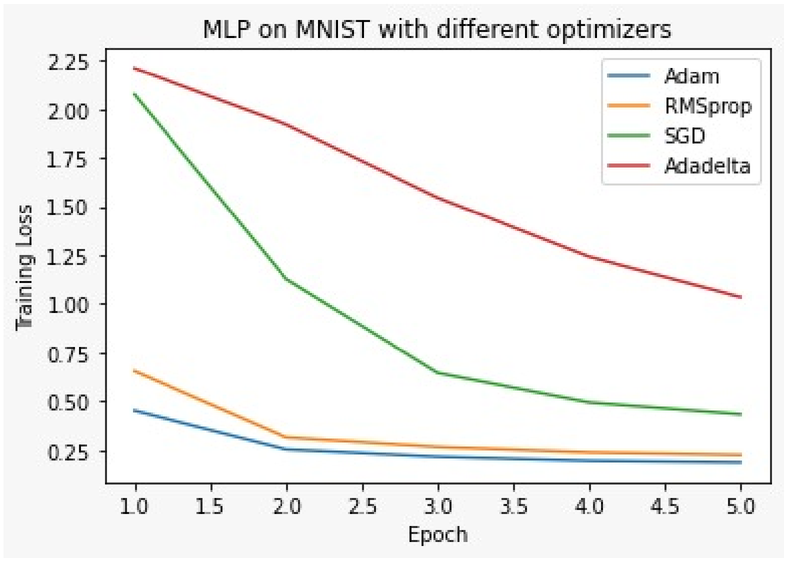
    💡 Adam had two constants $\beta_1, \beta_2 \in [0, 1)$: and also ε (Exponential decay rates for the moment estimates) We had to set these values, The chosen values are β_1 = 0.9, β_2 = 0.99, log(ε) = -8 because the authors proposed these values and these were the default values for other library implementations of Adam.

2. Our implementation almost always differed from PyTorch's in each of the above experiments. We suspect this to be due to floating points and rounding-offs, as PyTorch's implementation stores these gradients in lists and passes them to another function before using them to update the weights while our's didn't. This difference was not really noticeable in most cases.

3. **Sometimes, SGD converged much quickly than Adam**. Since the number of epochs was low, this conclusion shouldn't be taken. However, we could say for sure that SGD had better first steps and was faster than Adam in the initial phases of training. **This will happen when the initial terrain has very large gradients** (in such case, greedy SGD moves faster, while Adam takes time to pick up speed (controlled by the momentum factor in the numerator)).



Adam outperforms SGD in cases where there are large regions of near plain terrain in the optimization problems. In most other cases (ignoring deep networks as we didn't test them), SGD was very competitive with ADAM (SGD was reliably the second-best optimizer among the ones chosen by us in these experiments).



4. Observations from the visualizer results:

    a. In most cases, SGD and Adam were quite competitive. Both looked promising and produced similar results and rates of convergence.

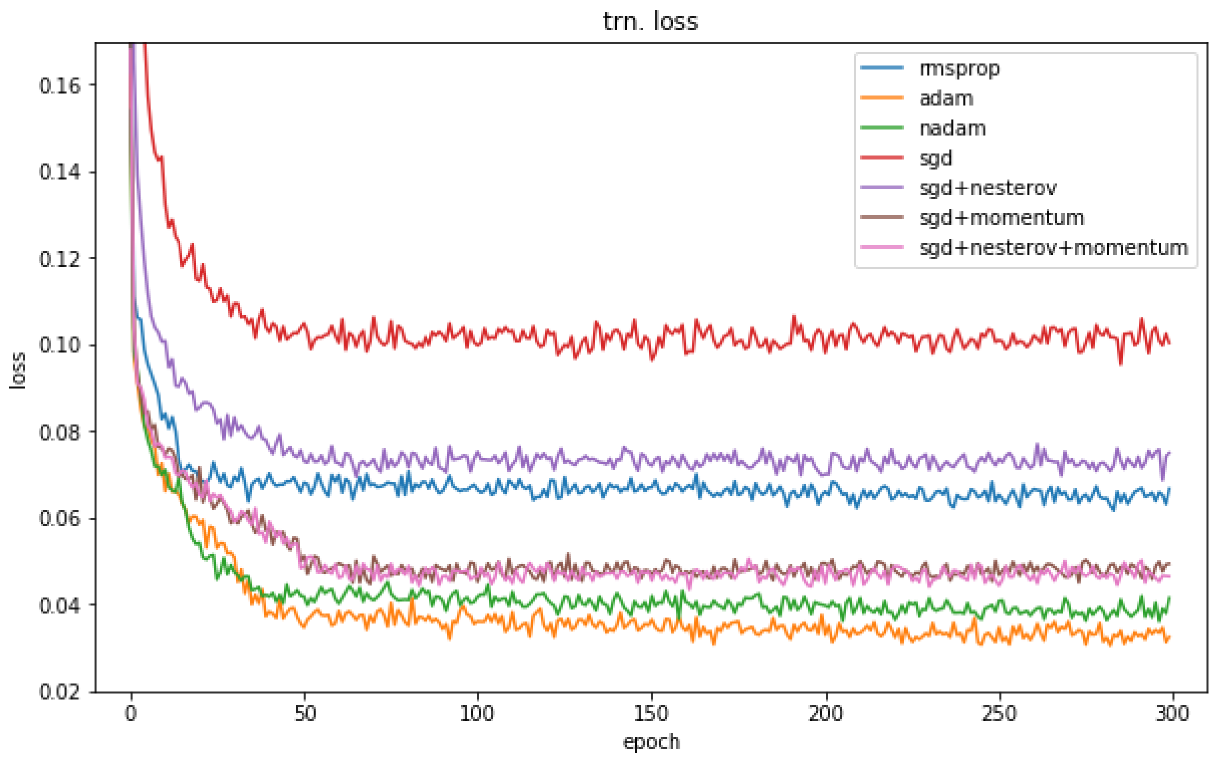
    b. In some cases where the point of initialization lies on the near-vertical terrains, SGD moves very fast taking large strides while Adam and RMSprop, speed up slowly due to the momentum factor. This is generally advantageous for Adam because if a minimum lies at the edge of this region, SGD will end up oscillating about that point (due to large step sizes) and will take too long to converge. While Adam and RMSprop can slow down (as they are helped by the momentum factor) and reach the minimum quickly with fewer oscillations (adaptable step-sizes).

    c. **One problem that still persisted is with the saddle/critical points**. When initialized at these points, all the optimizers get stuck and will not be able to move. This can be tackled by running the model with random initializations of multiple points. This kind of random initializations can also help deal with problems posed by local minima.

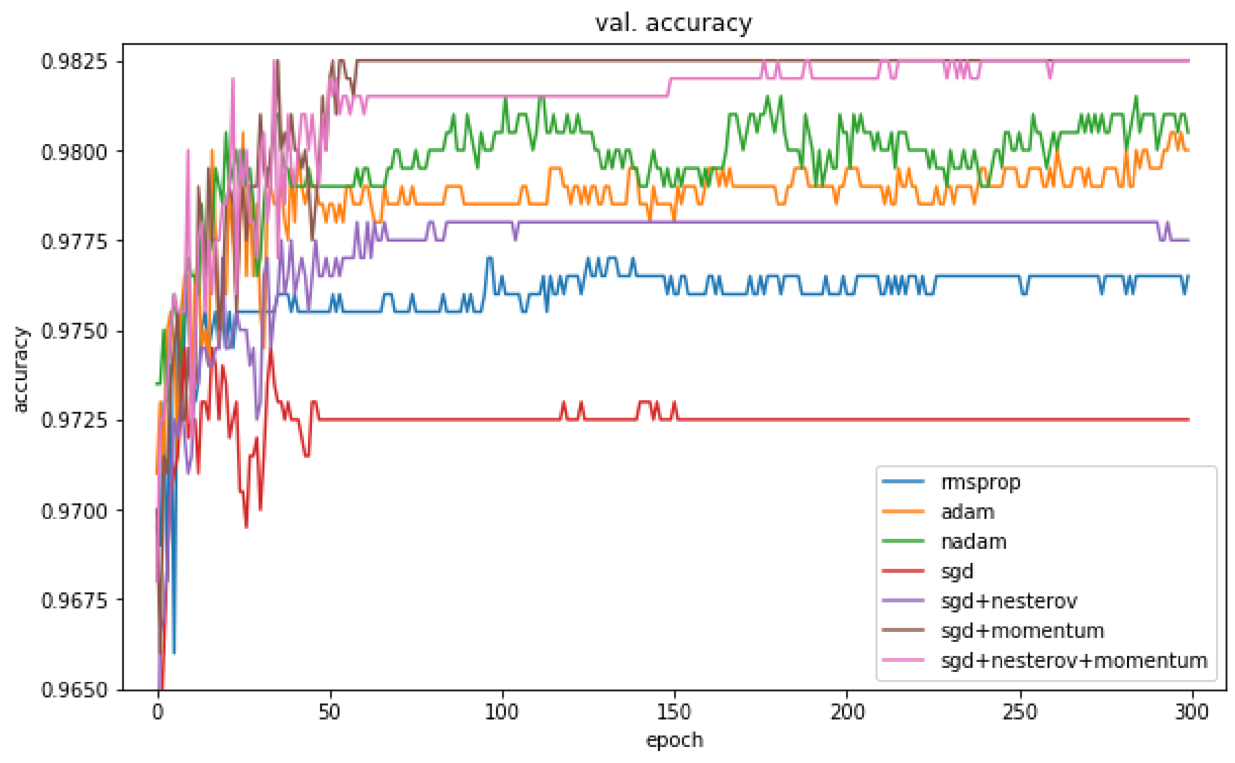
    d. **However, one striking feature of Adam** (and related optimizers like RMSprop, ...etc.) is that **they are able to go past the saddle points** when initialized at non-critical points, unlike other optimizers. This is mainly due to the momentum factor in the numerator, which refuses to zero down at saddle points and helps keep the optimizer moving.

Conclusion

There are many optimizers to choose from few are adaptive like (Adam, Adadelta, RMSprop) and others are not like SGD. The paper claimed and compared training loss and showed how adam converges quicker however this doesn't mean the model and updated weights are the best because when compared with a validation error, and accuracy there are some differences between the models and in most cases even when SGD takes a lot of time and energy it performs better eventually



Taken from https://shaoanlu.wordpress.com/2017/05/29/sgd-all-which-one-is-the-best-optimizer-dogs-vs-cats-toy-experiment/



Taken from https://shaoanlu.wordpress.com/2017/05/29/sgd-all-which-one-is-the-best-optimizer-dogs-vs-cats-toy-experiment/

This doesn't mean that we have to use SGD all the time because it isn't that worthy either based on the situation we have to choose an optimizer if the data is sparse we waste so much time if we use nonadaptive methods like SGD, in those cases, we can use adaptive methods like Adam or Adadelta or RMSprop.

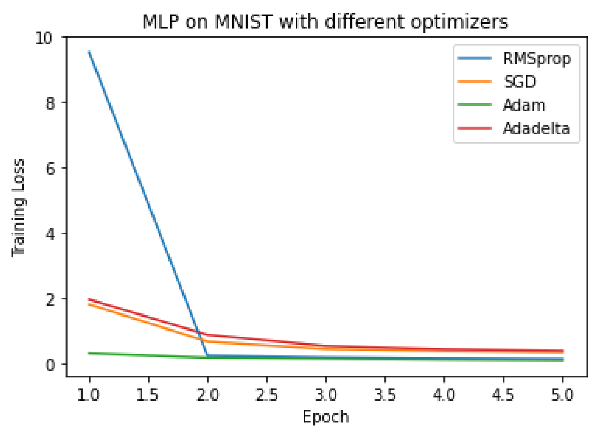
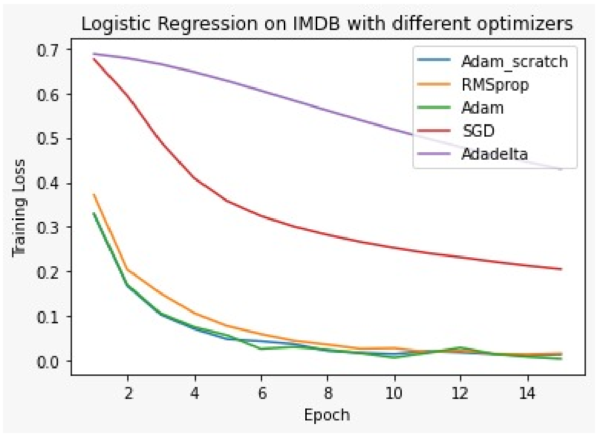


Image with original size (28*28). 3-layered network (1000, 1000, 10) with linear output (hidden layers use ReLu)



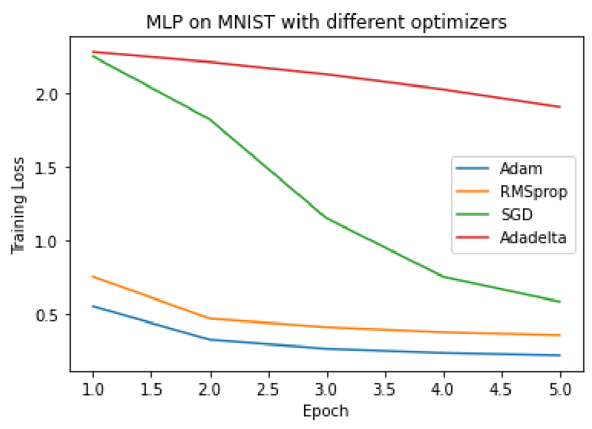


Image resized to 10*10, a 3-layered neural network (32, 16, 10) with linear output (hidden layers use ReLU)

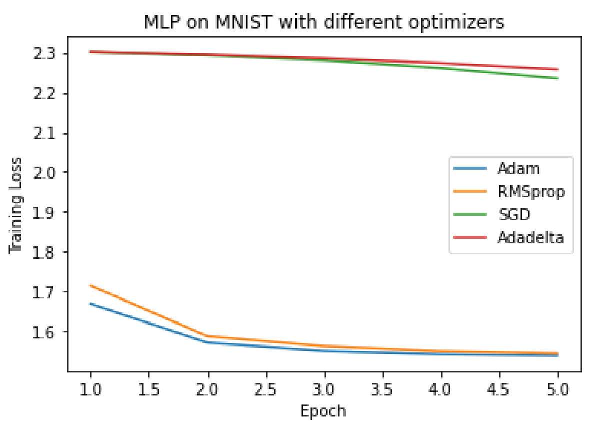
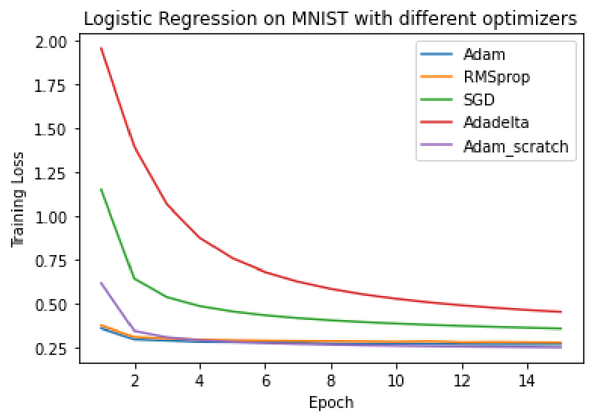


Image resized to 10*10. 3-layered neural network (32, 16, 10) with sigmoid output (hidden layers use ReLU)



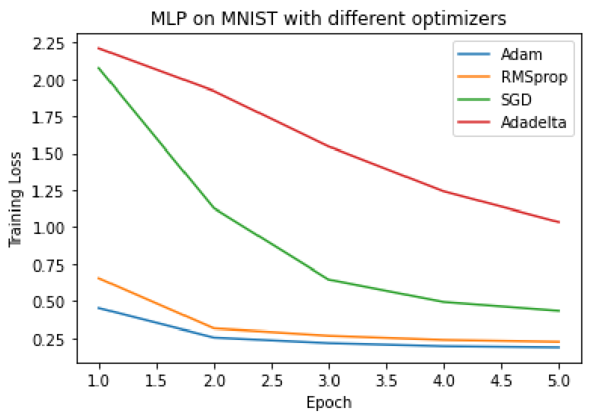


Image resized to 14*14, a 3-layered neural network (64, 32, 10) with linear output (no activation function used in the end) (hidden layers use ReLU)

Work Distribution

- MLP for MNIST: Naman and Pranshul
- Implementing Adam: Vishal
- CNN for CIFAR-10: Ruthwik and Vishal
- Logistic regression on MNIST (using softmax): Ruthwik
- Logistic Regression on IMDB Bow: Pranshul and Naman

- Visualizer: Vishal
- Report: Ruthwik, Vishal, Pranshul

References

1. https://arxiv.org/abs/1412.6980
2. https://github.com/Jaewan-Yun/optimizer-visualization,
3. https://github.com/3springs/viz_torch_optim
4. https://en.wikipedia.org/wiki/Rosenbrock_function