Learning how to learn

Descending down steep curves

Amish Mittal

Acknowledgements

I would like to thank my supervisor and mentor Prof Jimson for always supporting me in my work and helping me achieve my career goals.

Gradient Descent

ANN: $Y = h(\theta, X)$

Goal: Approximate correct value of Y by fixing h and then estimating θ using loss function f

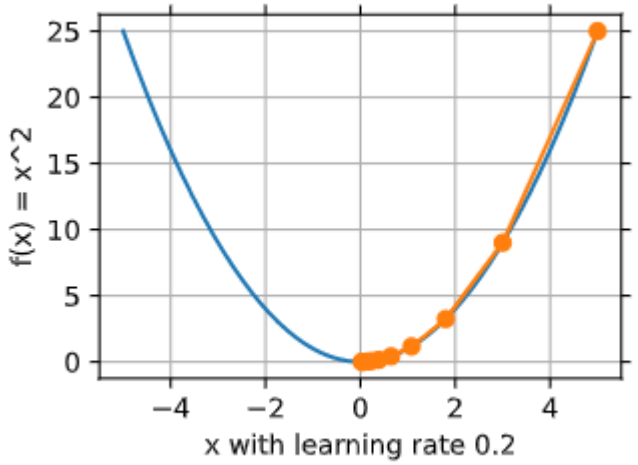
Ideal: $f(\theta) = \mathbb{E}_{(x,y)\sim p_{\text{data}}} L(h(\theta, X), Y)$

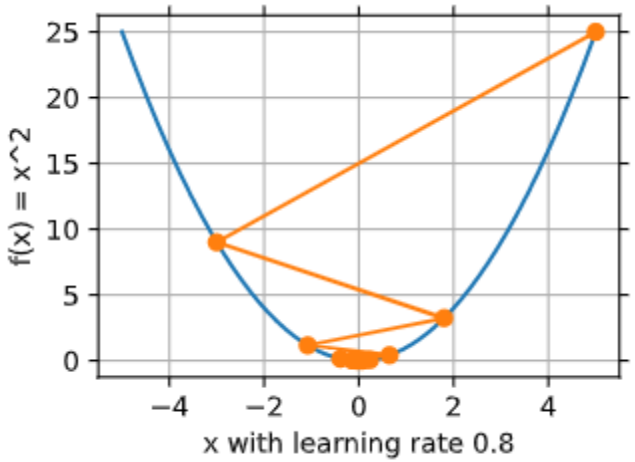
$$f^*(\theta) = \mathbb{E}_{(x,y)\sim \tilde{p}_{\text{data}}} L(h(\theta, X), Y) = \frac{1}{m} \sum_i L(h(\theta, x^i), y^i)$$

Gradient descent: $\theta_t = \theta_{t-1} - \eta \, \nabla_\theta \, f^*(\theta)$

Vanilla Stochastic GD: $\Delta\theta = -\eta\nabla f$

$f = x^2$





Issue:

1. Highly susceptible to chosen learning rate. We don't know appropriate LR beforehand.
2. The pattern of descent changes. Can we make it more deterministic?

Vanilla Stochastic GD: $\Delta\theta = -\eta\nabla f$

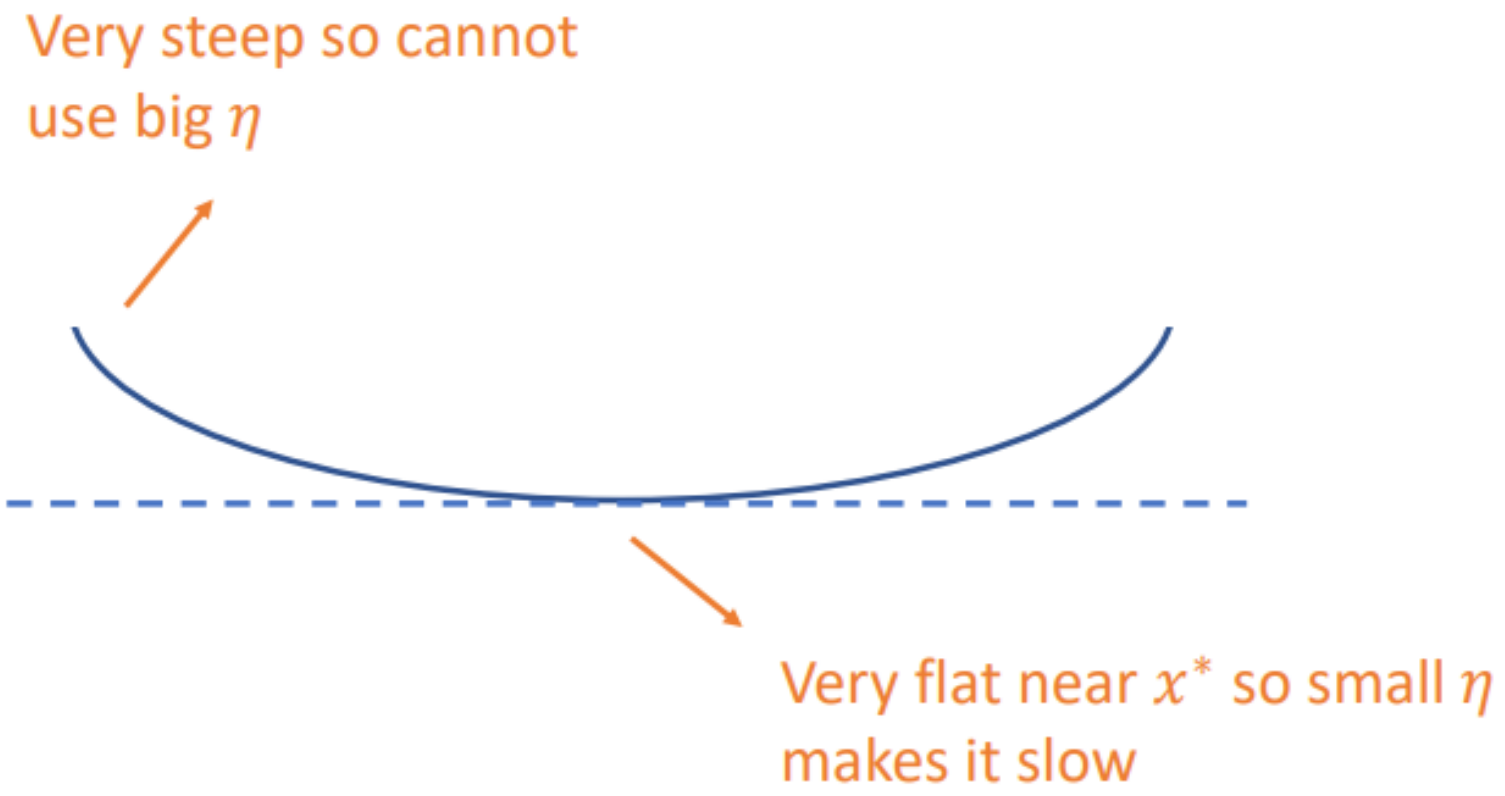
$f = 10x^2$

```
show_trace(gd(0.8, f_grad, 10), f, "10x^2", 0.8)
⊗ 0.4s
epoch 10, x: 2883251953125.000000
```

```
show_trace(gd(0.12, f_grad, 10), f, "10x^2", 0.12)
✓ 5.5s
epoch 10, x: 144.627327
```

Issue:
Gradient is way too high at our initialization point (89.4^o). The optimization hence diverges even with low learning rate.

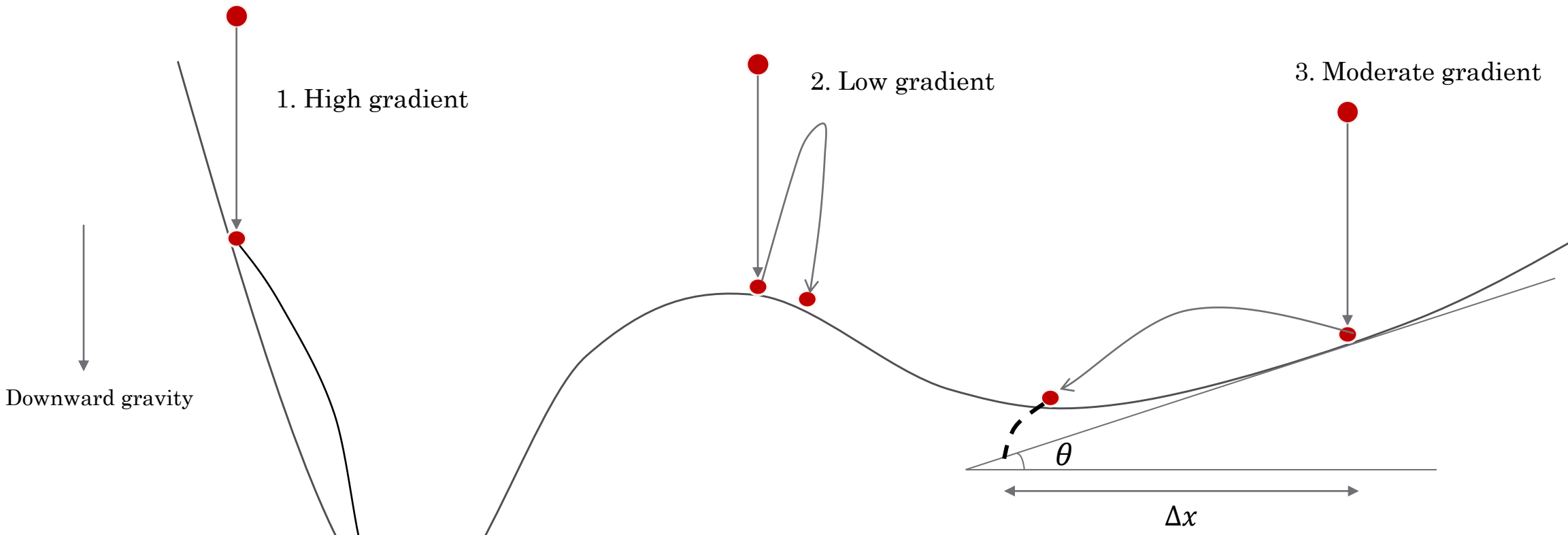
Gradient Descent Convergence



Source: https://praneethnetrapalli.org/

Taking inspiration from physics

Projectile motion on inclined plane



Let θ be the gradient at the point at which our ball collides

Then, $\Delta x \approx \sin(\theta)\cos(\theta)$

$\Delta x \rightarrow$ movement along parameter space assuming complete projectile motion without collision from loss function

With known $\tan(\theta) = \nabla f$,

$$\therefore\ \Delta x \approx -\sin(\theta)\cos(\theta) \approx -\frac{\nabla f}{1 + |\nabla f|^2}$$

Expanding this to $\mathbb{R}^n$ as $\theta \in \mathbb{R}^n$,

$$\Delta\theta_i = -\frac{1}{1 + \nabla f_i^2}.\nabla f_i$$

Vectorizing to improve time complexity,

$$\Delta\theta = -\frac{1}{1 + F} \odot \nabla f$$

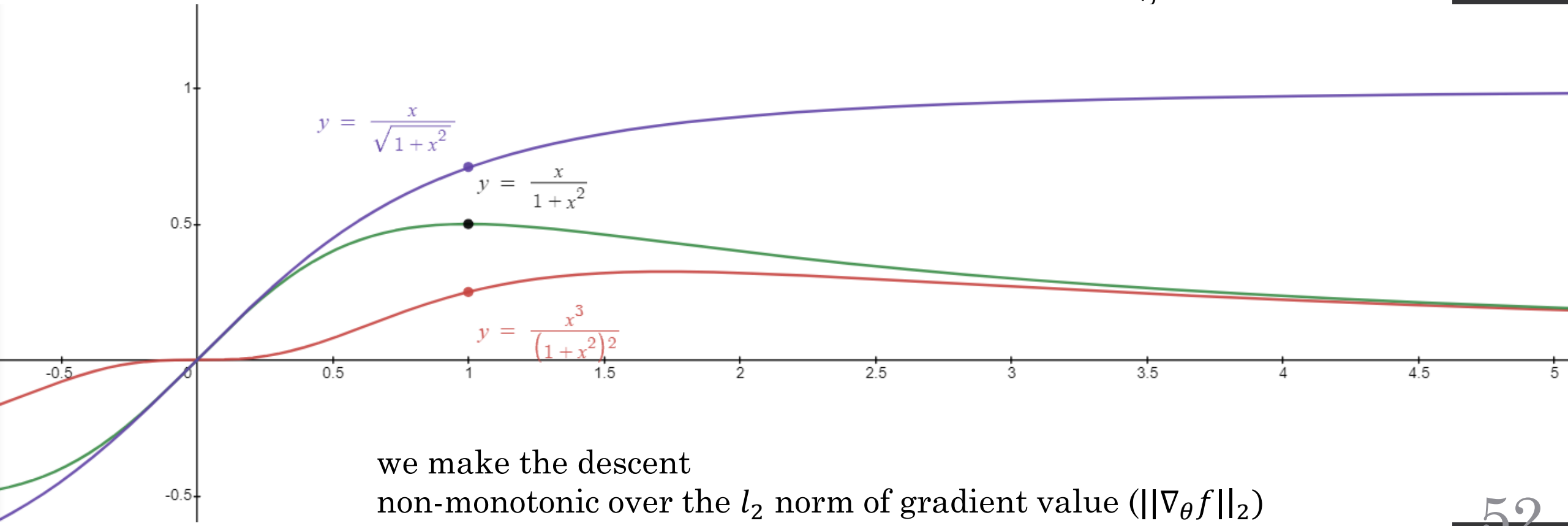
where,

$$F = \begin{bmatrix} \nabla f_0^2 & 0 & 0 & 0 \\ 0 & \nabla f_1^2 & 0 & 0 \\ 0 & 0 & \nabla f_2^2 & 0 \\ 0 & 0 & 0 & \nabla f_3^2 \end{bmatrix}$$

Nature of $\Delta\theta = \frac{\eta}{1+F} \odot \nabla f$

$$\lim_{\nabla f \to 0} \Delta\theta = 0$$

$$\lim_{\nabla f \to \infty} \Delta\theta = 0$$



we make the descent
non-monotonic over the l_2 norm of gradient value ($||\nabla_\theta f||_2$)

Optimizer Expression

Proposing a new update rules, of the family,

$$\Delta\theta_i = -\frac{|\nabla f_i|^h}{\left|1 + \nabla f_i^2\right|^h} . \nabla f_i$$

Vectorized implementation: (to reduce time complexity)

$$\Delta\theta = -\frac{F^h}{\left|1 + F^2\right|^h} . \nabla f$$

Constraints of this exercise

- Have a GD optimizer which:
  1. Does not perform worse
  2. Diverges in minimum number of cases
  3. Convergence can be proved
  4. Has similar asymptotic time complexity
- Best case:
  - Gives a better convergence value
  - Takes less number of iterations
  - Converges towards global minima instead of local minima
  - Regret bound is $O(\sqrt{T})$

Analysis

- Theoretical
  - Convergence Validity Theorem
  - Convergence Rate Theorem
- Empirical
  - 2D non-convex functions
    - Does it converge?
    - Score over number of epochs required to converge
    - Score over converging at global minimum
  - Neural Networks
    - Regression
      - Airfoil normalized and unnormalized
    - Classification
      - Fashion-MNIST
      - CIFAR-10
    - Word Embeddings (CBoW)
      - The Penn Treebank

Theoretical Analysis

4.1 Convergence Validity Theorem

Theorem 4.1.1. *A function* $f : \mathbb{R}^n -> \mathbb{R}$ *is convex and differentiable, and that its gradient is Lipschitz continuous with constant* $L > 0$, *i.e. we have* $||\nabla f_x - \nabla f_y|| \leq L||x - y||_2$ *for any* x, y. *Then, if we run gradient descent with update rule as* $\Delta\theta_i = \frac{|\nabla f_i|^h}{|1+\nabla f_i^2|^h}.\nabla f_i$, *it will always converge provided* $h > log_2 L - 1$.

$$t = \frac{|\nabla f_x|^h}{|1+\nabla f_x^2|^h} . \nabla f_x,$$

$$f(x^+) \leq f(x) - (1 - \frac{1}{2} L_i t)\, t ||\nabla f(x)||_2^2$$

$$L_i < \frac{2}{t}$$

$t(\nabla f_x)$ would be maximum at points where $\frac{d(t(\nabla f_x))}{d\nabla f_x} = 0$ and $\frac{d^2 t(\nabla f_x)}{d\nabla f_x^2} < 0$.

$\frac{2}{t}$'s minimum value would be 2^{h+1}.

$$L_i < 2^{h+1}$$

We now expand the result to n - dimensions considering $f : \mathbb{R}^n -> \mathbb{R}$ and $x \in \mathbb{R}^n$ with Lipschitz constant as L.

$$L \geq \max_i L_i$$

Theoretical Analysis

4.2 Convergence Rate Theorem

Theorem 4.2.1. *A function* $f : \mathbb{R}^n -> \mathbb{R}$ *is convex and differentiable, and that its gradient is Lipschitz continuous with constant* $L > 0$. *Then, if we run gradient descent for* k *iterations with update rule as* $\Delta\theta_i = -\max\left(\epsilon, \frac{|\nabla f_i|^h}{|1+\nabla f_i^2|^h}\right).\nabla f_i$, *it will lead to a solution* $f^{(k)}$ *satisfying*

$$f\left(x^{(k)}\right) - f\left(x^*\right) \leq \frac{\left\|x^{(0)} - x^*\right\|_2^2}{2\epsilon k}$$

provided that $h > log_2 L$.

$$f\left(x^{+}\right)-f\left(x^{*}\right) \leq \frac{1}{2t}\left(\left\|x-x^{*}\right\|_{2}^{2}-\left\|x^{+}-x^{*}\right\|_{2}^{2}\right) \leq \frac{1}{2t_{\min}}\left(\left\|x-x^{*}\right\|_{2}^{2}-\left\|x^{+}-x^{*}\right\|_{2}^{2}\right)$$

This inequality holds for x^{+} on every epoch of gradient descent. Summing over multiple epochs, we can deduce:

$$\sum_{i=1}^{k} f(x^{(i)})-f\left(x^{*}\right) \leq \sum_{i=1}^{k} \frac{1}{2\epsilon}\left(\left\|x^{(i-1)}-x^{*}\right\|_{2}^{2}-\left\|x^{(i)}-x^{*}\right\|_{2}^{2}\right)$$

Using the fact that f is decreasing on every iteration, we can conclude that,

$$\begin{aligned} f\left(x^{(k)}\right)-f\left(x^{*}\right) &\leq \frac{1}{k} \sum_{i=1}^{k} f\left(x^{(i)}\right)-f\left(x^{*}\right) \\ &\leq \frac{\left\|x^{(0)}-x^{*}\right\|_{2}^{2}}{2\epsilon k} \end{aligned}$$

Empirical Validation Parameters

1. Does it converge?

   True if there exists some c such that $|\theta_{t+1} - \theta_t| \leq k$ for all $t > c$,

   $k \rightarrow error\ parameter, \qquad c \rightarrow epochs\ required\ to\ converge$

Empirical Validation Parameters

2. Score over number of epochs required to converge

$$s_1 = \begin{cases} \frac{1}{c} & \text{if } \exists\ c : |x_{t+1} - x_t| \leq k\ \ \forall\ \ t > c\ \ \text{(descent converges)} \\ 0 & \text{otherwise (descent does not converge)} \end{cases}$$

Empirical Validation Parameters

3. Score over converging at global minimum

$$s_2 = \begin{cases} 10 & \text{if } |x_c - x^*| \leq k' \\ 0.1 & \text{otherwise} \end{cases}$$

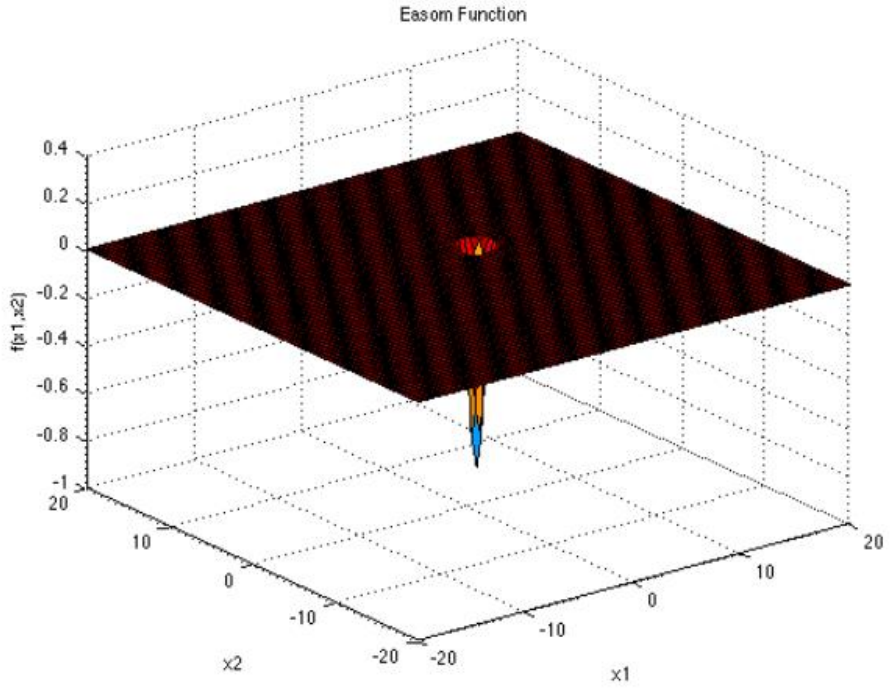
Empirical Validation Parameters

Overall Score: (Average over 300 runs for $s_1 * s_2$)

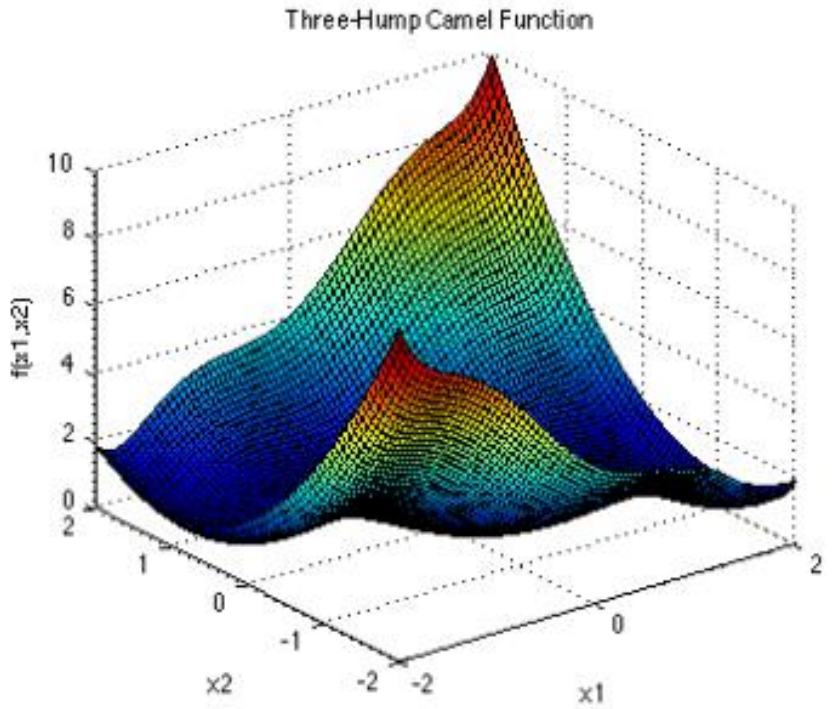
$$\text{Score} = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} s_1(f_i, x_{i,j}) s_2(f_i, x_{i,j})}{n + m}$$

10 random initializations for 30 functions each = 300

2D Test Functions

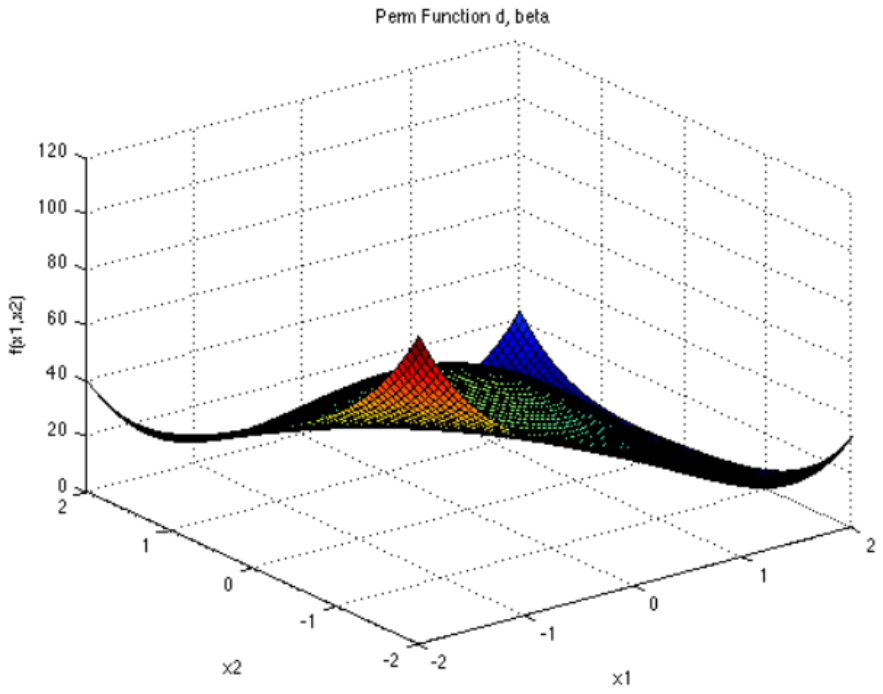


$$f(\mathbf{x}) = -\cos(x_1)\cos(x_2)\exp\left(-(x_1-\pi)^2-(x_2-\pi)^2\right)$$

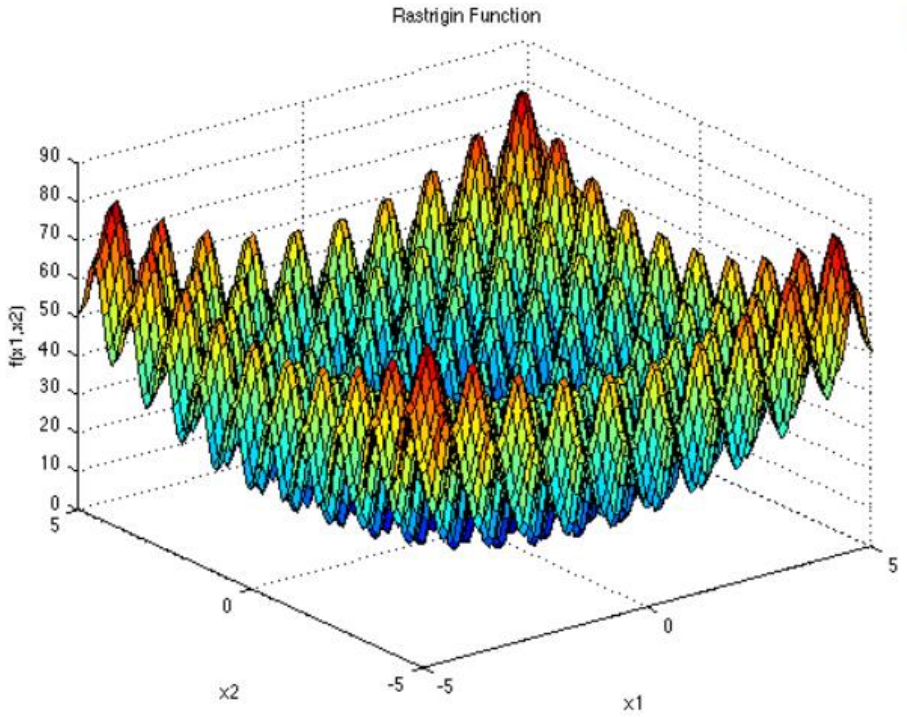


$$f(\mathbf{x}) = 2x_1^2 - 1.05x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2$$

30 such odd non-convex functions taken from https://www.sfu.ca/~ssurjano/optimization.html



$$f(\mathbf{x}) = \sum_{i=1}^{d} \left(\sum_{j=1}^{d} (j^i + \beta) \left(\left(\frac{x_j}{j} \right)^i - 1 \right) \right)^2$$



$$f(\mathbf{x}) = 10d + \sum_{i=1}^{d} \left[x_i^2 - 10\cos(2\pi x_i) \right]$$

2D Test Functions

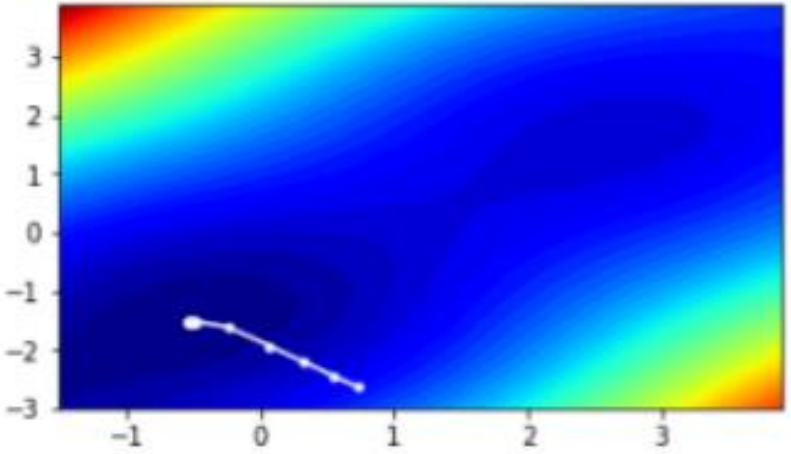
30 such odd non-convex functions taken from https://www.sfu.ca/~ssurjano/optimization.html

Scores over 2D Test Functions

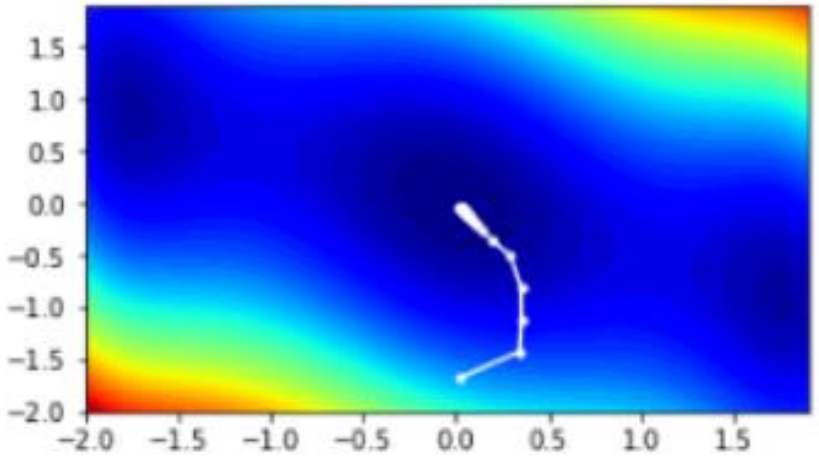
Optimizer	Average Score	Number of divergences
SGD	0.314	28
SGD with Momentum	0.322	41
Adagrad	0.282	14
RMSProp	0.520	8
Adam	0.885	4
My optimizer	0.825	**0**
My optimizer + Momentum	0.641	8
My optimizer + Momentum + EMA	**0.944**	**0**

Table 5.1 Score of various optimizers after hyperparameter tuning over 2D non-convex test suite. The average is taken over $30 * 10 = 300$ runs.

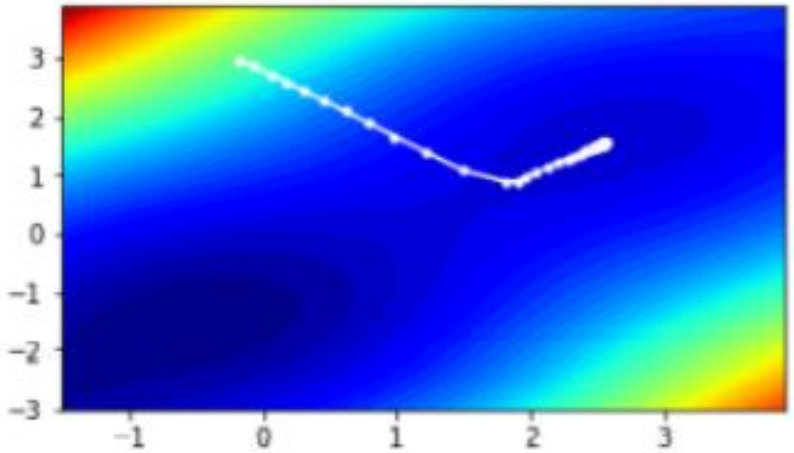
```
Using function: MCCORMICK with seed: 2
Converged:  True
Converged at Global Minima:  True
Number of epochs required to converge:  7
Score:  1.4285714285714286
```



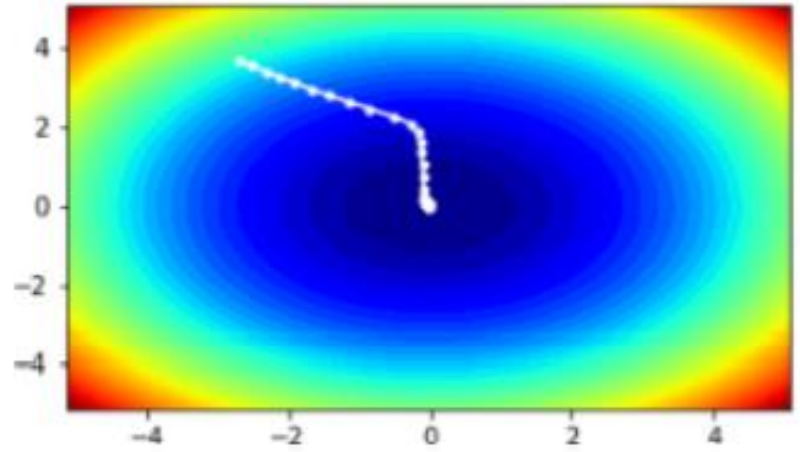
```
Using function: THREEHUMP_CAMEL with seed: 2
Converged:  True
Converged at Global Minima:  True
Number of epochs required to converge:  7
Score:  1.4285714285714286
```



```
Using function: MCCORMICK with seed: 5
Converged:  True
Converged at Global Minima:  False
Number of epochs required to converge:  13
Score:  0.015384615384615385
```



```
Using function: SPHERE with seed: 5
Converged:  True
Converged at Global Minima:  True
Number of epochs required to converge:  19
Score:  0.5263157894736842
```



Variations of optimizer

Vanilla + Momentum

We add a first order momentum term to accelerate our descent and to add a fraction of the previous update to the current update vector.

$$\Delta\theta_{i,t} = -\max\left(\epsilon, \frac{|\nabla f_i|^h}{|1+\nabla f_i^2|^h}\right).\nabla f_i - \gamma\ \Delta\theta_{i,t-1}$$

Variations of optimizer

Vanilla + EMA

We use the Exponential Moving Average (EMA) of the previous gradients to account for those values in a similar fashion as of RMSProp

$$E[\nabla f^2]_t = 0.9E[\nabla f^2]_{t-1} + 0.1\nabla f_t^2$$

$$\Delta\theta_i = -\max\left(\epsilon, \frac{|\nabla f_i|^h}{|1+\nabla f_i^2|^h}\right).\frac{\eta}{\sqrt[2]{E[\nabla f^2]_t + \epsilon}}.\nabla f_i$$

Variations of optimizer

Vanilla + Momentum + EMA

We add a momentum term to the previous update rule with EMA.

$$E[\nabla f^2]_t = 0.9E[\nabla f^2]_{t-1} + 0.1\nabla f_t^2$$

$$\Delta\theta_{i,t} = -\max\left(\epsilon, \frac{|\nabla f_i|^h}{|1+\nabla f_i^2|^h}\right) \cdot \frac{\eta}{\sqrt[2]{E[\nabla f^2]_t + \epsilon}} . \nabla f_i - \gamma\ \Delta\theta_{i,t-1}$$

Analysis over Neural Networks

- Each dataset is used in two formats
  - Input features normalized
  - Input features unnormalized
- Each of this data combination is passed to both a shallow and deep neural network and they are trained
- Each of this data-network pair is trained using 5 different optimizers and the results are logged
- Total = 10*5 = 50 convergence results
- Word embeddings using CBoW over Penn Treebank are trained as a specialized task.

Analysis over Neural Networks

-	SGD	Adagrad	Adam	My optimizer	My opt. + Mom. + EMA
Shallow Regression over Airfoil	inf	1	1	1	1
Shallow Regression over Airfoil (Normalized)	2	1	1	1	1
Deep Regression over Airfoil	inf	2	2	2	1
Deep Regression over Airfoil (Normalized)	6	5	1	5	3
Shallow Classification over Fashion-MNIST	inf	inf	inf	3	3
Shallow Classification over Fashion-MNIST (Normalized)	6	2	2	3	3
Deep Classification over Fashion-MNIST	inf	23	inf	40	14
Deep Classification over Fashion-MNIST (Normalized)	4	3	3	5	3
Shallow Classification over CIFAR-10 (Normalized)	32	18	20	52	19
Deep Classification over CIFAR-10 (Normalized)	44	40	18	28	24
Word Embeddings using Penn Treebank (CBoW)	inf	122	90	151	111

Number of epochs to converge

Convergence Graphs

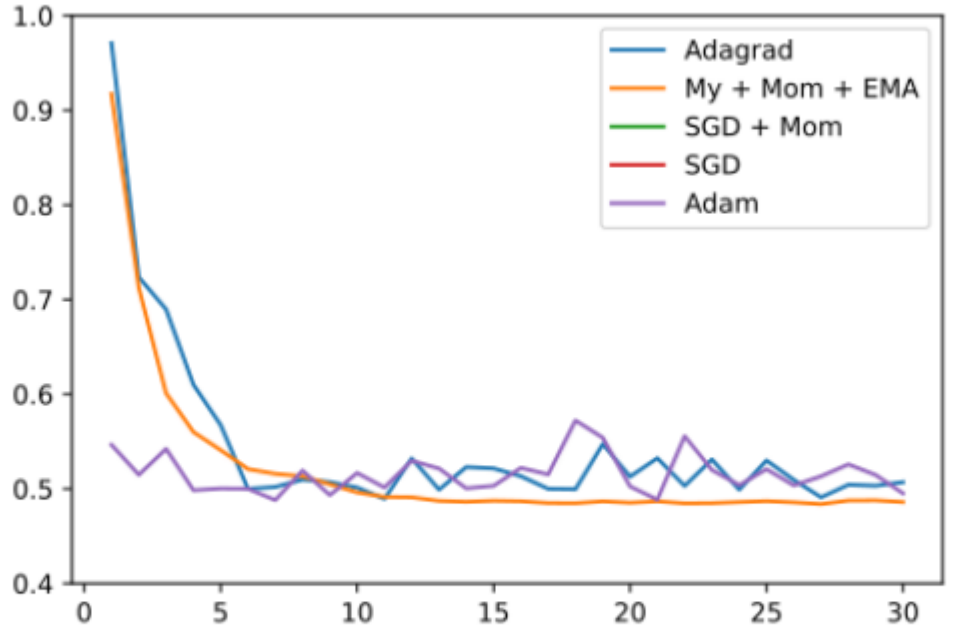


Fig. 5.6 Convergence graph for shallow regression over Airfoil (Normalized)

Convergence Graphs

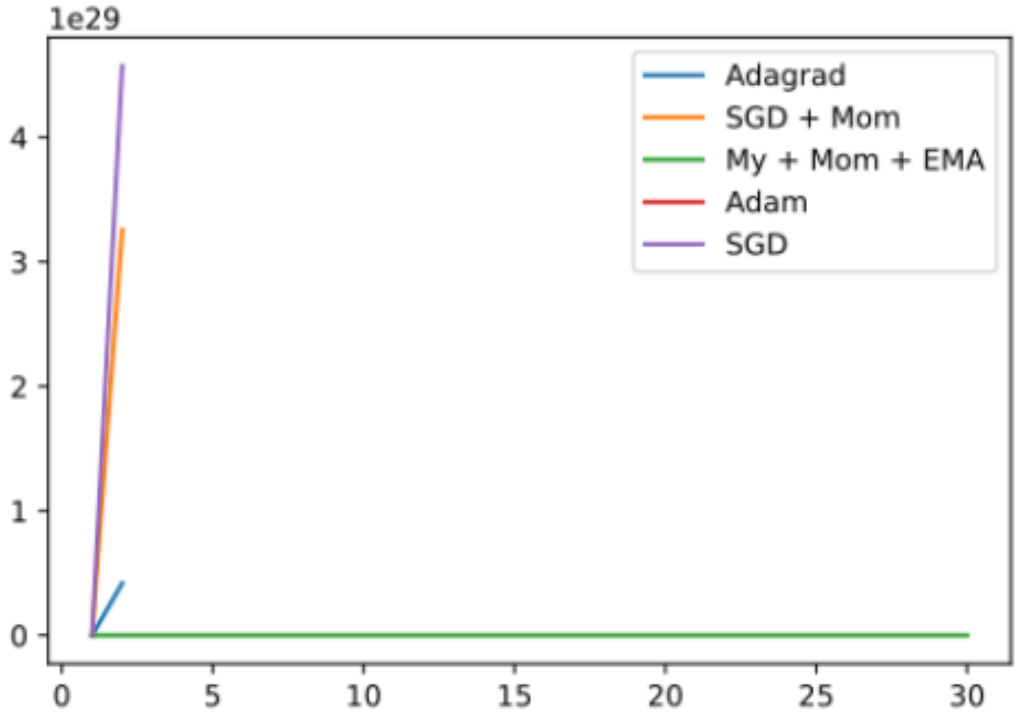
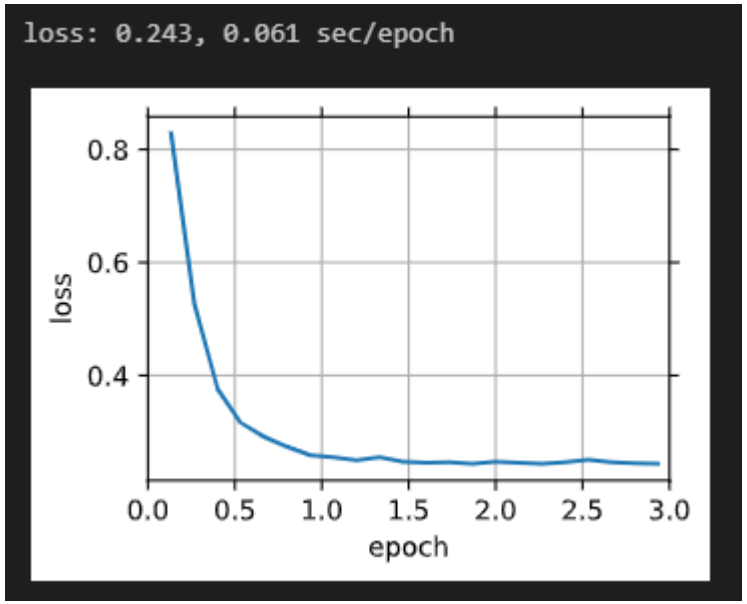


Fig. 5.7 Convergence graph for shallow regression over Airfoil (Unormalized)

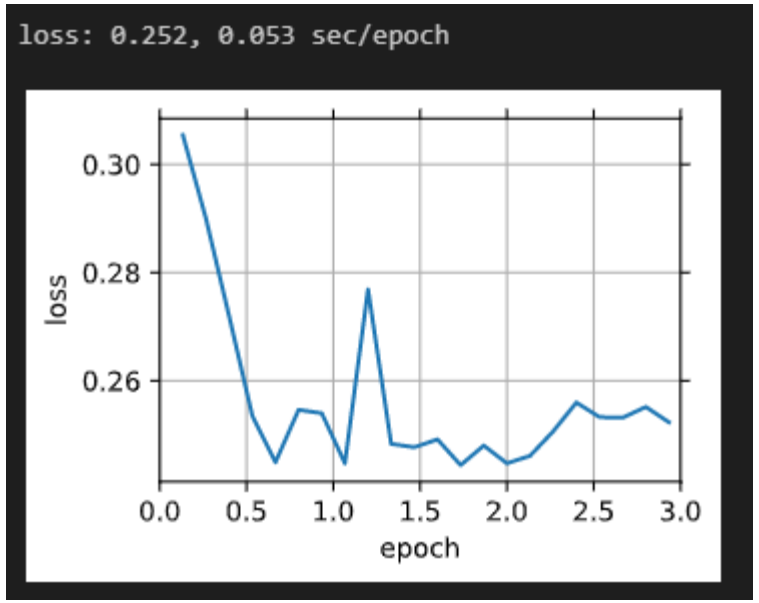
Results on Shallow NN (Airfoil dataset) – Regression with data normalization

- With h = 3 my optimizer



Uniform convergence guaranteed

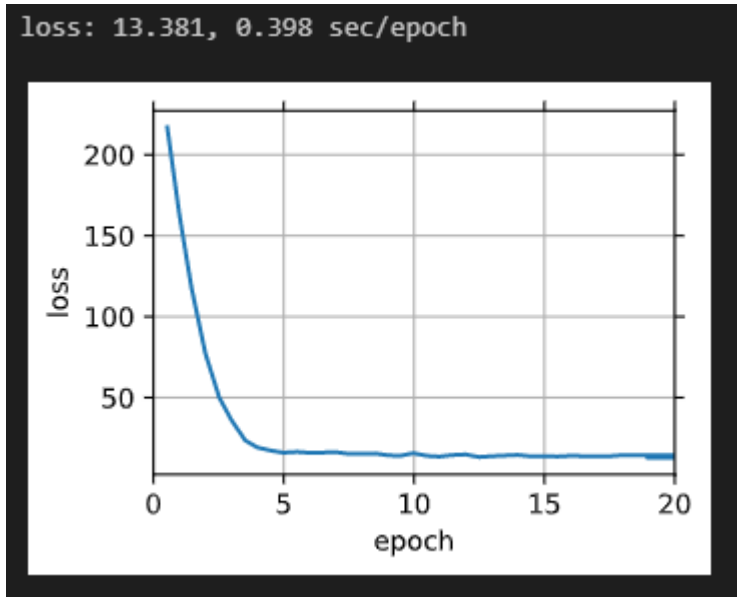
Normal SGD



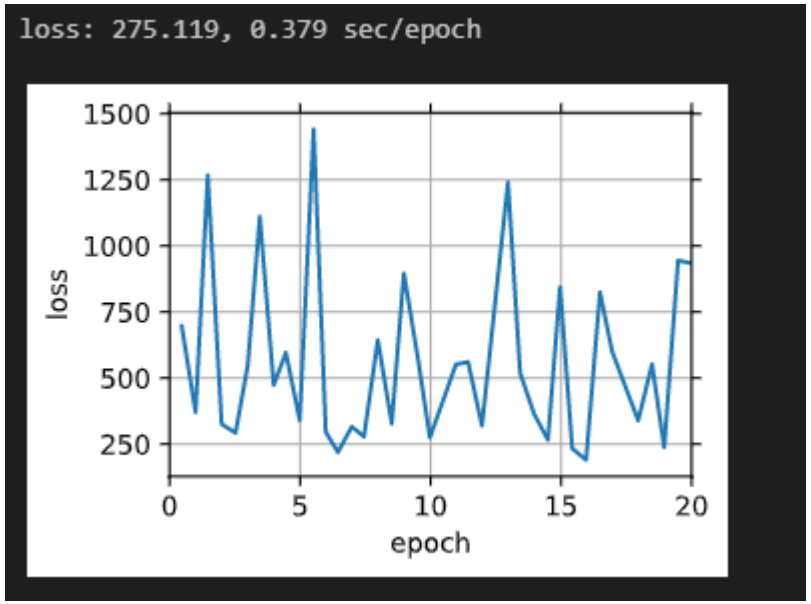
Lr = 0.05

Shallow Fashion MNIST without normalization

- my optimizer



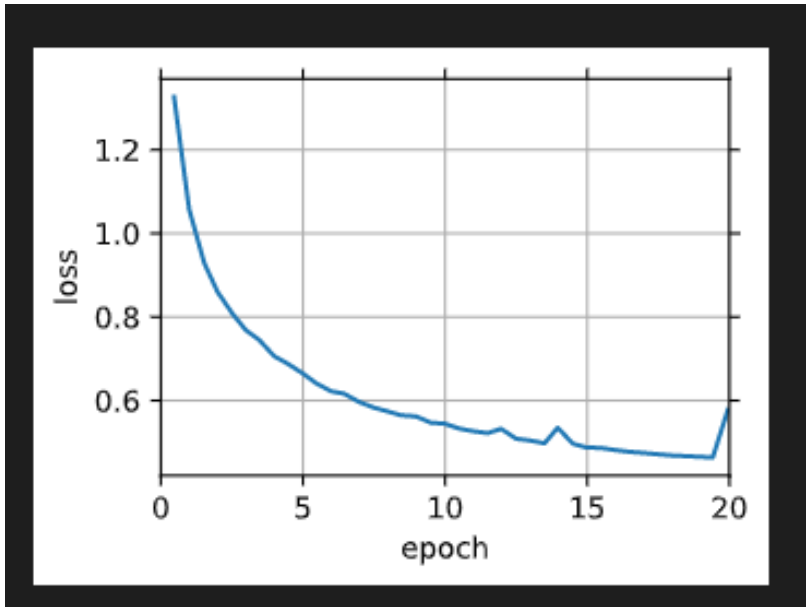
Normal SGD



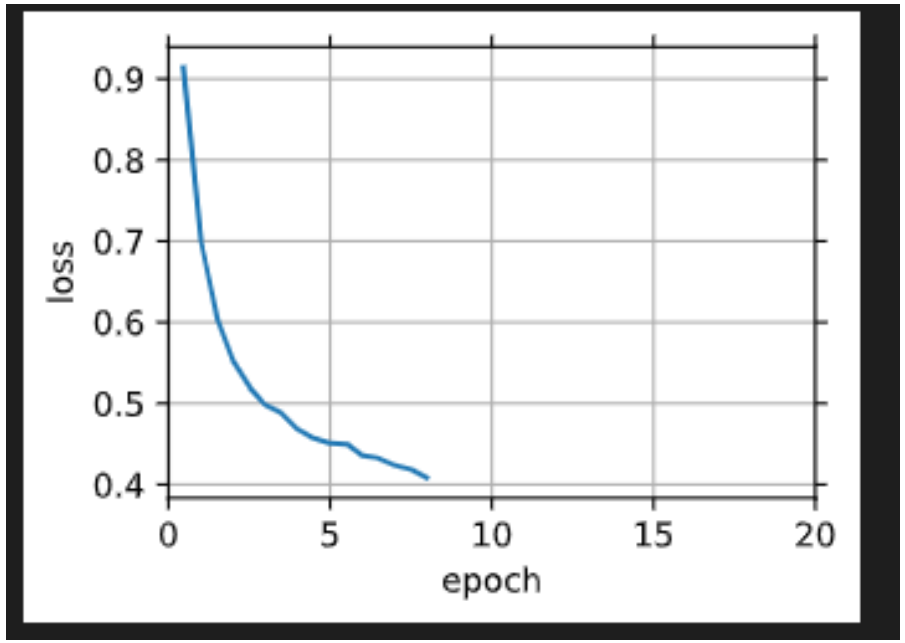
Very small lr = 0.005

Deep Fashion MNIST with normalization

- my optimizer

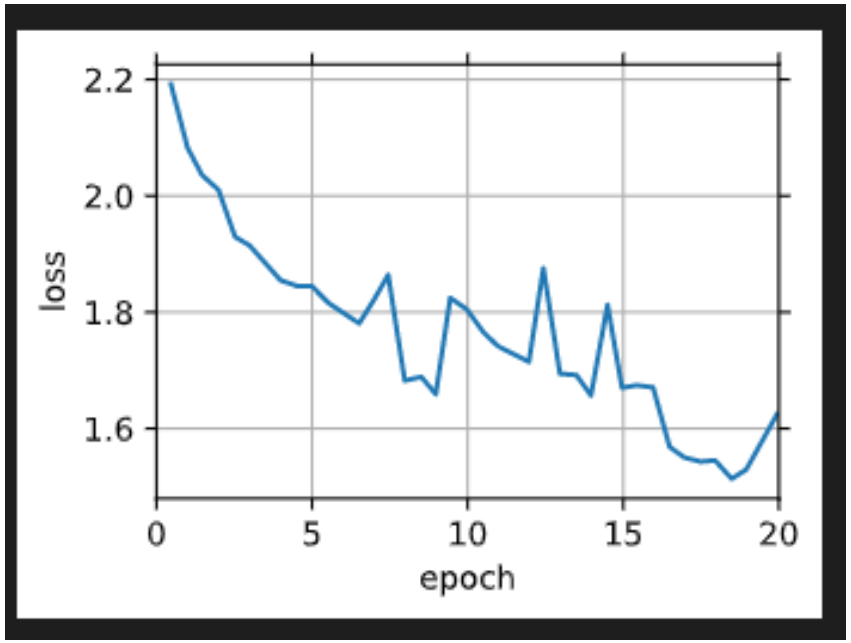


Normal SGD

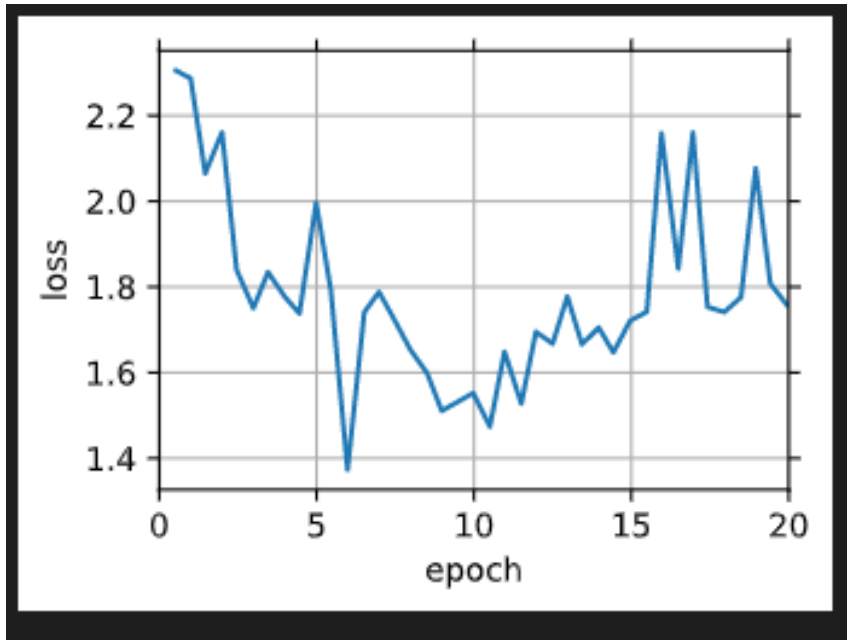


Deep Fashion MNIST without normalization

- my optimizer



Normal SGD



General Training Framework API

- Give any model, any dataset, any loss function, any optimizer (either custom or in-built) – it SHOULD train! and plot the loss curve
- Framework built over Tensorflow/Keras and can accept neural network models, data iterators and loss functions in Keras in-built format and directly train over it.

```
get_classification_data(batch_size, normalized)
get_regression_data(batch_size, normalized)

general_trainer(tf.keras.Model, tf.data.Iterator, tf.keras.losses,
optimizer, hyperparameters, epochs)
```

General Training Framework API

```
data_iter, test_iter = get_classification_data(batch_size=256, normalized = True)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
general_trainer(fashion_mnist, data_iter, loss, my_optimizer, {'power_factor': 2}, 40)
```

Future Work

- 1. Proving the REGRET bounds of this optimizer
- 2. Proving the series-limit bounds of this optimizer
- 3. Using the optimizer over Attention-based models
- 4. Building a better benchmarking strategy to compare performance over neural networks