Optimization of Neural Networks - Part 1

Contents

1. What am I trying to do?
2. Batch Gradient Descent vs. Stochastic Gradient Descent vs. Mini-Batch Gradient Descent
3. Issues with Gradient Descent
   A. Diffculty in choosing the fixed learning rate
      a. Slow with low lr and can osciallte/diverge for high lr
   B. Learning rate remains constant through time
      a. Ideally we would like to start fast and slow down near the minima to avoid overshooting
   C. Learning rate remains same across parameters
      a. When signal for a particular parameters are sparse, we would like to make larger updates compared to parameters with frequest signal.
   D. Issues dealing with saddle points and pathological curvatures
4. Fixing Gradient Descent
   A. Second-order optimization methods: Newton's method
   B. RMSProp
   C. Momentum
   D. Nesterov's Accelerated Gradient
5. Realworld Optimizers
   A. SGD + Momentum
   B. AdaGrad
   C. AdaDelta
   D. Adam
   E. AdaMax
   F. AMSGrad
   G. AdamW
6. Which optimizer should I use?
7. Initialization of Neural Networks
   A. Why is initialization important?
   B. Zero initialization
   C. Unifrom
   D. Normal
   E. Kaiming
   F. Pretraining
8. Which initialization strategy should I use?

How did I get here?

1. We have our data
2. We have finalized our initial neural network architecture to train
3. How will my neural network learn?
   A. Minimize the loss function with respect to the network parameters
   B. Calculus to rescue -> Iterative approach -> Gradient Descent

Batch Gradient Descent vs. Stochastic Gradient Descent vs. Mini-batch Gradient Descent

Batch Gradient Descent

1. Batch gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces
2. Not online

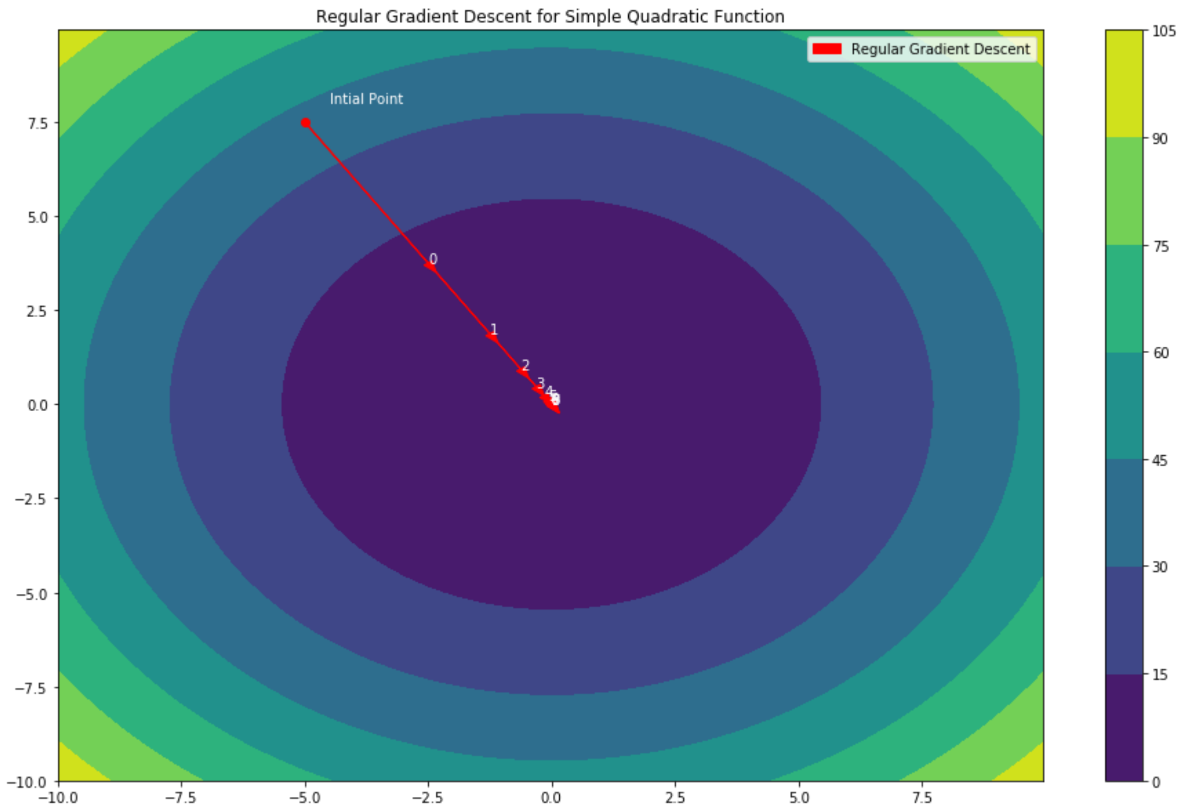
Stochastic Gradient Descent

1. Batch gradient descent computes derivative with respect to all samples in training set. This computation can be very redundant in terms of new information a training sample provides
2. Online
3. Faster. Frequent updates but high variance
4. Convergence can become an issue but with appropriate learning rate scheduling, convergence behaviour is close to that of Bath Gradient Descent
5. Opportunity to jump to a better local minimas

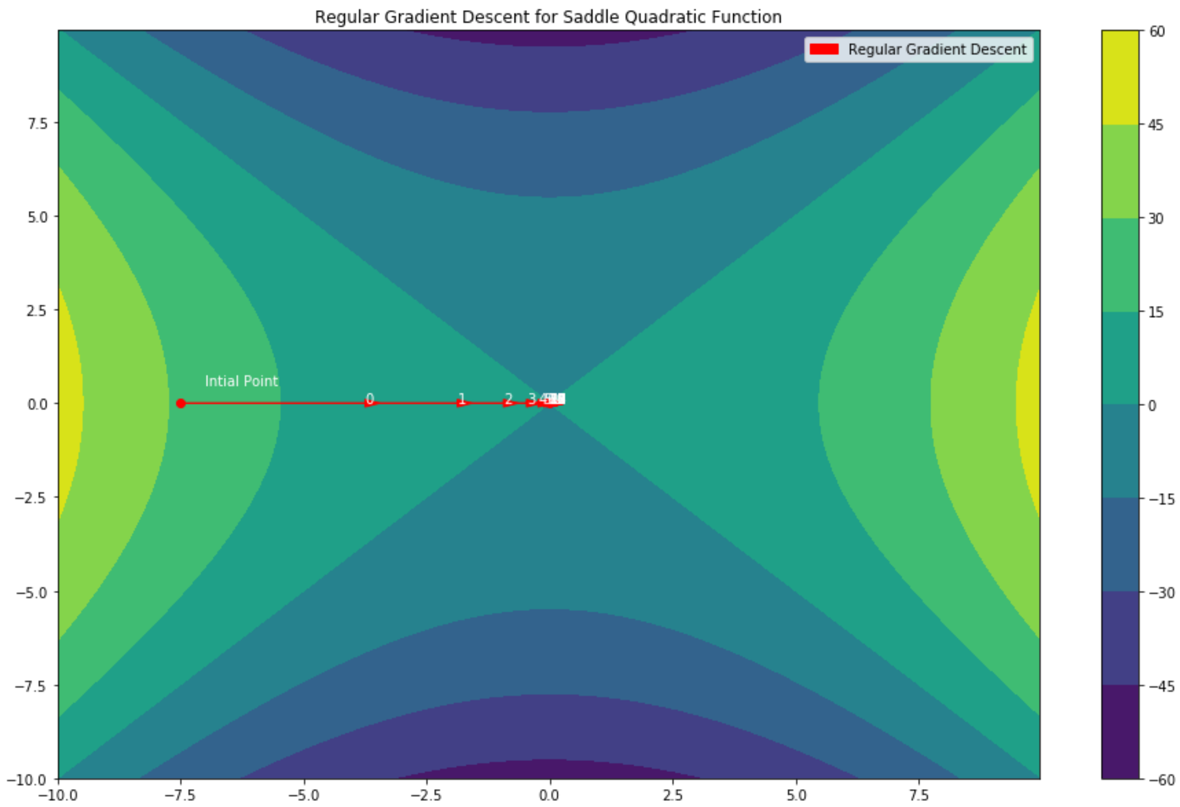
Mini-batch Gradient Descent

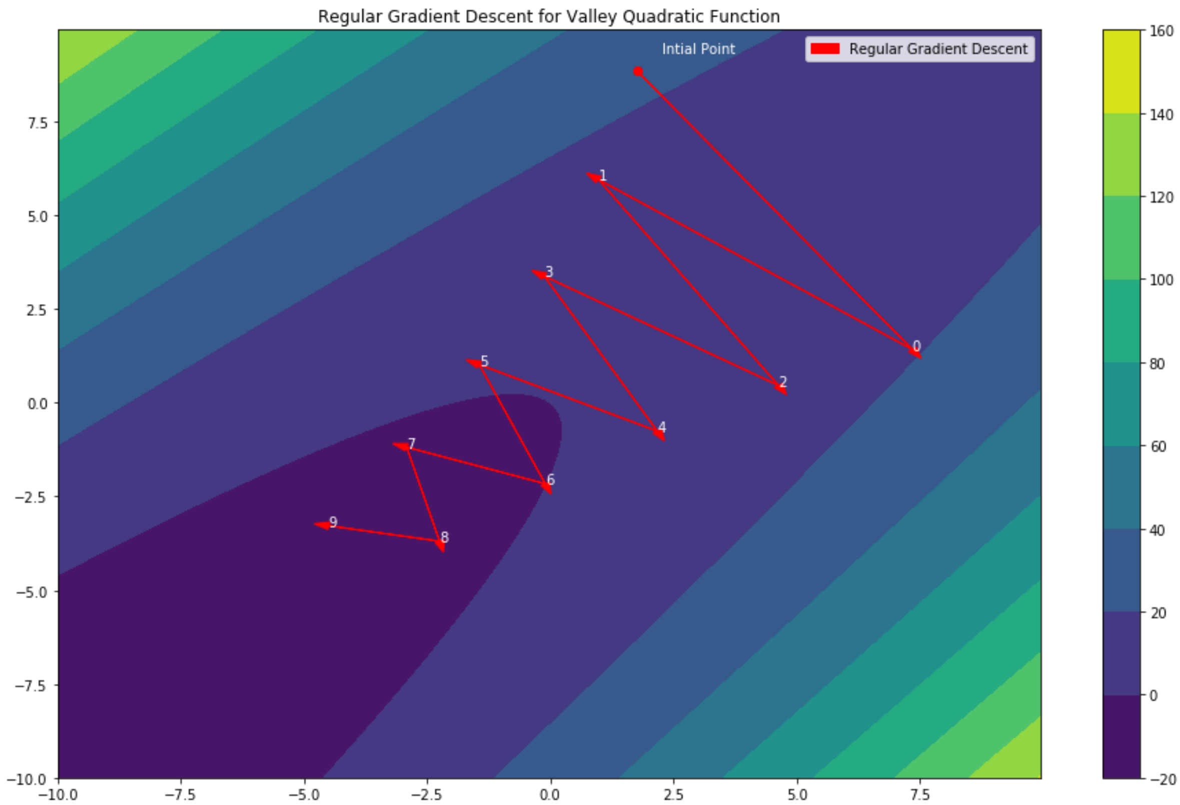
1. More stable convergence as parameters update variance reduces
2. Can be as fast as SGD due to parallelization
3. Searches through a larger part of the parameter space (based on empirical data)

Out[1]: Click here to toggle on/off the raw code.



Issues wih Gradient Descent



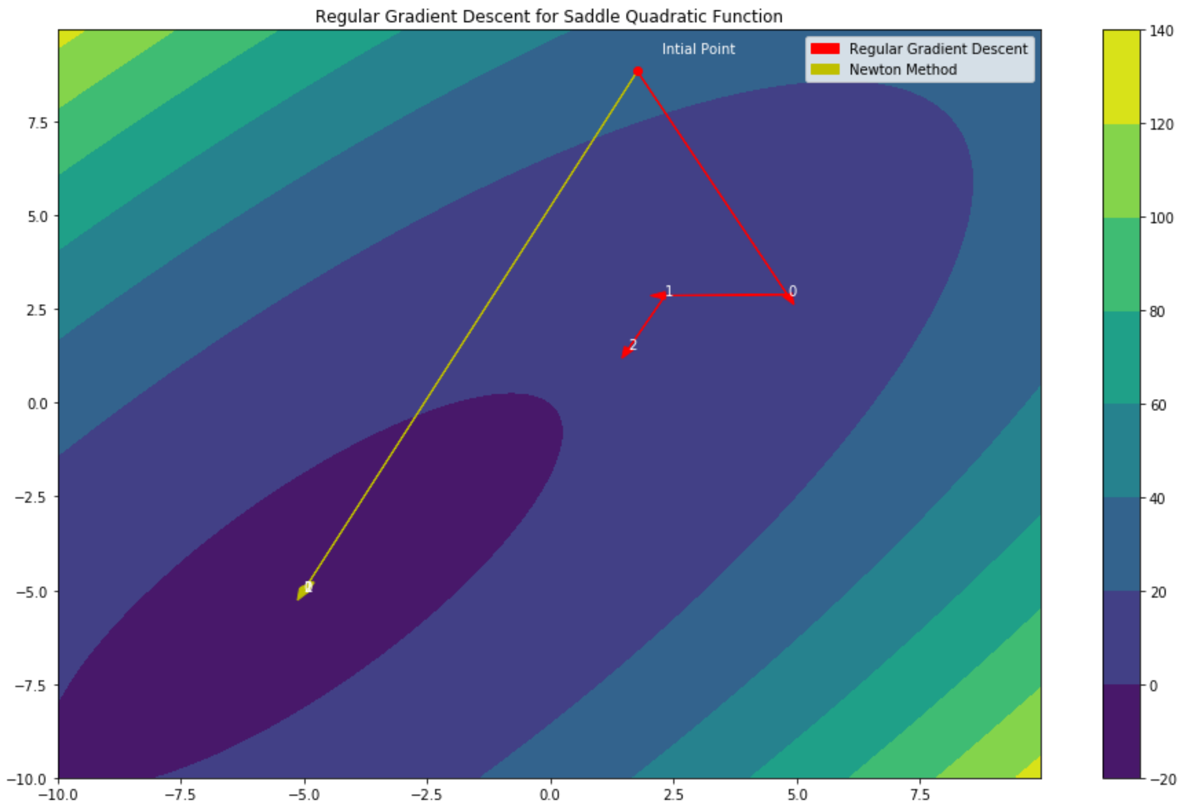


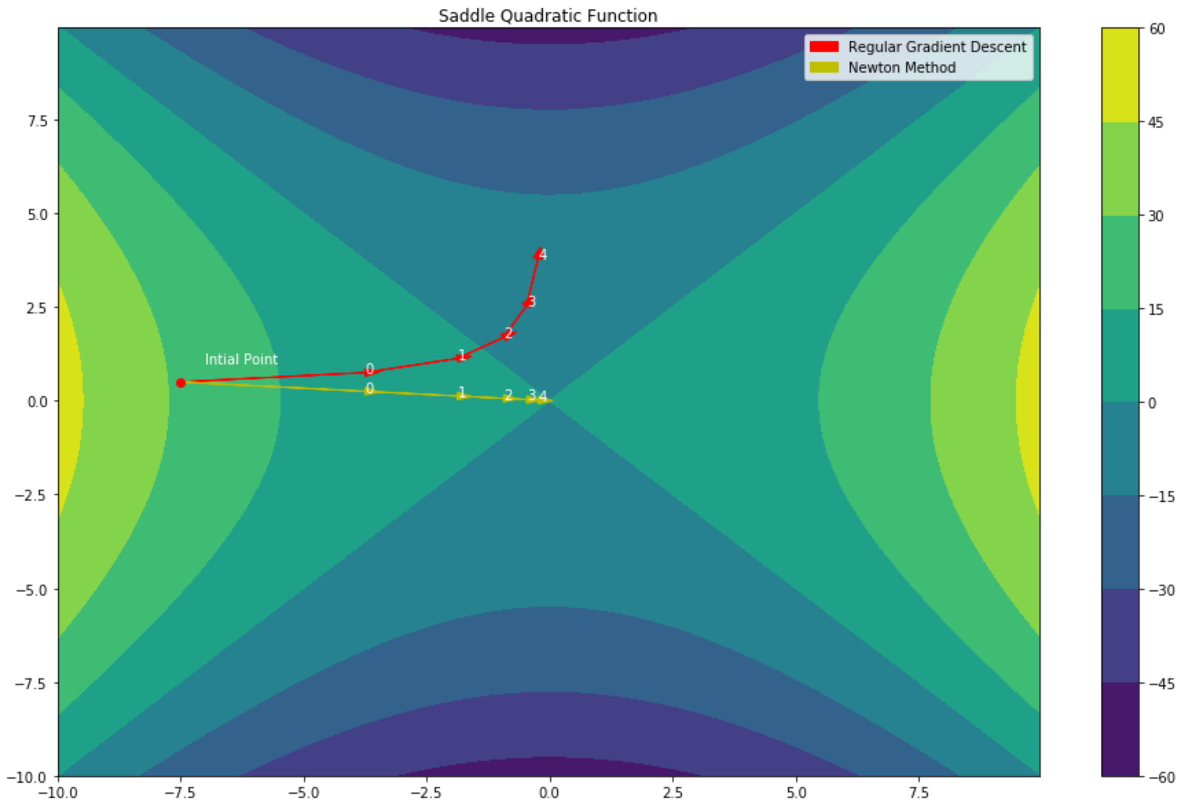
Fixing Gradient Descent

Newton's Method

$$\theta_{t+1} = \theta_t - \eta * H_t^{-1} * g_t$$

θ_t is the parameter at time-step t
η is the learning rate
H_t^{-1} is the inverse Hessian at time-step t
g_t is the gradient at time-step t


Regular Gradient Descent for Saddle Quadratic Function
Intial Point
Regular Gradient Descent
Newton Method
0
1
2


Saddle Quadratic Function
Regular Gradient Descent
Newton Method
Intial Point
0
1
2
3
4

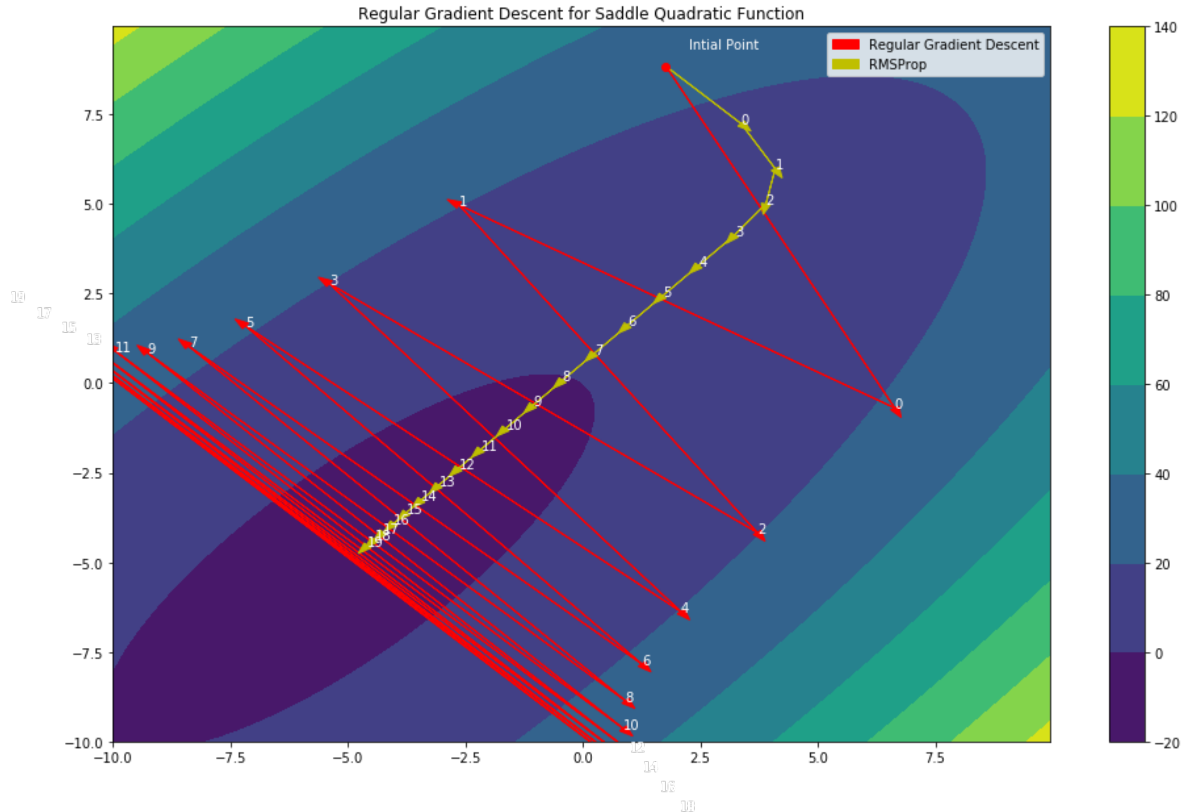
RMSProp

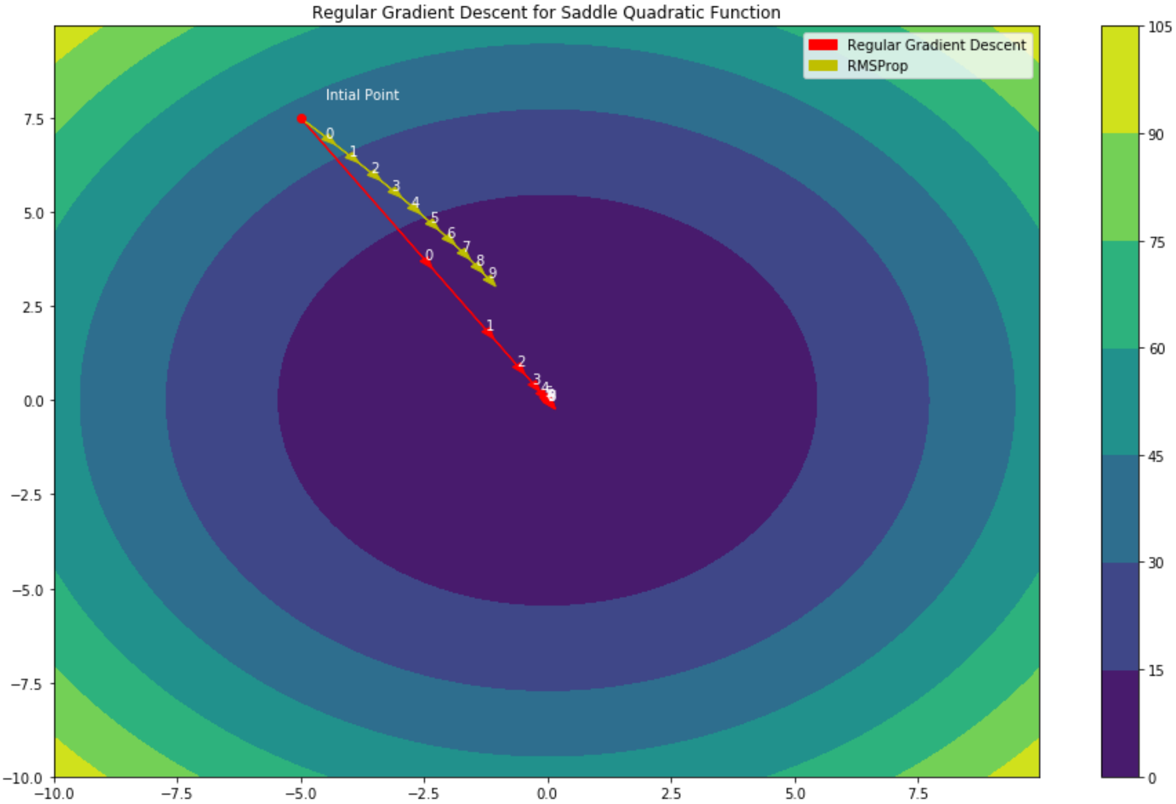
$$v_{t+1} = \gamma v_t + (1-\gamma) g_t^2$$

$$\Delta\theta_{t+1} = \frac{-\eta}{\sqrt{v_{t+1}+\epsilon}} g_t$$

$$\theta_{t+1} = \theta_t + \Delta\theta_{t+1}$$

θ_t is the parameter at time-step t
η is the learning rate
g_t is the gradient at time-step t
v_t is the exponentially decaying average of squared gradients at time-step t
ϵ is used to avoid division by zero



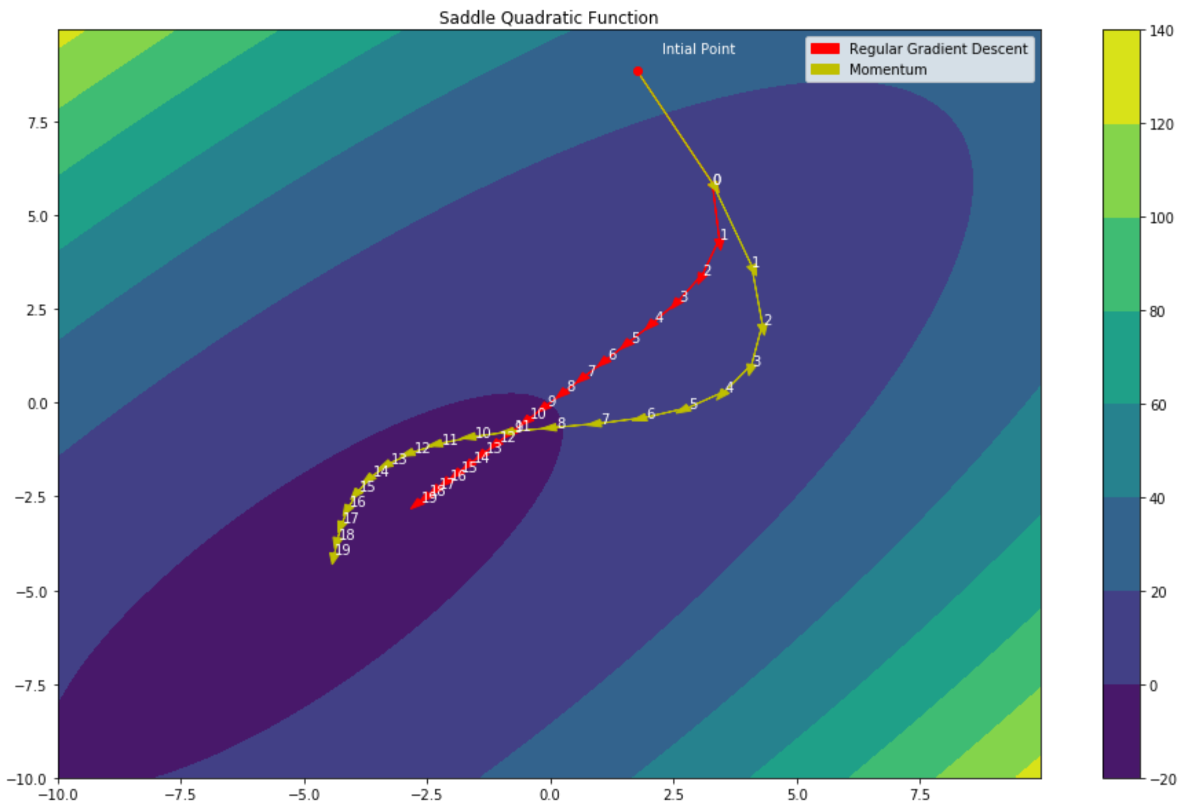


Momentum

$$m_{t+1} = \gamma m_t + \eta g_t$$

$$\theta_{t+1} = \theta_t - m_{t+1}$$

θ_t is the parameter at time-step t
η is the learning rate
g_t is the gradient at time-step t
m_t is the exponentially decaying average of gradients at time-step t



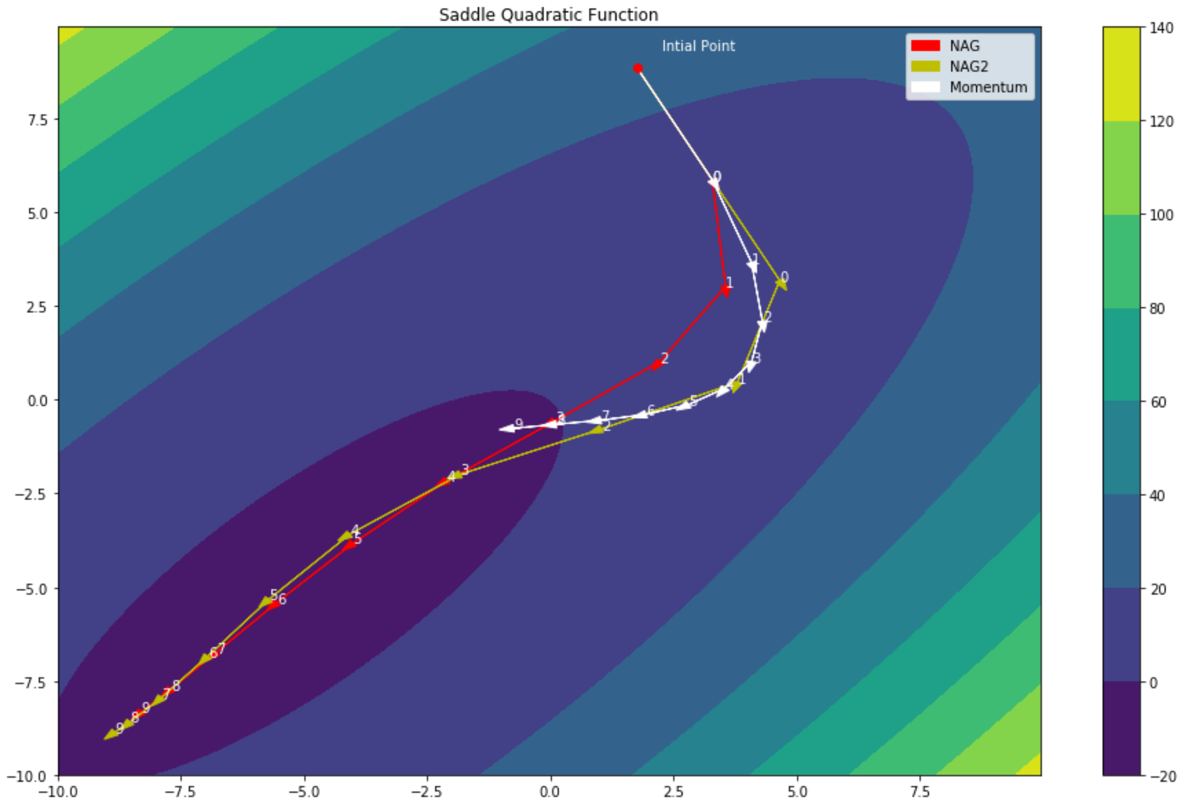
Nesterov's Accelerated Gradient

$$g_t = \nabla f(\theta_t - \eta\gamma m_t)$$

$$m_{t+1} = \gamma m_t + g_t$$

$$\theta_{t+1} = \theta_t - \eta m_{t+1}$$

θ_t is the parameter at time-step t
η is the learning rate
g_t is the gradient at time-step t
m_t is the exponentially decaying average of gradients at time-step t



Adam

$v_{t+1} = \gamma_1 v_t + (1 - \gamma_1) g_t^2$

$m_{t+1} = \gamma_2 m_t + (1 - \gamma_2) g_t$

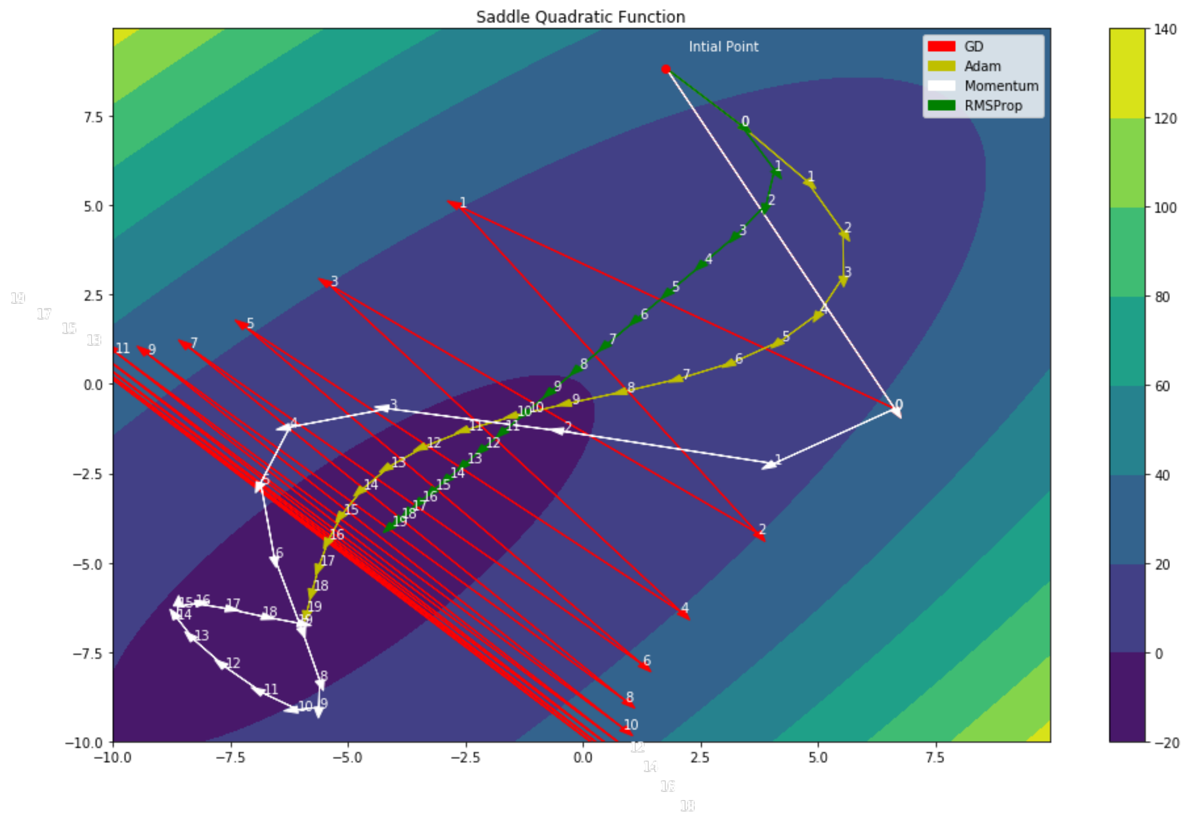
$\hat{v}_{t+1} = \frac{v_{t+1}}{1-\gamma_1^{t+1}}$

$\hat{m}_{t+1} = \frac{m_{t+1}}{1-\gamma_2^{t+1}}$

$\Delta\theta_{t+1} = \frac{-\eta}{\sqrt{\hat{v}_{t+1}}+\epsilon} \hat{m}_{t+1}$

$\theta_{t+1} = \theta_t + \Delta\theta_{t+1}$

θ_t is the parameter at time-step t
η is the learning rate
g_t is the gradient at time-step t
v_t is the exponentially decaying average of squared gradients at time-step t
m_t is the exponentially decaying average of gradients at time-step t
ϵ is used to avoid division by zero



Realworld Optimizers

[https://pytorch.org/docs/stable/optim.html (https://pytorch.org/docs/stable/optim.html)](https://pytorch.org/docs/stable/optim.html)

Which optimizer should I use?

Helpful Heuristics (NOT RULES):

1. Sparse features/data -> Adaptive learning-rate methods
2. Faster convergence -> Adaptive learning-rate methods
3. Better minima -> SGD + momentum

Initialization of Neural Networks

Why is initialization important

1. Resolves the issue of exploding/vanishing gradients/activations (to some extent)
2. Faster convergence
3. Helps reach better minima

QUESTION: Initialize the network with 0? With a contant value?

Short Proof for Xavier and Kaiming Initialization

Forward Pass

$y_l = W_l x_l + b_l$

$x_{l+1} = Relu(y_l)$

Assumptions:

1. W_l is $n_{l+1} \times n_l$ matrix with all it's elements being iid and each distribution symmetric around the mean with $E[W_l] = 0$
2. x_l is $n_l \times 1$ vector with all elements being iid
3. x_l and W_l are mutually independent (element-wise)

$Var[y_{,l}] = n_l Var[w_{,l} x_{,l}]$
$Var[y_{,l}] = n_l Var[w_{,l}] E[x_{,l}]$
$Var[y_{,l}] = \frac{1}{2} n_l Var[w_{,l}] Var[y_{,l-1}]$

And behold....
$Var[y_{,L}] = Var[y_{,1}] \prod_i \frac{1}{2} n_i Var[w_{,i}]$

Kaiming's idea:
Initialize the weights such that $\frac{1}{2} n_i Var[w_{,i}] = 1$ Terefore initialize W_i using a gaussian using mean 0 and std $\sqrt{\frac{2}{n_i}}$

Backward Pass

$\Delta x_l = W_l^T \Delta y_l$

$\Delta y_l = Relu^{'}(y_l) \Delta x_{l+1}$

Assumptions:

1. Δy_l is $n_{l+1} \times 1$ vector with all elements being iid
2. Δy_l and W_l are mutually independent (element-wise)
3. Δx_{l+1} and $Relu^{'}(y_l)$ are mutually independent

$Var[\Delta x_{,l}] = n_{l+1} Var[w^T_{,l} \Delta y_{,l}]$
$Var[\Delta x_{,l}] = n_{l+1} Var[w^T_{,l}] Var[\Delta y_{,l}]$
$Var[\Delta x_{,l}] = \frac{1}{2} n_{l+1} Var[w^T_{,l}] Var[\Delta x_{,l+1}]$

Finally:
$Var[\Delta x_{,2}] = Var[\Delta x_{,L+1}] \prod_{i=2}^{L} \frac{1}{2} n_{i+1} Var[w^T_{,i}]$

Kaiming's idea:
Initialize the weights such that $\frac{1}{2} n_{i+1} Var[w_i^T] = 1$
Terefore initialize W_i using a gaussian using mean 0 and std $\sqrt{\frac{2}{n_{i+1}}}$

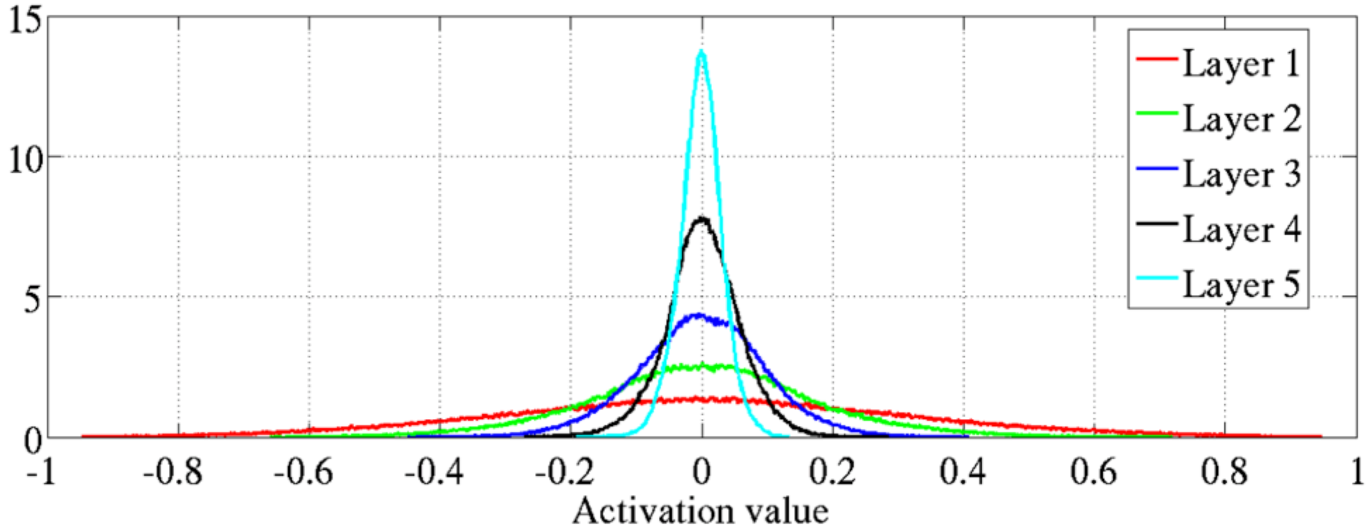
Xavier's Initialization
Similar to Kaiming but did not consider the Relu activation and by default assumed a linear activation. Therefore the factor $\frac{1}{2}$ that we see in the Kaiming's initialization is not present in Xavier. He takes a harmonic mean of the two results for initialization.

Terefore initialize W_i using a gaussian using mean 0 and std $\sqrt{\frac{2}{n_{i+1}+n_i}}$

Kaiming's Initialization Use either of the forward-based initialization or the backward-based initialization. The difference isn't much!

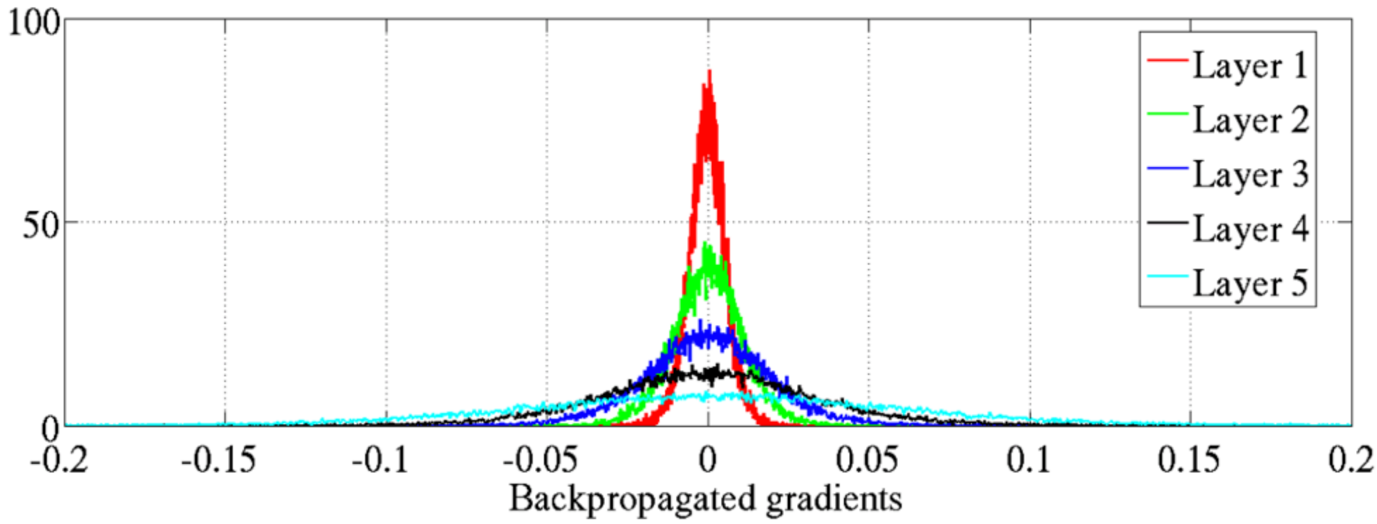
That's all great....but show me the real world results

Activation values with standard initialization



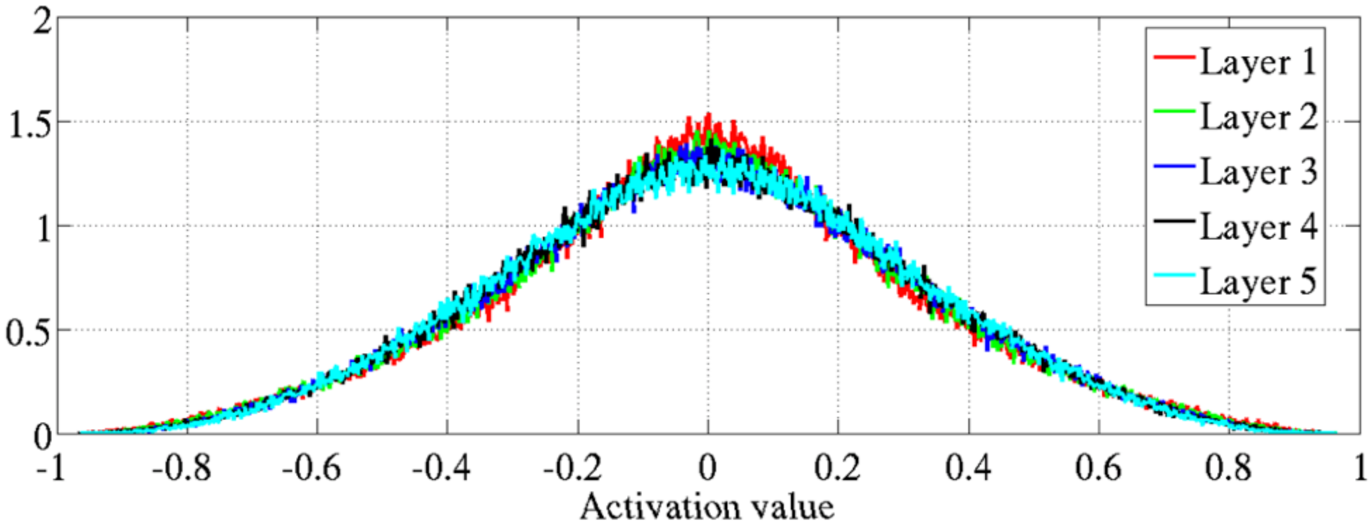
source: [Xavier's Paper (http://proceedings.mlr.press/v9/glorot10a.html)](http://proceedings.mlr.press/v9/glorot10a.html)

Gradient values with standard initialization



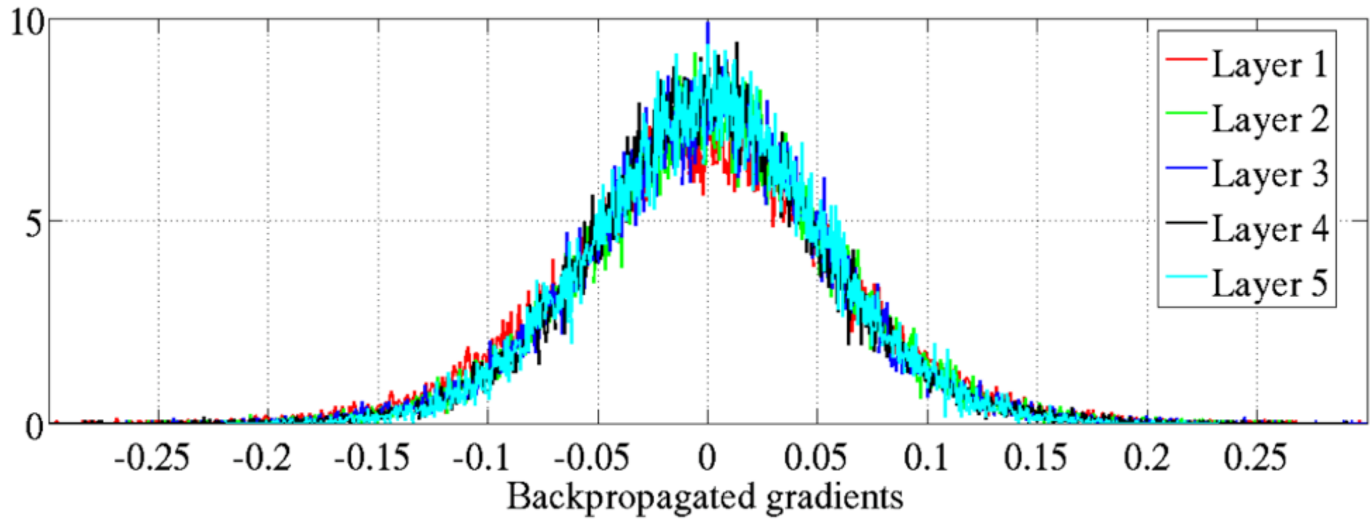
source: [Xavier's Paper (http://proceedings.mlr.press/v9/glorot10a.html)](http://proceedings.mlr.press/v9/glorot10a.html)

Activation values with Xavier's initialization



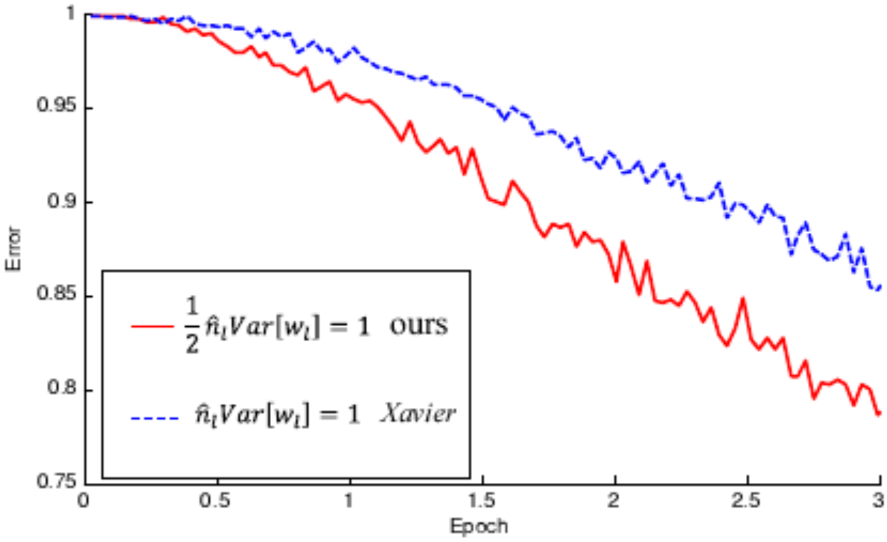
source: [Xavier's Paper (http://proceedings.mlr.press/v9/glorot10a.html)](http://proceedings.mlr.press/v9/glorot10a.html)

Gradient values with Xavier's initialization



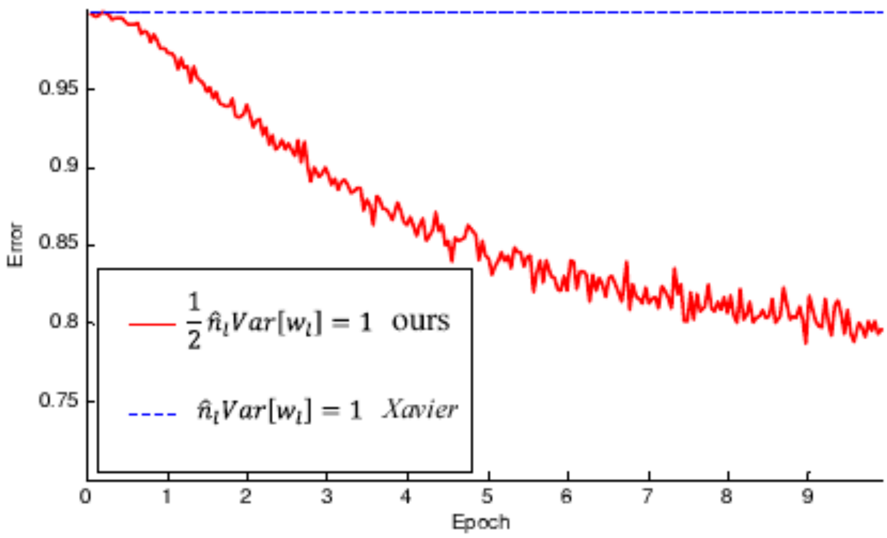
source: [Xavier's Paper (http://proceedings.mlr.press/v9/glorot10a.html)](http://proceedings.mlr.press/v9/glorot10a.html)

Error rate as a function of epochs with Xavier vs Kaiming initialization, 22-layer modelon



source: [Kaiming's Paper (https://arxiv.org/abs/1502.01852)](https://arxiv.org/abs/1502.01852)

Error rate as a function of epochs with Xavier vs Kaiming initialization, 30-layer model



source: [Kaiming's Paper (https://arxiv.org/abs/1502.01852)](https://arxiv.org/abs/1502.01852)

References

[https://pouannes.github.io/blog/initialization/ (https://pouannes.github.io/blog/initialization/)](https://pouannes.github.io/blog/initialization/)
[https://ruder.io/optimizing-gradient-descent/index.html#gradientdescentvariants (https://ruder.io/optimizing-gradient-descent/index.html#gradientdescentvariants)](https://ruder.io/optimizing-gradient-descent/index.html#gradientdescentvariants)