Overview of Stochastic Gradient Descent Algorithms

Srikumar Ramalingam

Reference

Sebastian Ruder, An overview of gradient descent optimization algorithms, 2017

https://arxiv.org/pdf/1609.04747.pdf

Notations

- Objective function $J(\theta)$, where θ is the parameter set $\theta \in \mathbb{R}^n$
- Gradient of the object function is $\nabla_\theta J(\theta)$ with respect to the parameters.
- η is the learning rate.

Batch Gradient Descent

- We compute the gradient of the cost function with respect to the parameters for the entire dataset:

$$\theta = \theta - \eta . \nabla_\theta J(\theta)$$

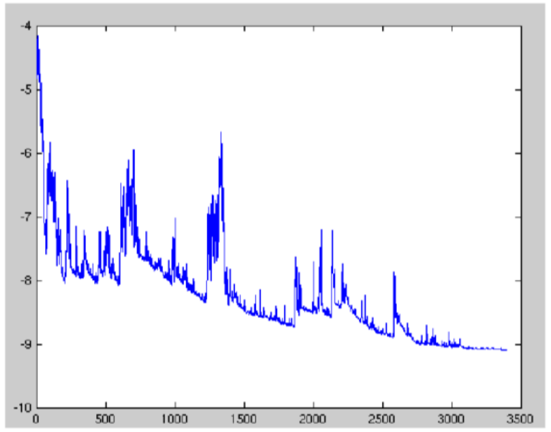
- As we need to calculate the gradients for the whole dataset to perform just one update, batch gradient descent can be very slow and is intractable for datasets that do not fit in memory.
- Batch gradient descent is guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces

Stochastic Gradient Descent

- Parameter update is done for each training example $\left(x^{(i)}, y^{(i)}\right)$.

$$\theta = \theta - \eta . \nabla_{\theta} J\left(\theta ; x^{(i)}, y^{(i)}\right)$$

- SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily.
- SGD's fluctuation enables it to jump to new and potentially better local minima. However, this also complicates convergence to the exact minimum, as SGD will keep overshooting.
- when we slowly decrease the learning rate, SGD shows the same convergence behaviour as batch gradient descent, almost certainly converging to a local or the global minimum for non-convex and convex optimization respectively



SGD fluctuation (Source: Wikipedia)

Mini-batch gradient descent

- Mini-batch gradient descent finally takes the best of both worlds and performs an update for every mini-batch of n training examples:

$$\theta = \theta - \eta . \nabla_{\theta} J\left(\theta; x^{(i:i+n)}, y^{(i:i+n)}\right)$$

- Common mini-batch sizes range between 50 and 256, but can vary for different applications.
- Mini-batch gradient descent is typically the algorithm of choice when training a neural network and the term SGD usually is employed also when mini-batches are used.

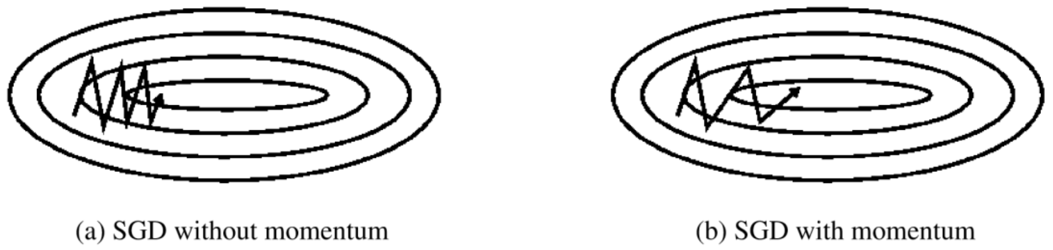
Momentum

- Here we add a fraction γ of the update vector of the past time step to the current update vector.

$$\nu_t = \gamma \nu_{t-1} + \eta . \nabla_\theta J(\theta)$$
$$\theta = \theta - \nu_t$$

- The idea of momentum is similar to a ball rolling down a hill. The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions.
- The momentum term γ is

usually set to 0.9 or a similar value.


(a) SGD without momentum (b) SGD with momentum

Source: Genevieve B. Orr

Nesterov accelerated gradient

- We would like to have a smarter ball, a ball that has a notion of where it is going so that it knows to slow down before the hill slopes up again.
- We can now effectively look ahead by calculating the gradient not w.r.t. to our current parameters θ but w.r.t. the approximate future position of our parameters:

$$\nu_t = \gamma \nu_{t-1} + \eta . \nabla_\theta J(\theta - \eta \nu_{t-1})$$

$$\theta = \theta - \nu_t$$

Adagrad

- For brevity, we set $g_{t,i}$ to be the gradient of the objective function w.r.t. to the parameter θ_i at time step t.
- The SGD update for every parameter $\theta_i$is given by:

$$\theta_{t+1,i} = \theta_{t,i} - \eta . g_{t,i}$$

- The Adagrad update for every parameter $\theta_i$is given by:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i}^2 + \epsilon}} . g_{t,i}$$

- $G_{t,i}$is the sum of squares of the gradients w.r.t. $\theta_i$upto time t, and ϵ is a smoothing variable that avoid division by zero.
- One of Adagrad’s main benefits is that it eliminates the need to manually tune the learning rate. Most implementations use a default value of 0.01 and leave it at that. On the other hand, the learning rate may eventually become infinitesimally small.

Adadelta

- This can be seen as a slight modification of Adagrad. The sum of gradients is recursively defined as a decaying average of all past squared gradients.

$$E[g_t^2] = \gamma E[g_{t-1}^2] + (1-\gamma) g_t^2$$

The update while using Adadelata is given below:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g_{t,i}^2] + \epsilon}} . g_{t,i}$$

- $E[g_{t,i}^2]$ is the the decaying average over past squared gradients w.r.t. θ_i upto time t, and ϵ is a smoothing variable that avoid division by zero.

RMSProp

- RMSprop and Adadelta have both been developed independently around the same time stemming from the need to resolve Adagrad’s radically diminishing learning rates.

$$E[g_t^2] = 0.9E[g_{t-1}^2] + 0.1g_t^2$$

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{E[g_{t,i}^2] + \epsilon}} \cdot g_{t,i}$$

Adam

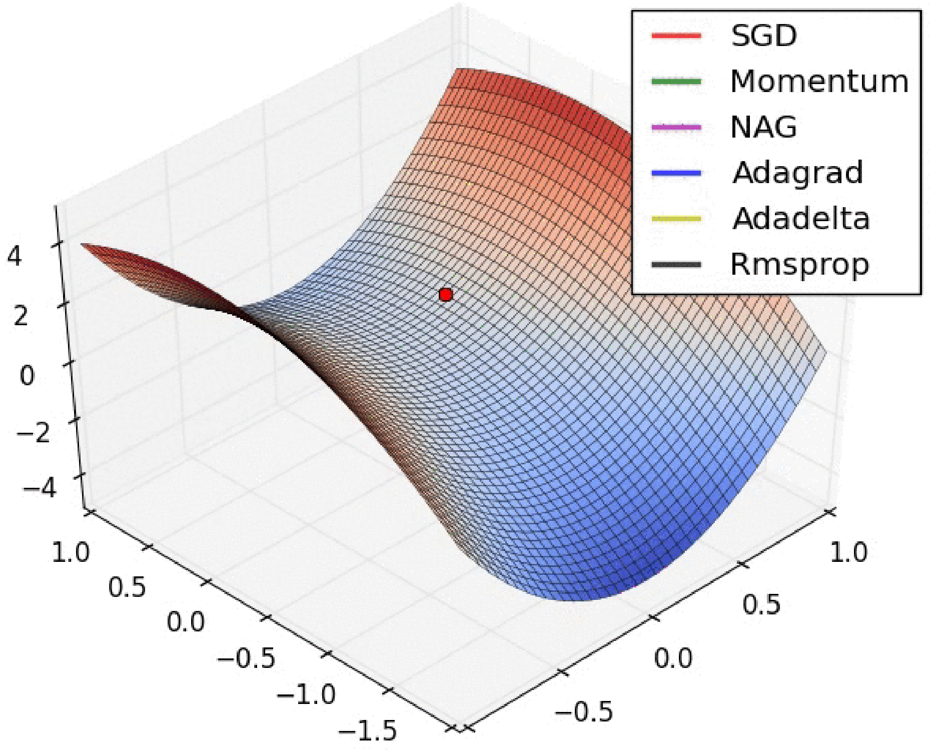
- Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter.
- In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum:

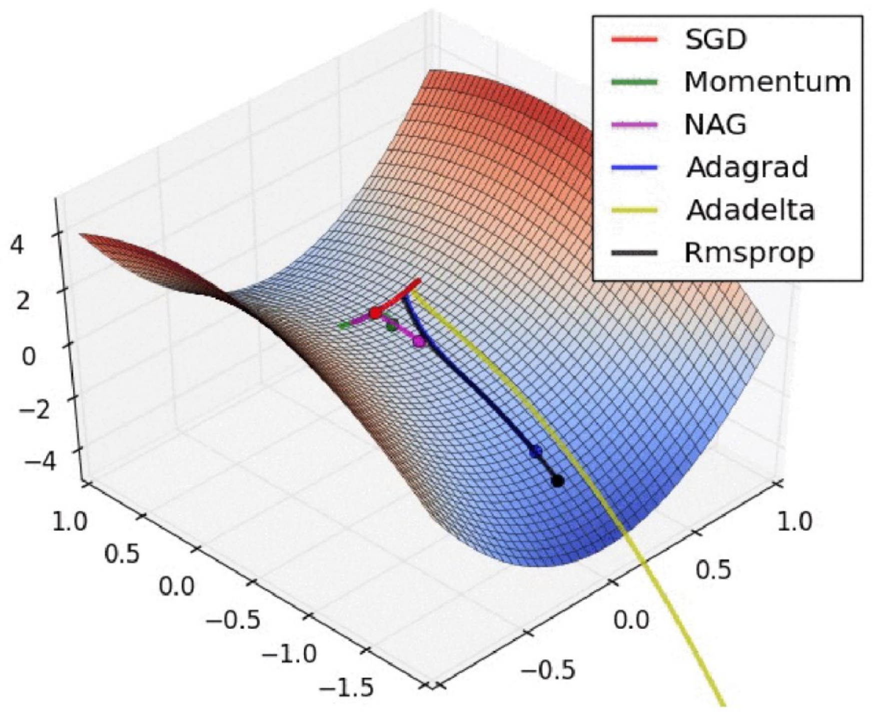
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t$$

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{v_{t,i}} + \epsilon} . m_{t,i}$$

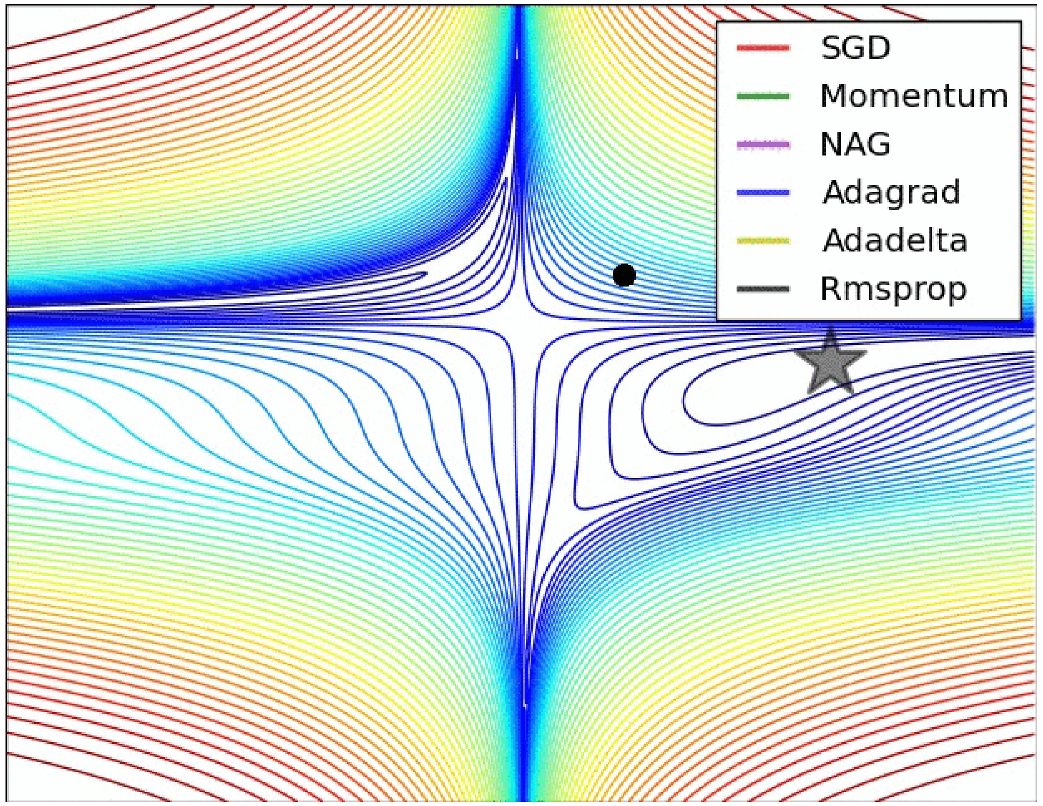
Visualization around a saddle point

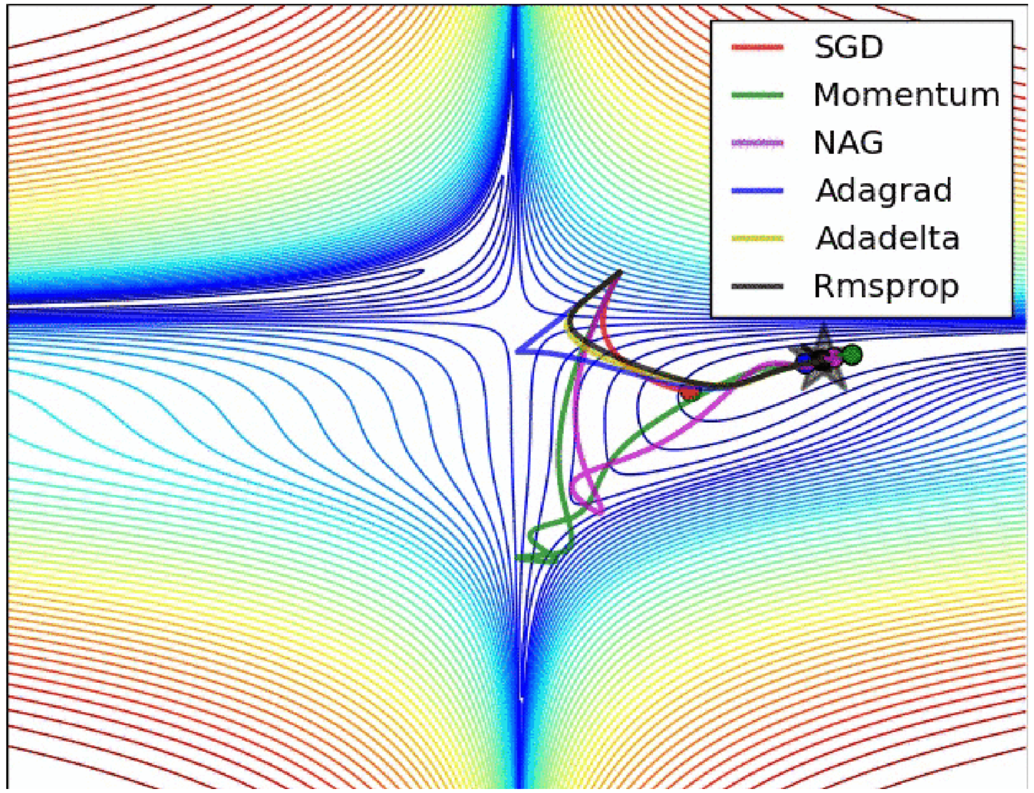




Here SGD, Momentum, and NAG find it difficulty to break symmetry, although the latter two eventually manage to escape the saddle point, while Adagrad, RMSprop, and Adadelta quickly head down the negative slope, with Adadelta leading the charge.

Visualization on Beale Function





- Adadelta, Adagrad, and Rmsprop headed off immediately in the right direction and converged fast.
- Momentum and NAG were off track, and NAG corrected it course eventually.