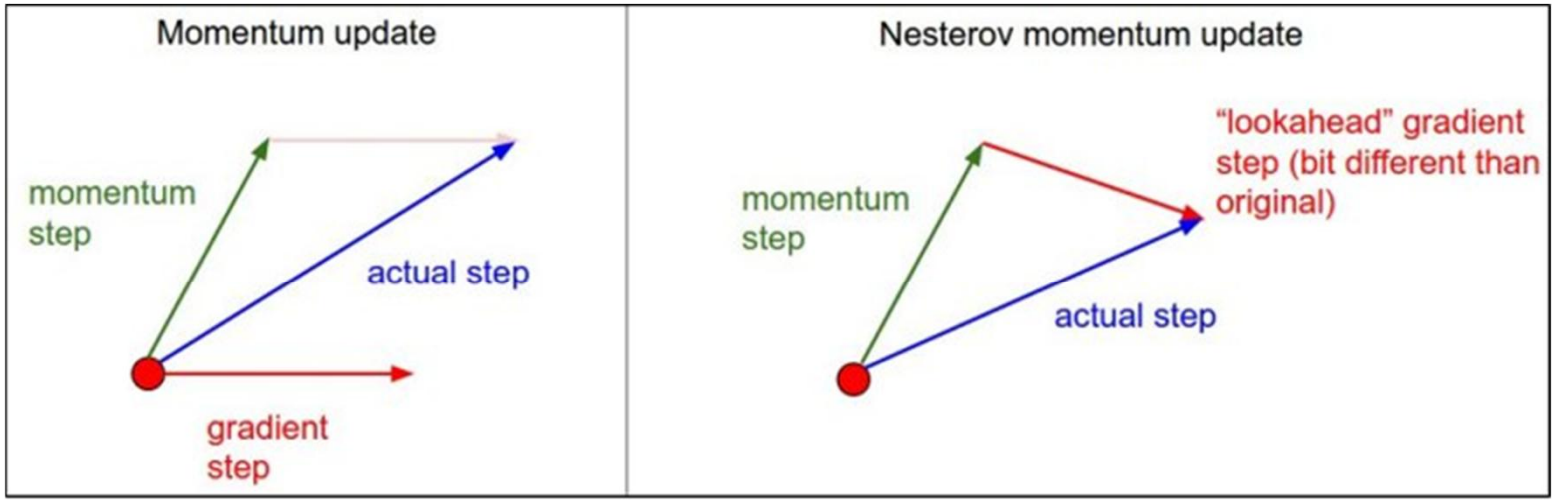
# Momentum vs. Nesterov

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \beta\left(\mathbf{x}_k - \mathbf{x}_{k-1}\right) - s\nabla f\left(\mathbf{x}_k + \gamma\left(\mathbf{x}_k - \mathbf{x}_{k-1}\right)\right)$$

| | step size | momentum | $\gamma$ | convergence |
|---|---|---|---|---|
| gradient descent | $s$ | $\beta = 0$ | 0 | $\frac{\kappa - 1}{\kappa + 1}$ |
| heavy ball | $s$ | $\beta$ | 0 | $\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}$ |
| Nesterov | $s$ | $\beta$ | shift $\nabla f$ by $\gamma\Delta\mathbf{x}$ | |




Difference between Momentum and NAG. Picture from CS231.

# Stochastic Gradient Descent

- Two different problems with classical steepest descent
  - 1. Computing the gradient at every descent step is too expensive
  - 2. The number of weights is even larger → poor results on unseen test data (fail to “generalize”)
- Stochastic gradient descent
    - Use only a “minibatch” of the training data at each step
    - n samples will be chosen randomly
    - Full batch of all the training data → minibatch: $L(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} l_i(\mathbf{x})$
  - 1. Computing $\nabla l_i$ by back propagation on n samples is much faster, often n=1
  - 2. The stochastic algorithm produces weights that succeed on unseen data

training data: $\{(x_1, y_1), \ldots, (x_n, y_n)\} \in \mathbb{R}^d$

$$\begin{cases} d: \text{ dimension of each input sample} \\ n: \text{ number of training data points / samples} \end{cases} \rightarrow \text{both } d \text{ and } n \text{ are large: Lage-scale ML}$$

$$\begin{cases} x_1, \ldots, x_n: \text{ raw images in ImageNet, image data set, text document} \\ y_1, \ldots, y_n: \ \pm 1 \text{ in classical ML, real number in regression} \end{cases}$$

$$\begin{cases} \text{Least-squares}: \ \frac{1}{n}\|\mathbf{Ax}-\mathbf{b}\|_2^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{a}_i{}^T\mathbf{x}-\mathbf{b}_i\right)^2 = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x}) \\ \text{LASSO}\left(l_1\text{-least squares}\right): \ \frac{1}{n}\|\mathbf{Ax}-\mathbf{b}\|_2^2 + \lambda\|\mathbf{x}\|_1 = \frac{1}{n}\sum_{i=1}^{n}\left(\mathbf{a}_i{}^T\mathbf{x}-\mathbf{b}_i\right)^2 + \lambda\sum_{j=1}^{n}|x_j| \\ \text{Deep Neural Network}: \ \frac{1}{n}\sum_{i=1}^{n} loss\left(\mathbf{y}_i, DNN(\mathbf{x};\mathbf{a}_i)\right) \end{cases}$$

optimizaton problems we are solving in ML: $\min_{\mathbf{x}} f(\mathbf{x}) \rightarrow \min_{\mathbf{x}} \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x})$

find x over a cost function where it can be written as a sum, "finite sum problems"

$$\mathbf{x}_{k+1} = \mathbf{x}_k - s\nabla f(\mathbf{x}_k) = \mathbf{x}_k - s\underbrace{\nabla\left[\frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x})\right]}_{\text{gradient of sum}} = \mathbf{x}_k - s\underbrace{\frac{1}{n}\sum_{i=1}^{n}\nabla f_i(\mathbf{x})}_{\text{sum of gradient}}$$

- Incremental gradient method
  - What if at iteration k, we randomly pick an integer, $i(k) \in \{1,\ldots,n\}$
  - And instead just perform the update? $\mathbf{x}_{k+1} = \mathbf{x}_k - s\nabla f_{i(k)}(\mathbf{x}_k)$
  - Update requires only gradient info, $\nabla f_{i(k)}$ $\rightarrow n$ times faster than using $\nabla f(\mathbf{x})$
  - But does this make sense?
    - Full gradient → gradient at a singly randomly chosen data point
- Gradient descent(GD): at every step, it descends
- Stochastic Gradient descent(SGD): at every step, it doesn't do any descent
  - Stochastically still seems to be making progress towards the minimum, much more sensitive to step sizes
  - More stable progress in the beginning
  - Fluctuating more when it comes close to the solution

- ML: solving optimization problem (X), finding solutions that work well on unseen data (◯)

$$\min f(x) = \frac{1}{2}\sum_{i=1}^{n}(a_i x - b_i)^2 \rightarrow \nabla f(x) = \sum a_i (a_i x - b_i) = 0 \rightarrow x^* = \frac{\sum a_i b_i}{\sum a_i^2}$$

$$\min f_i(x) = \frac{1}{2}(a_i x - b_i)^2 \rightarrow \nabla f_i(x) = a_i (a_i x - b_i) = 0 \rightarrow x^* = \frac{b_i}{a_i}$$

$$x^* \in \left[\min_i x_i^*, \max_i x_i^*\right] = R(\text{region of confusion})$$

$$\begin{cases} \text{if we have a scalar } x \text{ that lies outside } R? \\ \nabla f_i(x) \text{ has same sign as } \nabla f(x) \rightarrow \text{using } \nabla f_i(x) \text{ instead of } \nabla f(x) \text{ also ensures progress} \\ \text{if inside } R? \text{ this behavior breaks down some SG same sign, some may not} \end{cases}$$

$$\begin{cases} \text{If } \dfrac{B}{A} \text{ is the largest ratio } \dfrac{b_i}{a_i}, \text{ then the true solution } x^* \text{ is below } \dfrac{B}{A}. \\ \text{all } \dfrac{b_i}{a_i} \le \dfrac{B}{A} \rightarrow A a_i b_i \le B a_i^{\,2} \rightarrow A\left(\sum a_i b_i\right) \le B\left(\sum a_i^{\,2}\right) \rightarrow x^* = \dfrac{\sum a_i b_i}{\sum a_i^{\,2}} \le \dfrac{B}{A} \\ \text{Similarly } x^* \text{ is above the smallest ratio } \dfrac{\beta}{\alpha}. \end{cases}$$

$$\begin{cases} \text{If } x_k \text{ is outside } I, \text{ then } x_{k+1} \text{ moves toward the interval } \dfrac{\beta}{\alpha} \le x \le \dfrac{B}{A}. \\ \text{If } x_k \text{ is inside } I, \text{ then so is } x_{k+1}. \text{ The iterations can bounce around inside } I. \end{cases}$$

SGD uses "stochastic gradient" $g(x)$ such that $E\left[g(x)\right] = \nabla f(x)$

$$E\left[g(x)\right] = E\left[\nabla f_i(x)\right] = \sum_{i=1}^{n} \frac{1}{n} \nabla f_i(x) = \nabla f(x)$$

variance: key thing that governs the speed

$$\text{randomness} \begin{cases} \text{option 1: randomly pick an index } i \text{ with replacement} \leftarrow \text{theory} \\ \text{option 2: pick an index } i \text{ without replacement} \leftarrow \text{practice} \end{cases}$$

- Practical challenges
  - How to pick step size?
  - Which mini-batch to use?
  - How to compute stochastic gradient?
    - Back propagation algorithm
- Theoretical challenges
  - Prove that the method “works”
  - Theoretical analysis lagging behind practice
  - Why does SGD work so well for neural networks (generalization theory) than NN with fancy optimization method

# SGD Convergence

$$\begin{cases} \text{Lipschitz smoothness of } \nabla f(x): \ \left\|\nabla f(x)-\nabla f(y)\right\| \le L\left\|x-y\right\| \\ \text{Bounded gradient}: \ \left\|\nabla f_{i(k)}(x)\right\| \le G \\ \text{Unbiased stochastic gradient}: \ E\left[\nabla f_{i(k)}(x)-\nabla f(x)\right]=0 \end{cases}$$

$$f(x_{k+1}) \le f(x_k)+\left(\nabla f(x_k), x_{k+1}-x_k\right)+\frac{1}{2}Ls^2\left\|\nabla f_{i(k)}(x_k)\right\|^2, \ s=\frac{C}{\sqrt{T}}: \text{ step size}$$

$$f(x_{k+1}) \le f(x_k)+\left(\nabla f(x_k), -s\nabla f_{i(k)}(x)\right)+\frac{1}{2}Ls^2\left\|\nabla f_{i(k)}(x_k)\right\|^2$$

$$E\left[f(x_{k+1})\right] \le E\left[f(x_k)\right]-sE\left[\left\|\nabla f(x_k)\right\|^2\right]+\frac{1}{2}Ls^2G^2$$

$$E\left[\left\|\nabla f(x_k)\right\|^2\right] \le \frac{1}{s}E\left[f(x_k)-f(x_{k+1})\right]+\frac{1}{2}Ls^2G^2$$

$$\xrightarrow[k=1,\ldots,T]{\text{add up from}} \frac{1}{T}\sum_{k=1}^{T}E\left[\left\|\nabla f(x_k)\right\|^2\right] \le \frac{1}{T}\left(\frac{f(x_1)-f(x^*)}{C}+\frac{LC}{2}G^2\right)=\frac{C}{\sqrt{T}}$$

$$\min_{1\le k\le T} E\left[\left\|\nabla f(x_k)\right\|^2\right] \le \frac{C}{\sqrt{T}}$$

# Adaptive Methods Using Earlier Gradients

$$x_{k+1} = x_k + s_k D_k \text{ where } \begin{cases} D_k = D\left(\nabla L_k, \nabla L_{k-1}, \ldots, \nabla L_0\right) \leftrightarrow D_k = D\left(\nabla L_k\right) \\ s_k = s\left(\nabla L_k, \nabla L_{k-1}, \ldots, \nabla L_0\right) \leftrightarrow s_k = \dfrac{s}{\sqrt{k}} \end{cases}$$

$\nabla L_k\left(x_k, n\right)$

ADAGRAD(Adaptive Gradient): adapts the learning rate to the parameters

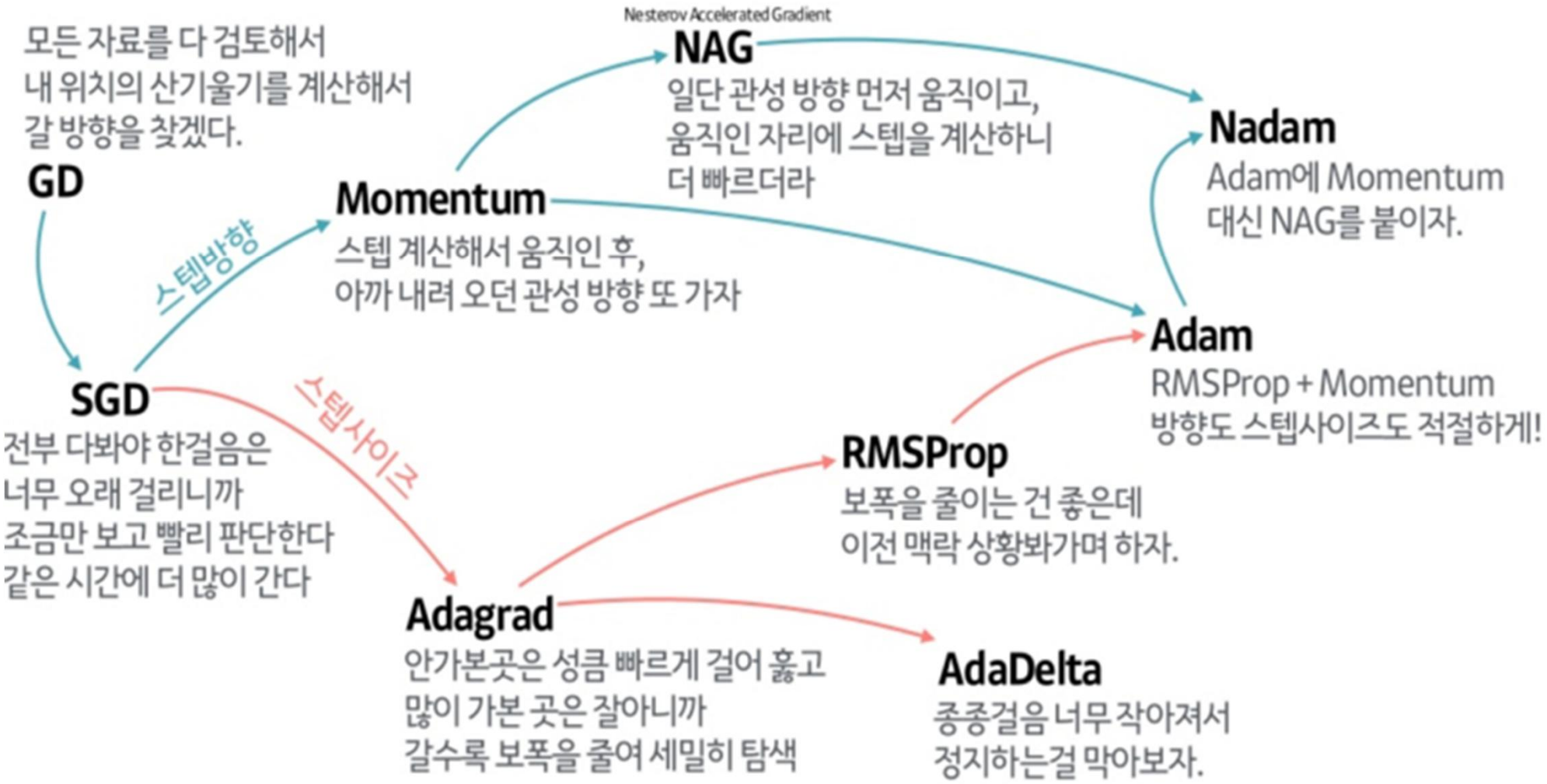
large updates for infrequent parameters, small upate for frequent parameters

$$s_k = \left(\frac{\alpha}{\sqrt{k}}\right)\left[\frac{1}{k} diag\left(\sum_{i=1}^{k} \left\|\nabla L_i\right\|^2\right)\right]^{1/2}$$

ADAM(Adaptive Momentum Estimation): exponential moving averages

$$\begin{cases} D_k = \left(1-\delta\right)\sum_{i=1}^{k} \delta^{k-i} \nabla L\left(x_i\right) \xrightarrow{\delta=0.9} D_k = \delta D_{k-1} + \left(1-\delta\right)\nabla L\left(x_k\right) \\ s_k = \left(\dfrac{\alpha}{\sqrt{k}}\right)\left[\left(1-\beta\right) diag\left(\sum_{i=1}^{k} \beta^{k-i} \left\|\nabla L\left(x_i\right)\right\|^2\right)\right]^{1/2} \xrightarrow{\beta=0.999} {s_k}^2 = \beta {s_{k-1}}^2 + \left(1-\beta\right)\left\|\nabla L\left(x_k\right)\right\|^2 \end{cases}$$

# Optimizer




https://onevision.tistory.com/entry/Optimizer-의-종류와-특성-momentum-RMSProp-Adam

# Gradient Descent Variants (1)

- amount of data used for update
  - Batch gradient descent
  - Stochastic gradient descent
  - Mini-batch gradient descent

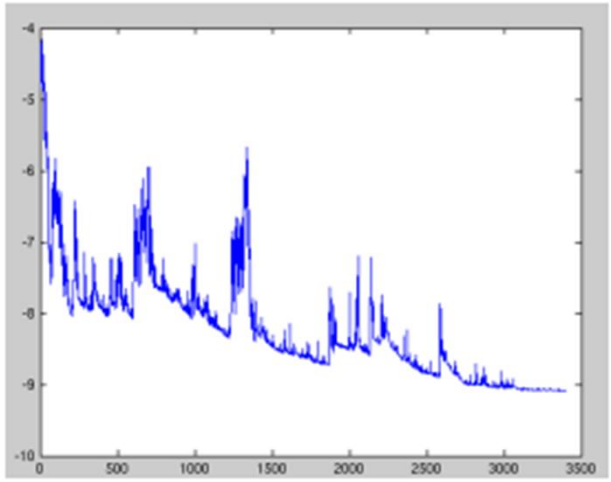



Figure: SGD fluctuation (Source: Wikipedia)

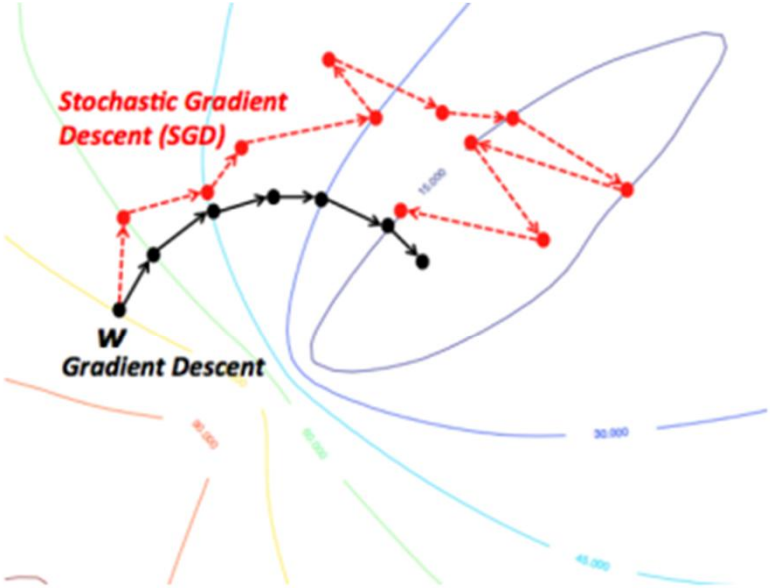



Figure: Batch gradient descent vs. SGD fluctuation (Source: wikidocs.net)

# Gradient Descent Variants (2)

- Mini-batch idea

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{s_k}{|I_k|} \sum_{j \in I_k} \nabla f_j(\mathbf{x}_k)$$

  - Useful in parallel setting
  - Averaging thing → reduce the variance
  - The more quantities you average, the less noise you have
  - For very large mini-batch, SGD → full GD (batch gradient descent)
  - Decrease noise so much → region of confusion shrinks → bad for ML (overfitting your NN)
  - Common size: 50~256

| Gradient Descent | Accuracy | Time | Memory usage | Online learning |
|---|---|---|---|---|
| Batch | ◯ | Slow | High | X |
| Mini-batch | ◯ | Medium | Medium | ◯ |
| Stochastic | △ | Fast | Low | ◯ |

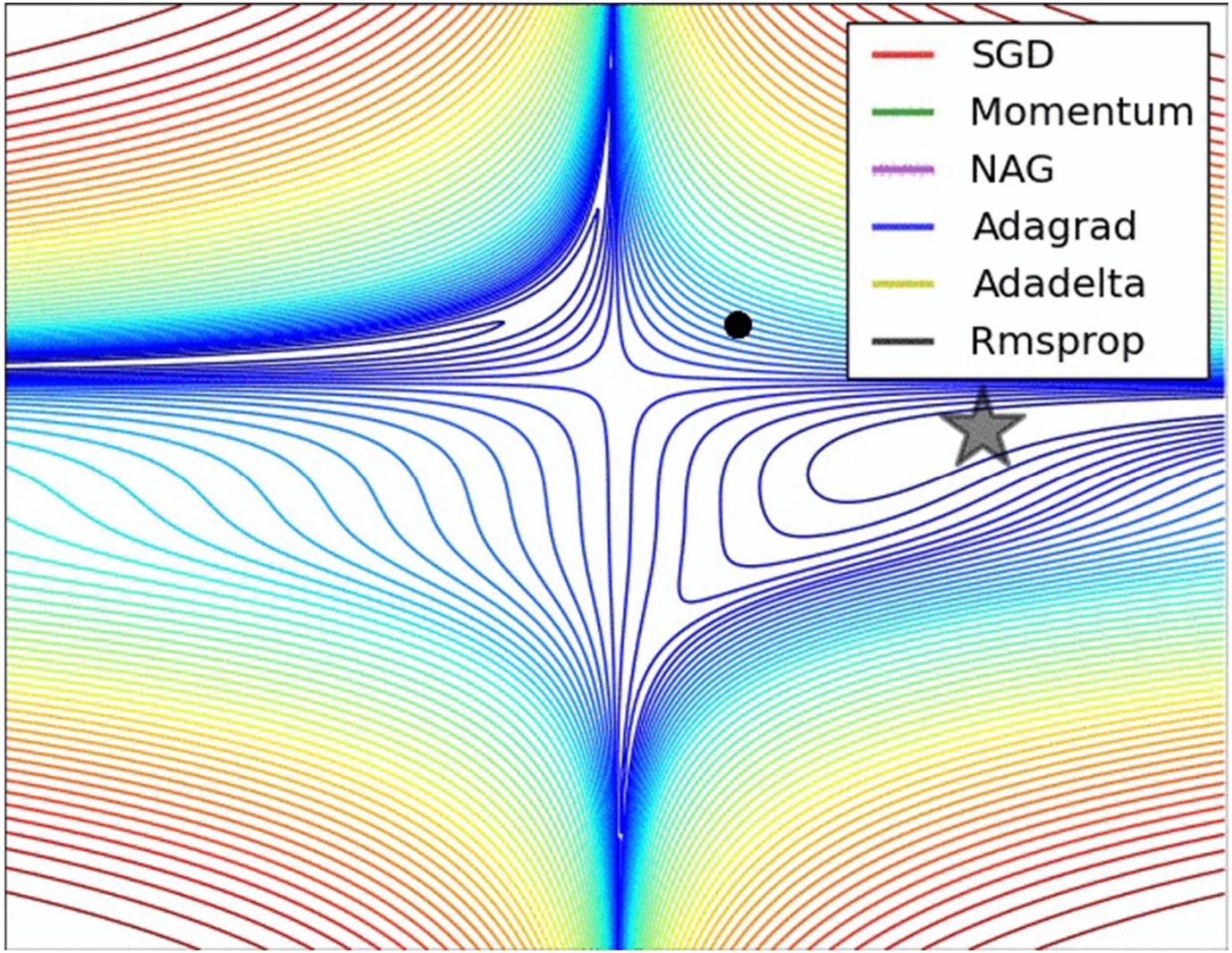
# Challenges

- Choosing a proper learning rate
  - too small, too large, schedules
  - Same learning rate applies to all parameter updates
- Defining an annealing schedule
- Updating features to different extent
- Avoiding suboptimal minima
  - Avoiding getting trapped in their numerous local minima

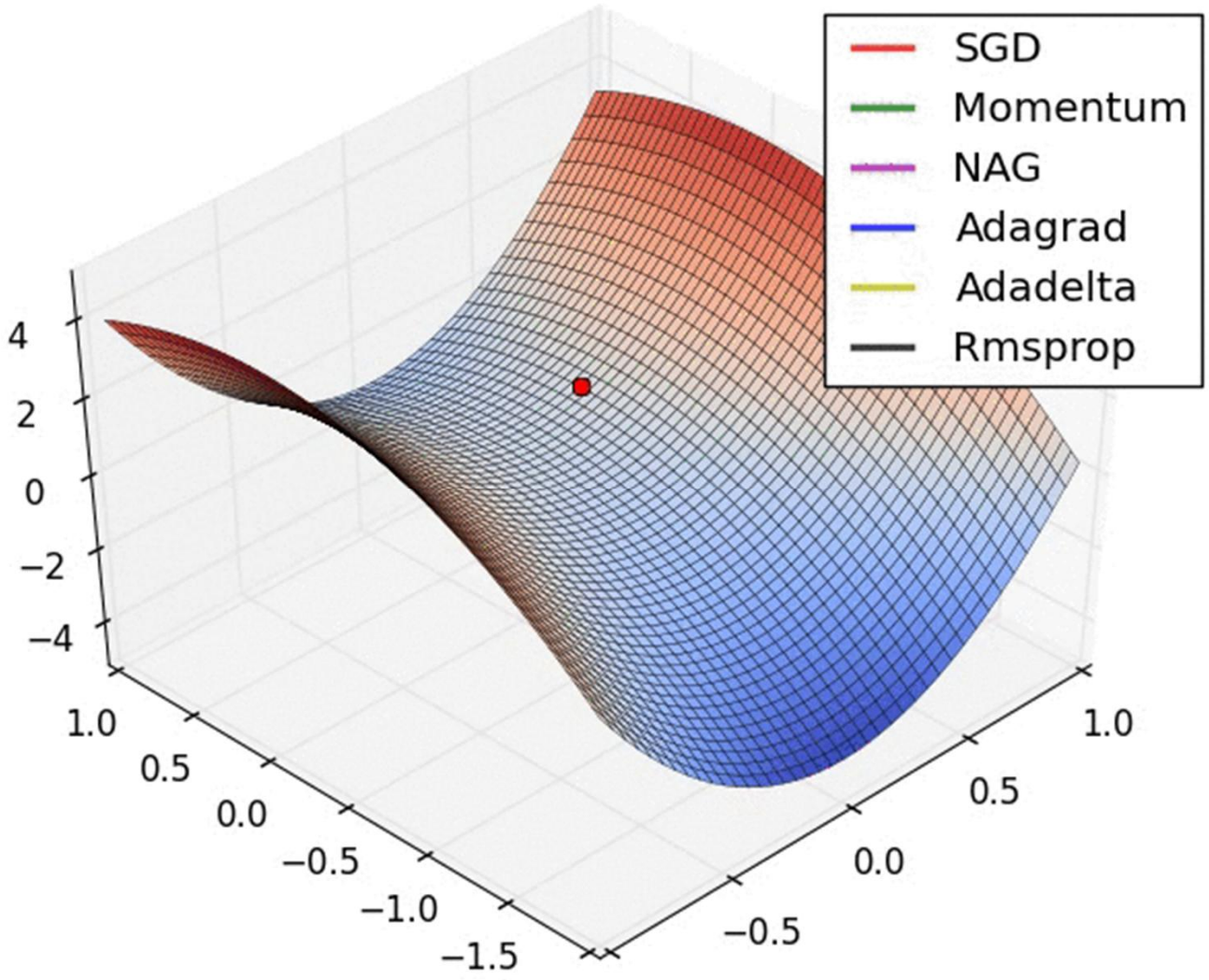
# Gradient Descent Optimization Algorithms

- Momentum
- Nesterov Accelerated Gradient
- Adagrad
- Adadelta
- RMSprop
- Adam (Adaptive Moment Estimation)
- Adam extensions
  - AdaMax: Adam with $l_\infty$ norm
  - Nadam: Adam with Nesterov accelerated gradient

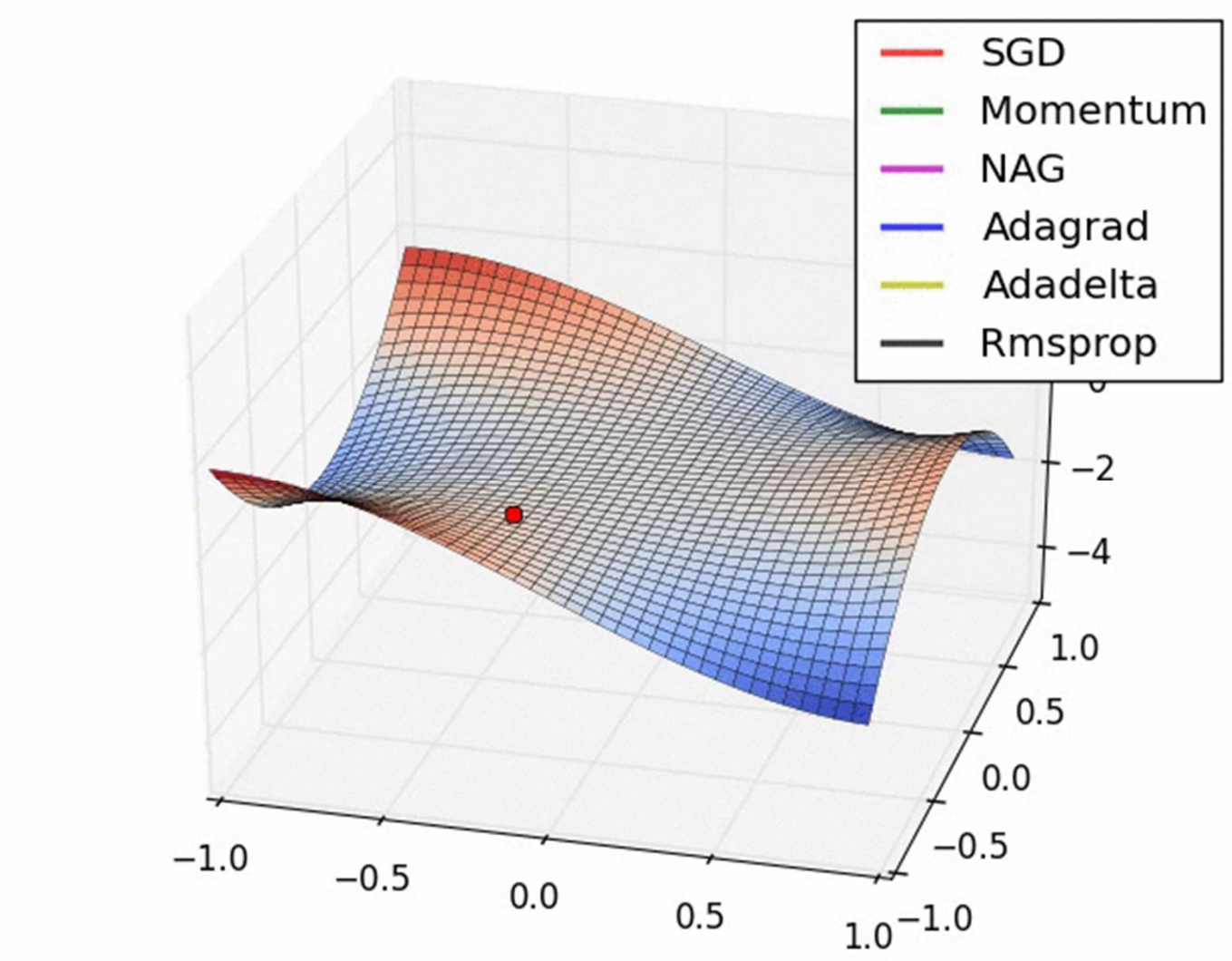
# loss surface (the Beale function)



# behaviour of the algorithms at a long valley



# behaviour of the algorithms at a saddle point



| Method | Update equation |
|---|---|
| | $g_t = \nabla_{\theta_t} J(\theta_t)$ |
| SGD | $\Delta\theta_t = -\eta \cdot g_t$ |
| | $\theta_t = \theta_t + \Delta\theta_t$ |
| Momentum | $\Delta\theta_t = -\gamma\, v_{t-1} - \eta g_t$ |
| NAG | $\Delta\theta_t = -\gamma\, v_{t-1} - \eta \nabla_\theta J(\theta - \gamma v_{t-1})$ |
| Adagrad | $\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$ |
| Adadelta | $\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$ |
| RMSprop | $\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$ |
| Adam | $\Delta\theta_t = -\frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$ |

# Which optimizer to choose?

- Adaptive learning rate methods (Adagrad, Adadelta, RMSprop, Adam) are particularly useful for sparse features.
- Adagrad, Adadelta, RMSprop, and Adam work well in similar circumstances.
- [Kingma and Ba, 2015] show that bias-correction helps Adam slightly outperform RMSprop