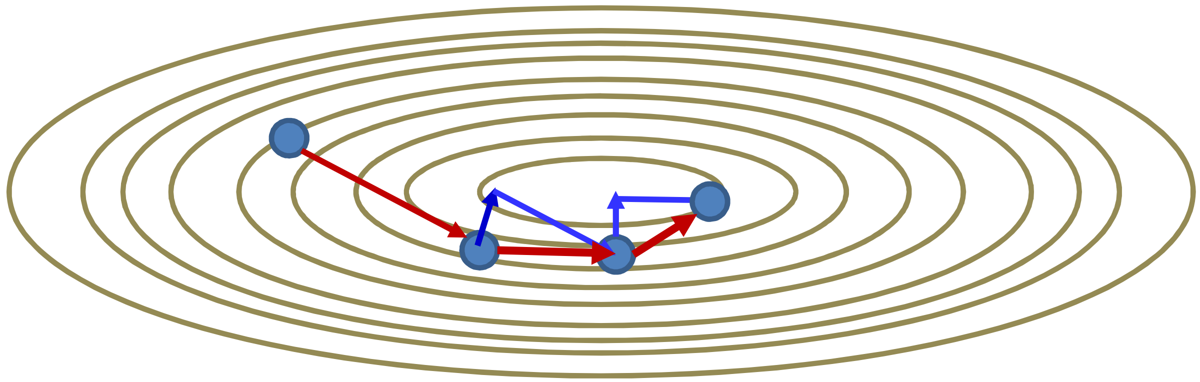
Recall: Momentum

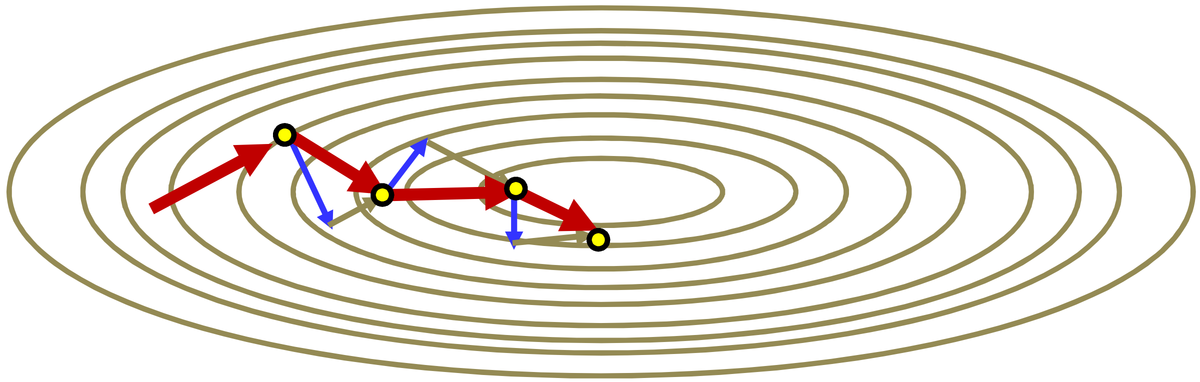


- The momentum method

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Err\left(W^{(k-1)}\right)$$

- Updates using a running average of the gradient

Momentum and incremental updates

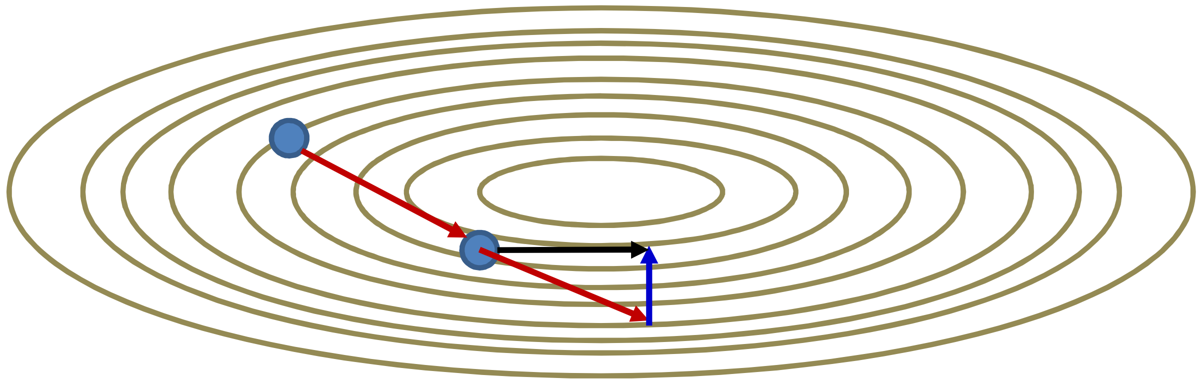


- The momentum method

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Loss\left(W^{(k-1)}\right)^T$$

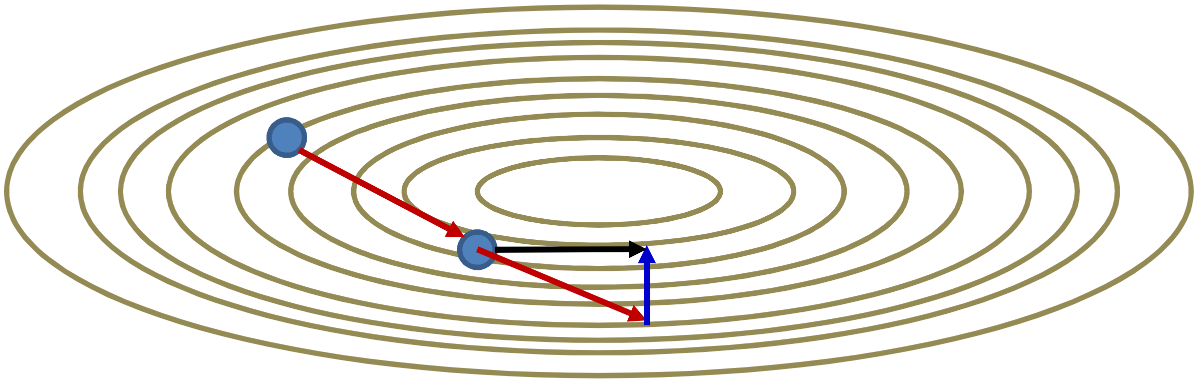
- Incremental SGD and mini-batch gradients tend to have high variance
- Momentum smooths out the variations
  - Smoother and faster convergence

Nestorov's Accelerated Gradient



- At any iteration, to compute the current step:
  - First extend the previous step
  - Then compute the gradient at the resultant position
  - Add the two to obtain the final step
- This also applies directly to incremental update methods
  - The accelerated gradient smooths out the variance in the gradients

Nestorov's Accelerated Gradient



- Nestorov's method

$$\Delta W^{(k)} = \beta \Delta W^{(k-1)} - \eta \nabla_W Loss(W^{(k-1)} + \beta \Delta W^{(k-1)})^T$$

$$W^{(k)} = W^{(k-1)} + \Delta W^{(k)}$$

Incremental Update: Mini-batch update

- Given (X_1, d_1), (X_2, d_2),…, (X_T, d_T)
- Initialize all weights $W_1, W_2, \ldots, W_K$; $j = 0$, $\Delta W_k = 0$
- Do:
  - Randomly permute (X_1, d_1), (X_2, d_2),…, (X_T, d_T)
  - For $t = 1 : b : T$
    - $j = j + 1$
    - For every layer k:
      - $W_k = W_k + \beta \Delta W_k$
      - $\nabla_{W_k} Loss = 0$
    - For t’ = t : t+b-1
      - For every layer k:
        - » Compute $\nabla_{W_k} Div(Y_t, d_t)$
        - » $\nabla_{W_k} Loss \mathrel{+}= \frac{1}{b} \nabla_{W_k} \boldsymbol{Div}(Y_t, d_t)$
    - Update
      - For every layer k:

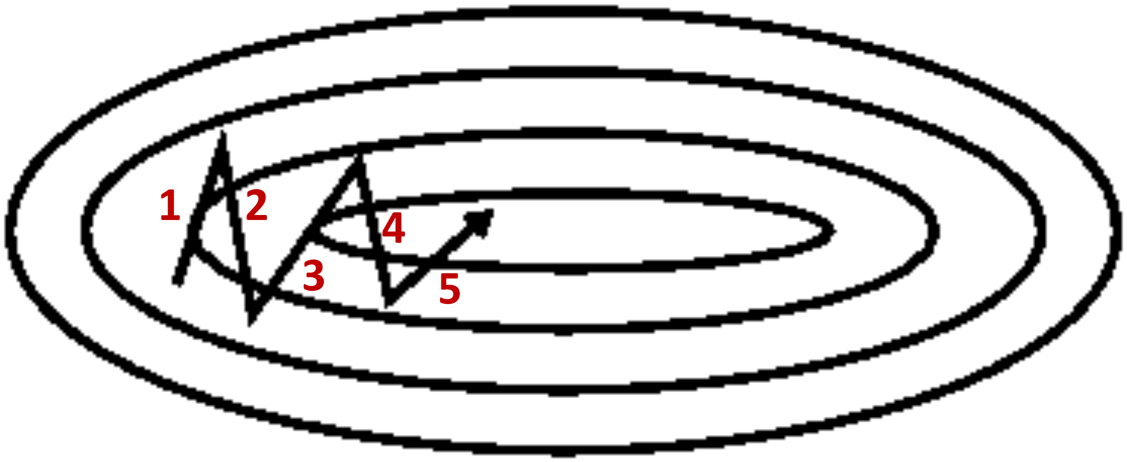
$$W_k = W_k - \eta_j \nabla_{W_k} Loss^T$$
$$\Delta W_k = \beta \Delta W_k - \eta_j \nabla_{W_k} Loss^T$$

- Until $Loss$ has converged

Still higher-order methods

- Momentum and Nestorov's method improve convergence by normalizing the *mean* of the derivatives
- More recent methods take this one step further by also considering their variance
  - RMS Prop
  - Adagrad
  - AdaDelta
  - ADAM: very popular in practice
  - …
- All roughly equivalent in performance

Smoothing the trajectory



Step	X component	Y component
1	1	+2.5
2	1	-3
3	3	+2.5
4	1	-2
5	2	1.5

- Simple gradient and acceleration methods still demonstrate oscillatory behavior in some directions
  - Depends on magic step size parameters
- Observation: Steps in “oscillatory” directions show large total movement
  - In the example, total motion in the vertical direction is much greater than in the horizontal direction
- Improvement: Dampen step size in directions with high motion
  - ***Second order term***

Normalizing steps by second moment

- In recent past
  - Total movement in *Y* component of updates is high
  - Movement in *X* components is lower
- Current update, modify usual gradient-based update:
  - Scale *down* Y component
  - Scale *up* X component
  - *According to their variation (and not just their average)*
- A variety of algorithms have been proposed on this premise
  - We will see a popular example

RMS Prop

- Notation:
  - Updates are ***by parameter***
  - Sum derivative of divergence w.r.t any individual parameter w is shown as $\partial_w D$
  - The ***squared*** derivative is $\partial_w^2 D = (\partial_w D)^2$
    - Short-hand notation represents the squared derivative, not the second derivative
  - The ***mean squared*** derivative is a running estimate of the average squared derivative. We will show this as $E[\partial_w^2 D]$
- Modified update rule: We want to
  - scale down updates with large mean squared derivatives
  - scale up updates with small mean squared derivatives

RMS Prop

- This is a variant on the *basic* mini-batch SGD algorithm

- **Procedure:**
  - Maintain a running estimate of the mean squared value of derivatives for each parameter
  - Scale update of the parameter by the *inverse* of the *root mean squared* derivative

$$E[\partial_w^2 D]_k = \gamma E[\partial_w^2 D]_{k-1} + (1-\gamma)(\partial_w^2 D)_k$$

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{E[\partial_w^2 D]_k + \epsilon}} \partial_w D$$

RMS Prop

- This is a variant on the *basic* mini-batch SGD algorithm

- **Procedure:**
  - Maintain a running estimate of the mean squared value of derivatives for each parameter
  - Scale update of the parameter by the *inverse* of the *root mean squared* derivative

$$E[\partial_w^2 D]_k = \gamma E[\partial_w^2 D]_{k-1} + (1-\gamma)(\partial_w^2 D)_k$$

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{E[\partial_w^2 D]_k + \epsilon}} \partial_w D$$

Note similarity to RPROP
The magnitude of the derivative is being normalized out

RMS Prop (updates are for each weight of each layer)

- Do:
  - Randomly shuffle inputs to change their order
  - Initialize: $k = 1$; for all weights w in all layers, $E[\partial_w^2 D]_k = 0$
  - For all $t = 1:B:T$ (incrementing in blocks of B inputs)
    - For all weights in all layers initialize $(\partial_w D)_k = 0$
    - For $b = 0:B-1$
      - Compute
        - » Output $\boldsymbol{Y}(\boldsymbol{X}_{t+b})$
        - » Compute gradient $\frac{\boldsymbol{dDiv}(\boldsymbol{Y}(\boldsymbol{X}_{t+b}),\boldsymbol{d}_{t+b})}{\boldsymbol{dw}}$
        - » Compute$(\partial_w D)_k \mathrel{+}= \frac{1}{B}\frac{\boldsymbol{dDiv}(\boldsymbol{Y}(\boldsymbol{X}_{t+b}),\boldsymbol{d}_{t+b})}{\boldsymbol{dw}}$
    - update:

$$\boldsymbol{E[\partial_w^2 D]_k = \gamma E[\partial_w^2 D]_{k-1} + (1-\gamma)(\partial_w^2 D)_k}$$

$$\boldsymbol{w_{k+1} = w_k - \frac{\eta}{\sqrt{E[\partial_w^2 D]_k + \epsilon}}\partial_w D}$$

    - $k = k + 1$
- Until $E(\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)}, \dots, \boldsymbol{W}^{(K)})$has converged

ADAM: RMSprop with momentum

- RMS prop only considers a second-moment normalized version of the current gradient
- ADAM utilizes a smoothed version of the *momentum-augmented* gradient
  - Considers both first and second moments
- **Procedure:**
  - Maintain a running estimate of the mean derivative for each parameter
  - Maintain a running estimate of the mean squared value of derivatives for each parameter
  - Scale update of the parameter by the *inverse* of the *root mean squared* derivative

$$m_k = \delta m_{k-1} + (1-\delta)(\partial_w D\ )_k$$

$$v_k = \gamma v_{k-1} + (1-\gamma)(\partial_w^2 D)_k$$

$$\widehat{m}_k = \frac{m_k}{1-\delta^k}, \qquad \hat{v}_k = \frac{v_k}{1-\gamma^k}$$

$$w_{k+1} = w_k - \frac{\eta}{\sqrt{\hat{v}_k + \epsilon}} \widehat{m}_k$$

ADAM: RMSprop with momentum

- RMS prop only considers a second-moment normalized version of the current gradient
- ADAM utilizes a smoothed version of the *momentum-augmented* gradient

- **Procedure:**
  - Maintain a running estimate of the mean derivative for each parameter
  - Maintain a running estimate of the mean squared value ... h parameter
  - Scale update of the parameter by the *inverse* of the ... derivative

Ensures that the δ and γ terms do not dominate in early iterations

$$m_k = \delta m_{k-1} + (1-\delta)(\partial_w D\ )_k$$

$$v_k = \gamma v_{k-1} + (1-\gamma)(\partial_w^2 D)_k$$

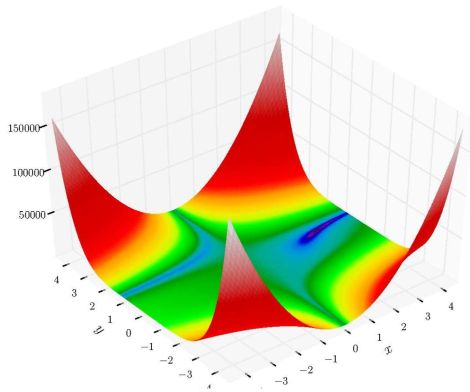
$$\boxed{\hat{m}_k = \frac{m_k}{1-\delta^k}, \qquad \hat{v}_k = \frac{v_k}{1-\gamma^k}}$$

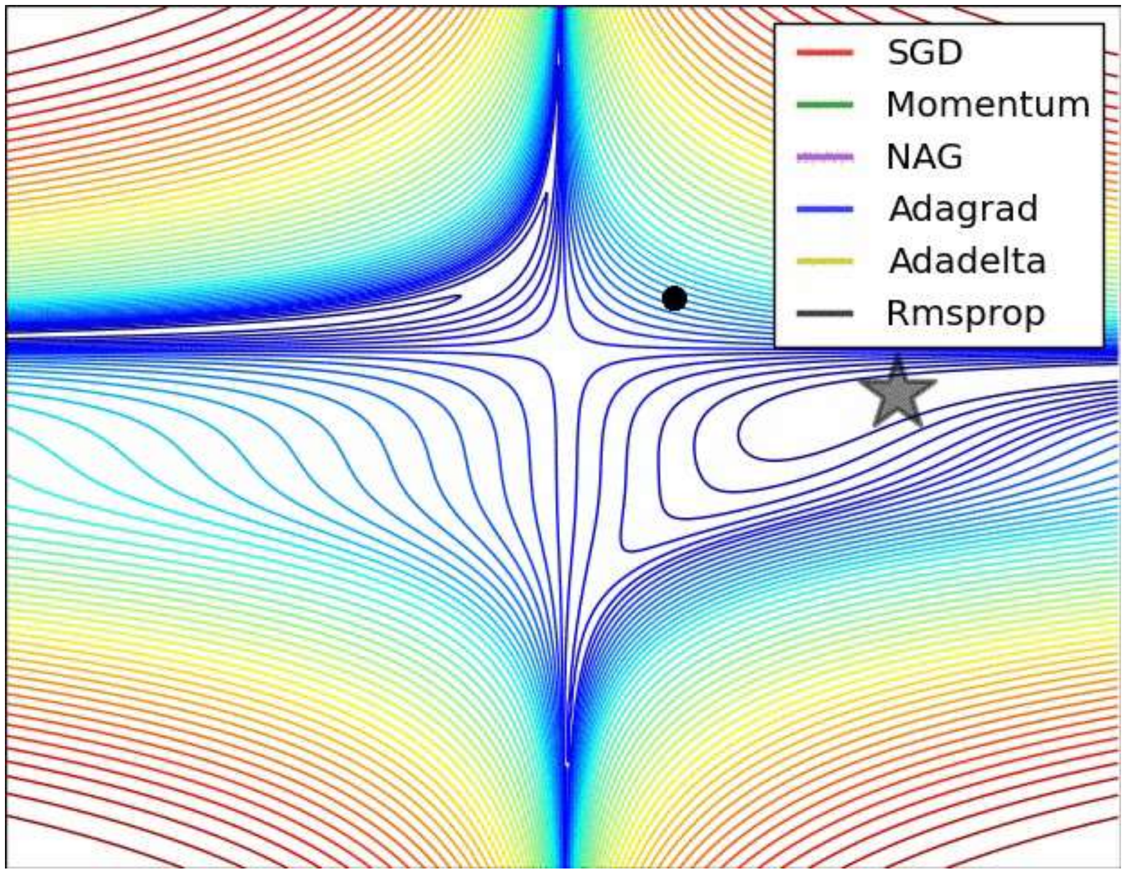
$$w_{k+1} = w_k - \frac{\eta}{\sqrt{\hat{v}_k + \epsilon}}\hat{m}_k$$

Other variants of the same theme

- Many:
  - Adagrad
  - AdaDelta
  - ADAM
  - AdaMax
  - …
- Generally no explicit learning rate to optimize
  - But come with other hyper parameters to be optimized
  - Typical params:
    - RMSProp: $\eta = 0.001, \gamma = 0.9$
    - ADAM: $\eta = 0.001, \delta = 0.9, \gamma = 0.999$

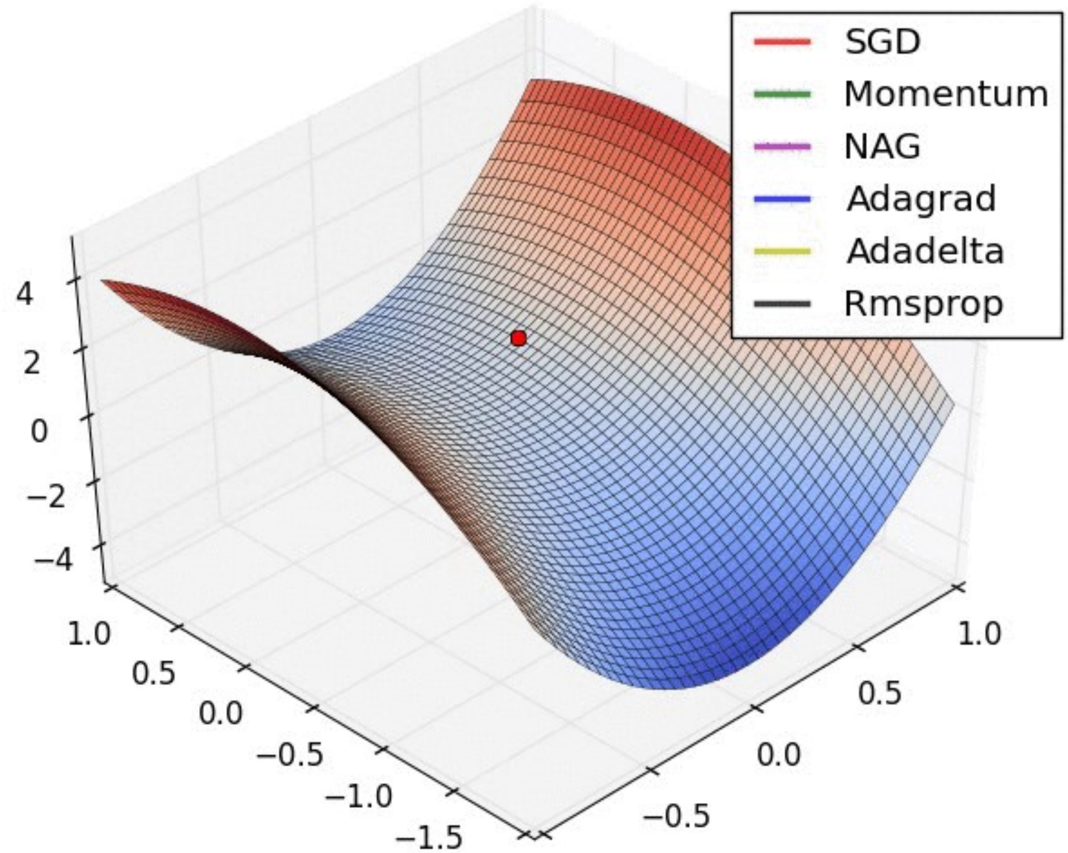
Visualizing the optimizers: Beale's Function





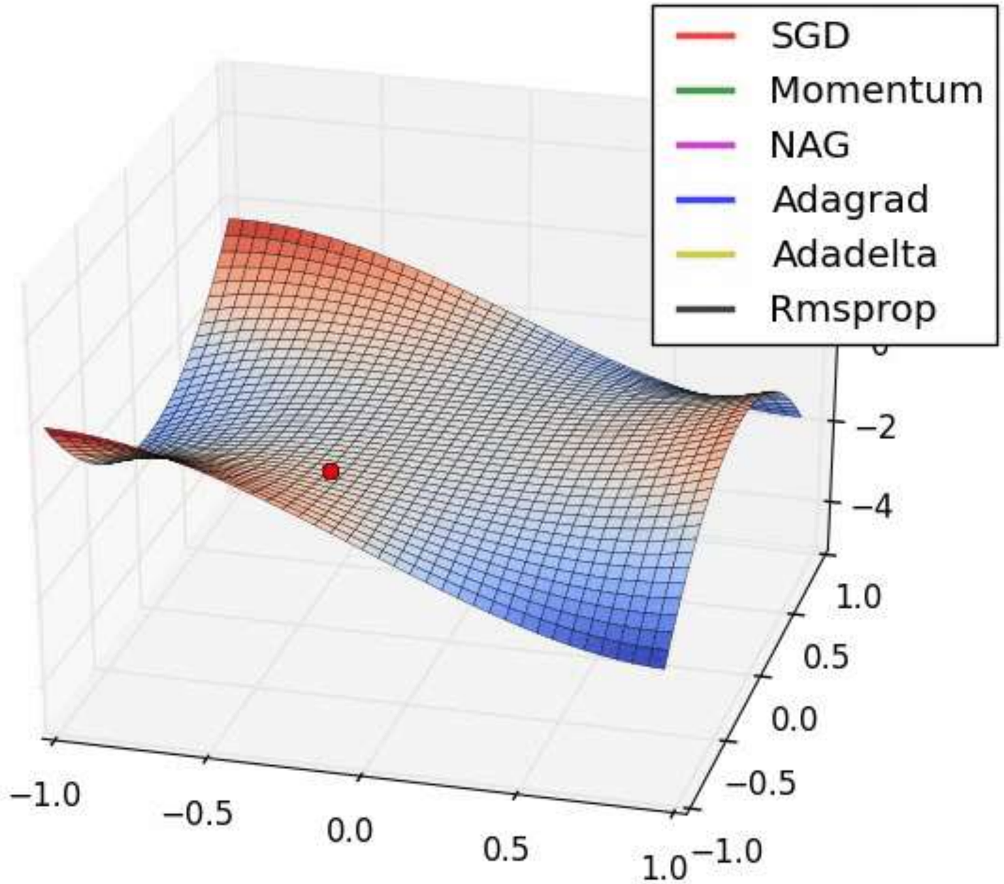
- http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Visualizing the optimizers: Long Valley



- http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html

Visualizing the optimizers: Saddle Point



- http://www.denizyuret.com/2015/03/alec-radfords-animations-for.html