Training Neural Networks
: Optimization (part2)

CMU 11-785 Introduction to Deep Learning, Fall 2020

- Lecture 7 -

TAVE Research DL001
Changdae Oh

2021. 02. 07

Topics

1. Stochastic Gradient Descent
2. The Variance of Loss & mini-batch updates
3. "Trend" Algorithms

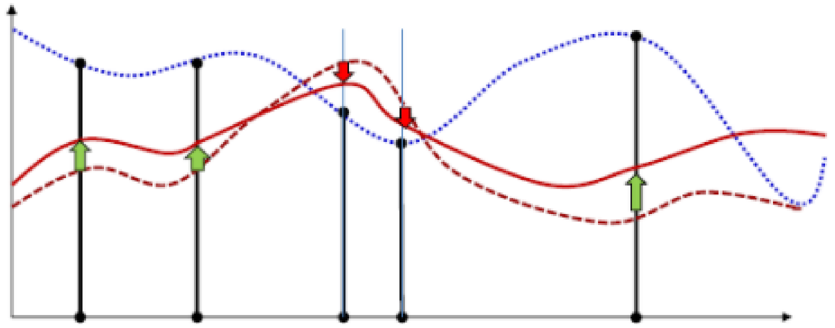
Topics

1. Stochastic Gradient Descent
2. The Variance of Loss & mini-batch updates
3. "Trend" Algorithms

Stochastic Gradient Descent

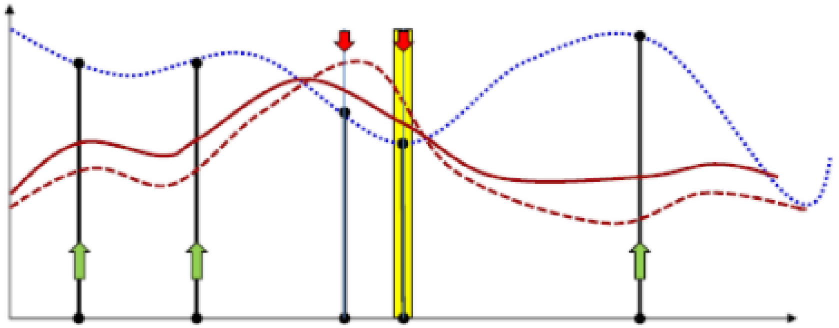
Problem with vanilla gradient descent

: Must process all training points before making a single adjustment (High cost)



"Batch" update

Alternative : Incremental update



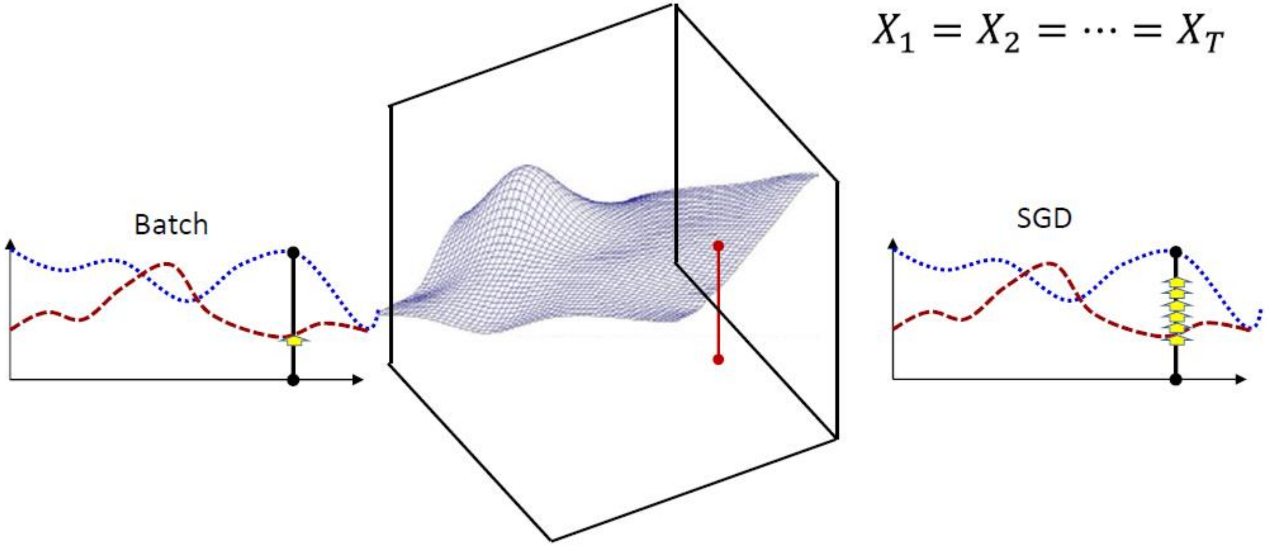
Accumulating updates of training instances

Randomly permute sample points !!

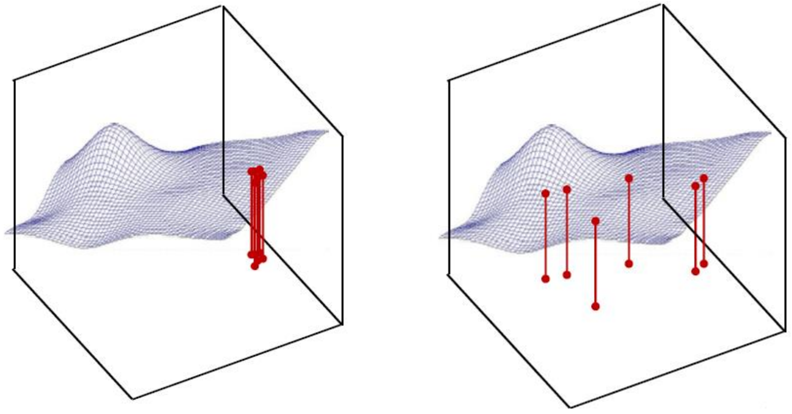
➡ More effective

Stochastic Gradient Descent

Why it works?



$$X_1 = X_2 = \cdots = X_T$$



As data get increasingly diverse,
the benefits of incremental updates decrease,
but do not entirely vanish

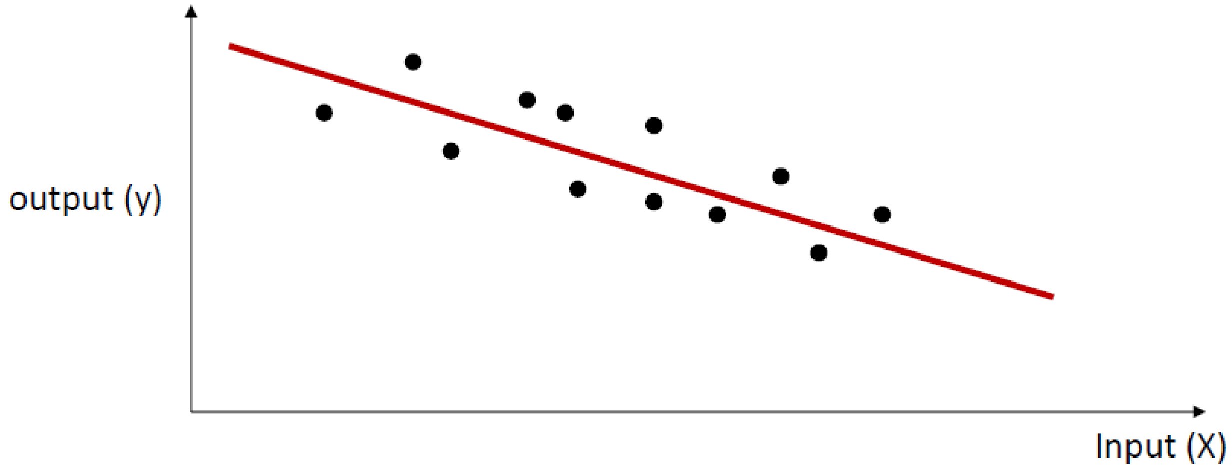
$$\boxed{\frac{dE}{dw_{i,j}^{(k)}} = \frac{1}{T}\sum_i \frac{dDiv(Y(X_i), d_i)}{dw_{i,j}^{(k)}} = \frac{dDiv(Y(X_i), d_i)}{dw_{i,j}^{(k)}}}$$

The final gradient points is simply the gradient for an individual instance

: a single update vs T updates (same computation)

Stochastic Gradient Descent

When does it work?



Shrink the learning rate with iterations to prevent never-ending updates

Sufficient conditions

1. Entire parameter space can be searched

$$\sum_k \eta_k = \infty$$

2. The steps shrink

$$\sum_k \eta_k^2 < \infty$$

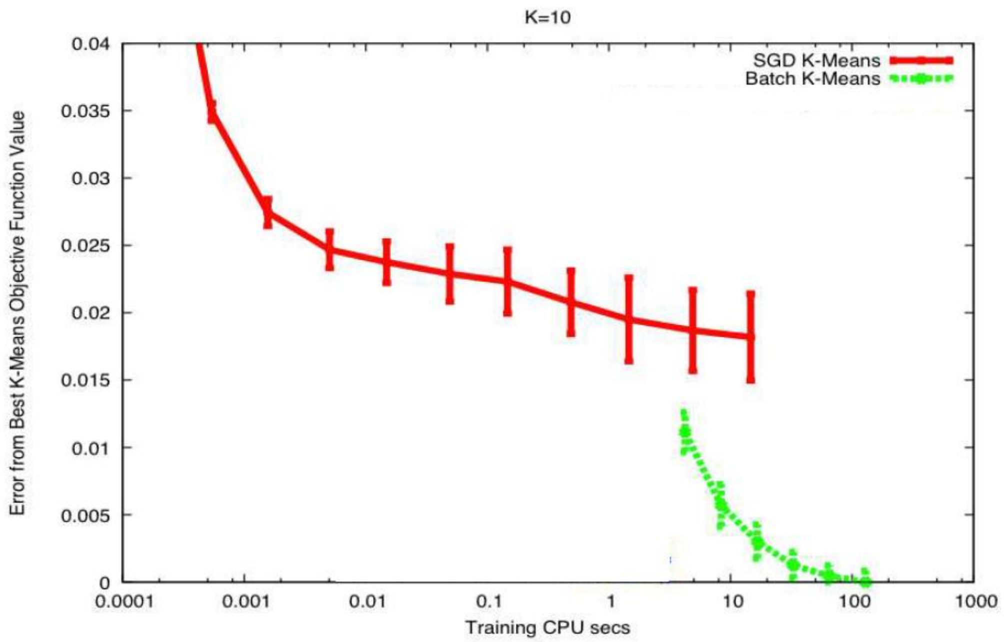
Optimal rate of shrinking the step size (in convex) : $\eta_k \propto \frac{1}{k}$

Stochastic Gradient Descent

Algorithm

- Given $(X_1, d_1), (X_2, d_2), \ldots, (X_T, d_T)$
- Initialize all weights $W_1, W_2, \ldots, W_K$; $j = 0$
- Do:
  - Randomly permute $(X_1, d_1), (X_2, d_2), \ldots, (X_T, d_T)$
  - For all $t = 1:T$
    - $j = j + 1$
    - For every layer k:
      - Compute $\nabla_{W_k} \boldsymbol{Div}(\boldsymbol{Y}_t, \boldsymbol{d}_t)$
      - Update

        $W_k = W_k - \eta_j \nabla_{W_k} \boldsymbol{Div}(\boldsymbol{Y}_t, \boldsymbol{d}_t)^T$
- Until $Loss$ has converged



Speed and Quality of Convergence

Topics

1. Stochastic Gradient Descent
2. The Variance of Loss & mini-batch updates
3. "Trend" Algorithms

The Variance of Loss & mini-batch updates

V(LOSS) of GD & SGD

In Batch GD

$$Loss(w) = \frac{1}{N}\sum_i div(f(x_i; w), d_i)$$

$$Var(Loss(w)) = \frac{1}{N^2} Var\left(\sum_i div(f(x_i; w), d_i)\right)$$

$$= \frac{1}{N^2}\sum_i Var(div(f(x_i; w), d_i))$$

$$= \frac{1}{N} \cdot Var(div(f(x_i; w), d_i))$$

$\Rightarrow$ more stable

In SGD

$$Loss(w) = div(f(x_i; w), d_i)$$

$$Var(Loss(w)) = Var(div(f(x_i; w), d_i))$$

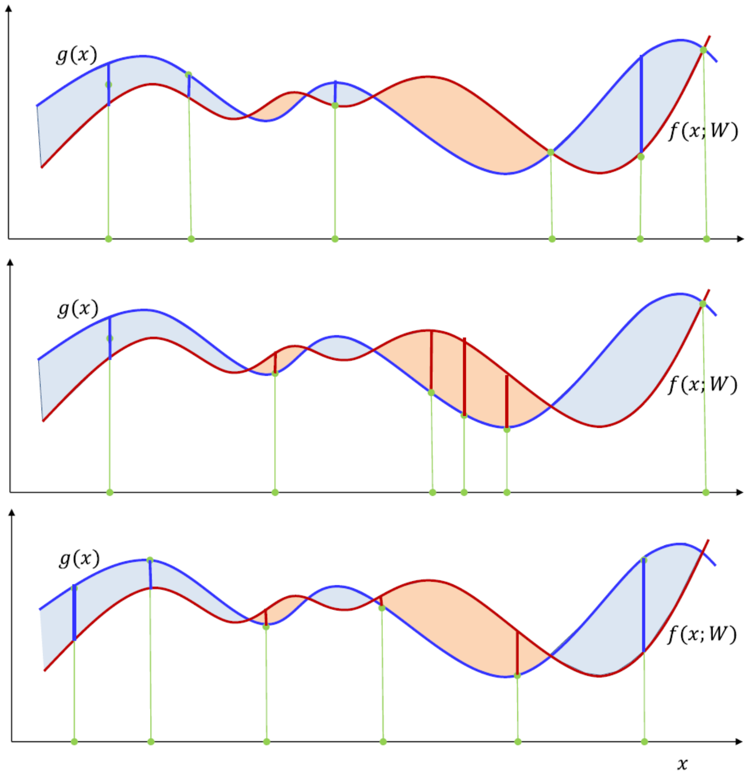
$\uparrow$

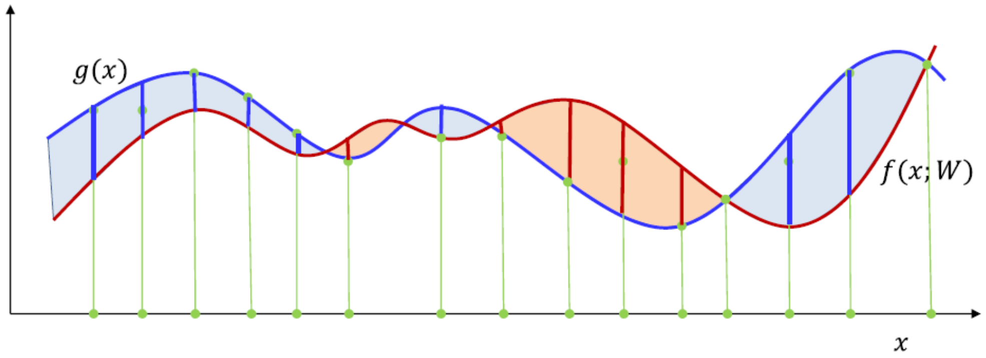
N times !!

The Variance of Loss & mini-batch updates

Explaining the Variance

The larger the variance of Loss(W),
the greater the likelihood that the W that minimizes the empirical risk (estimator of E[div])
will differ significantly from the W that minimizes the expected divergence





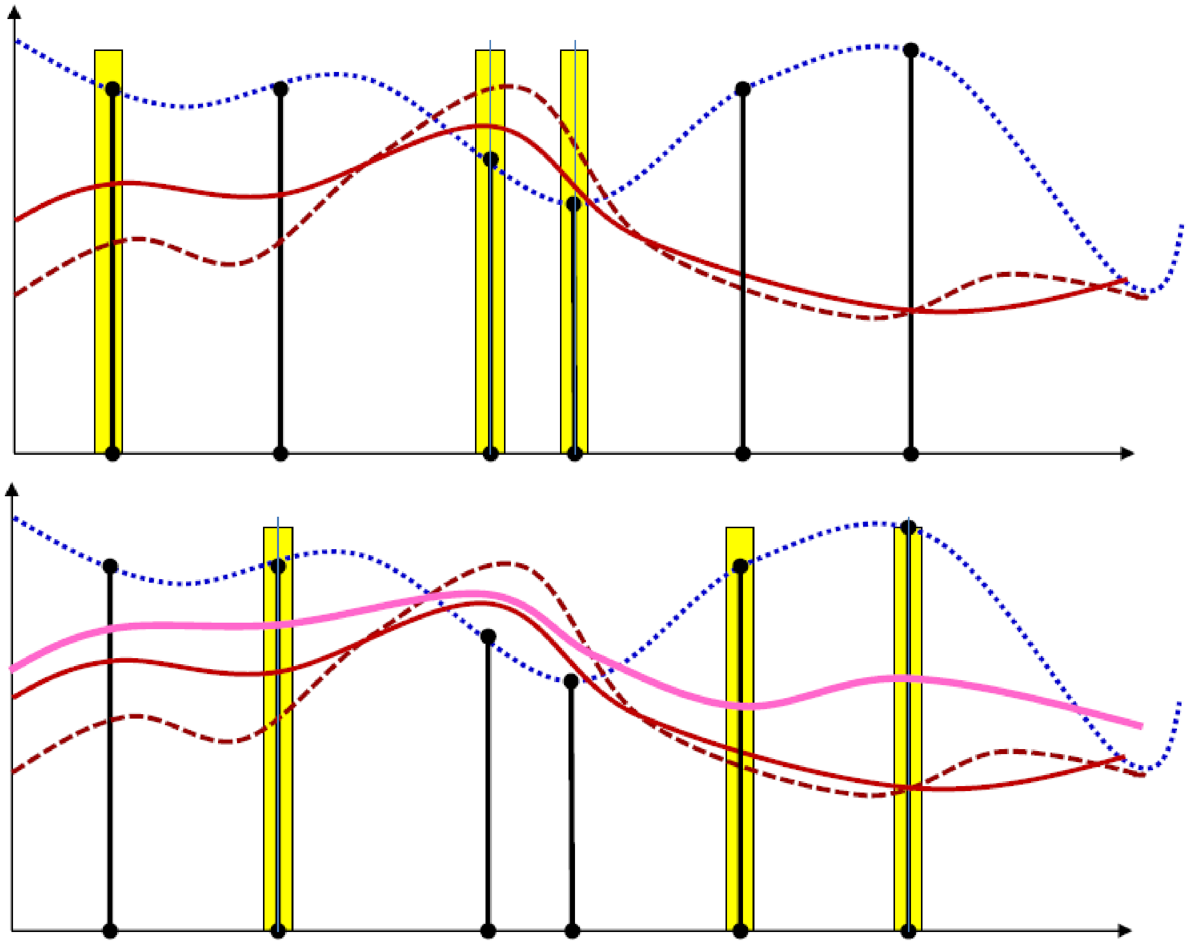
More Robust !

Loss is drastically changed with position of the few samples

The Variance of Loss & mini-batch updates

Mini-batch update

: adjust the function at a small, randomly chosen subset of points



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

$$W_k = W_k - \eta_j \Delta W_k$$

Shrinking step size

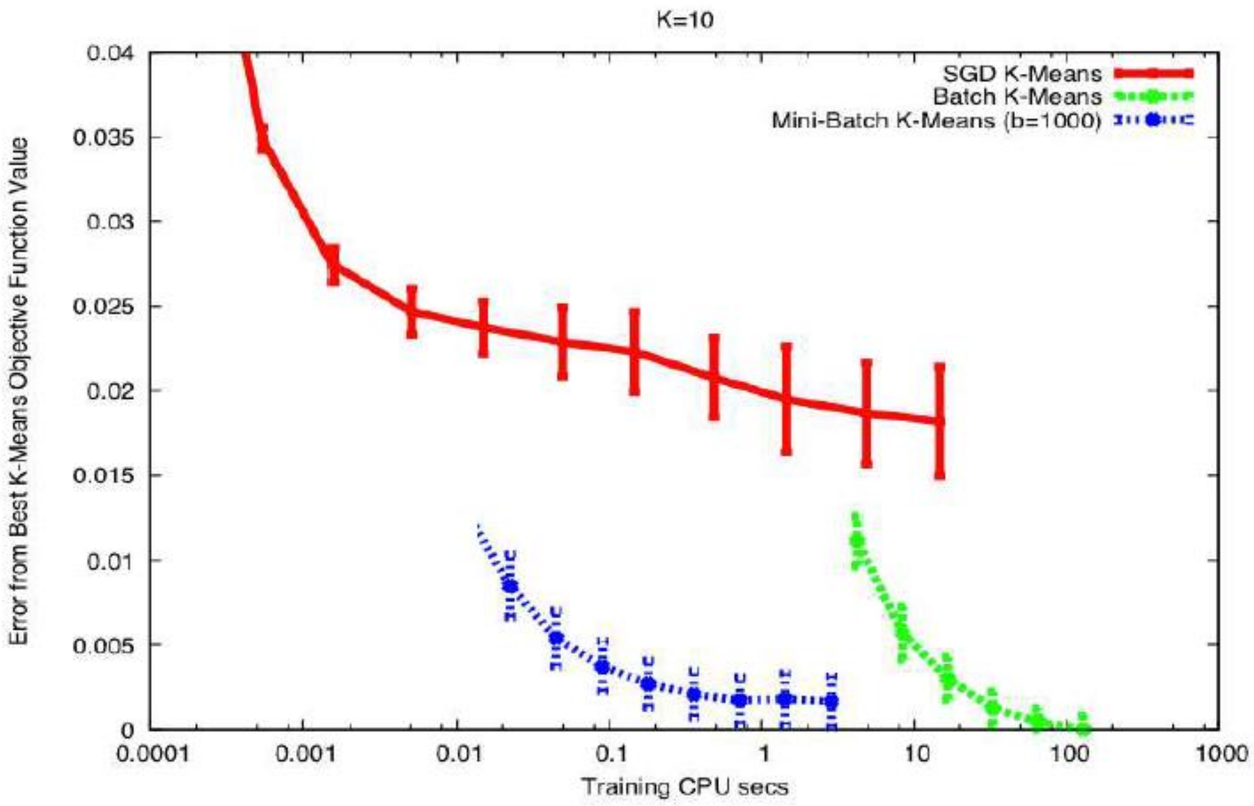
- Until Err has converged

The Variance of Loss & mini-batch updates

Mini-batch update

The variance of the minibatch loss

: V(LOSS(W)) = 1/B * V(div(fi, di))



Fast & Nice convergence

Topics

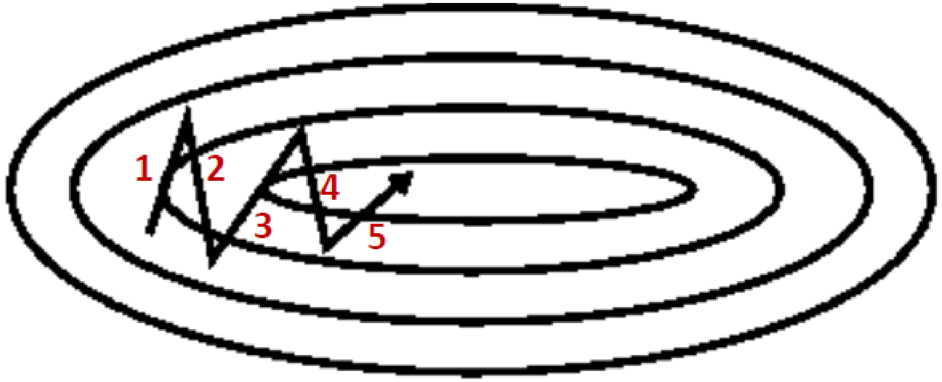
1. Stochastic Gradient Descent
2. The Variance of Loss & mini-batch updates
3. "Trend" Algorithms

"Trend" Algorithms

- Momentum & Nesterov's method improve convergence by normalizing the mean of the derivatives

- More recent methods take this one step further
  by considering their variance

Smoothing the trajectory !



"Trend" Algorithms

RMS Prop

- Scale down updates with large mean squared derivatives
- Scaled up updates with small mean squared derivatives

$\boxed{E[\partial_w^2 D]}$: Quantifying the volatility of trajectories !

Procedure

decaying rate

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

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

"Trend" Algorithms

ADAM

- RMS Prop with momentum
- <u>Considers both first and second mements</u>

Procedure

decaying rate

Momentum term

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

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

Adaptive updates

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

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