Machine Learning

Stochastic Optimization

Gradient Descent (cont'd)

- Training data $D = \{x_n, y_n\}_{n \in [N]}$
- Loss function $L(W; D)$; gradient $g(W) = \nabla_W L(W; D)$

GD Algorithm:

Initialize W_0

For $i = 1,2, \ldots$ (until termination criteria)

$$W_i = W_{i-1} - \mu\, g(W_{i-1})$$

- Appropriate selection of step size μ
- Loss $L(W; D)$ drops at each iteration.
- If $L(W; D)$ is convex, GD will converge to global solution.
- If $L(W; D)$ is non-convex, GD converges to arbitrary local minimum (depending on W_0)

Stochastic Gradient Descent (cont'd)

- Might not have access of all data at once
- Might be too expensive to compute gradient on all data at once
- Might want to avoid overfitting by not minimizing the training loss exactly
- Instead, estimate on a subset of the data, named batch or mini-batch
    - Correspond to gradient of loss on the same mini-batch

Stochastic Gradient Descent

- Training data $D = \{x_n, y_n\}_{n \in [N]}$

SGD Algorithm:

Initialize W_0

For $i = 1,2, \ldots$ (until termination criteria)

- $W_i = W_{i-1} - \mu g_i(W_{i-1})$

Return W_i

- where, for all i, D_i is a random subset (called a minibatch) of training data at step i and, for any W,

$$g_i(W) = \nabla_W L(W; D_i)$$

- is a random estimate of $g(W)$ on the random data of D_i.
- The method is called stochastic because it uses random gradient estimates.

SGD Convergence

$$g(W) = \nabla_W L(W; D);\ g_i(W) = \nabla_W L(W; D_i)\ \forall i$$

Theorem: For any W, the loss, and its gradient, evaluated on a minibatch is an unbiased estimate of the loss on the whole training set. That is,

$$E[g_i(W)] = g(W)$$

SGD Convergence (cont'd)

- The empirical loss, and its gradient, evaluated on a minibatch is an *unbiased* estimate of the empirical loss on the whole training set.
- This means that in expectation, we get the same result. The method is called stochastic because it uses random gradient estimates.
- The smaller the minibatch, the larger the variance of this random gradient is going to be in each step - the more stochastic the optimization algorithm becomes.

SGD Variants: Momentum SGD

Incorporates a fraction of the previous update to accelerate convergence.

Reduces oscillations and speeds up the descent:

Momentum SGD Algorithm:

Initialize W_0; $\Delta_0 = 0$

For $i = 1,2, \dots$ (until termination criteria)

- $W_i = W_{i-1} - \mu g_i + \alpha \Delta_{i-1}$
- $\Delta_i = W_i - W_{i-1}$

Return W_i

where, for brevity, $g_i = g_i(W_{i-1})$ and $g_i(W) = \nabla_W L(W; D_i)$. $\alpha \in [0,1]$ is the forgetting or decay factor.

We notice, for $i = 1,2, \dots, \Delta_i = -\mu g_i + \alpha \Delta_{i-1}$

Momentum SGD (cont'd)

W_0 arbitrary and $\Delta_0 = 0$; For $i \geq 1$: $W_i = W_{i-1} - \mu g_i + \alpha \Delta_{i-1}$; $\Delta_i = W_i - W_{i-1} = \Delta_i = -\mu g_i + \alpha \Delta_{i-1}$

That is,

- $\alpha\, \Delta_{i-1} = -\mu\, \alpha\, g_{i-1} + \alpha^2 \Delta_{i-2}$
- $= -\mu\, \alpha\, g_{i-1} + \alpha^2(-\mu g_{i-2} + \alpha \Delta_{i-3})$
- $= -\mu\, (\alpha\, g_{i-1} + \alpha^2 g_{i-2}) + \alpha^3 \Delta_{i-3}$
- …
- $= -\mu\left(\sum_{j \in \{1,\ldots,i-1\}} \alpha^{i-j} g_j\right)$

$$W_i = W_{i-1} - \mu g_i - \mu \left(\sum_{j \in \{1,\ldots,i-1\}} \alpha^{i-j} g_j \right) = W_{i-1} - \mu \left(\sum_{j \in \{1,\ldots,i\}} \alpha^{i-j} g_j \right)$$

Momentum SGD (cont'd)

- Stems from theory in the 1960's on solving functional equations.
- Stochastic gradient descent with momentum remembers the update at each iteration and determines the next update as a linear combination of the gradient and the previous update.
- Unlike in classical stochastic gradient descent, it tends to keep traveling in the same direction, preventing oscillations.
- Momentum has been used successfully by computer scientists in the training machine learning for several decades.

SGD Variants: Average

Records an average of parameters vector over time.

When iterations terminate, return average parameters

Momentum SGD Algorithm:

Initialize W_0; $\bar{W}_i = 0$

For $i = 1,2, \ldots$ (until termination criteria)

- $W_i = W_{i-1} - \mu g_i$
- $\bar{W}_i = \frac{i-1}{i} \bar{W}_{i-1} + \frac{1}{i} W_i$

Return $\bar{W}_i$

where, for brevity, $g_i = g_i(W_{i-1})$ and $g_i(W) = \nabla_W L(W; D_i)$.

Average SGD (cont'd)

- Stemming from work in late 1980s.
- Title: Merits of Averaged Stochastic Gradient Descent (ASGD)
- By averaging the parameter vectors over time, ASGD reduces the variance in the parameter updates which often leads to a reduction in the noise inherent in the stochastic gradient descent process.
- ASGD tends to have better convergence properties especially in the case of non-convex optimization problems. The averaging process can lead to a more stable convergence to a local minima.
- The averaging procedure makes ASGD more robust to the choice of learning rate and initial conditions compared to standard SGD. Possibly better generalization.

SGD Variants: AdaGrad

Records and sums the squares of the gradient over time.

Uses them to weight gradient per coordinate. Thus, the gradient is adaptive per parameter.

AdaGrad Algorithm:

Initialize W_0; $s_0 = 0$

For $i = 1,2, \ldots$ (until termination criteria)

- $s_i: [s_i]_j = [s_{i-1}]_j + [g_i]_j^2, j \in [d]$
- $W_i = W_{i-1} - \mu \, \mathrm{diag}(s_i)^{-\frac{1}{2}} g_i$

Return W_i

SGD Variants: AdaGrad

- AdaGrad (for adaptive gradient algorithm) was first published in 2011.
- Designed for convex problems but has been successful to non-convex optimization and ML.
- Individual learning rates for each parameter can handle the sparse and dense parts of the data differently.
- Adjusts the learning rate during training, which can be beneficial for dealing with problems of varying scale and curvature.
- Often improves convergence speed over standard SGD in settings where data is sparse and sparse parameters are more informative (e.g., NLP and image recognition).

SGD Variants: RMSProp

Modification of AdaGrad.

Again, learning rate adapted for each of the parameters.

Use running average of the magnitudes of recent gradients for that weight.

RMSProp SGD Algorithm:

Initialize W_0; $s_0 = 0$

For $i = 1,2, \ldots$ (until termination criteria)

- $s_i: [s_i]_j = \gamma\, [s_{i-1}]_j + (1-\gamma)[g_i]_j^2, j \in [d]$
- $W_i = W_{i-1} - \mu\, \mathrm{diag}(s_i)^{-\frac{1}{2}}\, g_i$

Return W_i

γ is the forgetting factor.

SGD Variants: RMSProp

- Introduced about 2012.
- RMSProp modifies the Adagrad algorithm to work better in non-convex optimization settings, making it more suitable for training deep neural networks.
- Unlike Adagrad's aggressive, monotonically decreasing learning rate, RMSProp employs a smoother decay that can adaptively change over time. Suitable for problems requiring long-term.

SGD Variants: Adam

RMSProp SGD + Momentum SGD, with bias correction.

Adam Algorithm:

Initialize $W_0;\ s_0 = 0;\ \Delta_0 = 0$

For $i = 1,2,\ldots$ (until termination criteria)

- $s_i: [s_i]_j = \gamma\,[s_{i-1}]_j + (1-\gamma)[g_i]_j^2, j \in [d]$
  - $s = s_i \frac{1}{1-\gamma^i}$
- $V_i = \alpha V_{i-1} + (1-\alpha) g_i$
  - $V = V_i \frac{1}{1-\alpha^i}$
- $W_i = W_{i-1} - \mu \left(\text{diag}(s)^{\frac{1}{2}} + \epsilon \boldsymbol{I}\right)^{-1} V$

Return W_i

Momentum SGD Algorithm:

Initialize $W_0;\ V_0 = 0$

For $i = 1,2,\ldots$ (until termination criteria)

- $V_i = \alpha V_{i-1} + g_i$
- $W_i = W_{i-1} - \mu V_i$
- Return W_i

RMSProp SGD Algorithm:

Initialize $W_0;\ s_0 = 0;\ \Delta_0 = 0$

For $i = 1,2,\ldots$ (until termination criteria)

- $s_i: [s_i]_j = \gamma\,[s_{i-1}]_j + (1-\gamma)[g_i]_j^2, j \in [d]$
- $W_i = W_{i-1} - \mu\,\text{diag}(s_i)^{-\frac{1}{2}} g_i$
- $\Delta_i = W_i - W_{i-1}$

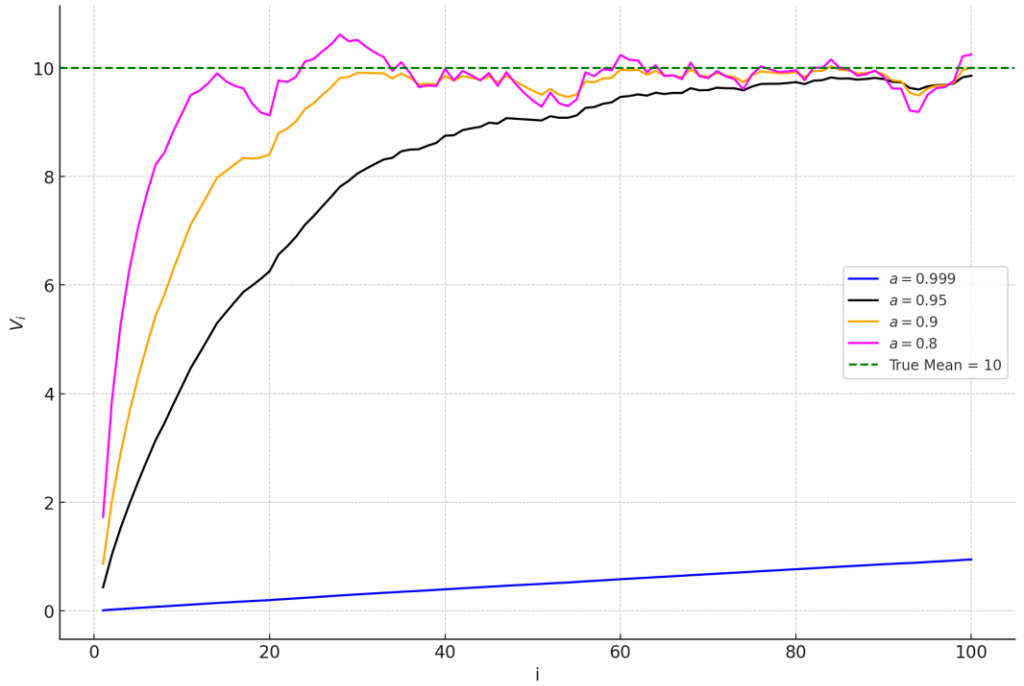
Return W_i

SGD Variants: Adam (cont'd)

- The moving averages
  - $\gamma\,[s_{i-1}]_j + (1-\gamma)[g_i]_j^2$
  - $\alpha V_{i-1} + (1-\alpha)g_i$
- are estimates of the 1st moment (the mean) and the second raw moment (the uncentered variance) of the gradient.
- These moving averages are initialized to 0, leading to moment estimates that are biased towards zero, especially during the initial iterations and small decay rates γ and α close to 1.

SGD Variants: Adam (cont'd)

- Moving average:
- $V_i = \alpha V_{i-1} + (1-\alpha) g_i$
- $= \alpha(\alpha V_{i-2} + (1-\alpha) g_{i-1}) + (1-\alpha) g_i$
- $= \alpha^2 V_{i-2} + (1-\alpha)\alpha\, g_{i-1} + (1-\alpha) g_i$
- $= \alpha^2(\alpha V_{i-3} + (1-\alpha) g_{i-2}) + (1-\alpha)\alpha\, g_{i-1} + (1-\alpha) g_i$
- $= \alpha^3 V_{i-3} + (1-\alpha)(\alpha^2 g_{i-2} + \alpha\, g_{i-1} + g_i)$
- $V_i = \alpha^i V_0 + (1-\alpha)\left(\sum_{j=\{1,..,i\}} \alpha^{i-j}\, g_i\right)$
- $V_i = (1-\alpha)\left(\sum_{j=\{1,..,i\}} \alpha^{i-j}\, g_i\right)$
- Exponentially diminishing gradient of older gradients

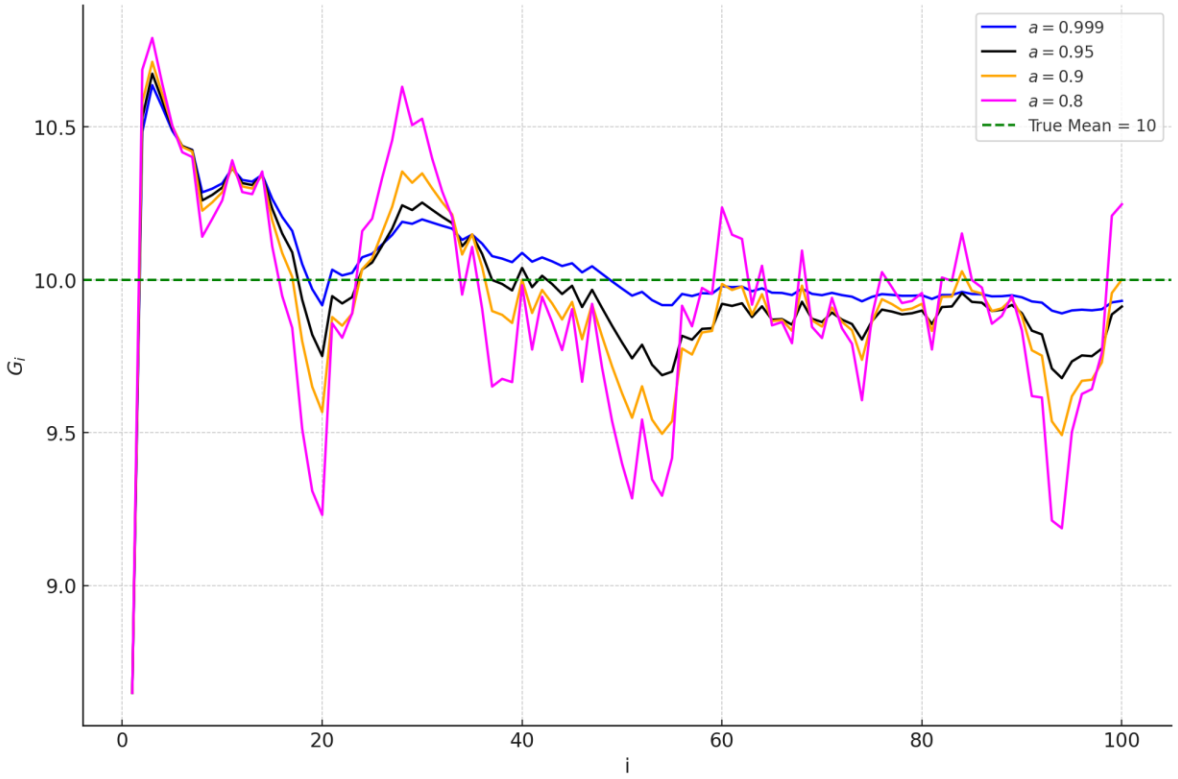


Trade-off between convergence speed and stability.

SGD Variants: Adam (cont'd)

- Bias correction

- $G_i = \frac{1}{1-\alpha^i}(1-\alpha)\left(\sum_{j=\{1,..,i\}} \alpha^{i-j}\, g_i\right)$



SGD Variants: Adam (cont'd)

- Adam (Adaptive Moment Estimation) combines the benefits of both Momentum SGD and RMSprop, making it a powerful and versatile optimizer.
- Utilizes moment estimates to adaptively change the learning rate, like RMSprop, while also incorporating momentum to accelerate convergence.
- Adam includes bias-correction mechanisms to account for the initialization of moment estimates at zero, providing more accurate estimations especially in the early stages of training.
- Handles noisy data and sparse gradients more effectively compared to Momentum SGD and RMSprop, which can be particularly useful in real-world scenarios with imperfect data.
- Often requires less parameter tuning to achieve good performance, making it a more user-friendly optimizer for a wide range of problems.