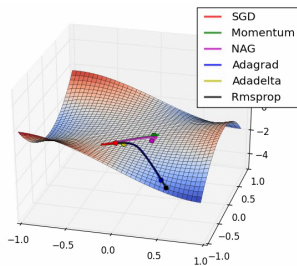


Optimization in Machine Learning

First order methods

Adam and friends

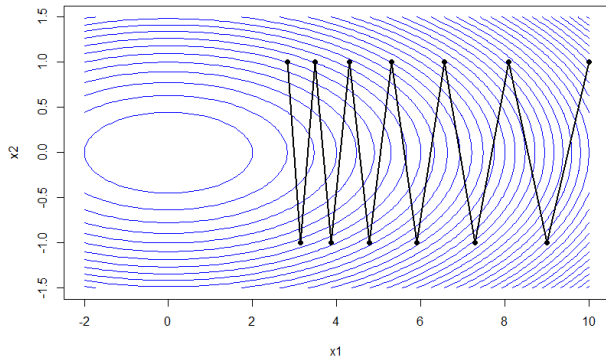
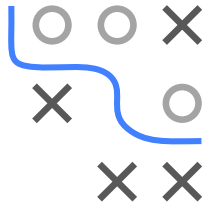


Learning goals

- Adaptive step sizes
- AdaGrad
- RMSProp
- Adam

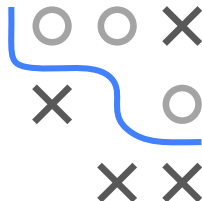
ADAPTIVE STEP SIZES

- Step size is probably the most important control parameter
- Has strong influence on performance
- Natural to use different step size for each input individually and automatically adapt them



ADAGRAD

- AdaGrad adapts step sizes by scaling them inversely proportional to square root of the sum of the past squared derivatives
 - Inputs with large derivatives get smaller step sizes
 - Inputs with small derivatives get larger step sizes
- Accumulation of squared gradients can result in premature small step sizes (Goodfellow et al., 2016)

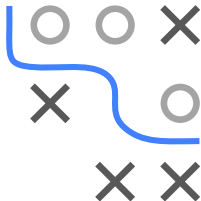


ADAGRAD / 2

Algorithm AdaGrad

- 1: **require** Global step size α
 - 2: **require** Initial parameter θ
 - 3: **require** Small constant β , perhaps 10^{-7} , for numerical stability
 - 4: **Initialize** gradient accumulation variable $\mathbf{r} = \mathbf{0}$
 - 5: **while** stopping criterion not met **do**
 - 6: Sample minibatch of m examples from the training set $\{\tilde{\mathbf{x}}^{(1)}, \dots, \tilde{\mathbf{x}}^{(m)}\}$
 - 7: Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(y^{(i)}, f(\tilde{\mathbf{x}}^{(i)} | \theta))$
 - 8: Accumulate squared gradient $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
 - 9: Compute update: $\nabla \theta = -\frac{\alpha}{\beta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ (operations element-wise)
 - 10: Apply update: $\theta \leftarrow \theta + \nabla \theta$
 - 11: **end while**
-

\odot : element-wise product (Hadamard)



RMSPROP

- Modification of AdaGrad
- Resolves AdaGrad's radically diminishing step sizes.
- Gradient accumulation is replaced by exponentially weighted moving average
- Theoretically, leads to performance gains in non-convex scenarios
- Empirically, RMSProp is a very effective optimization algorithm. Particularly, it is employed routinely by DL practitioners.



RMSPROP / 2

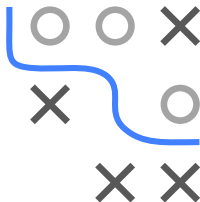
Algorithm RMSProp

- 1: **require** Global step size α and decay rate $\rho \in [0, 1)$
 - 2: **require** Initial parameter θ
 - 3: **require** Small constant β , perhaps 10^{-6} , for numerical stability
 - 4: Initialize gradient accumulation variable $\mathbf{r} = \mathbf{0}$
 - 5: **while** stopping criterion not met **do**
 - 6: Sample minibatch of m examples from the training set $\{\tilde{\mathbf{x}}^{(1)}, \dots, \tilde{\mathbf{x}}^{(m)}\}$
 - 7: Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(y^{(i)}, f(\tilde{\mathbf{x}}^{(i)} | \theta))$
 - 8: Accumulate squared gradient $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
 - 9: Compute update: $\nabla \theta = -\frac{\alpha}{\beta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$
 - 10: Apply update: $\theta \leftarrow \theta + \nabla \theta$
 - 11: **end while**
-



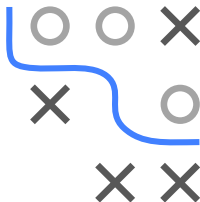
ADAM

- Adaptive Moment Estimation also has adaptive step sizes
- Uses the 1st and 2nd moments of gradients
 - Keeps an exponentially decaying average of past gradients (1st moment)
 - Like RMSProp, stores an exp-decaying avg of past squared gradients (2nd moment)
 - Can be seen as combo of RMSProp + momentum.



Algorithm Adam

- 1: **require** Global step size α (suggested default: 0.001)
 - 2: **require** Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$ (suggested defaults: 0.9 and 0.999 respectively)
 - 3: **require** Small constant β (suggested default 10^{-8})
 - 4: **require** Initial parameters θ
 - 5: Initialize time step $t = 0$
 - 6: Initialize 1st and 2nd moment variables $\mathbf{s}^{[0]} = 0, \mathbf{r}^{[0]} = 0$
 - 7: **while** stopping criterion not met **do**
 - 8: $t \leftarrow t + 1$
 - 9: Sample a minibatch of m examples from the training set $\{\tilde{\mathbf{x}}^{(1)}, \dots, \tilde{\mathbf{x}}^{(m)}\}$
 - 10: Compute gradient estimate: $\hat{\mathbf{g}}^{[t]} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(y^{(i)}, f(\tilde{\mathbf{x}}^{(i)} | \theta))$
 - 11: Update biased first moment estimate: $\mathbf{s}^{[t]} \leftarrow \rho_1 \mathbf{s}^{[t-1]} + (1 - \rho_1) \hat{\mathbf{g}}^{[t]}$
 - 12: Update biased second moment estimate: $\mathbf{r}^{[t]} \leftarrow \rho_2 \mathbf{r}^{[t-1]} + (1 - \rho_2) \hat{\mathbf{g}}^{[t]} \odot \hat{\mathbf{g}}^{[t]}$
 - 13: Correct bias in first moment: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}^{[t]}}{1 - \rho_1^t}$
 - 14: Correct bias in second moment: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}^{[t]}}{1 - \rho_2^t}$
 - 15: Compute update: $\nabla \theta = -\alpha \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}} + \beta}}$
 - 16: Apply update: $\theta \leftarrow \theta + \nabla \theta$
 - 17: **end while**
-



- Initializes moment variables \mathbf{s} and \mathbf{r} with zero \Rightarrow Bias towards zero

$$\mathbb{E}[\mathbf{s}^{[t]}] \neq \mathbb{E}[\hat{\mathbf{g}}^{[t]}] \quad \text{and} \quad \mathbb{E}[\mathbf{r}^{[t]}] \neq \mathbb{E}[\hat{\mathbf{g}}^{[t]} \odot \hat{\mathbf{g}}^{[t]}]$$

(\mathbb{E} calculated over minibatches)

- Indeed: Unrolling $\mathbf{s}^{[t]}$ yields

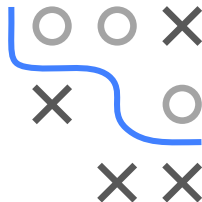
$$\mathbf{s}^{[0]} = 0$$

$$\mathbf{s}^{[1]} = \rho_1 \mathbf{s}^{[0]} + (1 - \rho_1) \hat{\mathbf{g}}^{[1]} = (1 - \rho_1) \hat{\mathbf{g}}^{[1]}$$

$$\mathbf{s}^{[2]} = \rho_1 \mathbf{s}^{[1]} + (1 - \rho_1) \hat{\mathbf{g}}^{[2]} = \rho_1 (1 - \rho_1) \hat{\mathbf{g}}^{[1]} + (1 - \rho_1) \hat{\mathbf{g}}^{[2]}$$

$$\mathbf{s}^{[3]} = \rho_1 \mathbf{s}^{[2]} + (1 - \rho_1) \hat{\mathbf{g}}^{[3]} = \rho_1^2 (1 - \rho_1) \hat{\mathbf{g}}^{[1]} + \rho_1 (1 - \rho_1) \hat{\mathbf{g}}^{[2]} + (1 - \rho_1) \hat{\mathbf{g}}^{[3]}$$

- Therefore: $\mathbf{s}^{[t]} = (1 - \rho_1) \sum_{i=1}^t \rho_1^{t-i} \hat{\mathbf{g}}^{[i]}$.
- **Note:** Contributions of past $\hat{\mathbf{g}}^{[i]}$ decreases rapidly



- We continue with

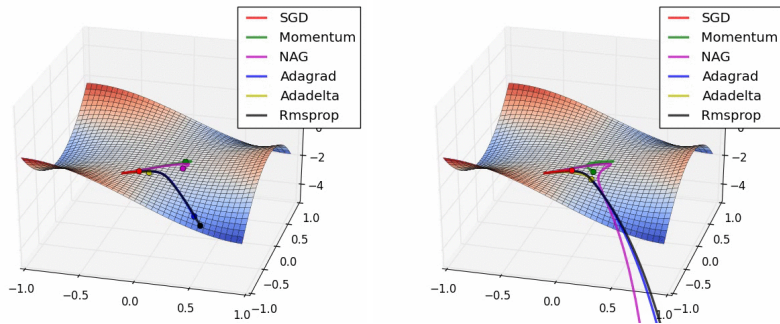
$$\begin{aligned}
 \mathbb{E}[\mathbf{s}^{[t]}] &= \mathbb{E}[(1 - \rho_1) \sum_{i=1}^t \rho_1^{t-i} \hat{\mathbf{g}}^{[i]}] \\
 &= \mathbb{E}[\hat{\mathbf{g}}^{[t]}](1 - \rho_1) \sum_{i=1}^t \rho_1^{t-i} + \zeta \\
 &= \mathbb{E}[\hat{\mathbf{g}}^{[t]}](1 - \rho_1^t) + \zeta,
 \end{aligned}$$

where we approximated $\hat{\mathbf{g}}^{[i]}$ by $\hat{\mathbf{g}}^{[t]}$. The resulting error is put in ζ and be kept small due to the exponential weights of past gradients.

- Therefore: $\mathbf{s}^{[t]}$ is a biased estimator of $\hat{\mathbf{g}}^{[t]}$
- But bias vanishes for $t \rightarrow \infty$ ($\rho_1^t \rightarrow 0$)
- Ignoring ζ , we correct for the bias by $\hat{\mathbf{s}}^{[t]} = \frac{\mathbf{s}^{[t]}}{(1 - \rho_1^t)}$
- Analogously: $\hat{\mathbf{r}}^{[t]} = \frac{\mathbf{r}^{[t]}}{(1 - \rho_2^t)}$



COMPARISON OF OPTIMIZERS: ANIMATION



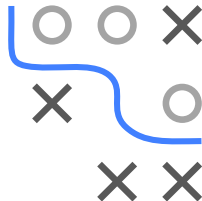
Credits: Dettmers (2015) and Radford

Comparison of SGD optimizers near saddle point.

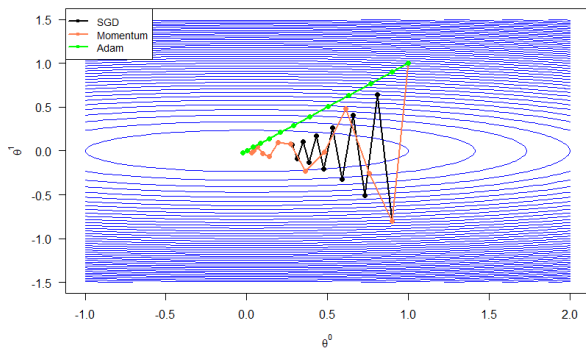
Left: After start. **Right:** Later.

All methods accelerate compared to vanilla SGD.

Best is RMSProp, then AdaGrad. (Adam is missing here.)



COMPARISON ON QUADRATIC FORM



SGD vs. SGD with Momentum vs. Adam on a quadratic form.