Table of Contents

1 Advanced First-Order Optimization Algorithms
   - Vanilla SGD
   - Momentum
   - Parameter Adaptive Learning Rate

2 Keras Primer
Table of Contents

1 Advanced First-Order Optimization Algorithms
   • Vanilla SGD
   • Momentum
   • Parameter Adaptive Learning Rate

2 Keras Primer
Vanilla SGD Pitfalls

- **Ill-conditioned problem:** When the loss changes significantly in one dimension more than it changes in others.

- **Noisy estimation of gradient:** Comes from the fact that we are working in mini-batches, but can also be due to inherent stochasticity in the model (e.g. dropout layer).

- *Examples in notebook*
Vanilla SGD Pitfalls

- **Local minima and saddle points**: Gradient is zero so we cannot descend in the direction of the negative gradient. Saddle points are common in high-dimensional optimization problems\(^1\).

\(^1\)Dauphin et al., “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization”.
SGD + Momentum

- A simple and effective heuristic that mitigates the mentioned pitfalls is the introduction of a momentum term to the gradient descent step.
- The new update rule is:

\[ v_{t+1} = \mu v_t - \eta \nabla w f(w_t) \]
\[ w_{t+1} = w_t + v_{t+1} \]

- We decay the current velocity ("friction"), and add the current (negative) gradient to it. Typically, \( \mu = 0.9 \).
Nesterov momentum

- Instead of adding the gradient at the current point to the velocity, we add the gradient at the point in which we would have landed had we taken a step with the current velocity.
- The Nesterov update rule is:

\[
v_{t+1} = \mu v_t - \eta \nabla_w f(w_t + \mu v_t)
\]

\[
w_{t+1} = w_t + v_{t+1}
\]
Nesterov Momentum

If we set \( \tilde{w}_t = w_t + \mu v_t \), and then change variables, we get a different formulation for this update:

\[
\begin{align*}
    v_{t+1} &= \mu v_t - \eta \nabla_w f(\tilde{w}_t) \\
    \tilde{w}_{t+1} &= \tilde{w}_t - \mu v_t + (1 + \mu) v_{t+1}
\end{align*}
\]

What do we gain from the new formulation?
Ill-conditioning is still a problem

- Momentum can mitigate the problem of an ill-conditioned objective to some degree, but it may still oscillate and take a long time to converge.
- We can address this issue by taking smaller steps in volatile dimensions, and larger steps in stable ones.
- How can we do this automatically?
AdaGrad

- The Adaptive Gradient (AdaGrad) algorithm scales each parameter’s learning rate by a value representing the volatility of the gradient w.r.t it.
- The update rule is:

\[
g_{t+1} = g_t + (\nabla w f(w_t))^2
\]

\[
w_{t+1} = w_t - \frac{\eta}{\sqrt{g_{t+1}} + \epsilon} \odot \nabla w f(w_t)
\]

- What might be a problem with this algorithm over time?
RMSProp

- In order to prevent the learning rate from decreasing too much over time, the Root Mean Square Propagation algorithm (RMSProp) decays the accumulated sum of squares.

- The update rule is (typically, $\gamma = 0.9$):

$$g_{t+1} = \gamma g_t + (1 - \gamma)(\nabla_w f(w_t))^2$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{g_{t+1}} + \epsilon} \odot \nabla_w f(w_t)$$
Adam

- **Adaptive Moment Estimation** (Adam) was published by Kingma and Ba in 2015, and aims to incorporate both momentum and adaptive learning rate into a single algorithm.

- A rough idea for the update rule is (typically, $\beta_1 = 0.9, \beta_2 = 0.999$):

  \[
  m_{t+1} = \beta_1 m_t + (1 - \beta_1) \nabla_w f(w_t) \\
  v_{t+1} = \beta_2 v_t + (1 - \beta_2) (\nabla_w f(w_t))^2 \\
  w_{t+1} = w_t - \frac{\eta}{\sqrt{v_{t+1} + \epsilon}} \odot m_{t+1}
  \]

- But there is a problem here. What happens early in the training process?
The solution is to scale the moments’ estimates in order to correct the zero-bias (a well known issue with exponential averaging).

The final update rule is (typically, $\beta_1 = 0.9$, $\beta_2 = 0.999$):

\[
\begin{align*}
  m_{t+1} &= \beta_1 m_t + (1 - \beta_1) \nabla_w f(w_t) \\
  \tilde{m}_{t+1} &= m_{t+1} \frac{1}{1 - \beta_1^t} \\
  v_{t+1} &= \beta_2 v_t + (1 - \beta_2)(\nabla_w f(w_t))^2 \\
  \tilde{v}_{t+1} &= v_{t+1} \frac{1}{1 - \beta_2^t} \\
  w_{t+1} &= w_t - \frac{\eta}{\sqrt{\tilde{v}_{t+1}} + \epsilon} \odot \tilde{m}_{t+1}
\end{align*}
\]
Adam

- Empirically, Adam performs well on a wide range of deep learning task.
- Most of recent years’ papers use either Adam or SGD+Nesterov for optimization.
# Table of Contents

1. Advanced First-Order Optimization Algorithms
   - Vanilla SGD
   - Momentum
   - Parameter Adaptive Learning Rate

2. Keras Primer
Open your notebooks :)