Adam (Adaptive Moment Estimation)

Siddhant Chaudhary

CMI, October 2021

Moving Averages

- Suppose $\{a_1, a_2, a_3, ...\}$ is some sequence. For each time step t, we want to determine the *moving average* of this sequence.

Moving Averages

- Suppose $\{a_1, a_2, a_3, ...\}$ is some sequence. For each time step t, we want to determine the *moving average* of this sequence.
- Let β be some positive number. We will call β the *decaying factor*.

Moving Averages

- Suppose $\{a_1, a_2, a_3, ...\}$ is some sequence. For each time step t, we want to determine the *moving average* of this sequence.
- Let β be some positive number. We will call β the *decaying factor*.
- Define the sequence A_t by the following recursive formula.

$$A_1 = a_1$$
$$A_t = \beta A_{t-1} + (1 - \beta) a_t$$

Moving Averages

- Suppose $\{a_1, a_2, a_3, ...\}$ is some sequence. For each time step t, we want to determine the *moving average* of this sequence.
- Let β be some positive number. We will call β the *decaying factor*.
- Define the sequence A_t by the following recursive formula.

$$A_1 = a_1$$
$$A_t = \beta A_{t-1} + (1 - \beta) a_t$$

- This is also called an *exponential decay moving average*; very old terms of the sequence are given exponentially small weight.

Adam: Adaptive Moment Estimation

- *Adam* is an optimized version of *minibatch gradient descent*.

Adam: Adaptive Moment Estimation

- *Adam* is an optimized version of *minibatch gradient descent*.
- Uses *adaptive step sizes* to do gradient descent; very useful for training neural networks.

Adam: Adaptive Moment Estimation

- *Adam* is an optimized version of *minibatch gradient descent*.
- Uses *adaptive step sizes* to do gradient descent; very useful for training neural networks.
- Combines the ideas of *momentum* and *adaptive weights*.

Adam: Adaptive Moment Estimation

- *Adam* is an optimized version of *minibatch gradient descent*.
- Uses *adaptive step sizes* to do gradient descent; very useful for training neural networks.
- Combines the ideas of *momentum* and *adaptive weights*.
- Adam has become the default method for many neural network packages these days!

Adam: Adaptive Moment Estimation

- *Adam* is an optimized version of *minibatch gradient descent*.
- Uses *adaptive step sizes* to do gradient descent; very useful for training neural networks.
- Combines the ideas of *momentum* and *adaptive weights*.
- Adam has become the default method for many neural network packages these days!
- Easy to implement and very efficient, and magnitudes of parameter updates are invariant to scaling of the gradient.

The Algorithm

- Let $f(\theta)$ be the cost function we want to optimize, where the parameters are θ.

The Algorithm

- Let $f(\theta)$ be the cost function we want to optimize, where the parameters are θ.
- f will be *stochastic* if we are doing minibatch/stochastic gradient descent. At time step t, we will assume that the function is f_t.

The Algorithm

- Let $f(\theta)$ be the cost function we want to optimize, where the parameters are θ.
- f will be *stochastic* if we are doing minibatch/stochastic gradient descent. At time step t, we will assume that the function is f_t.
- θ_0 : Our initial parameter vector.

The Algorithm

- Let $f(\theta)$ be the cost function we want to optimize, where the parameters are θ.
- f will be *stochastic* if we are doing minibatch/stochastic gradient descent. At time step t, we will assume that the function is f_t.
- θ_0 : Our initial parameter vector.
- We set $m_0 = 0$ and $v_0 = 0$ (these are vectors of the same dimension as that of the gradients of f). Also, $t = 0$ (the initial time step).

The Algorithm

- Let $f(\theta)$ be the cost function we want to optimize, where the parameters are θ.
- f will be *stochastic* if we are doing minibatch/stochastic gradient descent. At time step t, we will assume that the function is f_t.
- θ_0 : Our initial parameter vector.
- We set $m_0 = 0$ and $v_0 = 0$ (these are vectors of the same dimension as that of the gradients of f). Also, $t = 0$ (the initial time step).
- At time t, we set the following.

$$m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

(contd.)

- Here, g_t is the gradient at time step t, i.e $g_t = \nabla_\theta f_t(\theta_{t-1})$ and the quantity $g_t^2 = g_t \odot g_t$ denotes the coordinate-wise product.

(contd.)

- Here, g_t is the gradient at time step t, i.e $g_t = \nabla_\theta f_t(\theta_{t-1})$ and the quantity $g_t^2 = g_t \odot g_t$ denotes the coordinate-wise product.
- So, m_t and v_t respectively are estimates of first and second moments of g_t respectively.

(contd.)

- Here, g_t is the gradient at time step t, i.e $g_t = \nabla_\theta f_t(\theta_{t-1})$ and the quantity $g_t^2 = g_t \odot g_t$ denotes the coordinate-wise product.
- So, m_t and v_t respectively are estimates of first and second moments of g_t respectively.
- But there's a problem: initialization to 0 leads to a *bias* towards zero, i.e the quantities m_t and v_t will initially be small in magnitude.

(contd.)

- Here, g_t is the gradient at time step t, i.e $g_t = \nabla_\theta f_t(\theta_{t-1})$ and the quantity $g_t^2 = g_t \odot g_t$ denotes the coordinate-wise product.
- So, m_t and v_t respectively are estimates of first and second moments of g_t respectively.
- But there's a problem: initialization to 0 leads to a *bias* towards zero, i.e the quantities m_t and v_t will initially be small in magnitude.
- Not a problem, as there are bias corrected estimates (we will see how these estimates come about).

$$\hat{m}_t \leftarrow \frac{m_t}{(1-\beta_1^t)}$$
$$\hat{v}_t \leftarrow \frac{v_t}{(1-\beta_2^t)}$$

(contd.)

- Finally, we do the parameter update.

$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

(contd.)

- Finally, we do the parameter update.

$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- Note that the step sizes are different for each coordinate!

(contd.)

- Finally, we do the parameter update.

$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- Note that the step sizes are different for each coordinate!
- Adam uses ideas from two *techniques*: *momentum* and *adadelta*.

(contd.)

- Finally, we do the parameter update.

$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- Note that the step sizes are different for each coordinate!
- Adam uses ideas from two *techniques*: *momentum* and *adadelta*.
- *Momentum* ensures that when we are doing stochastic GD, by taking the whole history of gradients into account (moving averages), the current update will not jump around.

(contd.)

- Finally, we do the parameter update.

$$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

- Note that the step sizes are different for each coordinate!
- Adam uses ideas from two *techniques*: *momentum* and *adadelta*.
- *Momentum* ensures that when we are doing stochastic GD, by taking the whole history of gradients into account (moving averages), the current update will not jump around.
- *Adadelta* does the following: if the magnitude of gradient is large, we want to take small steps; if the magnitude is small, we want to take large steps.

Pseudocode

Input: α and $\beta_1, \beta_2 \in [0, 1)$.

Required: $f(\theta)$ (objective) and θ_0 (initial parameter)

$m_0 \leftarrow 0$

$v_0 \leftarrow 0$

$t \leftarrow 0$

while (some convergence critrion on θ_t) **do**

  $t \leftarrow t + 1$

  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$

  $m_t \leftarrow \beta_1 m_{t-1} + (1 - \beta_1) g_t$

  $v_t \leftarrow \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

  $\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$

  $\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$

  $\theta_t \leftarrow \theta_{t-1} - \alpha \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$

end while