An overview of gradient descent optimization algorithm

Presenter: Yeu-Chern Harn
Overview

• Intro to gradient descent
• Challenges
• Gradient descent optimization algorithms
• Additional strategies for optimizing

This overview is mostly based on This blog, it also has a paper version
Intro to gradient descent

- Gradient descent is a way to minimize an objective function $J(w)$ parameterized by a model's parameters $w$.

- It updates the parameters in the opposite direction of the gradient of the objective function w.r.t. to the parameters ($\nabla_w J(w)$).
Intro to gradient descent

• The *learning rate* $\eta$ determines the size of the steps we take to reach a (local) minimum.

![Diagram showing large and small learning rates](image)

- Large learning rate: Overshooting.
- Small learning rate: Many iterations until convergence and trapping in local minima.
Intro to gradient descent – Three variants

• Batch gradient descent
  • $w = w - \eta \cdot \nabla J(w)$
  • The gradient is computed from the entire training dataset
  • Pros: Accurate.
  • Cons: If the data size is very large, 1) slow, 2) cannot fit into memory
Intro to gradient descent – Three variants

• Stochastic gradient descent (SGD)
  • \[ w = w - \eta \cdot \nabla_w J(w; x^{(i)}, y^{(i)}) \]
  • Compute gradient from just one sample and update.
  • Pros: fast
  • Cons: keep overshooting and hard to converge (can be solved by decreasing learning rate)
Intro to gradient descent – Three variants

• Mini-batch gradient descent
  • \( w = w - \eta \cdot \nabla_w J(w; x^{(i:i+n)}, y^{(i:i+n)}) \)
  • Compute gradient and update for every mini-batch of n samples.
    • Reduce variance, stable update.
    • Compute faster. (Make use of hardware/software strength).
  • This is the algorithm for training a neural network.
Intro to gradient descent – Challenges

Choosing a proper learning rate and learning schedules could be difficult.
Intro to gradient descent – Challenges

• The same learning rate applies to all parameter update.

• When minimizing highly non-convex error functions common for neural networks, it is easy to get trapped in their numerous suboptimal local minima or saddle point.
Gradient descent optimization algorithms

• Some algorithms that are widely used by the deep learning community
  • Momentum
  • Nesterov accelerate gradient
  • Adagrad
  • Adadelta
  • Rmsprop
  • Adam
Gradient descent optimization algorithms - Momentum

• Motivation: SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another

• Formulation:
  • \( v_t = \gamma v_{t-1} + \eta \nabla_w J(w) \)
  • \( w = w - v_t \)
• Momentum term \( \gamma \) usually set to .9
Gradient descent optimization algorithms - Momentum

• Adding momentum:

  - Without momentum
  - With momentum

• With Momentum update, the parameter vector will build up velocity in any direction that has consistent gradient.
Gradient descent optimization algorithms - Nesterov accelerate gradient

• An improved momentum algorithm
• Idea: Better estimate gradient.
  • We know that we will use our momentum term ($\gamma v_{t-1}$) to move the parameters.
  • Computing $w - \gamma v_{t-1}$ gives us a rough idea where our parameters are going to be
  • Computing the gradient w.r.t. the approximate future position of our parameters
Gradient descent optimization algorithms - Nesterov accelerate gradient

- Formulation:
  - $v_t = \gamma v_{t-1} + \eta \nabla_w J (w - \gamma v_{t-1})$
  - $w = w - v_t$

- Momentum term $\gamma$ usually set to .9

- Bengio et. al.[1] show it significantly increased the performance of RNNs on a number of tasks

Gradient descent optimization algorithms - Adagrad

• Idea: Have different step size for every parameters.
• Set $w_i$ as the parameter at index $i$ in the parameter vector $w$
• Set $g_{t,i}$ to be the gradient of the objective function w.r.t. to the parameter $w_i$ at time step $t$, $g_{t,i}=\nabla_w J(w_i)$
• Formulation

$$w_{t+1,i} = w_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \varepsilon}} * g_{t,i}$$

• $G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix where each diagonal element $(i, i)$ is the sum of the squares of the gradients w.r.t. $w_i$ up to time step $t$
• $\varepsilon$ is the smooth term (usually 1e-8)
Gradient descent optimization algorithms - Adagrad

- Formulation in vector form:
  \[
  w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t+\varepsilon}} g_t
  \]

- Pros: Reduce the need to tune the learning rate. Usually set \( \eta \) to 0.01.

- Cons: accumulation of the squared gradients. The learning rate eventually becomes very small.

- It performs well on DistBelief (Distributed FCN)[1] and on GloVe[2] compared to SGD.

Gradient descent optimization algorithms - Adadelta

• Idea: Reduce Adagrad’s monotonically decreasing learning rate -> Running average of gradients

• First update:
  • $E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g^2_t$ ($\gamma$ usually set to 0.9)
  • $w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
  • Let $\Delta w_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$
Gradient descent optimization algorithms - Adadelta

• Second update: Unit should match.
  • When we update $\Delta w$ to $w$, their unit should match.
  • The unit in SGD is
    
    \[
    \text{units of } \Delta x \propto \text{units of } g \propto \frac{\partial f}{\partial x} \propto \frac{1}{\text{units of } x}
    \]
  
  • Newton method have the correct unit.
    \[
    \Delta x \propto H^{-1} g \propto \frac{\partial f}{\partial x} \propto \frac{\partial f}{\partial^2 f} \propto \text{units of } x
    \]
  
  • With some manipulation we have
    
    \[
    \Delta x = \frac{\partial f}{\partial x} \Rightarrow \frac{1}{\partial^2 f} = \frac{\Delta x}{\partial f / \partial x}
    \]
  
  • This indicate the a good step size for the gradient.
Gradient descent optimization algorithms - Adadelta

• Since we already have an good approximation for the gradient, we only need one for the $\Delta x$

• Apply the same trick (running average)
  • $E[\Delta w^2]_t = \gamma E[\Delta w^2]_{t-1} + (1 - \gamma)\Delta w^2_t$

• Combined with the first update, the formulation is
  • $\Delta w_t = -\frac{\sqrt{E[\Delta w^2]_t + \varepsilon}}{\sqrt{E[g^2]_t + \varepsilon}} \; g_t$
  • $w_{t+1} = w_t + \Delta w_t$

• In [1], it has better performance on a 2-layer FCN, compared to SGD with momentum and adagrad.

Gradient descent optimization algorithms - RMSprop

• Idea: Also reduce Adagrad’s monotonically decreasing learning rate -> Running average of gradients.

• Formulation:
  • \( E[g^2]_t = \gamma E[g^2]_{t-1} + (1 - \gamma) g^2_t \) (\( \gamma \) usually set to 0.9)
  • \( w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_t} + \varepsilon} g_t \) (\( \eta \) usually set to 0.001)

• Unpublished. But Karpathy uses it to optimize his char-rnn model.
Gradient descent optimization algorithms - Adam

• Idea: Estimate the first moment and the second moment of gradients to do the update.

• Let
  • $m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$ (first moment estimate)
  • $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ (second moment estimate)

• Formulation:
  • $w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \varepsilon}} m_t$
Gradient descent optimization algorithms - Adam

• They found estimations of $m_t, v_t$, because initialized as zero, are biased toward zero.

• Derivation (same for $m_t$)
  • $v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$ can be written as
    $$v_t = (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} g_t^2$$
  • $E[v_t] = E \left[ (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} \cdot g_i^2 \right]$
    $$= E[g_t^2] \cdot (1 - \beta_2) \sum_{i=1}^{t} \beta_2^{t-i} + \zeta$$
    $$= E[g_t^2] \cdot (1 - \beta_2^t) + \zeta$$
Gradient descent optimization algorithms - Adam

• Formulation:
  • $\hat{m}_t = \frac{m_t}{1-\beta_1^t}$
  • $\hat{v}_t = \frac{v_t}{1-\beta_2^t}$
  • $w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t+\epsilon}} \hat{m}_t$
  • Usually $\beta_1=.9$, $\beta_2=.999$, $\epsilon=10^{-8}, \eta=10^{-3}$
  • In [1], it has better performance on a FCN and CNN, compared to AdaGrad, RMSProp, SGD with Momentum and Adam.

[1] ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION
Conclusion on all these methods

contours of a loss surface and time evolution of different optimization algorithms

A visualization of a saddle point in the optimization landscape
Conclusion on all these methods

• Which to use
  • Data is sparse => should use adaptive learning rate methods.
  • Among all adaptive learning rate methods, Adam seems performs the best; though there is no strong evidence show Adam is always the best.
  • The author observed that SGD usually achieves to find a minimum, but it might take significantly longer time. => If care about optimization speed, should use adaptive learning rate methods or SGD with momentum.
Additional strategies for optimizing

• Shuffle the data. Generally, we want to avoid providing the training examples in a meaningful order to our model as this may bias the optimization algorithm.

• Batch normalization.
  • People usually pre-processing data to have zero mean and unit variance.
  • With multi-layer neural net, the new input from the layer might not have this property.
  • Batch normalization reestablishes these normalizations for the input and changes are back-propagated through the operation as well.
  • Applied in Residual network.
Additional strategies for optimizing

• Early stopping.
  • You should thus always monitor error on a validation set

• Curriculum Learning.
  • For some cases we supply the training samples in a meaningful order (order by samples’ difficulties) may actually lead to improved performance and better convergence [1].
  • But when training a LSTM model with this approach actually harm the performance. They found mixing ordinary samples with at least one difficulties helps [2].

• Gradient noise.
  • [3] found adding Gaussian noise to the gradient makes networks more robust to poor initialization.