An overview of gradient descent optimization algorithms

Pandu Raharja

Siemens Corporate Technology (CT RDA BAM SMR-DE)
Technische Universität München
Ludwig-Maximillians-Universität München

May 24, 2017

- Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv preprint arXiv:1609.04747 (2016).
- Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." Proceedings of the 3rd International Conference on Learning Representations (ICLR). 2015

Optimization problem in machine learning

Machine learning is an optimization problem.
Sometimes it's convex:

- Support Vector Machine: $\min_{w} \|w\| \; s.t. \; y_n(w^T x_n) \leq 1, \forall n$
- Linear Regression with L1-Regularization: $\min_{x} \frac{1}{n} \|Xw - Y\| + \lambda \|w\|_1$

Often times it's non-convex:

- **Neural networks**

"Hardness" of non-convex optimization

- No unique global minimum guaranteed
- Hard to model asymptotic behaviour

(First order) Strategies in Non-Convex Optimization

- Gradient descent variants:
  - Batch gradient descent
  - Stochastic gradient descent
  - Mini-batch gradient descent
- Gradient descent optimizations:
  - Momentum
  - Nesterov accelerated gradient
  - Adagrad
  - RMSprop
  - Adadelta
  - Adam

Batch Gradient Descent (vanilla)

Compute the gradient for the entire training dataset:

$$\theta \leftarrow \theta - \eta . \nabla_{\theta} J(\theta)$$

- Slow
- Intractable for datasets larger than memory capacity
- Redundant gradient computations for each parameter update

Batch gradient descent is a **first-order but not stochastic method**.

First-order stochastic method

Define general minimization problem:

$$\min_\theta \{J(\theta) := \frac{1}{m}\sum_{i=1}^{m} l(\theta; x_i, y_i) + \lambda r(\theta)\}$$

- θ – network parameters to learn
- $l(\theta; x, y)$ – loss function
- $r(\theta)$ – regularization function

First-order stochastic method

In deep learning, we usually have:

- Many network parameters ($\dim(\theta) >> 10^8$) – computing Hessian is expensive
- Many training examples ($m >> 10^6$) – computing full objective function per iteration is expensive

Thus, a gradient optimization method should be **first order stochastic**:

- **First order** – update based on objective value and gradient only
- **Stochastic** – update based on subset of training examples

There are several second-order methods developed (e.g. Martens, 2010, ICML & LM-BFGS), but it's not within the scope of presentation.

Stochastic Gradient Descent (SGD)

$$\theta \leftarrow \theta - \eta . \nabla_\theta \boldsymbol{J}(\boldsymbol{\theta}; \boldsymbol{x}^{(i)}; \boldsymbol{y}^{(i)})$$

- Frequent gradient updates on higher variance data
- Variance-induced fluctuation exposes new local minima
- Slowly decreasing learning rate shown to show convergence behaviour as vanila GD.

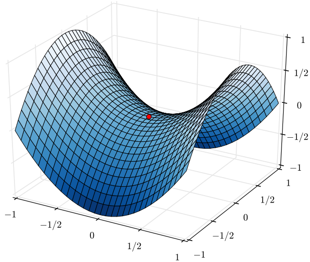
Mini-batch Gradient Descent

$$\theta \leftarrow \theta - \eta . \nabla_\theta J(\theta; \boldsymbol{x}^{(i:i+n)}; \boldsymbol{y}^{(i:i+n)})$$

- Update with n training samples at a time
- Reduced variance leading to more stable convergence behaviour

Challenges of non-optimized GD methods

- How to properly select learning rate?
- How to embed apropriate learning rate scheduling, i.e. how and when to adjust the learning rate?
- Features have very different frequency and yet are updated with same learning rate
- Suboptimal local minima might pose a risk. Moreover, sattle points are notoriously hard due to zero gradients.



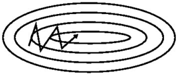
Momentum SGD

$$v_t = \gamma \boldsymbol{v_{t-1}} + \eta . \nabla \theta J(\theta)$$

$$\theta \leftarrow \theta - v_t$$

- Accelerates SGD in relevant direction:
  - Updates increased for dimensions with same direction to gradient's
  - Decreased for dimensions with opposite direction to gradient's
- Faster convergence and reduced oscillation
- Useful for landscape of unequal gradient (e.g. ravine around local minimum)

(a) SGD without momentum



(b) SGD with momentum

Nesterov Accelerated Gradient (NAG)

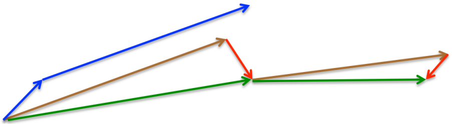
- Momentum does not anticipate gradient change in the near future
- Adjust the momentum by looking at the future:
  1. **First** make a big jump in the direction of the previous accumulated gradient
  2. **Then** measure the gradient where you end up and make correction

$$v_t = \gamma v_{t-1} + \eta . \nabla \theta J(\boldsymbol{\theta - \gamma v_{t-1}})$$

$$\theta \leftarrow \theta - v_t$$

- Embeds the response of future gradient change
- Reduces oscillation

Nesterov Accelerated Gradient (NAG)



brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum

Adaptive Gradient (AdaGrad)

- So far, NAG manages to adapt the rate to slope of error function.
- We'd like to adapt the rate to how often a parameter is updated – important so that sparse representation could also be learned

Idea: normalize update of parameter θ_i to the **magnitude of update** for θ_i so far

$$\theta_{t,i} \leftarrow \theta_{t-1,i} - \eta . \frac{\nabla_{\theta_i J_t(\theta)}}{\sqrt{\sum_{t'=1}^{t} \nabla_{\theta_i} J_{t'}(\theta)^2 + \epsilon}}$$

Surpresses frequently updated parameters and enchances rarely updated parameters

Adaptive Gradient (AdaGrad)

$$\theta_{t,i} \leftarrow \theta_{t-1,i} - \eta . \frac{\nabla_{\theta_i J_t(\theta)}}{\sqrt{\sum_{t'=1}^{t} \nabla_{\theta_i} J_{t'}(\theta)^2 + \epsilon}}$$

- **Main problem:** learning rate keeps falling due to denominator

AdaDelta

- Restrict the accumulated past gradients to some fixed size w
- **Problem:** storing and updating last w gradients for each θ_i is not very efficient
- **Idea:** recursively define running average of gradient $g := \nabla_\theta J(\theta)$:

$$E[g^2]_t = \gamma.E[g^2]_{t-1} + (1-\gamma).g_t^2$$

The parameter update at time t is thus defined as:

$$\Delta\theta_t := -\frac{\eta}{\sqrt{\boldsymbol{E[g^2]_t + \epsilon}}} g_t = -\frac{\eta}{\boldsymbol{RMS[g]_t}} g_t$$

$$\theta_{t,i} \leftarrow \theta_{t-1,i} + \Delta\theta_t$$

Root Mean Square Propagation (RMSProp)

Independently proposed by Hinton at the same time of AdaDelta:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t,i} \leftarrow \theta_{t-1,i} - \frac{0.001}{RMS[g]_t} g_t$$

Adaptive Moment Estimation (ADAM)

Combine the advantages of:

- **AdaGrad**, which works well with sparse gradients
- **RMSProp**, which works well with non-stationary setting

Idea:

- Maintain exponential of gradients and its square
- Update proportional to $\frac{\textit{average gradient}}{\textit{average squared gradient}}$

Adaptive Moment Estimation (ADAM)

Keeps exponential decaying average of past gradients m_t and momentum v_t (the *first* and *second* moment of g):

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

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

Problem: Since m_t and v_t are initialized as zero vectors, they are biased towards zero, especially in initial time steps.
Solution: Bias correction of m_t and v_t

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Adaptive Moment Estimation (ADAM)

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

with

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \text{(first moment correction)}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \qquad \text{(second moment correction)}$$

and

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \qquad \text{(first moment estimate)}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \qquad \text{(second moment estimate)}$$

ADAM: Experiments

Logistic regression on MNIST and IMDB Bag of Words

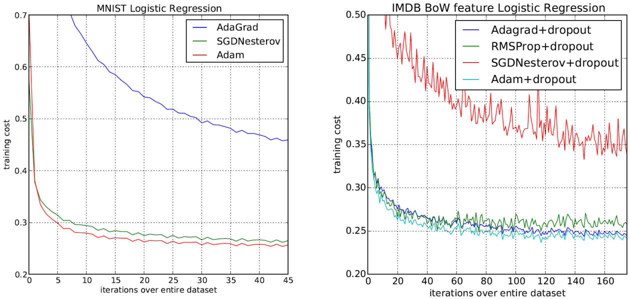
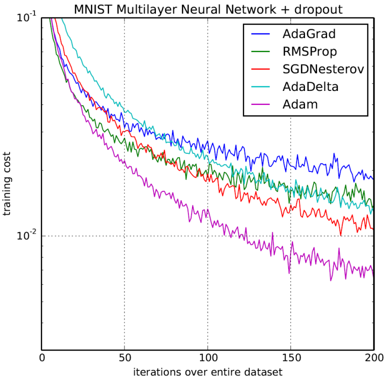


Figure 1: Logistic regression training negative log likelihood on MNIST images and IMDB movie reviews with 10,000 bag-of-words (BoW) feature vectors.

ADAM: Experiments

NN + dropout on MNIST



ADAM: Experiments

CNN on CIFAR10

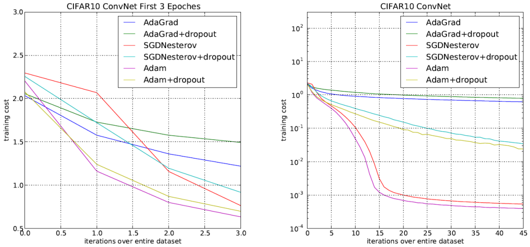


Figure 3: Convolutional neural networks training cost. (left) Training cost for the first three epochs. (right) Training cost over 45 epochs. CIFAR-10 with c64-c64-c128-1000 architecture.

ADAM: Extension (AdaMax)

Recall ADAM's update rule:

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

- We could generalize the L_2 norm into L_p norm for $p \to \infty$
- This surpisingly stabilizes and simplifies the algorithm

AdaMax

$$\theta_t \leftarrow \theta_{t-1} - \frac{\eta}{(\boldsymbol{\hat{v}_t + \epsilon})^{\boldsymbol{1/p}}} \hat{m}_t$$

with

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \text{(first moment bias correction)}$$

$$\hat{v}_t = \frac{v_t}{1 - \boldsymbol{\beta_2^{pt}}} \qquad \text{(p-th moment bias correction)}$$

and

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \qquad \text{(first moment estimate)}$$

$$v_t = \beta_2 v_{t-1} + \boldsymbol{(1 - \beta_2^p) g_t^p} \qquad \text{(p-th moment estimate)}$$

AdaMax

Recall the definition of L_p norm:

$$\|x\|_p := (x_1^p + \cdots + x_n^p)^{1/p}$$

For $p \to \infty$, this becomes maximum function. Hence we can rewrite the $(\hat{\boldsymbol{v}}_{\boldsymbol{t}} + \epsilon)^{\mathbf{1/p}}$-part:

$$\hat{v_t}^{1/p} = (\frac{v_t}{1 - \beta_2^{pt}})^{1/p} = (\frac{\beta_2 v_{t-1} + (1 - \beta_2^p) g_t^p}{1 - \beta_2^{pt}})^{1/p}$$

$$\overset{\text{inf. p}}{=} (\beta_2 v_{t-1} + g_t^p)^{1/p} \overset{\text{inf. p}}{=} max\{\beta_2 u_{t-1}, |g_{t-1}|\}$$

AdaMax

$$\theta_t \leftarrow \theta_{t-1} - \frac{\eta}{\boldsymbol{u_t}} \hat{m}_t$$

with

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \text{(first moment bias correction)}$$

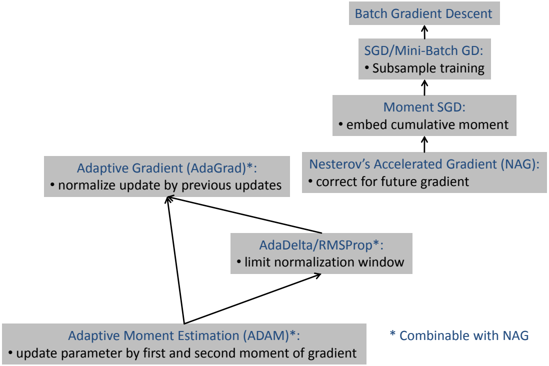
$$\hat{u}_t = max\{\beta_2 u_{t-1}, |g_{t-1}|\} \qquad \text{(∞-th moment etimate)}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \qquad \text{(first moment estimate)}$$

Additional Strategies

- **Shuffling** and **curriculum learning**
- **Batch norm**
- **Early stopping**
- **Gradient noise**

Methods evolution



Example

http://imgur.com/a/Hqolp

References

- Ruder, Sebastian. "An overview of gradient descent optimization algorithms." arXiv preprint arXiv:1609.04747 (2016).
- Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic optimization." Proceedings of the 3rd International Conference on Learning Representations (ICLR). 2015.
- Martens, James. "Deep learning via Hessian-free optimization." Proceedings of the 27th International Conference on Machine Learning (ICML-10). 2010.
- Qian, Ning. "On the momentum term in gradient descent learning algorithms." Neural networks 12.1 (1999): 145-151.
- Nesterov, Yurii. "A method of solving a convex programming problem with convergence rate O (1/k2)." Soviet Mathematics Doklady. Vol. 27. No. 2. 1983.

References

- Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." Journal of Machine Learning Research 12.Jul (2011): 2121-2159.
- Zeiler, Matthew D. "ADADELTA: an adaptive learning rate method." arXiv preprint arXiv:1212.5701 (2012).