Adam: A Method for Stochastic Optimization

Diederik P. Kingma and Jimmy Lei Ba

Nadav Cohen

The Hebrew University of Jerusalem

Advanced Seminar in Deep Learning (#67679)

October 18, 2015

Outline

Loss Minimization

Loss minimization problem:

$$\min_{W} \left\{ L(W) := \frac{1}{m} \sum\nolimits_{i=1}^{m} \ell(W; x_i, y_i) + \lambda r(W) \right\}$$

- $\{(x_i, y_i)\}_{i=1}^{m}$ – training instances (x_i) and corresponding labels (y_i)
- W – network parameters to learn
- $\ell(W; x_i, y_i)$ – loss of network parameterized by W w.r.t. (x_i, y_i)
- $r(W)$ – regularization function (e.g. $\|W\|_2^2$)
- $\lambda > 0$ – regularization weight

Large-Scale $\longrightarrow$ First-Order Stochastic Methods

Large-scale setting:

- Many network parameters (e.g. $dim(W) \sim 10^8$)
  $\implies$ computing Hessian (second order derivatives) is expensive
- Many training examples (e.g. $m \sim 10^6$)
  $\implies$ computing full objective at every iteration is expensive

Large-Scale $\longrightarrow$ First-Order Stochastic Methods

Large-scale setting:

- Many network parameters (e.g. $dim(W) \sim 10^8$)
  $\Longrightarrow$ computing Hessian (second order derivatives) is expensive
- Many training examples (e.g. $m \sim 10^6$)
  $\Longrightarrow$ computing full objective at every iteration is expensive

Optimization methods must be:

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

$$L_t(W) := \frac{1}{b}\sum_{j=1}^{b} \ell(W; x_{i_j}, y_{i_j}) + \lambda r(W)$$

$\{(x_{i_j}, y_{i_j})\}_{j=1}^{b}$ – random *mini-batch* chosen at iteration t

Stochastic Gradient Descent (SGD)

Update rule:

$$\begin{aligned} V_t &= \mu V_{t-1} - \alpha \nabla L_t(W_{t-1}) \\ W_t &= W_{t-1} + V_t \end{aligned}$$

- $\alpha > 0$ – *learning rate* (typical choices: $0.01, 0.1$)
- $\mu \in [0, 1)$ – *momentum* (typical choices: $0.9, 0.95, 0.99$)

Momentum smooths updates, enhancing stability and speed.

Nesterov's Accelerated Gradient (NAG)

Update rule:

$$\begin{aligned} V_t &= \mu V_{t-1} - \alpha \nabla L_t(W_{t-1} + \mu V_{t-1}) \\ W_t &= W_{t-1} + V_t \end{aligned}$$

Only difference from SGD is partial update $(+\mu V_t)$ in gradient computation. May increase stability and speed in ill-conditioned problems[1].

[1] See "On the Importance of Initialization and Momentum in Deep Learning" by Sutskever et al.

Adaptive Gradient (AdaGrad)

Update rule:

$$W_t = W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{\sqrt{\sum_{t'=1}^{t} \nabla L_{t'}(W_{t'-1})^2}}$$

Learning rate adapted per coordinate:

- Highly varying coordinate $\longrightarrow$ suppress
- Rarely varying coordinate $\longrightarrow$ enhance

Disadvantage in non-stationary settings:
All gradients (recent and old) weighted equally

Root Mean Square Propagation (RMSProp)

Update rule:

$$
\begin{aligned}
R_t &= \gamma R_{t-1} + (1-\gamma)\nabla L_t(W_{t-1})^2 \\
W_t &= W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{\sqrt{R_t}}
\end{aligned}
$$

Similar to AdaGrad but with an exponential moving average controlled by $\gamma \in [0,1)$ (smaller $\gamma \implies$ more emphasis on recent gradients).

Root Mean Square Propagation (RMSProp)

Update rule:

$$\begin{aligned} R_t &= \gamma R_{t-1} + (1-\gamma)\nabla L_t(W_{t-1})^2 \\ W_t &= W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{\sqrt{R_t}} \end{aligned}$$

Similar to AdaGrad but with an exponential moving average controlled by $\gamma \in [0, 1)$ (smaller $\gamma \implies$ more emphasis on recent gradients).

May be combined with NAG:

$$\begin{aligned} R_t &= \gamma R_{t-1} + (1-\gamma)\nabla L_t(W_{t-1} + \mu V_{t-1})^2 \\ V_t &= \mu V_{t-1} - \frac{\alpha}{\sqrt{R_t}} \nabla L_t(W_{t-1} + \mu V_{t-1}) \\ W_t &= W_{t-1} + V_t \end{aligned}$$

Outline

Rationale

Motivation

Combine the advantages of:

- AdaGrad – works well with sparse gradients
- RMSProp – works well in non-stationary settings

Idea

- Maintain exponential moving averages of gradient and its square
- Update proportional to $\frac{\text{average gradient}}{\sqrt{\text{average squared gradient}}}$

Algorithm

Adam

$M_0 = \mathbf{0}, R_0 = \mathbf{0}$ (Initialization)

For $t = 1, \ldots, T$:

$$M_t = \beta_1 M_{t-1} + (1 - \beta_1)\nabla L_t(W_{t-1}) \quad \text{(1st moment estimate)}$$

$$R_t = \beta_2 R_{t-1} + (1 - \beta_2)\nabla L_t(W_{t-1})^2 \quad \text{(2nd moment estimate)}$$

$$\hat{M}_t = M_t / (1 - (\beta_1)^t) \quad \text{(1st moment bias correction)}$$

$$\hat{R}_t = R_t / (1 - (\beta_2)^t) \quad \text{(2nd moment bias correction)}$$

$$W_t = W_{t-1} - \alpha \frac{\hat{M}_t}{\sqrt{\hat{R}_t + \epsilon}} \quad \text{(Update)}$$

Return W_T

Hyper-parameters:

- $\alpha > 0$ – learning rate (typical choice: 0.001)
- $\beta_1 \in [0, 1)$ – 1st moment decay rate (typical choice: 0.9)
- $\beta_2 \in [0, 1)$ – 2nd moment decay rate (typical choice: 0.999)
- $\epsilon > 0$ – numerical term (typical choice: 10^{-8})

Parameter Updates

Adam's step at iteration t (assuming $\epsilon = 0$):

$$\Delta_t = -\alpha \frac{\hat{M}_t}{\sqrt{\hat{R}_t}}$$

Parameter Updates

Adam's step at iteration t (assuming $\epsilon = 0$):

$$\Delta_t = -\alpha \frac{\hat{M}_t}{\sqrt{\hat{R}_t}}$$

Properties:

- **Scale-invariance**:

$$\begin{aligned} L(W) \rightarrow c \cdot L(W) \quad &\Longrightarrow \quad \hat{M}_t \rightarrow c \cdot \hat{M}_t \ \wedge \ \hat{R}_t \rightarrow c^2 \cdot \hat{R}_t \\ &\Longrightarrow \quad \Delta_t \text{ does not change} \end{aligned}$$

Parameter Updates

Adam's step at iteration t (assuming $\epsilon = 0$):

$$\Delta_t = -\alpha \frac{\hat{M}_t}{\sqrt{\hat{R}_t}}$$

Properties:

- **Scale-invariance**:

$$\begin{aligned} L(W) \rightarrow c \cdot L(W) \quad &\Longrightarrow \quad \hat{M}_t \rightarrow c \cdot \hat{M}_t \;\wedge\; \hat{R}_t \rightarrow c^2 \cdot \hat{R}_t \\ &\Longrightarrow \quad \Delta_t \text{ does not change} \end{aligned}$$

- **Bounded norm**:

$$\|\Delta_t\|_\infty \leq \begin{cases} \alpha \cdot (1 - \beta_1)/\sqrt{1 - \beta_2} & , (1 - \beta_1) > \sqrt{1 - \beta_2} \\ \alpha & , \text{otherwise} \end{cases}$$

Bias Correction

Taking into account the initialization $M_0 = \mathbf{0}$, we have:

$$\begin{aligned} M_t &= \beta_1 M_{t-1} + (1-\beta_1)\nabla L_t(W_{t-1}) \\ &= \sum_{i=1}^{t} (1-\beta_1)(\beta_1)^{t-i} \cdot \nabla L_i(W_{i-1}) \end{aligned}$$

$\sum_{i=1}^{t}(1-\beta_1)(\beta_1)^{t-i} = 1-(\beta_1)^t$, so to obtain an unbiased estimate we divided by $1-(\beta_1)^t$:

$$\hat{M}_t = M_t / \left(1-(\beta_1)^t\right)$$

An analogous argument derives the bias correction of R_t.

Outline

Convergence in Online Convex Regime

Regret at iteration T :

$$R(T) := \sum_{t=1}^{T} [L_t(W_t) - L_t(W^*)]$$

where:

$$W^* := \underset{W}{\operatorname{argmin}} \sum_{t=1}^{T} L_t(W)$$

Convergence in Online Convex Regime

Regret at iteration T :

$$R(T) := \sum_{t=1}^{T} [L_t(W_t) - L_t(W^*)]$$

where:

$$W^* := \underset{W}{\text{argmin}} \sum_{t=1}^{T} L_t(W)$$

In convex regime, Adam gives regret bound comparable to best known:

Theorem

If all batch objectives $L_t(W)$ are convex and have bounded gradients, and all points W_t generated by Adam are within bounded distance from each other, then for every $T \in \mathbb{N}$:

$$\frac{R(T)}{T} = \mathcal{O}\left(\frac{1}{\sqrt{T}}\right)$$

Outline

Relation to SGD

Setting $\beta_2 = 1$, Adam's update rule may be written as:

$$\begin{aligned} M_t &= \mu M_{t-1} - \eta \nabla L_t(W_{t-1}) \\ W_t &= W_{t-1} + M_t \end{aligned}$$

where:

$$\mu := \frac{\beta_1(1-(\beta_1)^{t-1})}{1-(\beta_1)^t} \qquad \eta := \frac{\alpha(1-\beta_1)}{(1-(\beta_1)^t)\sqrt{\epsilon}}$$

<u>**Conclusion**</u>

Disabling 2nd moment estimation ($\beta_2 = 1$) reduces Adam to SGD with:

- Learning rate descending towards $\alpha(1-\beta_1)/\sqrt{\epsilon}$
- Momentum ascending towards β_1

Relation to AdaGrad

Setting $\beta_1 = 0$ and $\beta_2 \rightarrow 1^-$ (and assuming $\epsilon = 0$), Adam's update rule may be written as:

$$W_t = W_{t-1} - \alpha \frac{\nabla L_t(W_{t-1})}{t^{1/2}\sqrt{\sum_{i=1}^{t} \nabla L_t(W_{t-1})^2}}$$

Conclusion

In the limit $\beta_2 \rightarrow 1^-$, with $\beta_1 = 0$, Adam reduces to AdaGrad with annealing learning rate $\alpha \cdot t^{-1/2}$.

Relation to RMSProp

RMSProp with momentum is the method most closely related to Adam.

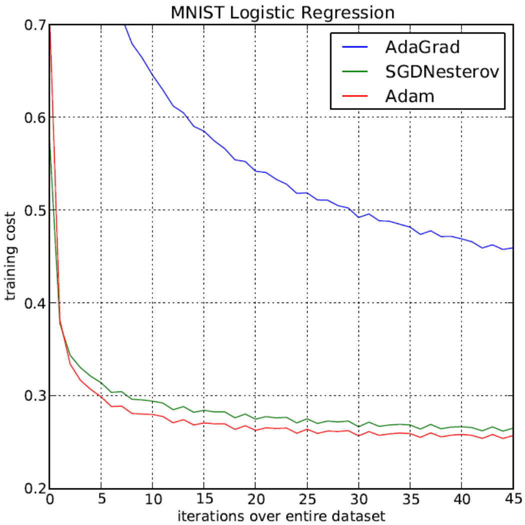
Main differences:

- RMSProp rescales gradient and then applies momentum, Adam first applies momentum (moving average) and then rescales.
- RMSProp lacks bias correction, often leading to large stepsizes in early stages of run (especially when β_2 is close to 1).

Outline

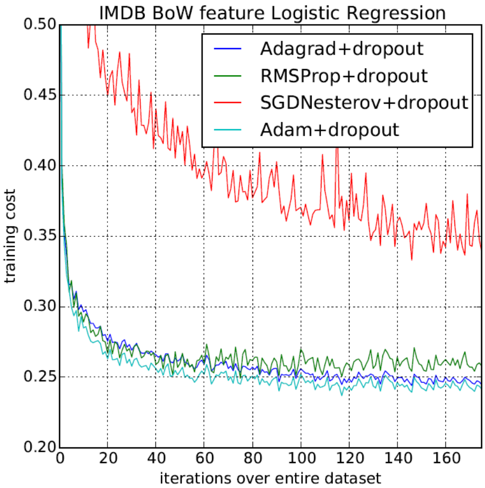
Logistic Regression on MNIST

L^2-regularized logistic regression applied directly to image pixels:



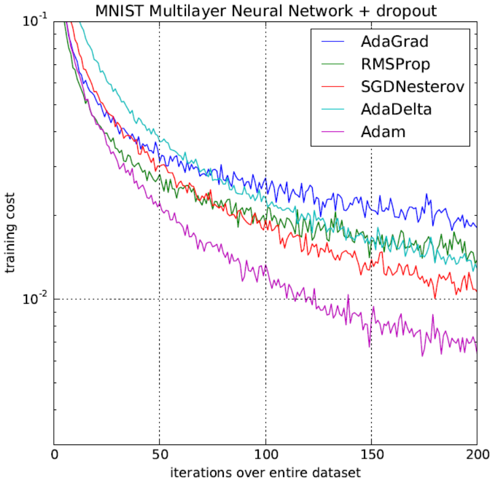
Logistic Regression on IMDB

Dropout regularized logistic regression applied to sparse Bag-of-Words features:



Multi-Layer Neural Networks (Fully-Connected) on MNIST

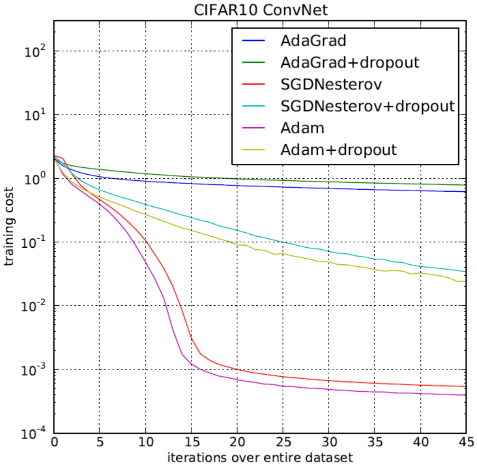
2 hidden layers, 1000 units each, ReLU activation, dropout regularization:



Convolutional Neural Networks on CIFAR-10

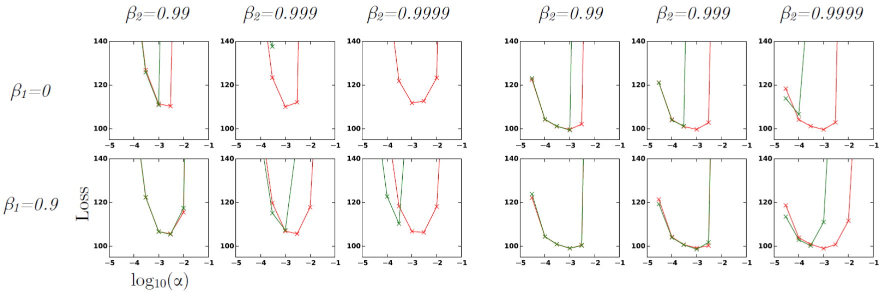
Network architecture:

conv-5x5@64 → pool-3x3(stride-2) → conv-5x5@64 → pool-3x3(stride-2) → conv-5x5@128 → pool-3x3(stride-2) → dense@1000 → dense@10:



Bias Correction Term on Variational Auto-Encoder

Training variational auto-encoder (single hidden layer network) with (red) and without (green) bias correction, for different values of β_1, β_2, α:



(a) after 10 epochs

(b) after 100 epochs

Outline

L^p Generalization

Adam may be generalized by replacing gradient L^2 norm with L^p norm:

Adam – $\mathbf{L^p}$ Generalization

$M_0 = \mathbf{0}, R_0 = \mathbf{0}$ (Initialization)

For $t = 1, \ldots, T$:

$M_t = \beta_1 M_{t-1} + (1 - \beta_1) \nabla L_t(W_{t-1})$ (1st moment estimate)

$R_t = \beta_2 R_{t-1} + (1 - (\beta_2)^p) \nabla L_t(W_{t-1})^{\mathbf{p}}$ (p'th moment estimate)

$\hat{M}_t = M_t / (1 - (\beta_1)^t)$ (1st moment bias correction)

$\hat{R}_t = R_t / (1 - (\beta_2)^{pt})$ (p'th moment bias correction)

$W_t = W_{t-1} - \alpha \frac{\hat{M}_t}{(\hat{R}_t + \epsilon)^{\mathbf{1/p}}}$ (Update)

Return W_T

(β_2 re-parameterized as $(\beta_2)^p$ for convenience)

$p \to \infty \implies$ AdaMax

When $p \to \infty$, $\|\cdot\|_p \to \max\{\cdot\}$ and we get:

AdaMax

$M_0 = \mathbf{0}, U_0 = \mathbf{0}$ (Initialization)

For $t = 1, \ldots, T$:

$M_t = \beta_1 M_{t-1} + (1 - \beta_1) \nabla L_t(W_{t-1})$ (1st moment estimate)

$U_t = \max\{\beta_2 U_{t-1}, |\nabla L_t(W_{t-1})|\}$ ("∞" moment estimate)

$\hat{M}_t = M_t / (1 - (\beta_1)^t)$ (1st moment bias correction)

$W_t = W_{t-1} - \alpha \frac{\hat{M}_t}{U_t}$ (Update)

Return W_T

In AdaMax step size is always bounded by α:

$$\|\Delta_t\|_\infty \leq \alpha$$

Outline

Conclusion

Adam:

Conclusion

Adam:

- Efficient first-order stochastic optimization method

Conclusion

Adam:

- Efficient first-order stochastic optimization method
- Combines the advantages of:
  - AdaGrad – works well with sparse gradients
  - RMSProp – deals with non-stationary objectives

Conclusion

Adam:

- Efficient first-order stochastic optimization method
- Combines the advantages of:
  - AdaGrad – works well with sparse gradients
  - RMSProp – deals with non-stationary objectives
- Parameter updates:
  - Have bounded norm
  - Are scale-invariant

Conclusion

Adam:

- Efficient first-order stochastic optimization method
- Combines the advantages of:
  - AdaGrad – works well with sparse gradients
  - RMSProp – deals with non-stationary objectives
- Parameter updates:
  - Have bounded norm
  - Are scale-invariant
- *Widely used in deep learning community (e.g. Google DeepMind)*

Outline

Thank You