Advanced gradient descent

Olivier Sigaud

Sorbonne Université
http://people.isir.upmc.fr/sigaud

Outline

- A reminder about vanilla gradient descent

  Standard background knowledge...

- Advanced gradient descent methods

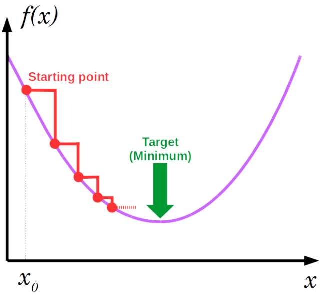
  Pierrot, T., Perrin, N., & Sigaud, O. (2018) First-order and second-order variants of the gradient descent: a unified framework. *arXiv preprint arXiv:1810.08102*

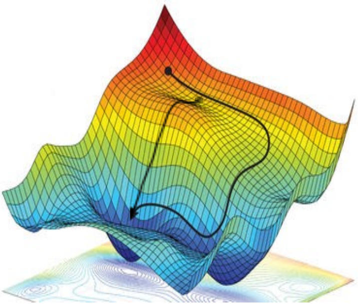
- I won't cover adaptive gradient descent methods, see:

  Ruder, S. (2016) An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*

- The underlying motivation is explaining advanced gradient descent concepts used in deep RL

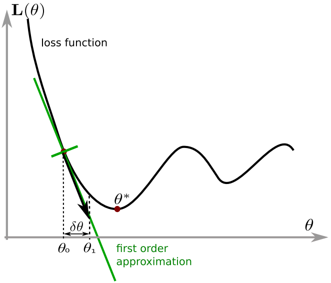
Vanilla gradient descent





- Gradient descent is an iterative process with local steps
- We want to reach a minimum
- At each step, we need a direction and a step size

I. Getting the right direction

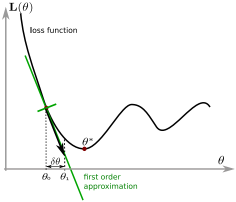


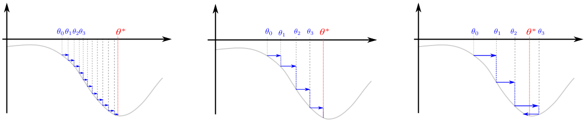
- We want to minimize $\mathbf{L}(\boldsymbol{\theta}_i + \delta\boldsymbol{\theta}_i)$ over $\delta\boldsymbol{\theta}_i$.
- How to choose $\delta\boldsymbol{\theta}_i$?

First order approximation of the loss

- The optimum on $\delta\boldsymbol{\theta}$ is reached when $\dfrac{\partial \mathbf{L}(\boldsymbol{\theta} + \delta\boldsymbol{\theta})}{\partial \delta\boldsymbol{\theta}} = 0$
- $\mathbf{L}(\boldsymbol{\theta} + \delta\boldsymbol{\theta})$ can be approximated at the first order as $\mathbf{L}(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})\delta\boldsymbol{\theta} + \nu\delta\boldsymbol{\theta}^T\delta\boldsymbol{\theta}$ + higher order terms
- $\dfrac{\partial \mathbf{L}(\boldsymbol{\theta} + \delta\boldsymbol{\theta})}{\partial \delta\boldsymbol{\theta}} \sim \nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta}) + 2\nu\delta\boldsymbol{\theta}$
- Thus $\delta\boldsymbol{\theta}^* = -\frac{1}{2\nu}\nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})$
- We rewrite it $\delta\boldsymbol{\theta}^* = -\alpha\nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})$
- And the iteration rule is $\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i - \alpha_i\nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})$
- The steepest descent direction is given by the first order derivative!

II. Finding the right step size

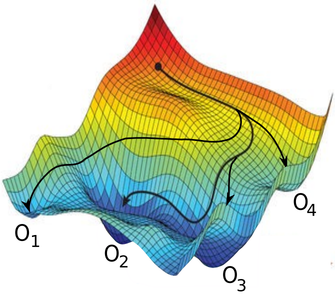




- The first order local derivative $\nabla_{\boldsymbol{\theta}} \mathbf{L}(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_i}$ gives the right direction
- But minimizing the first order derivative is not lower bounded
- We need a step size α to determine how far to go
- (a) α_i are too small, (b) α_i are adequate, (c) α_i are too large
- If too small, too many steps. If too large, may miss a local optimum
- Line search: iterate to find the best step size (used e.g. in TRPO)

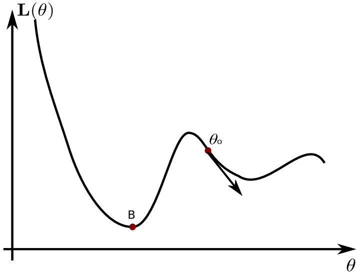
Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2015) Trust region policy optimization. *CoRR, abs/1502.05477*

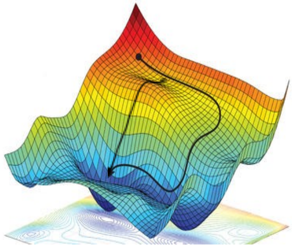
Local Optima



- Gradient descent is a local improvement approach where, at each step, we follow the steepest descent direction
- But we do not know where is the target optimum
- Unless the cost function is convex in the parameter space, gradient descent can end-up in different local optima depending on the starting point
- Adding small noise can result in very different local optima and paths
- Anything that basically goes down will end up somewhere low!

Steepest descent: limitations





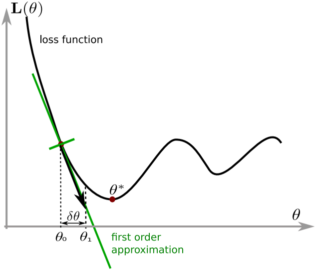
- Always following the steepest descent direction does not necessarily:
  - Result in the lowest minimum
  - Result in the shortest path to a given optimum
- Using momentum can help escape from local minima and "climb bumps" when useful
- Adaptive and second order gradient descent can help finding better minima or shorter paths

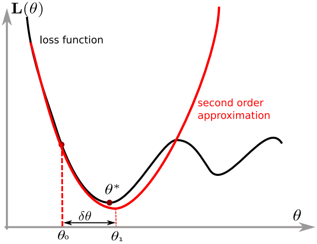
Other limitations

- Vanilla gradient descent may converge very slowly if the step size is not properly tuned.
- Its efficiency depends on arbitrary parameterizations θ
- We focus on two lines of improvement:
  - First-order methods such as the natural gradient introduce particular metrics to restrict gradient steps and make them independent from parametrization choices
  - Second-order methods use the Hessian matrix of the loss or its approximations to take into account its local curvature

Shun-ichi Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998

First order versus second order derivative





- In first order methods, need to define a step size
- Second order methods provide a more accurate approximation
- They can even provide a true minimum, when the Hessian matrix is a symmetric positive-definite matrix (SPD)
- In both cases, the derivative is very local
- The gradient should not be applied too far away from the current point
- Find a trust region where gradient can be trusted

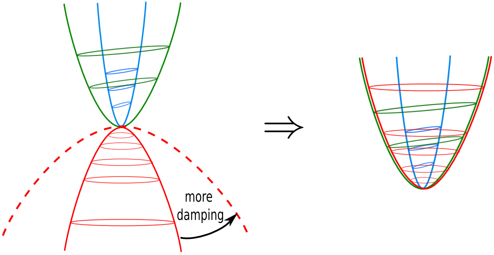
Defining a trust region: formulation

- To perform safe steps, we put a constraint on how far we can go
- We still want to find the $\delta\boldsymbol{\theta}$ providing the best decrease of $\mathbf{L}(\boldsymbol{\theta} + \delta\boldsymbol{\theta})$
- General constrained optimization problem:

$$\begin{cases} \min_{\delta\boldsymbol{\theta}} \mathbf{L}(\boldsymbol{\theta} + \delta\boldsymbol{\theta}) \\ \text{under constraint } \delta\boldsymbol{\theta}^T \mathbf{M}(\boldsymbol{\theta}) \delta\boldsymbol{\theta} \leq \epsilon^2, \end{cases} \tag{1}$$

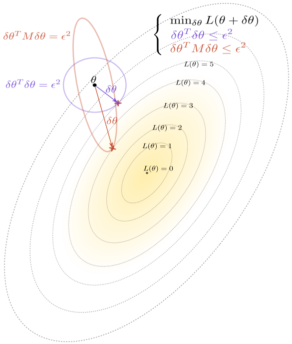
- $\mathbf{M}(\boldsymbol{\theta})$ has to be an SPD matrix (because we will invert it)

Damping



- Matrices $\mathbf{M}(\boldsymbol{\theta})$ are often symmetric but might not be definite positive
- To invert it, we add a damping term $\lambda\mathbf{I}$
- We use a λ as small as possible so that the matrix is positive in all dimensions
- Thus we invert $\mathbf{M}(\boldsymbol{\theta}) + \lambda\mathbf{I}$

Defining a metrics on the step size



- Different metrics $\mathbf{M}(\boldsymbol{\theta})$ affect both the gradient step size and direction
- The colored ellipses correspond to trust regions

First order approximation of the loss

- We still want to find the $\delta\boldsymbol{\theta}$ providing the best decrease of $\mathbf{L}(\boldsymbol{\theta} + \delta\boldsymbol{\theta})$
- $\mathbf{L}(\boldsymbol{\theta} + \delta\boldsymbol{\theta})$ can be approximated at the first order as $\mathbf{L}(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})^T\delta\boldsymbol{\theta}+$ higher order terms
- $\mathbf{L}(\boldsymbol{\theta})$ does not depend on $\delta\boldsymbol{\theta}$, thus is can be removed from the minimization problem
- Problem (1) can be reformulated as

$$\begin{cases} \min_{\delta\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})^T\delta\boldsymbol{\theta} \\ \text{under constraint } \delta\boldsymbol{\theta}^T\mathbf{M}(\boldsymbol{\theta})\delta\boldsymbol{\theta} \leq \epsilon^2. \end{cases} \tag{2}$$

- How can we solve it?

Finding the steepest descent direction: Lagrangian method

- The Lagrangian of (2) is

$$\mathcal{L}(\delta\boldsymbol{\theta}) = \mathbf{L}(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})^T\delta\boldsymbol{\theta} + \nu(\delta\boldsymbol{\theta}^T\mathbf{M}(\boldsymbol{\theta})\delta\boldsymbol{\theta} - \epsilon^2)$$

  where ν is the Lagrange multiplier
- The optimum on $\delta\boldsymbol{\theta}$ is when the derivative of the Lagrangian is null
- $(\mathcal{L}(\delta\boldsymbol{\theta}))' = \nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})^T + 2\nu\mathbf{M}(\boldsymbol{\theta})\delta\boldsymbol{\theta}$
- Thus $\delta\boldsymbol{\theta}^* = -\frac{1}{2\nu}\mathbf{M}(\boldsymbol{\theta})^{-1}\nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})$ ($\mathbf{M}(\boldsymbol{\theta})$ is SPD, thus invertible)
- We rewrite it $\delta\boldsymbol{\theta}^* = \alpha\mathbf{M}(\boldsymbol{\theta})^{-1}\nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})$ with $\alpha = -\frac{\epsilon}{\sqrt{\nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})^T\mathbf{M}(\boldsymbol{\theta})^{-1}\nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})}}$
- When $\mathbf{M}(\boldsymbol{\theta}) = \mathbf{I}$, we recover the standard gradient descent equation
- $\boldsymbol{\theta}_{i+1} \leftarrow \boldsymbol{\theta}_i - \alpha_i\nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})$

Six different choices for $\mathbf{M}(\boldsymbol{\theta})$

- $\mathbf{M}(\boldsymbol{\theta})$ defines metrics transforming from parameters θ to another space

$\mathbf{M}(\boldsymbol{\theta})$	Corresponding approach
$\mathbf{I}$	Vanilla gradient descent
$\mathbb{E}_{\mathbf{x}}\left[J(\mathbf{x},\boldsymbol{\theta})^T J(\mathbf{x},\boldsymbol{\theta})\right] + \lambda\mathbf{I}$	Classical Gauss-Newton
$\mathbb{E}_{(\mathbf{x},\mathbf{y})}\left[\nabla_{\boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}(\mathbf{y}\mid\mathbf{x}))\nabla_{\boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}(\mathbf{y}\mid\mathbf{x}))^T\right] + \lambda\mathbf{I}$	Natural gradient (with empirical Fisher matrix)
$\mathbb{E}_{\mathbf{x}}\left[\nabla_{\boldsymbol{\theta}} l_{\boldsymbol{\theta}}(\mathbf{x})\nabla_{\boldsymbol{\theta}} l_{\boldsymbol{\theta}}(\mathbf{x})^T\right] + \lambda\mathbf{I}$	Gradient covariance matrix
$H(\boldsymbol{\theta}) + \lambda\mathbf{I}$	Newton's method
$\mathbb{E}_{(\mathbf{x},\mathbf{y})}\left[J(\mathbf{x},\boldsymbol{\theta})^T \mathcal{H}_{\mathbf{y}}(\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}))J(\mathbf{x},\boldsymbol{\theta})\right] + \lambda\mathbf{I}$	Generalized Gauss-Newton

- Correspond to 6 popular variants of the gradient descent
- In the first 3, $\mathbf{M}(\boldsymbol{\theta})$ does not depend on the loss, in the last 3, it does
- The first four are first order, the last two are second order methods

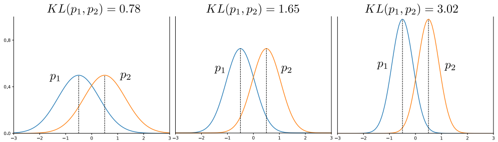
Pierrot, T., Perrin, N., & Sigaud, O. (2018) First-order and second-order variants of the gradient descent: a unified framework. *arXiv preprint arXiv:1810.08102*

I. Classical Gauss-Newton

- In vanilla gradient descent, the constraint $\delta\boldsymbol{\theta}^T\mathbf{I}(\boldsymbol{\theta})\delta\boldsymbol{\theta} \leq \epsilon^2$ acts as if all components of $\boldsymbol{\theta}$ had the same importance, which is not necessarily true
- Rather constrain the output $\mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x})$ of the function to optimize
- We want to minimize $\mathbb{E}_{\mathbf{x}}[||\mathbf{h}_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\mathbf{x}) - \mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x})||^2]$
- At first order, $\mathbf{h}_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\mathbf{x}) - \mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x}) \sim J_{\mathbf{x}}(\boldsymbol{\theta})\delta\boldsymbol{\theta}$ where $J_{\mathbf{x}}(\boldsymbol{\theta})$ is the Jacobian of the function $\boldsymbol{\theta} \rightarrow \mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x})$
- $\mathbb{E}_{\mathbf{x}}[||\mathbf{h}_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\mathbf{x}) - \mathbf{h}_{\boldsymbol{\theta}}(\mathbf{x})||^2] \sim \mathbb{E}_{\mathbf{x}}[||J_{\mathbf{x}}(\boldsymbol{\theta})\delta\boldsymbol{\theta}||^2] = \delta\boldsymbol{\theta}^T\mathbb{E}_{\mathbf{x}}\left[J(\mathbf{x},\boldsymbol{\theta})^T J(\mathbf{x},\boldsymbol{\theta})\right]\delta\boldsymbol{\theta}$
- Thus we want to minimize $\mathbf{M}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}}\left[J(\mathbf{x},\boldsymbol{\theta})^T J(\mathbf{x},\boldsymbol{\theta})\right]$
- The loss does not appear in the constraint

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018

II. Natural Policy Gradient



- To constrain stochastic objects to stay close to each other, one can constrain their KL divergence
- Under KL constraint, moving further away is easier when the variance is large
- Thus the mean converges first, then variance is reduced
- Ensures a large enough amount of exploration
- Other properties listed in the Pierrot et al. (2018) paper
- Many RL algorithms use the Natural Policy Gradient: NAC, eNAC, TRPO...
- $\mathbf{M}(\boldsymbol{\theta}) = \mathbb{E}_{(\mathbf{x},\mathbf{y})}[\nabla_{\boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}))\nabla_{\boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}(\mathbf{y}|\mathbf{x}))^T](+\lambda\mathbf{I})$

Sham M. Kakade. A natural policy gradient. In *Advances in neural information processing systems*, pp. 1531–1538, 2002

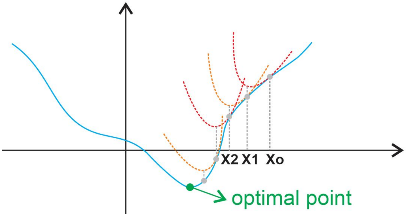
III. Gradient covariance matrix

- We want to use the loss to measure the magnitude of a change due to $\delta\boldsymbol{\theta}$
- Consider the expected squared difference between both losses at each atomic sample $\mathbf{x}$:
- We want to minimize $\mathbb{E}_{\mathbf{x}}[(l_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\mathbf{x}) - l_{\boldsymbol{\theta}}(\mathbf{x}))^2]$
- Let us replace $l_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\mathbf{x})$ by a first-order approximation: $l_{\boldsymbol{\theta}+\delta\boldsymbol{\theta}}(\mathbf{x}) \sim l_{\boldsymbol{\theta}}(\mathbf{x}) + \nabla_{\boldsymbol{\theta}} l_{\boldsymbol{\theta}}(\mathbf{x})^T \delta\boldsymbol{\theta}$
- The above expectation simplifies to $\mathbb{E}_{\mathbf{x}}[(\nabla_{\boldsymbol{\theta}} l_{\boldsymbol{\theta}}(\mathbf{x})^T \delta\boldsymbol{\theta})^2] = \delta\boldsymbol{\theta}^T \mathbb{E}_{\mathbf{x}}[(\nabla_{\boldsymbol{\theta}} l_{\boldsymbol{\theta}}(\mathbf{x}) \nabla_{\boldsymbol{\theta}} l_{\boldsymbol{\theta}}(\mathbf{x})^T] \delta\boldsymbol{\theta}$
- $\mathbf{M}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x}}[(\nabla_{\boldsymbol{\theta}} l_{\boldsymbol{\theta}}(\mathbf{x}) \nabla_{\boldsymbol{\theta}} l_{\boldsymbol{\theta}}(\mathbf{x})^T]$ is called the gradient covariance matrix or the outer product metric
- If the loss is the negative log-likelihood $\nabla_{\boldsymbol{\theta}} l_{\boldsymbol{\theta}}(\mathbf{x}) = \nabla_{\boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}(.|\mathbf{x}))$, we recover the empirical Fisher, hence a natural gradient method

Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in neural information processing systems*, pp. 161–168, 2008

Yann Ollivier. Riemannian metrics for neural networks I: feedforward networks. *Information and Inference: A Journal of the IMA*, 4(2):108–153, 2015

IV. Newton's method



- Newton's method is the second order counterpart of the gradient covariance matrix method
- The second order approximation of the loss is $\mathbf{L}(\boldsymbol{\theta}+\delta\boldsymbol{\theta}) \sim \mathbf{L}(\boldsymbol{\theta}) + \nabla_{\boldsymbol{\theta}}\mathbf{L}(\boldsymbol{\theta})^T\delta\boldsymbol{\theta} + \frac{1}{2}\delta\boldsymbol{\theta}^T\mathbf{H}(\boldsymbol{\theta})\delta\boldsymbol{\theta}$ where $\mathbf{H}(\boldsymbol{\theta})$ is the Hessian matrix of the loss
- The approximation of $\mathbf{L}(\boldsymbol{\theta}+\delta\boldsymbol{\theta})$ is probably accurate as long as $\frac{1}{2}\delta\boldsymbol{\theta}^T\mathbf{H}(\boldsymbol{\theta})\delta\boldsymbol{\theta}$ is small
- Thus we constrain on $\delta\boldsymbol{\theta}^T\mathbf{H}(\boldsymbol{\theta})\delta\boldsymbol{\theta} \leq \epsilon^2$
- Thus $\mathbf{M}(\boldsymbol{\theta}) = \mathbf{H}(\boldsymbol{\theta})$

Andreas Fischer. A special newton-type optimization method. *Optimization*, 24(3-4):269–284, 1992

V. Generalized Gauss-Newton

- The generalized Gauss-Newton approach combines several of the above ideas
- Well... it is complicated! ;)
- Read the Pierrot et al. (2018) paper if you want to know more
- The sixth method is vanilla gradient descent

O Knoth. A globalization scheme for the generalized Gauss-Newton method. *Numerische Mathematik*, 56(6):591–607, 1989

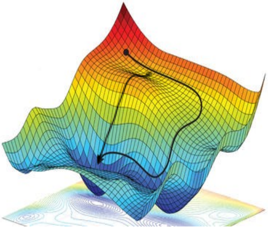
Computation methods

- To solve the constrained optimization problem (2), we need to invert $\mathbf{M}(\boldsymbol{\theta})$
- This is why we need it to be SPD
- But estimating and inverting the Fisher or Gauss-Newton matrix is costly
- KFAC and KFRA use block diagonal approximation of the Gauss-Newton matrix
- Lehman uses an even simpler diagonal approx of the Gauss-Newton
- TRPO uses a conjugate gradient method (avoids using matrices by only multiplying vectors)
- NAC and eNAC use the fact that, under some compatibility condition, the empirical Fisher calculation simplifies and even vanishes away
- Conjugate gradient is a “Hessian-free” method

Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O. Stanley. Safe mutations for deep and recurrent neural networks through output gradients. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 117–124, 2018

Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba (2017) Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. *arXiv preprint arXiv:1708.05144*

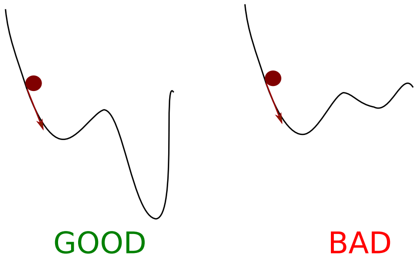
Using momentum



- It can be the case that at some point along the shortest path, you have to increase the cost function instead of decreasing it
- It can also be the case that you need to increase the cost function to escape from a local minimum
- In such a case, momentum can help you follow the shortest path by "climbing the bump"

Qian, N. (1999) On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151

Is momentum good or bad?



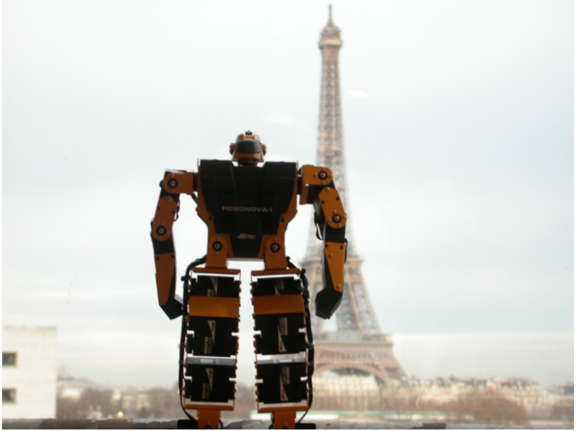
- Depends on the cost function landscape!
- Other adaptive gradient mechanisms: RmsProp, Adam, Nesterov...

Ruder, S. (2016) An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*

Messages

- Six different well-founded methods, six directions
- All six directions are in the same half-plane as vanilla gradient descent
- Does the direction really matter?
- Given the local steps, highly depends on the landscape
- No free lunch: each method might be the best in some landscape
- What about the step size? Use line search, see TRPO and PPO
- Open question: why is Adam often superior in many DNN applications?

Any question?



Send mail to: `Olivier.Sigaud@upmc.fr`

Shun-ichi Amari.
Natural gradient works efficiently in learning.
Neural Computation, 10(2):251–276, 1998.

Léon Bottou and Olivier Bousquet.
The tradeoffs of large scale learning.
In *Advances in neural information processing systems*, pp. 161–168, 2008.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal.
Optimization methods for large-scale machine learning.
Siam Review, 60(2):223–311, 2018.

Andreas Fischer.
A special newton-type optimization method.
Optimization, 24(3-4):269–284, 1992.

Sham M. Kakade.
A natural policy gradient.
In *Advances in neural information processing systems*, pp. 1531–1538, 2002.

O. Knoth.
A globalization scheme for the generalized gauss-newton method.
Numerische Mathematik, 56(6):591–607, 1989.

Joel Lehman, Jay Chen, Jeff Clune, and Kenneth O. Stanley.
Safe mutations for deep and recurrent neural networks through output gradients.
In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 117–124, 2018.

Yann Ollivier.
Riemannian metrics for neural networks I: feedforward networks.
Information and Inference: A Journal of the IMA, 4(2):108–153, 2015.

Thomas Pierrot, Nicolas Perrin, and Olivier Sigaud.

First-order and second-order variants of the gradient descent: a unified framework.
arXiv preprint arXiv:1810.08102, 2018.

Ning Qian.
On the momentum term in gradient descent learning algorithms.
Neural networks, 12(1):145–151, 1999.

Sebastian Ruder.
An overview of gradient descent optimization algorithms.
arXiv preprint arXiv:1609.04747, 2016.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel.
Trust region policy optimization.
CoRR, abs/1502.05477, 2015.

Yuhuai Wu, Elman Mansimov, Shun Liao, Roger Grosse, and Jimmy Ba.
Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation.
arXiv preprint arXiv:1708.05144, 2017.