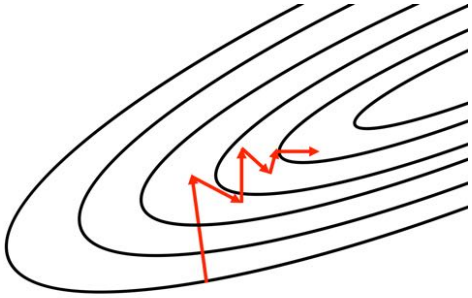
Tópicos Avanzados en Entrenamiento de Redes Profundas

Tasas Adaptativas

Complejidad del Entrenamiento



- Problema: $\min_{\boldsymbol{w}} E(\boldsymbol{w})$
- Algoritmo:

for $t = 1, \ldots, T$ **do**
  $\boldsymbol{w}^{(t)} \leftarrow \boldsymbol{w}^{(t-1)} - \eta \frac{\delta E}{\delta \boldsymbol{w}}$;
end

Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada

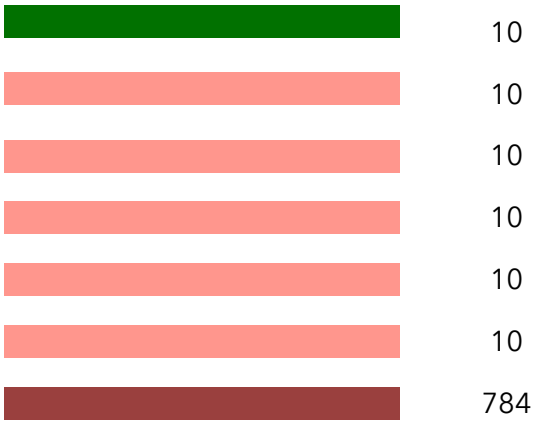
Abstract

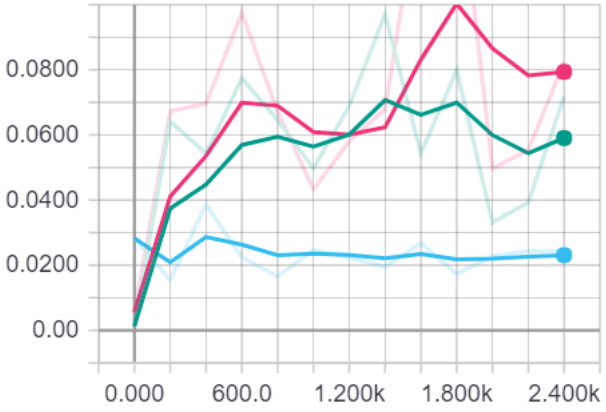
Whereas before 2006 it appears that deep multi-layer neural networks were not successfully trained, since then several algorithms have been shown to successfully train them, with experimental results showing the superiority of deeper vs less deep architectures. All these experimen-

learning methods for a wide array of *deep architectures*, including neural networks with many hidden layers (Vincent et al., 2008) and graphical models with many levels of hidden variables (Hinton et al., 2006), among others (Zhu et al., 2009; Weston et al., 2008). Much attention has recently been devoted to them (see (Bengio, 2009) for a review), because of their theoretical appeal, inspiration from biology and human cognition, and because of empirical

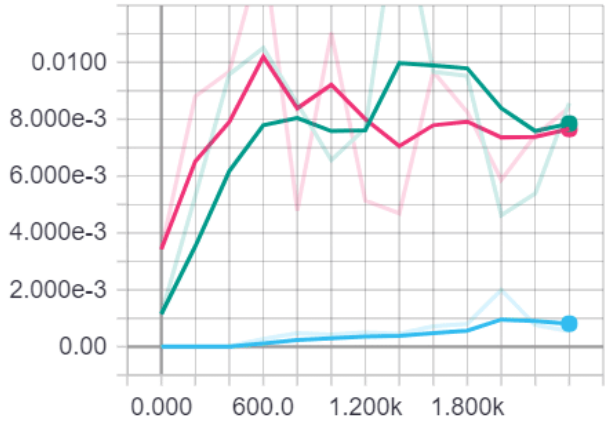
Xavier Glorot et al. Understanding the difficulty of training deep feedforward neural networks." *International Conference on Artificial Intelligence and Statistics*. 2010.

Gradientes Desvanecientes (y Explosivos)





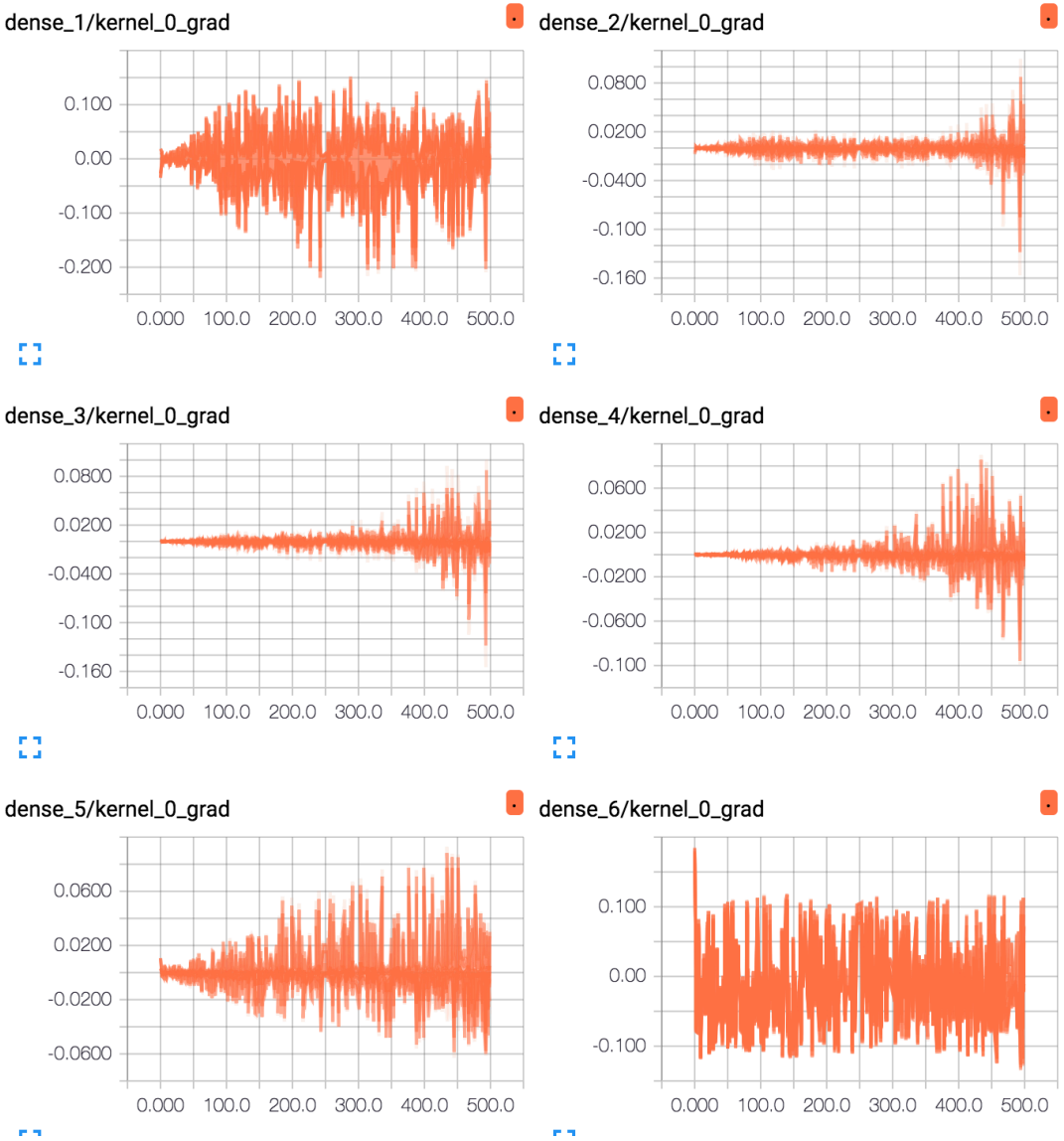
mean absolute gradients – output layer (6th layer) – sigmoid (blue), ReLU (red), Leaky ReLU (green)



mean absolute gradients – 1st layer – sigmoid (blue), ReLU (red), Leaky ReLU (green)

(*) créditos a Andrew Thomas

Gradientes Desvanecientes (y Explosivos)



Créditos a Jason Brownlee

Gradientes Desvanecientes (y Explosivos)

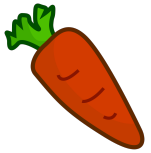
- Recordemos la forma en que entrenamos una capa (densa) de la red:

forward pass

$$a^{(\ell)} = g(W^{(\ell)} a^{(\ell-1)} - b^{(\ell)}) \quad \Leftrightarrow \quad \begin{aligned} p^{(\ell)} &= W^{(\ell)} a^{(\ell-1)} - b^{(\ell)} \\ a^{(\ell)} &= g(p^{(\ell)}) \end{aligned}$$

backward pass

$$\frac{\partial L}{\partial W_{ij}^{(\ell)}} = \frac{\partial L}{\partial a_i^{(\ell)}} \frac{\partial a_i^{(\ell)}}{\partial W_{ij}^{(\ell)}}$$



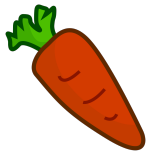
$$\delta_i^{(\ell)} = \sum_t \delta_t^{(\ell+1)} g'(p^{(\ell+1)})_t W_{ti}^{(\ell+1)}$$

$\delta_i^{(\ell)}$ $\quad g'(p_i^{(\ell)}) a_j^{(\ell-1)}$

Gradientes Desvanecientes (y Explosivos)

backward pass

$$\frac{\partial L}{\partial W_{ij}^{(\ell)}} = \boxed{\frac{\partial L}{\partial a_i^{(\ell)}}} \boxed{\frac{\partial a_i^{(\ell)}}{\partial W_{ij}^{(\ell)}}}$$



$$\delta^{(\ell)} = W^{(\ell+1)T} \text{diag}(g'(p^{(\ell+1)}))\, \delta^{(\ell+1)}$$

$$\delta^{(\ell)} = J^{(\ell+1)} \delta^{(\ell+1)}$$

Gradientes Desvanecientes (y Explosivos)

backward pass

$$\delta^{(\ell)} = J^{(\ell+1)} J^{(\ell+2)} \dots J^{(L)} \delta^{(L)}$$

$$\delta^{(\ell)} = \left(\prod_{s=\ell+1}^{L} J^{(s)} \right) \delta^{(L)}$$

Gradientes Desvanecientes (y Explosivos)

backward pass

$$\delta^{(\ell)} = \left(\prod_{s=\ell+1}^{L} J^{(s)} \right) \delta^{(L)}$$

$$\left\| \delta^{(\ell)} \right\| = \left\| \prod_{s=\ell+1}^{L} J^{(s)} \right\| \left\| \delta^{(L)} \right\|$$

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

Gradientes Desvanecientes (y Explosivos)

backward pass

$$\delta^{(\ell)} = \left(\prod_{s=\ell+1}^{L} J^{(s)} \right) \delta^{(L)}$$

$$\left\| \delta^{(\ell)} \right\| = \left\| \prod_{s=\ell+1}^{L} J^{(s)} \right\| \left\| \delta^{(L)} \right\|$$

Sea σ_{max^*} el más grande de los valores singulares principales de los Jacobianos en la red y sea σ_{min^*} el más pequeño de entre los valores singulares menores.

$$\|A\| = \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

$$\|A\|_2 = \sigma_{max}$$

$$\|A\|_2 \geq \sigma_{min}$$

Gradientes Desvanecientes (y Explosivos)

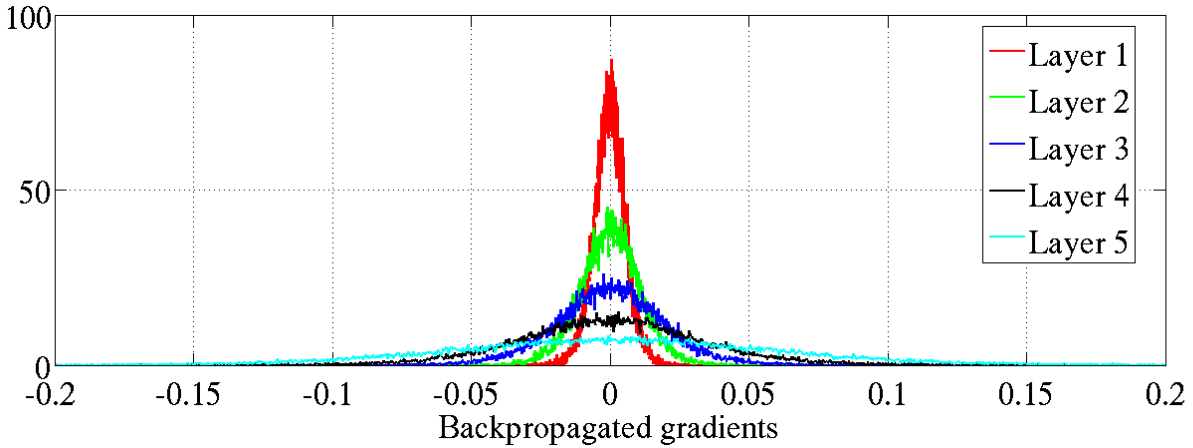
backward pass

$$\delta^{(\ell)} = \left(\prod_{s=\ell+1}^{L} J^{(s)} \right) \delta^{(L)}$$

$$\left\| \delta^{(\ell)} \right\| = \left\| \prod_{s=\ell+1}^{L} J^{(s)} \right\| \left\| \delta^{(L)} \right\|$$

$$\left\| \delta^{(\ell)} \right\| \leq \sigma_{\max^*}^{L-\ell} \left\| \delta^{(L)} \right\|$$

$$\left\| \delta^{(\ell)} \right\| \geq \sigma_{\min^*}^{L-\ell} \left\| \delta^{(L)} \right\|$$

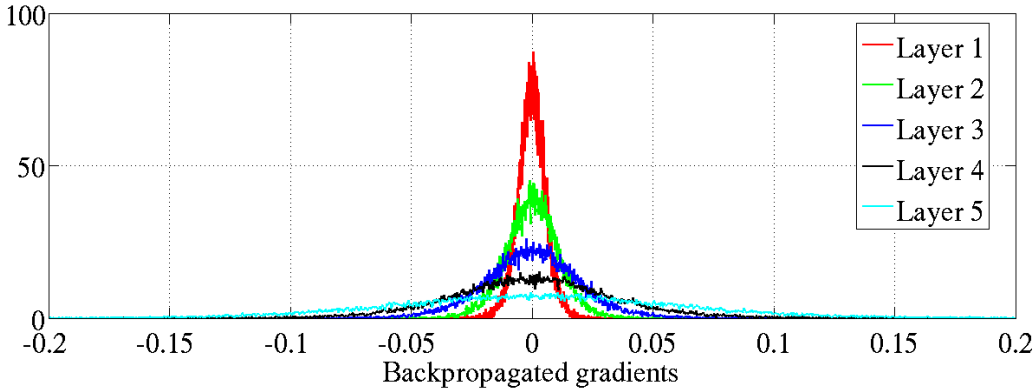


Gradientes Desvanecientes (y Explosivos)

backward pass

$$\left\|\delta^{(\ell)}\right\| \leq \sigma_{\max*}^{L-\ell} \left\|\delta^{(L)}\right\|$$

$$\left\|\delta^{(\ell)}\right\| \geq \sigma_{\min*}^{L-\ell} \left\|\delta^{(L)}\right\|$$



Understanding the difficulty of training deep feedforward neural networks

Xavier Glorot **Yoshua Bengio**
DIRO, Université de Montréal, Montréal, Québec, Canada

Abstract

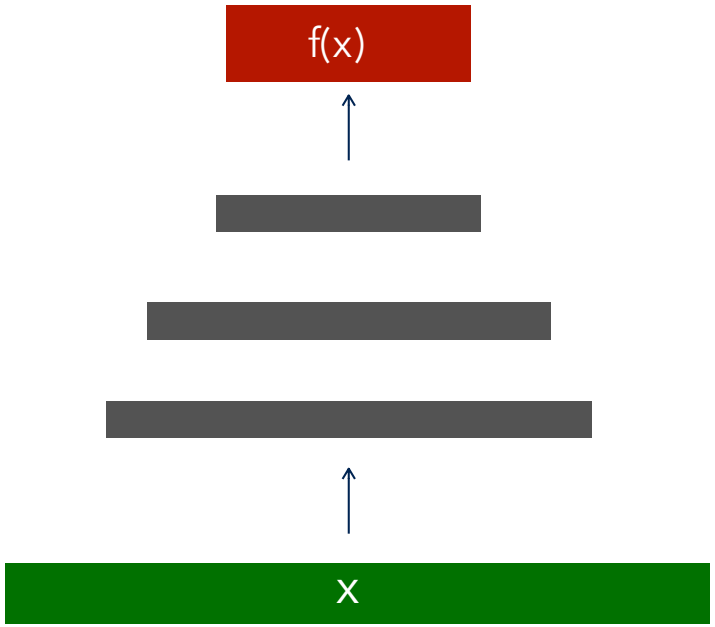
Whereas before 2006 it appears that deep multi-layer neural networks were not successfully trained, since then several algorithms have been shown to successfully train them, with experimental results showing the superiority of deeper vs less deep architectures. All these experimen-

learning methods for a wide array of *deep architectures*, including neural networks with many hidden layers (Vincent et al., 2008) and graphical models with many levels of hidden variables (Hinton et al., 2006), among others (Zhu et al., 2009; Weston et al., 2008). Much attention has recently been devoted to them (see (Bengio, 2009) for a review), because of their theoretical appeal, inspiration from biology and human cognition, and because of empirical

Xavier Glorot et al. Understanding the difficulty of training deep feedforward neural networks." *International Conference on Artificial Intelligence and Statistics*. 2010.

Tasas Adaptativas

- Una **idea sencilla** para manejar el problema consiste en usar **tasas de aprendizaje variables por capa**: más altas paras las primeras capas y más pequeñas para las capas más profundas. ¿Cómo exactamente?



$$W_{ij}^{(\ell)} \leftarrow W_{ij}^{(\ell)} - \eta_t^{(\ell)} \frac{\partial E}{W_{ij}^{(\ell)}}$$

$$\eta_t^{(\ell)} \propto \frac{1}{\| \frac{\partial E}{W^{(\ell)}} \|}$$

Tasas Adaptativas en Optimización Convexa

- Recordemos que SGD es usado no sólo en redes neuronales!
- Aún en modelos clásicos (shallow) tener una tasa puede ser problemático. Consideremos e.g. el caso de un sencillo regresor logístico $y = \sigma(\mathbf{w}^T \mathbf{x} - b)$

SGD

$$\mathbf{w}^{(0)} \leftarrow 0$$
$$\mathbf{x}^{(i)} \leftarrow \text{sample}(S)$$
$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta_t E'(\cdot)\sigma'(\cdot)\mathbf{x}^{(i)}$$

Journal of Machine Learning Research 12 (2011) 2121-2159 Submitted 3/10; Revised 3/11; Published 7/11

Adaptive Subgradient Methods for Online Learning and Stochastic Optimization*

John Duchi JDUCHI@CS.BERKELEY.EDU
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720 USA

Elad Hazan EHAZAN@IE.TECHNION.AC.IL
Technion - Israel Institute of Technology
Technion City
Haifa, 32000, Israel

Yoram Singer SINGER@GOOGLE.COM
Google
1600 Amphitheatre Parkway
Mountain View, CA 94043 USA

John Duchi et al. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research* 12.Jul (2011): 2121-2159.

Tasas Adaptativas en Optimización Convexa

SGD

$$\mathbf{w}^{(0)} \leftarrow 0$$

$$\mathbf{x}^{(i)} \leftarrow \text{sample}(S)$$

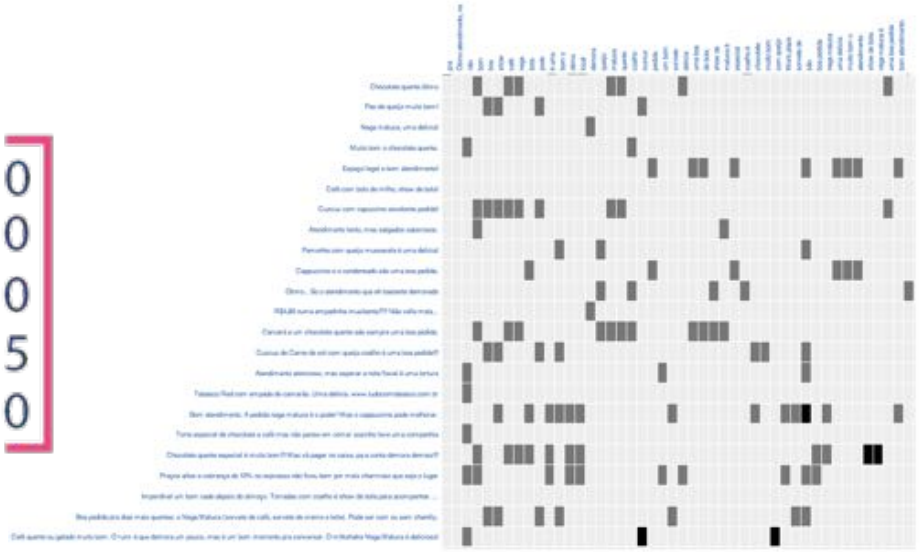
$$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \eta_t E'(\cdot)\sigma'(\cdot)\mathbf{x}^{(i)}$$

sparse x

¿qué sucede si un atributo x_i es disperso? e.g. bag of words.

attributes

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 8 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 \\ 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix}$$

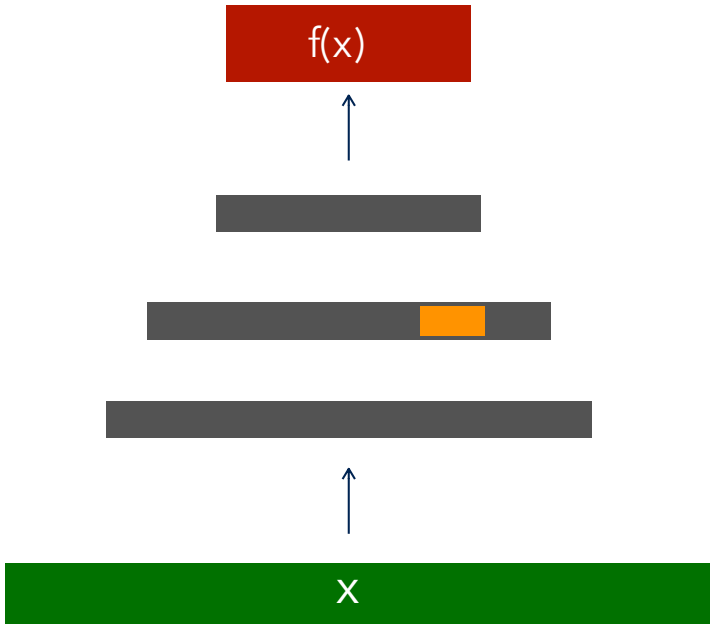


text data

John Duchi et al. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research* 12.Jul (2011): 2121-2159.

Tasas Adaptativas

- No sólo debiésemos tener tasas diferentes por capa sino que por cada peso de cada neurona!



$$W_{ij}^{(\ell)} \leftarrow W_{ij}^{(\ell)} - \eta_{ijt}^{(\ell)} \frac{\partial E}{W_{ij}^{(\ell)}}$$

$$\eta_{ijt}^{(\ell)} \propto \frac{1}{\| \frac{\partial E}{W_{ij}^{(\ell)}} \|}$$

AdaGrad

Adaptive Gradient es la más simple de las opciones populares hoy en día

$$g_i^{(t)} \leftarrow \text{Estimar } \nabla_i E(\theta^{(t)})$$

$$r_i^{(t)} \leftarrow \sum_{\tau=1}^{t} \|g_i^{(\tau)}\|^2$$

$$\eta_i^{(t)} \leftarrow \frac{\eta}{\sqrt{r_i^{(t)} + \epsilon}}$$

$$\theta_i^{(t)} \leftarrow \theta_i^{(t)} - \eta_i^{(t)} g_i^{(t)}$$

John Duchi et al. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research* 12.Jul (2011): 2121-2159.

AdaGrad

En forma vectorial:

$$g^{(t)} \leftarrow \text{Estimar } \nabla_\theta E(\theta^{(t)})$$

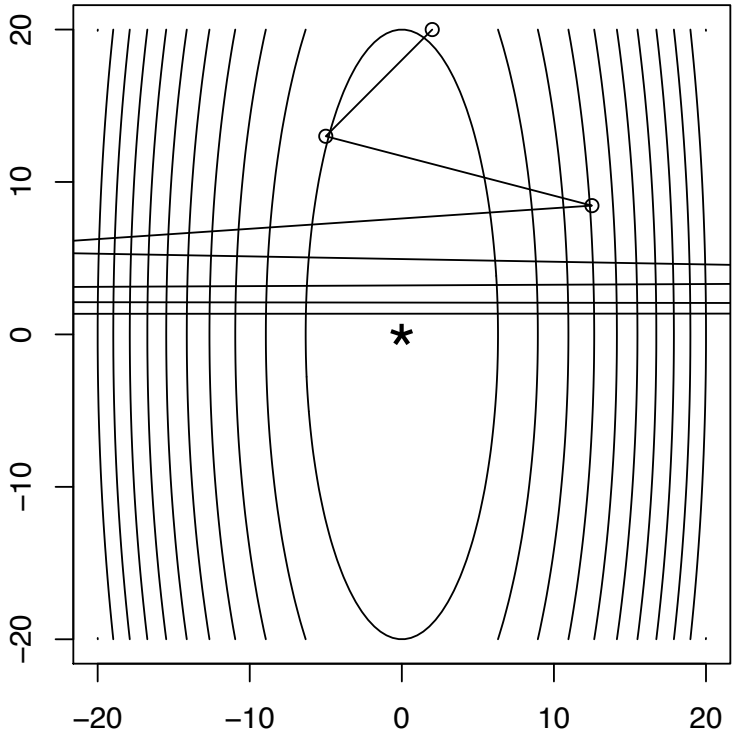
$$r^{(t)} \leftarrow \sum_{\tau=1}^{t} g^{(\tau)} \odot g^{(\tau)}$$

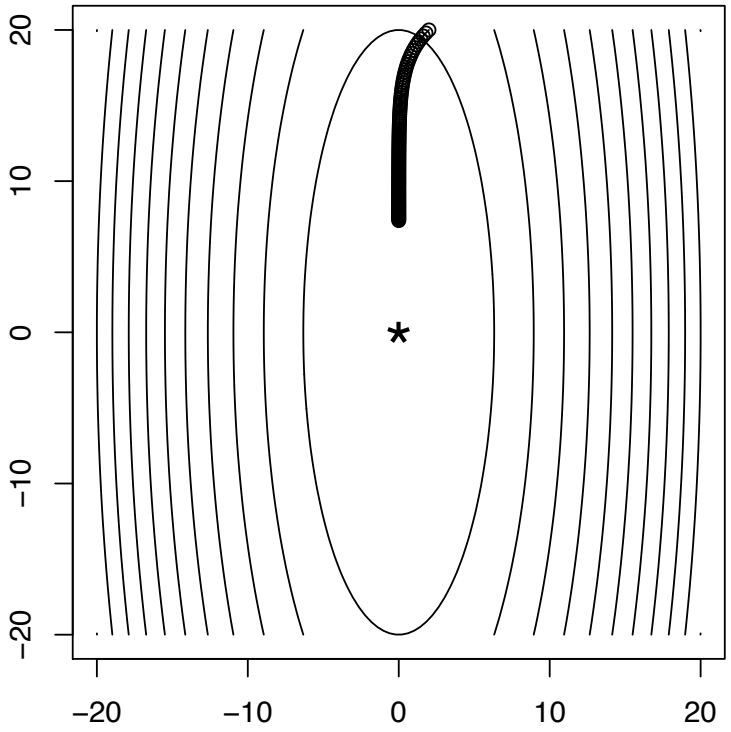
$$\theta^{(t)} \leftarrow \theta^{(t)} - \frac{\eta}{\sqrt{r^{(t)} + \epsilon}} \odot g^{(t)}$$

John Duchi et al. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research* 12.Jul (2011): 2121-2159.

AdaGrad

- Esto puede verse como un método que intenta (de manera aproximada) normalizar la geometría de la f.o.





Porqué Funciona AdaGrad?

Recordemos la motivación de GD y SGD:

$$E = \frac{1}{n}\sum_x L(f_\theta(x), y) \approx \mathbb{E}(L(f_\theta(x), y))$$

$$E(\theta^{(t+1)}) \approx E(\theta^{(t)}) + \nabla_\theta E^T \left(\theta^{(t+1)} - \theta^{(t)}\right)$$

$$E(\theta^{(t+1)}) - E(\theta^{(t)}) \approx \eta \nabla_\theta E^T d^{(t)}$$

$$d^{(t)} = -\nabla_\theta E(\theta^{(t)})$$

$$\nabla_\theta E(\theta^{(t)}) \approx g_x = \nabla_\theta L(f_\theta(x), y)$$

Porqué Funciona AdaGrad?

Por supuesto, una mejor aproximación se obtiene considerando información de segundo orden sobre la f.o.

$$E = \frac{1}{n}\sum_x L(f_\theta(x), y) \approx \mathbb{E}(L(f_\theta(x), y))$$

$$E(\theta^{(t+1)}) \approx E(\theta^{(t)}) + \nabla_\theta E^T \left(\theta^{(t+1)} - \theta^{(t)}\right) + \left(\theta^{(t+1)} - \tfrac{1}{2}\theta^{(t)}\right)^T H(\theta^{(t)}) \left(\theta^{(t+1)} - \theta^{(t)}\right)$$

$$E(\theta^{(t+1)}) - E(\theta^{(t)}) \approx \eta \nabla_\theta E^T d^{(t)} + \frac{\eta^2}{2} d^{(t)T} H d^{(t)}$$

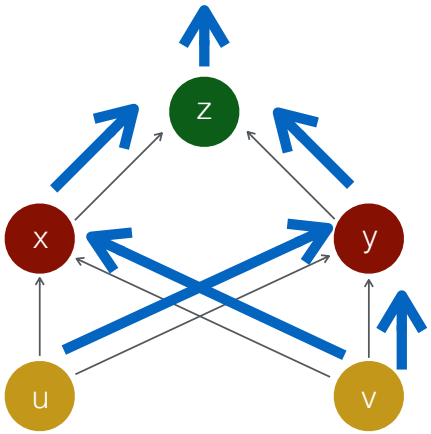
$$\eta d^{(t)} = -H^{-1}\nabla_\theta E(\theta^{(t)})$$

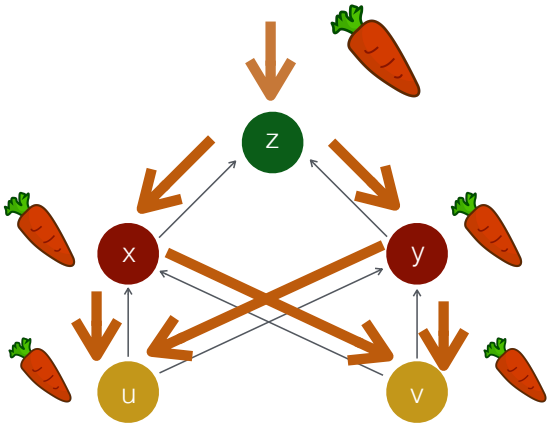
$$H_{ij} = \frac{\partial^2 E}{\partial\theta_i \partial\theta_j}$$

$$\theta^{(t+1)} = \theta^{(t)} + \eta d^{(t)} = \theta^{(t)} - H^{-1}\nabla_\theta E(\theta^{(t)})$$

Suavidad & Cota para la Mejora

Lamentablemente, en problemas no convexos con muchas variables, estimar H suele ser infactible computacionalmente. En nuestro caso, tendríamos que inventar un Backprop de segundas derivadas. Aún si pudiésemos computar esta matriz, sólo almacenarla es inviable.





Porqué Funciona AdaGrad?

Podemos intentar aproximar la Hessiana

$$p_\theta(x) \propto \exp(-L(f(x), y))$$

$$E = -\frac{1}{n}\sum_x \log p_\theta(x) \quad \approx -\mathbb{E}_{p_\theta} \log p_\theta(x)$$

$$\begin{aligned}
-\nabla^2_\theta \mathbb{E}_{p_\theta} \log p_\theta(x) &= -\mathbb{E}_{p_\theta} \nabla^2_\theta \log p_\theta(x) \\
&= -\boldsymbol{F} = \mathbb{E}_{p_\theta} \nabla_\theta \log p_\theta(x) \nabla_\theta \log p_\theta(x)^T \\
&= \mathbb{E}_{p_\theta} \nabla_\theta E(f(x), y) \nabla_\theta E(f(x), y)^T \\
&\approx \frac{1}{n}\sum_x g_x(\theta) g_x(\theta)^T
\end{aligned}$$

Aproximación de Gauss-Newton

Resuelve en parte el problema del cálculo (de todos modos habría que invertirla), no del almacenamiento.

Porqué Funciona AdaGrad?

Podemos intentar aproximar la Hessiana de modo que sea fácil almacenarla e invertirla.

$$H \approx \frac{1}{n} \sum_x g_x(\theta) g_x(\theta)^T$$

Aproximación de Gauss-Newton

$$H \approx \text{diag}\left(\frac{1}{n} \sum_x g_x(\theta) g_x(\theta)^T \right)$$

$$= \text{diag}\left(\frac{1}{n} \sum_x g_x(\theta) \odot g_x(\theta) \right)$$

Aproximación de Jacobi

Porqué Funciona AdaGrad?

Con esta aproximación, el paso quedaría como

$$\theta^{(t)} = \theta^{(t)} - H^{-1}\nabla_\theta E(\theta^{(t)})$$
$$\approx \theta^{(t)} - H^{-1} g_x$$

$$\nabla_\theta E(\theta^{(t)}) \approx g_x = \nabla_\theta L(f_\theta(x), y)$$

$$H^{-1} g_x \approx \text{diag}^{-1}\left(\frac{1}{n}\sum_x g_x(\theta) \odot g_x(\theta)\right)$$
$$= \frac{n}{\sum_x g_x(\theta) \odot g_x(\theta)} \odot g_x$$

Porqué Funciona AdaGrad?

Concluimos que AdaGrad es casi un método quasi Newton con decaimiento progresivo de la tasa (casi porque hay una raíz cuadrada demás)

Obtenido

$$g_i^{(t)} \leftarrow \text{Estimar } \nabla_i E(\theta^{(t)})$$

$$r^{(t)} \leftarrow \frac{1}{t} \sum_{\tau=1}^{t} g^{(\tau)} \odot g^{(\tau)}$$

$$\eta_t = \eta / t$$

$$\theta^{(t)} \leftarrow \theta^{(t)} - \frac{\eta_t}{r^{(t)} + \epsilon} \odot g^{(t)}$$

AdaGrad

$$g^{(t)} \leftarrow \text{Estimar } \nabla_\theta E(\theta^{(t)})$$

$$r^{(t)} \leftarrow \sum_{\tau=1}^{t} g^{(\tau)} \odot g^{(\tau)}$$

$$\theta^{(t)} \leftarrow \theta^{(t)} - \frac{\eta}{\sqrt{r^{(t)} + \epsilon}} \odot g^{(t)}$$

Porqué Funciona AdaGrad?

Realmente necesitamos la raíz?

Second-order Information in First-order Optimization Methods

Yuzheng Hu* **Licong Lin*** **Shange Tang***
Peking University, School of Mathematical Science
Beijing,China
{1700010613,llc2000,sangertang}@pku.edu.cn

Abstract

First-order optimization methods play a central role in modern deep learning. There are various types of first-order methods, such as GD, SGD, Momentum, NAG, Adagrad, RMSprop, Adam, etc. An interesting fact is that some algorithms are more favored in practice (SGD and Adam, for example) and experiments have revealed that these algorithms are indeed significantly better than others. Although there are many heuristic explanations trying to interpret this phenomenon, theoretical guarantees are still lacking.

In this paper, we aim to uncover part of the mystery from a novel perspective. We claim that some first-order methods, though Hessian-free, are actually utilizing the second-order information of the loss function while training. More specifically, we demonstrate that a powerful method, Nesterov Accelerated Gradient, explores second order information by making use of the difference of past and current gradients and thus accelerates the training. Furthermore, we rigorously prove that adaptive methods such as Adagrad and Adam can actually be regarded as relaxations of natural gradient descent—a well-known second-order technique in computation statistics. Based on this observation, we design a new algorithm, AdaSqrt, which has better performance on MNIST and CIFAR10 even compared to the most

Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients

Lukas Balles [1] **Philipp Hennig** [1]

Abstract

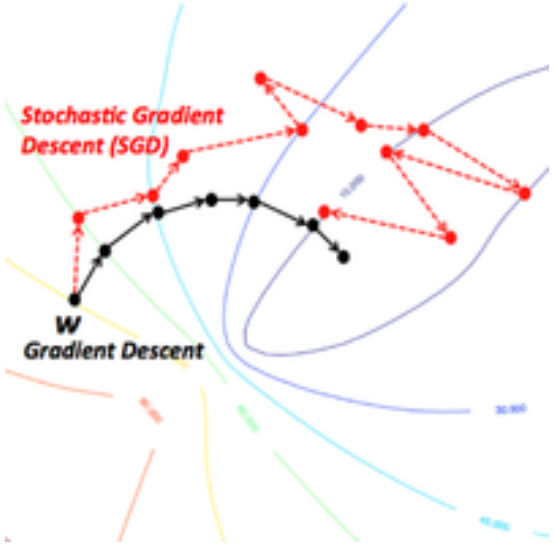
The ADAM optimizer is exceedingly popular in the deep learning community. Often it works very well, sometimes it doesn't. Why? We interpret ADAM as a combination of two aspects: for each weight, the update direction is determined by the *sign* of stochastic gradients, whereas the update magnitude is determined by an estimate of their *relative variance*. We disentangle these two aspects and analyze them in isolation, gaining insight into the mechanisms underlying ADAM. This analysis also extends recent results on ad-

which is a random variable with $\mathbf{E}[g(\theta)] = \nabla\mathcal{L}(\theta)$. An important quantity for this paper will be the (element-wise) variances of the stochastic gradient, which we denote by $\sigma_i^2(\theta) := \mathbf{var}[g(\theta)_i]$.

Widely-used stochastic optimization algorithms are stochastic gradient descent (SGD, Robbins & Monro, 1951) and its momentum variants (Polyak, 1964; Nesterov, 1983). A number of methods popular in deep learning choose per-element update magnitudes based on past gradient observations. Among these are ADAGRAD (Duchi et al., 2011), RMSPROP (Tieleman & Hinton, 2012), ADADELTA (Zeiler,

Porqué Funciona AdaGrad?

Otra explicación pasa por recordar que nuestras estimaciones del gradiente de la f.o. son estocásticas y que la falta de esfericidad en la varianza de esas estimaciones afecta negativamente la convergencia



$$E = \frac{1}{n}\sum_x L(f_\theta(x), y) \approx \mathbb{E}(L(f_\theta(x), y))$$

$$g_x = \nabla_\theta L(f_\theta(x), y)$$

$$\mathbb{E}(g_x) = \nabla_\theta \mathbb{E}(L(f_\theta(x), y))$$

Asumiendo:

$$g_x = (1 + \xi)\nabla_\theta E, \quad \xi \sim \mathcal{N}(0, \sigma^2)$$

Obtenemos:

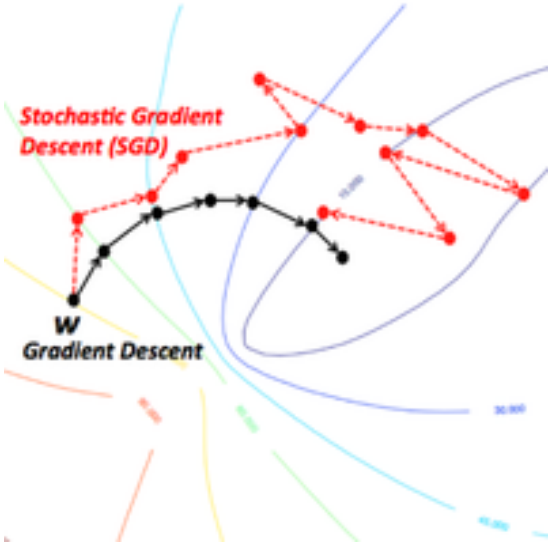
$$\mathbb{E}(g_x - \mathbb{E}(g_x)(g_x - \mathbb{E}(g_x))^T = \sigma^2 \nabla_\theta E \nabla_\theta E^T$$

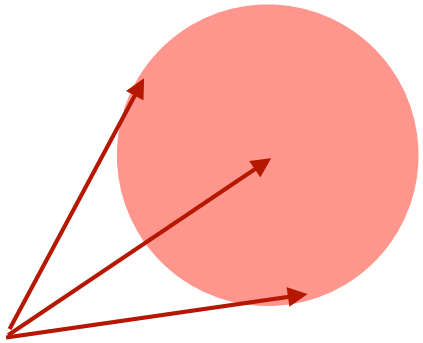
Porqué Funciona AdaGrad?

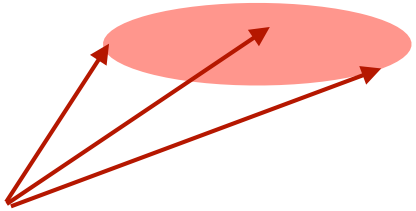
La falta de esfericidad en la varianza de esas estimaciones afecta negativamente la convergencia

$$\mathbb{E}(g_x - \mathbb{E}(g_x)(g_x - \mathbb{E}(g_x))^T = \sigma^2 \nabla_\theta E \nabla_\theta E^T$$

$$:= \sigma^2 \Sigma^2$$







$$P(g_x^T \boldsymbol{e}_i > \nabla E^T \boldsymbol{e}_i + \epsilon) = P(g_x^T \boldsymbol{e}_i - \mathbb{E}(g_x^T \boldsymbol{e}_i) > \epsilon)$$

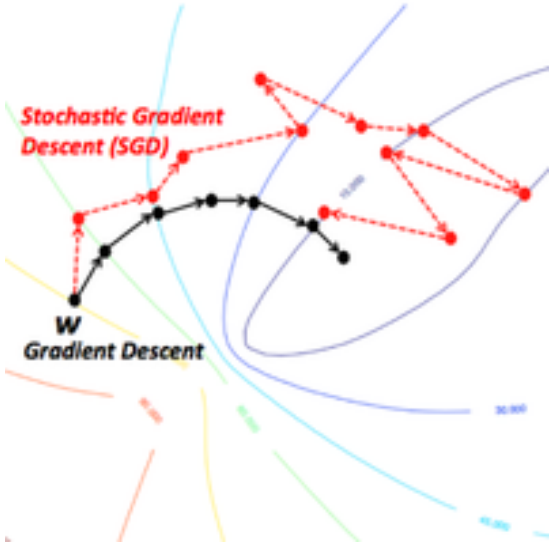
$$\leq P(|g_x^T \boldsymbol{e}_i - \mathbb{E}(g_x^T \boldsymbol{e}_i)| > \epsilon) \leq \frac{\sigma^2 \Sigma_{ii}^2}{\epsilon^2}$$

(*) Chebyshev

Porqué Funciona AdaGrad?

Una solución es escalar a variable aleatoria (gradiente), de modo que su varianza sea más esférica

$$\mathbb{E}(g_x - \mathbb{E}(g_x)(g_x - \mathbb{E}(g_x))^T = \sigma^2 \nabla_\theta E \nabla_\theta E^T$$



$$\tilde{g}_x = \Sigma^{-1} g_x$$

$$\mathbb{E}(\tilde{g}_x) = \Sigma^{-1} \nabla_\theta E$$

$$\mathbb{E}(\tilde{g}_x - \mathbb{E}(\tilde{g}_x)(\tilde{g}_x - \mathbb{E}(\tilde{g}_x))^T$$

$$= \sigma^2 \Sigma^{-1} \Sigma^2 \Sigma^{-1} = \sigma^2 I$$

(*) Chebyshev

Porqué Funciona AdaGrad?

No es difícil demostrar que a dirección sigue siendo de descenso

$$E(\theta^{(t+1)}) \approx E(\theta^{(t)}) + \nabla_\theta E^T \left(\theta^{(t+1)} - \theta^{(t)}\right) \qquad d^{(t)} = \tilde{g}_x = \Sigma^{-1} g_x$$

$$E(\theta^{(t+1)}) - E(\theta^{(t)}) \approx \eta \nabla_\theta E^T d^{(t)} \qquad \mathbb{E}(\tilde{g}_x) = \Sigma^{-1} \nabla_\theta E$$

$$\begin{aligned}
\mathbb{E}\left(E(\theta^{(t+1)}) - E(\theta^{(t)})\right) &\approx \eta \nabla_\theta E^T \mathbb{E}(d^{(t)}) \\
&= -\eta \nabla_\theta E^T \Sigma^{-1} \nabla_\theta E \\
&= -\eta \mathrm{Tr}\left(\nabla_\theta E^T \Sigma^{-1} \nabla_\theta E\right) \\
&= -\eta \mathrm{Tr}\left(\Sigma^{-1} \nabla_\theta E \nabla_\theta E^T\right) \\
&= -\eta \sigma^2 \mathrm{Tr}\left(\Sigma^{-1} \Sigma^2\right) \\
&= -\eta \sigma^2 \mathrm{Tr}\left(\Sigma\right) = -\eta \sigma^2 \mathrm{Tr}\left(\Sigma^2\right)^{1/2} = -\eta \sigma^2 \|\nabla_\theta E\|
\end{aligned}$$

Porqué Funciona AdaGrad?

¿Cómo estimar la curvatura eficientemente?

$$\Sigma^2 = \nabla_\theta E \nabla_\theta E^T \qquad E = \frac{1}{n}\sum_x L(f_\theta(x), y) \approx \mathbb{E}(L(f_\theta(x), y))$$

$$\Sigma^2 \approx \nabla_\theta E \odot \nabla_\theta E \qquad \text{Aproximación de Jacobi}$$

$$\approx \frac{1}{n}\sum_x g_x \odot g_x \qquad \text{Montecarlo Básico}$$

AdaGrad

Adaptive Gradient es la más simple de las opciones populares hoy en día

$$g_i^{(t)} \leftarrow \text{Estimar } \nabla_i E(\theta^{(t)})$$

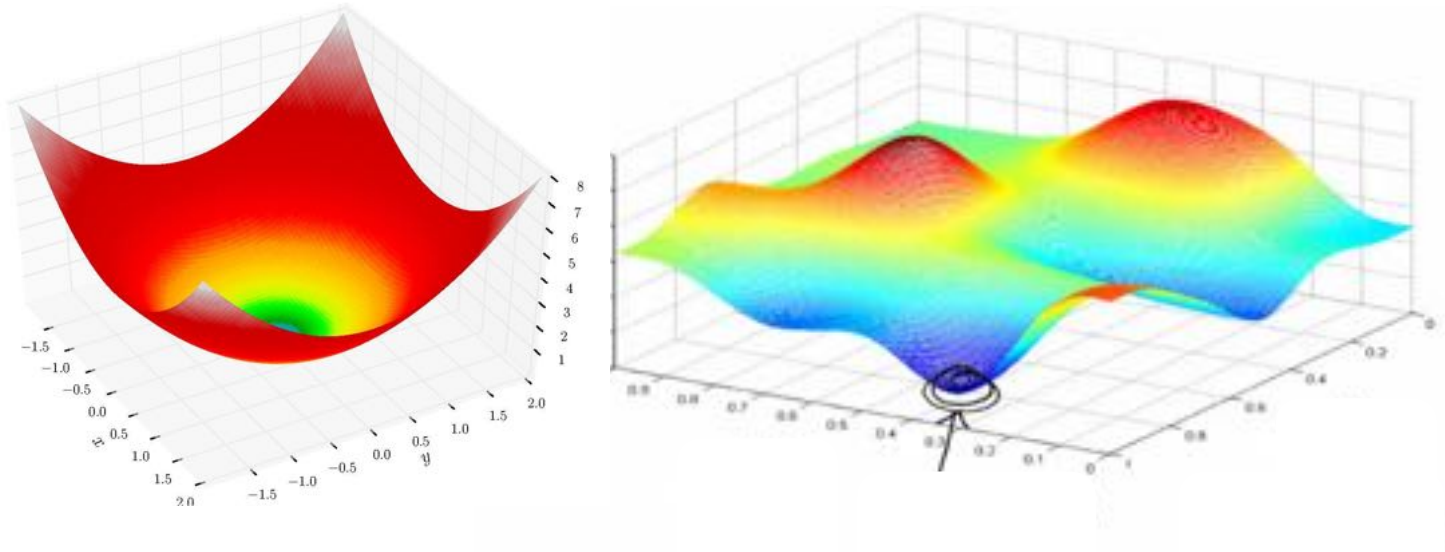
$$r_i^{(t)} \leftarrow \sum_{\tau=1}^{t} \|g_i^{(\tau)}\|^2$$

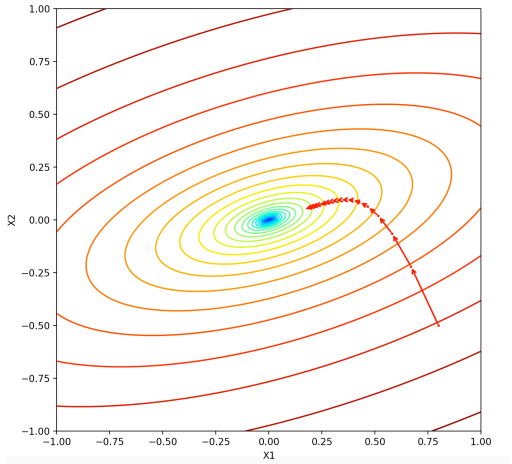
$$\eta_i^{(t)} \leftarrow \frac{\eta}{\sqrt{r_i^{(t)} + \epsilon}}$$

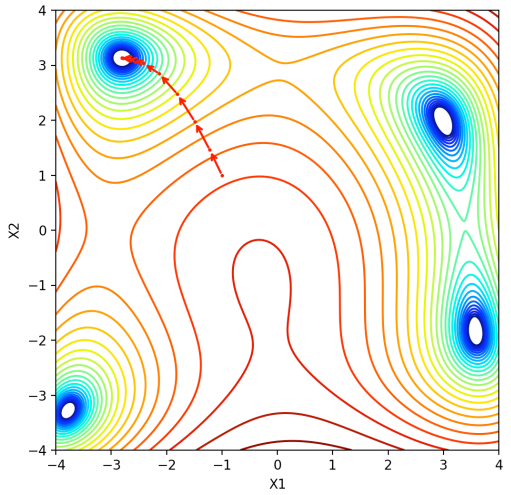
$$\theta_i^{(t)} \leftarrow \theta_i^{(t)} - \eta_i^{(t)} g_i^{(t)}$$

John Duchi et al. "Adaptive subgradient methods for online learning and stochastic optimization." *Journal of Machine Learning Research* 12.Jul (2011): 2121-2159.

RMS Prop







RMS Prop

Root mean squared error propagation (RMS Prop) es una algoritmo de tasa adaptativa que trata se superar las dificultades que aparecen en el caso no-convexo.

RMSProp

$$g_i^{(t)} \leftarrow \text{Estimar } \nabla_i E(\theta^{(t)})$$

$$r_i^{(t)} \leftarrow \gamma r_i^{(t-1)} + (1-\gamma)\|g_i^{(t)}\|^2$$

$$\eta_i^{(t)} \leftarrow \frac{\eta}{\sqrt{r_i^{(t)}+\epsilon}}$$

$$\theta_i^{(t)} \leftarrow \theta_i^{(t)} - \eta_i^{(t)} g_i^{(t)}$$

AdaGrad

$$g_i^{(t)} \leftarrow \text{Estimar } \nabla_i E(\theta^{(t)})$$

$$r_i^{(t)} \leftarrow \sum_{\tau=1}^{t} \|g_i^{(\tau)}\|^2$$

$$\eta_i^{(t)} \leftarrow \frac{\eta}{\sqrt{r_i^{(t)}+\epsilon}}$$

$$\theta_i^{(t)} \leftarrow \theta_i^{(t)} - \eta_i^{(t)} g_i^{(t)}$$

RMS Prop

En forma vectorial:

$$g^{(t)} \leftarrow \text{Estimar } \nabla_\theta E(\theta^{(t)})$$

$$r^{(t)} \leftarrow \gamma r^{(t)} + (1-\gamma) g^{(\tau)} \odot g^{(\tau)}$$

$$\theta^{(t)} \leftarrow \theta^{(t)} - \frac{\eta}{\sqrt{r^{(t)} + \epsilon}} \odot g^{(t)}$$

RMS Prop

Media móvil para el segundo momento

$$
\begin{aligned}
r^{(t)} &= \gamma r^{(t-1)} + (1-\gamma) g^{(t)} \odot g^{(t)} \\
&= \gamma^2 r^{(t-2)} + \gamma(1-\gamma) g^{(t-1)} \odot g^{(t-1)} + (1-\gamma) g^{(t)} \odot g^{(t)} \\
&= \gamma^3 r^{(t-3)} + \gamma^2(1-\gamma) g^{(t-2)} \odot g^{(t-2)} + \gamma(1-\gamma) g^{(t-1)} \odot g^{(t-1)} + (1-\gamma) g^{(t)} \odot g^{(t)}
\end{aligned}
$$

Valores típicos son $\eta = 0.01$ y $\gamma = 0.9$

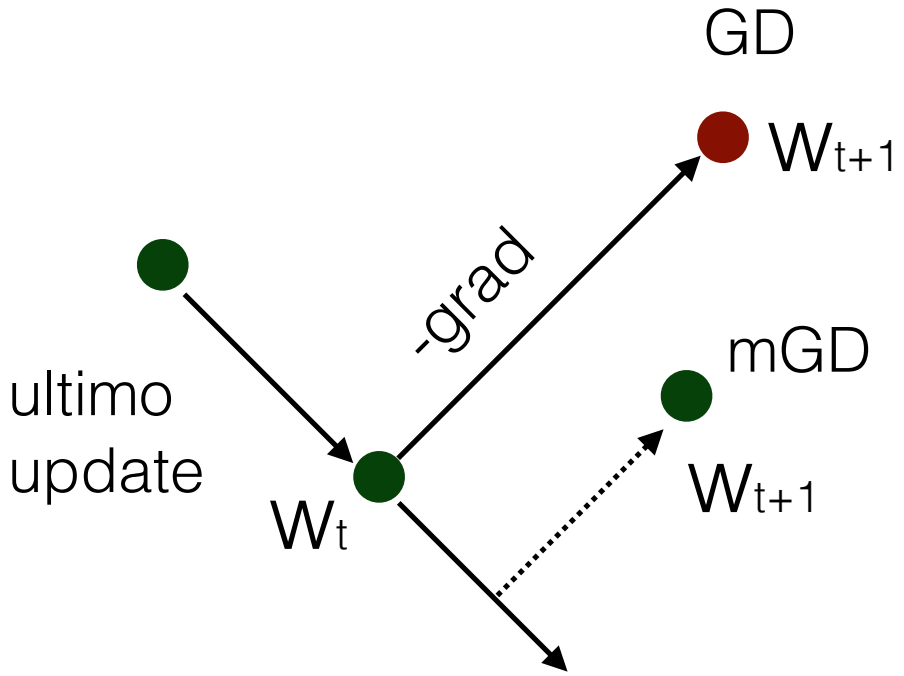
Momentum Clásico (o de Polyak)

¿Porqué no usar esta idea también para estimar el primer momento del gradiente?

$$g^{(t)} \leftarrow \text{Estimar } \nabla_\theta E(\theta^{(t)})$$

$$v^{(t)} \leftarrow \beta v^{(t)} - \eta_t g^{(t)}$$

$$\theta^{(t)} \leftarrow \theta^{(t)} + v^{(t)}$$



Polyak, Boris T. "Some methods of speeding up the convergence of iteration methods." *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964): 1-17.

Momentum

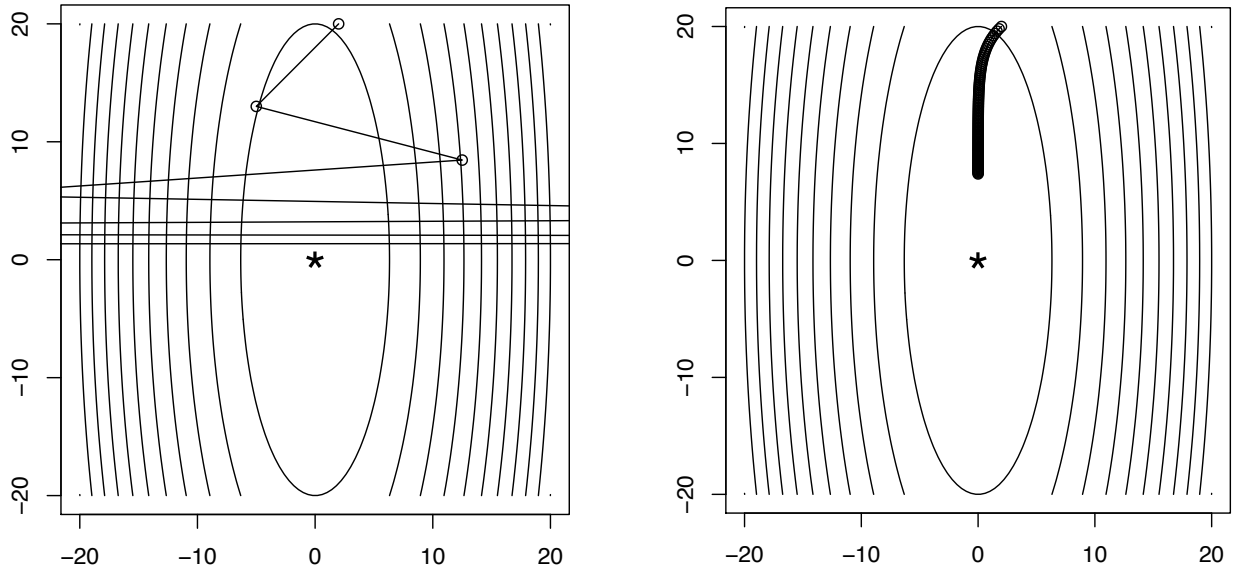
¿Porqué no usar esta idea también para estimar el primer momento del gradiente?

$$
\begin{aligned}
v^{(t)} &= \beta v^{(t-1)} - \eta g^{(t)} \\
&= \beta^2 v^{(t-2)} - \beta\eta g^{(t-1)} - \eta g^{(t)} \\
&= \beta^3 v^{(t-3)} - \beta^2\eta g^{(t-2)} - \beta\eta g^{(t-1)} - \eta g^{(t)}
\end{aligned}
$$

Momentum

Intutición: Por definición, en las direcciones con poca curvatura del espacio de búsqueda, el gradiente cambia lentamente. La media móvil permite que esas direcciones se "acumulen" en la velocidad y terminen teniendo un efecto en la actualización.

$$v^{(t)} = \beta^3 v^{(t-3)} - \beta^2 \eta g^{(t-2)} - \beta \eta g^{(t-1)} - \eta g^{(t)}$$



Momentum Clásico (o de Polyak)

Polyak mostró que eligiendo apropiadamente las constantes (*) este método converge $\sqrt{R/2}$ veces más rápido para funciones cuadráticas con número de condición R.

(*) Polyak usa constantes fijas para la velocidad y el último gradiente.

Sutskever, Ilya, et al. "On the importance of initialization and momentum in deep learning." *International conference on machine learning*. 2013.

Polyak, Boris T. "Some methods of speeding up the convergence of iteration methods." *USSR Computational Mathematics and Mathematical Physics* 4.5 (1964): 1-17.

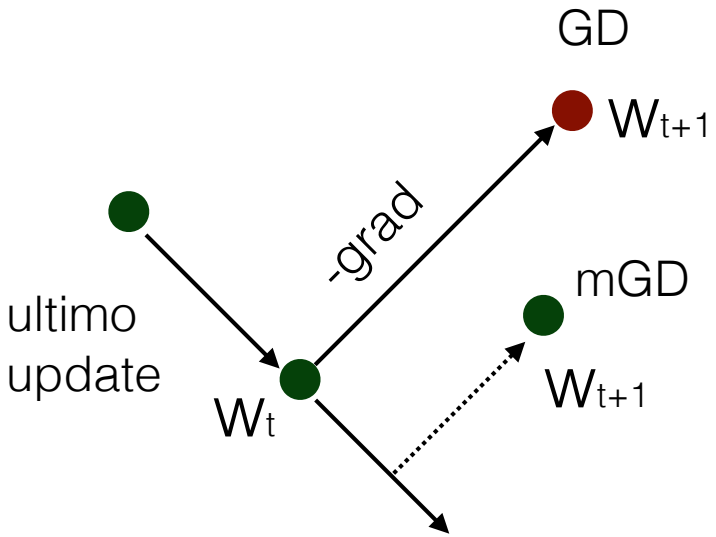
Momentum de Nesterov

Es posible mejorar este resultado. Notemos que el método anterior estima el último gradiente en la última solución encontrada.

$$g^{(t)} \leftarrow \text{Estimar } \nabla_\theta E(\theta^{(t)})$$

$$v^{(t)} \leftarrow \beta v^{(t)} - \eta_t g^{(t)}$$

$$\theta^{(t)} \leftarrow \theta^{(t)} + v^{(t)}$$



Y. Nesterov, “A method of solving a convex programming problem with convergence rate O(1/k2),” in Soviet Mathematics Doklady, vol. 27, no. 2, 1983, pp. 372–376.

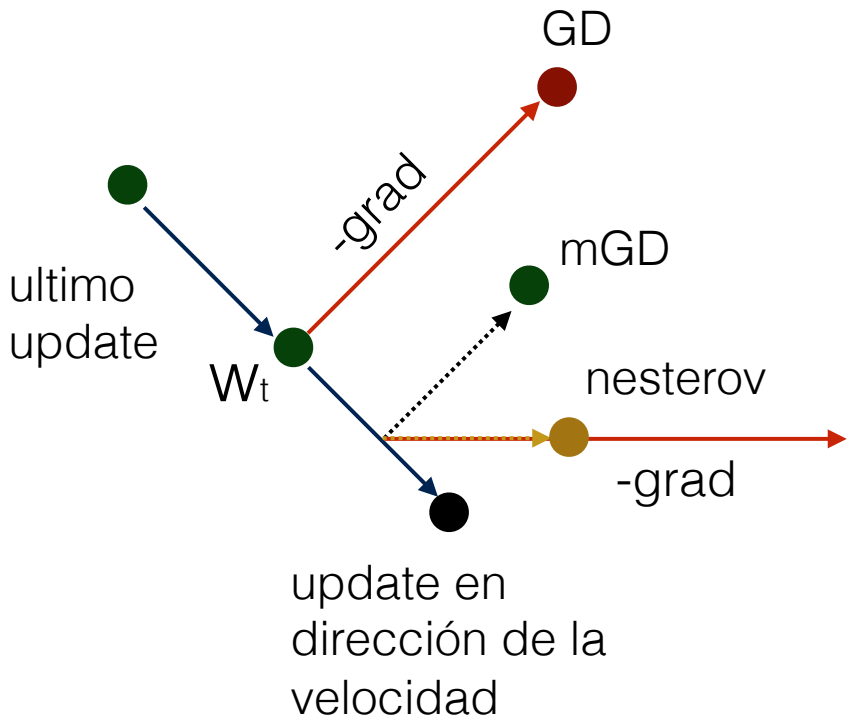
Momentum de Nesterov

Como tenemos a velocidad anterior, es posible estimar la próxima posición en la que estaremos

$$g^{(t)} \leftarrow \text{Estimar } \nabla_\theta E(\theta^{(t)} + \beta v^{(t)})$$

$$v^{(t)} \leftarrow \beta v^{(t)} - \eta_t g^{(t)}$$

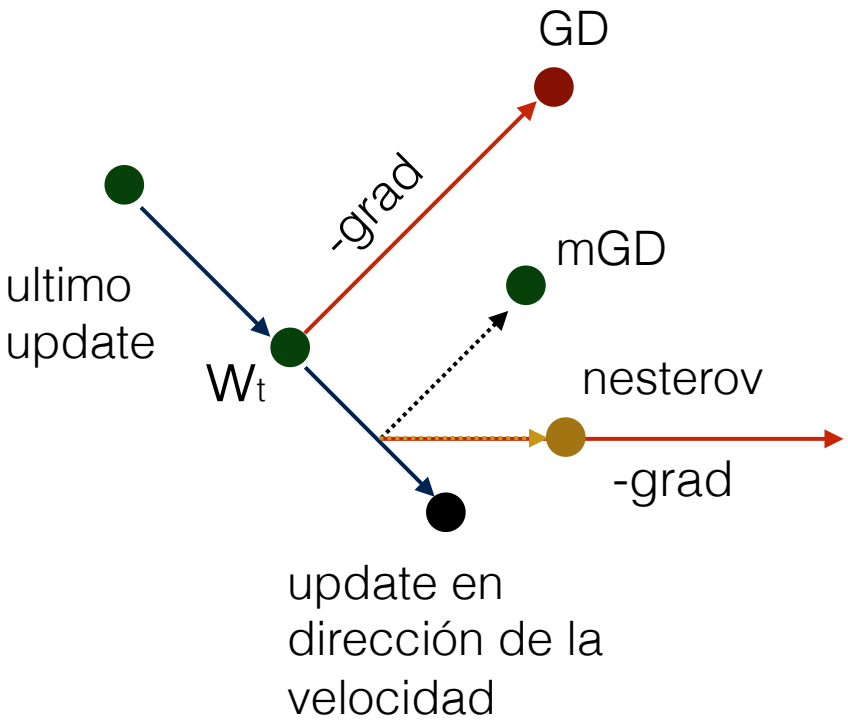
$$\theta^{(t)} \leftarrow \theta^{(t)} + v^{(t)}$$

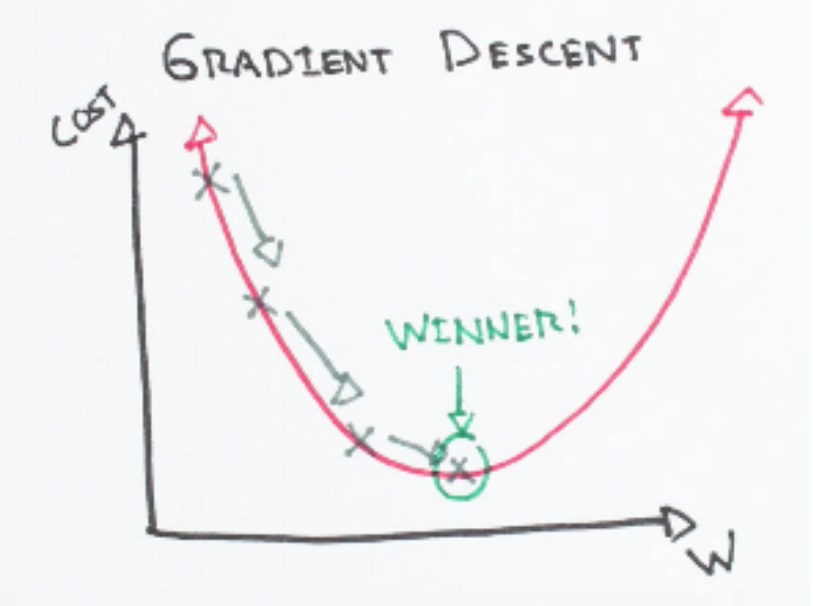


Y. Nesterov, “A method of solving a convex programming problem with convergence rate O(1/k2),” in Soviet Mathematics Doklady, vol. 27, no. 2, 1983, pp. 372–376.

Momentum de Nesterov

Intuición: es posible que el movimiento no nos lleve en a dirección correcta, estimando el gradiente en una posición más cercana a donde terminaremos tenemos la posibilidad de darnos cuenta y eventualmente revertir el paso sin necesidad de "gastar" una iteración más.





Momentum de Nesterov

Asumiendo convexidad y gradientes suaves (L-Lipschitz) es posible mostrar

1. Después de $\mathcal{O}(1/\sqrt{\epsilon})$ iteraciones, el método obtiene un valor del objetivo ta que $E(\theta^{(t)}) - E(\theta^*) < \epsilon$. Sin momentum, se logra esa mejora después de $\mathcal{O}(1/\epsilon)$ iteraciones.

2. No existe un método de primer orden que converja más rápido bajo los mismos supuestos.

Y. Nesterov, “A method of solving a convex programming problem with convergence rate O(1/k2),” in Soviet Mathematics Doklady, vol. 27, no. 2, 1983, pp. 372–376.

Adam

Adaptive Moment Estimation (Adam) es esencialmente RMS Prop combinado con momentum

$$g^{(t)} \leftarrow \text{Estimar } \nabla_\theta E(\theta^{(t)})$$

$$v^{(t)} \leftarrow \beta r^{(t-1)} + (1-\beta) g^{(t)}$$

$$r^{(t)} \leftarrow \gamma r^{(t-1)} + (1-\gamma) g^{(t)} \odot g^{(t)}$$

$$\theta^{(t)} \leftarrow \theta^{(t)} - \frac{\eta_t}{\sqrt{r^{(t)} + \epsilon}} \odot v^{(t)}$$

Kingma, Diederik P., and Jimmy Lei Ba. "Adam: A method for stochastic gradient descent." *ICLR: International Conference on Learning Representations*. 2015.

Adam

A esta combinación se le hace una modificación para obtener estimadores insesgados del primer y segundo momento, que por la media móvil quedan sesgados (sobre todo en las primeras iteraciones).

$$
\begin{aligned}
v^{(t)} &= \beta v^{(t-1)} + (1-\beta) g^{(t)} \\
&= \beta^2 v^{(t-2)} + \beta(1-\beta) g^{(t-1)} + (1-\beta) g^{(t)} \\
&= \beta^3 v^{(t-3)} + \beta^2(1-\beta) g^{(t-2)} + \beta(1-\beta) g^{(t-1)} + (1-\beta) g^{(t)} \\
&\quad \dots
\end{aligned}
$$

$$
v^{(t)} = (1-\beta) \sum_{\tau=0}^{t} \beta^{t-\tau} g^{(\tau)}
$$

$$
\mathbb{E}\, v^{(t)} = (1-\beta) \nabla_\theta E \sum_{\tau=0}^{t} \beta^{t-\tau}
$$

$$
\mathbb{E}\, v^{(t)} = (1-\beta^{t+1}) \nabla_\theta E
$$

Adam

A esta combinación se le hace una modificación para obtener estimadores insesgados del primer y segundo momento, que por la media móvil quedan sesgados (sobre todo en las primeras iteraciones).

$$
\begin{aligned}
r^{(t)} &= \gamma r^{(t-1)} + (1-\gamma) g^{(t)} \odot g^{(t)} \\
&= \gamma^2 r^{(t-2)} + \gamma(1-\gamma) g^{(t-1)} \odot g^{(t-1)} + (1-\gamma) g^{(t)} \odot g^{(t)} \\
&= \gamma^3 r^{(t-3)} + \gamma^2(1-\gamma) g^{(t-2)} \odot g^{(t-2)} + \gamma(1-\gamma) g^{(t-1)} \odot g^{(t-1)} + (1-\gamma) g^{(t)} \odot g^{(t)}
\end{aligned}
$$

$$\cdots$$

$$\mathbb{E}\, r^{(t)} \approx (1-\gamma^{t+1}) \nabla_\theta E \odot \nabla_\theta E$$

Adam

Para remover el sesgo agregamos dos correcciones sencillas.

$$g^{(t)} \leftarrow \text{Estimar } \nabla_\theta E(\theta^{(t)})$$

$$v^{(t)} \leftarrow \beta r^{(t-1)} + (1-\beta) g^{(t)}$$

$$r^{(t)} \leftarrow \gamma r^{(t-1)} + (1-\gamma) g^{(t)} \odot g^{(t)}$$

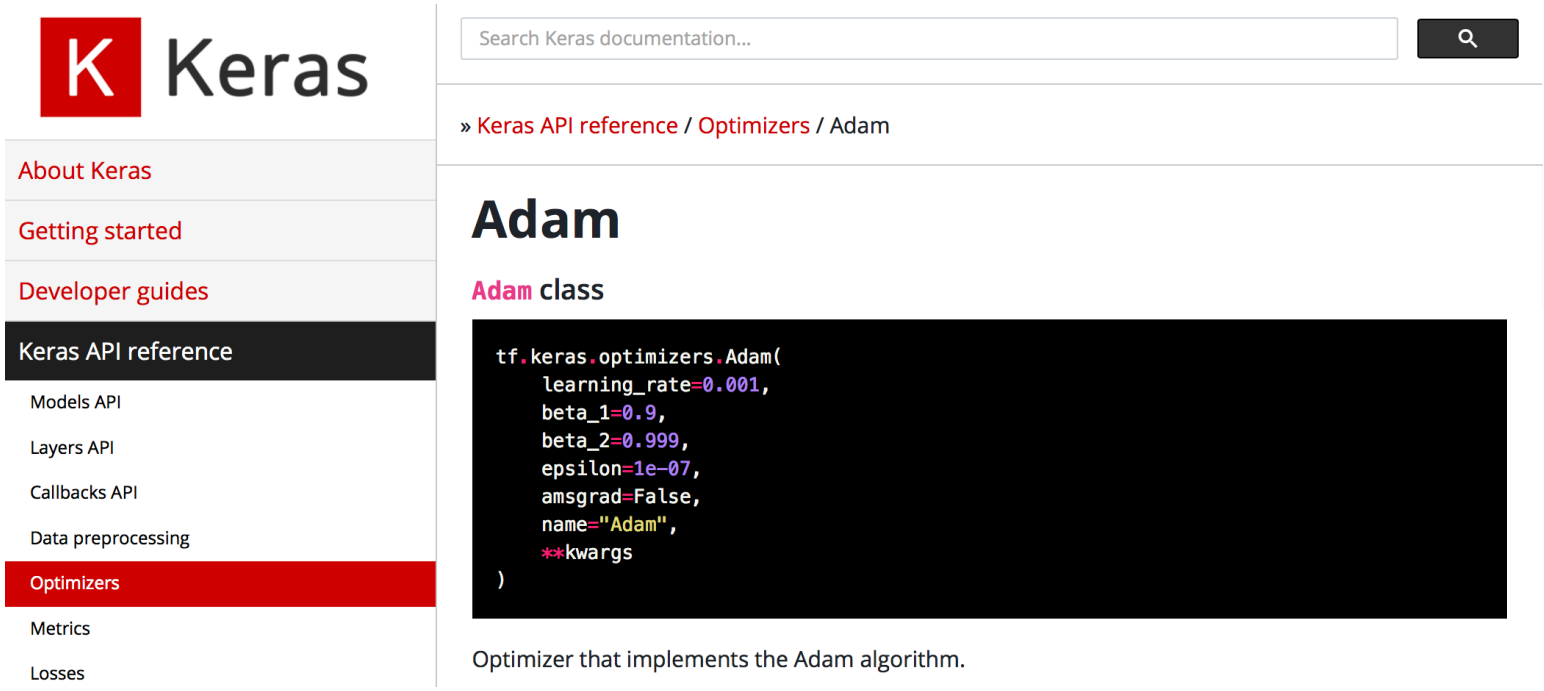
$$\tilde{v}^{(t)} = (1-\beta^{t+1})^{-1} v^{(t)}$$

$$\tilde{r}^{(t)} = (1-\gamma^{t+1})^{-1} r^{(t)}$$

$$\theta^{(t)} \leftarrow \theta^{(t)} - \frac{\eta_t}{\sqrt{\tilde{r}^{(t)} + \epsilon}} \odot \tilde{v}^{(t)}$$

Kingma, Diederik P., and Jimmy Lei Ba. "Adam: A method for stochastic gradient descent." *ICLR: International Conference on Learning Representations*. 2015.

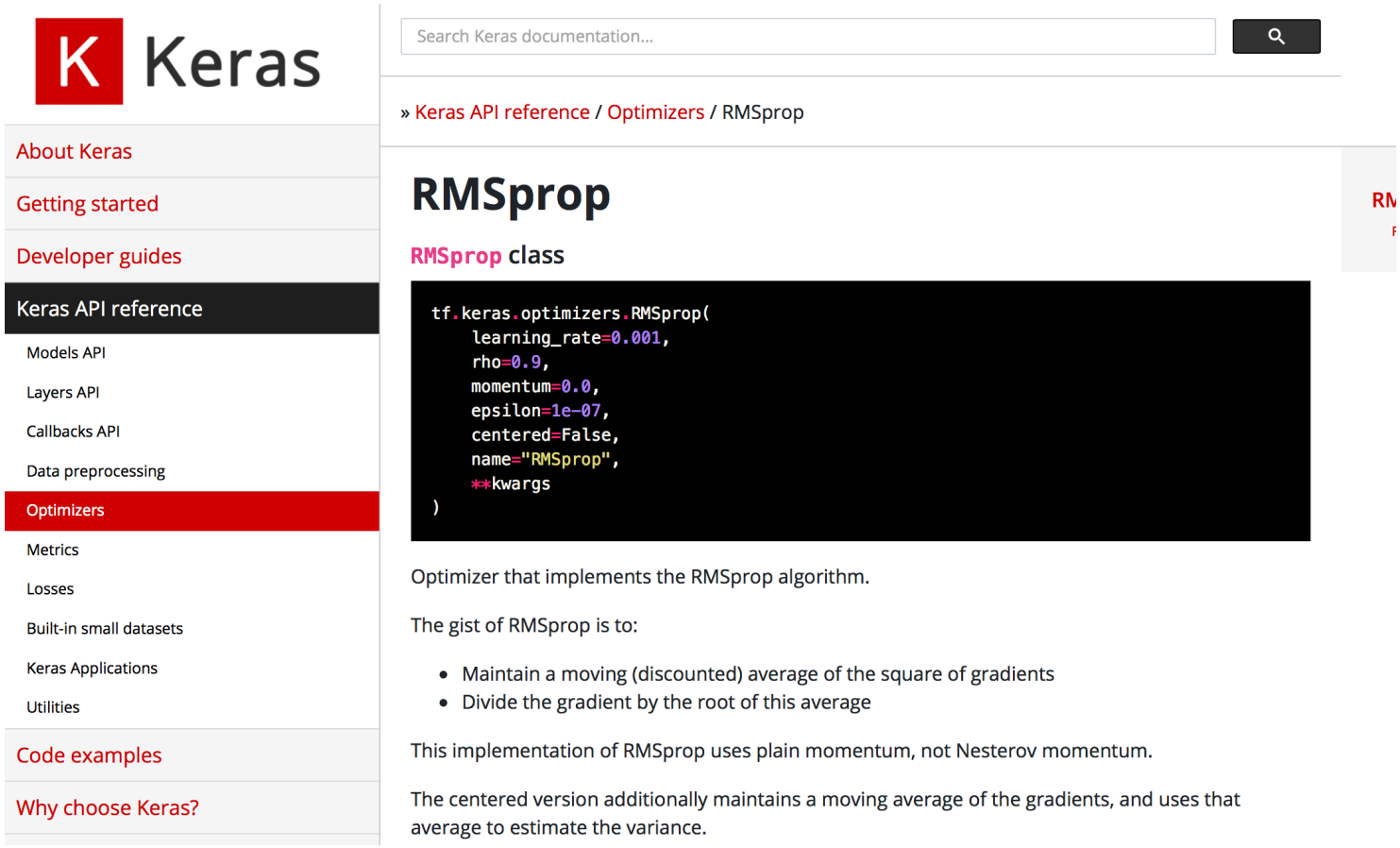
Adam en Keras



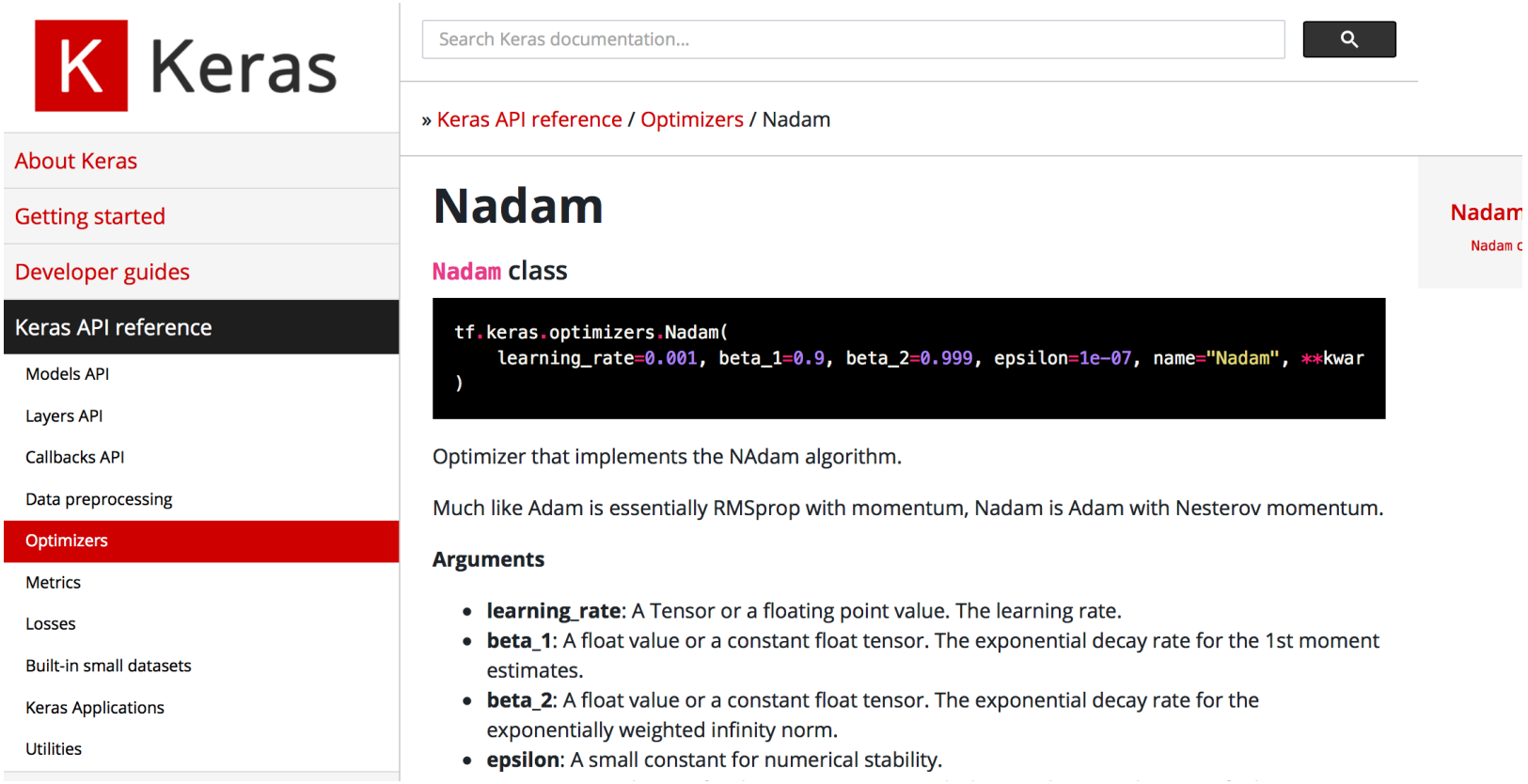
Arguments

- **learning_rate**: A `Tensor`, floating point value, or a schedule that is a `tf.keras.optimizers.schedules.LearningRateSchedule`, or a callable that takes no arguments and returns the actual value to use, The learning rate. Defaults to 0.001.
- **beta_1**: A float value or a constant float tensor, or a callable that takes no arguments and returns the actual value to use. The exponential decay rate for the 1st moment estimates. Defaults to 0.9.
- **beta_2**: A float value or a constant float tensor, or a callable that takes no arguments and returns the actual value to use, The exponential decay rate for the 2nd moment estimates. Defaults to 0.999.
- **epsilon**: A small constant for numerical stability. This epsilon is "epsilon hat" in the Kingma and Ba paper (in the formula just before Section 2.1), not the epsilon in Algorithm 1 of the paper. Defaults to 1e-7.

RMS Prop en Keras



Nadam


Keras
Search Keras documentation...
About Keras
Getting started
Developer guides
Keras API reference
Models API
Layers API
Callbacks API
Data preprocessing
Optimizers
Metrics
Losses
Built-in small datasets
Keras Applications
Utilities
» Keras API reference / Optimizers / Nadam
Nadam
Nadam class
tf.keras.optimizers.Nadam(
learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-07, name="Nadam", **kwar
)
Optimizer that implements the NAdam algorithm.
Much like Adam is essentially RMSprop with momentum, Nadam is Adam with Nesterov momentum.
Arguments
learning_rate: A Tensor or a floating point value. The learning rate.
beta_1: A float value or a constant float tensor. The exponential decay rate for the 1st moment estimates.
beta_2: A float value or a constant float tensor. The exponential decay rate for the exponentially weighted infinity norm.
epsilon: A small constant for numerical stability.
Nadam

Entonces ...

- Una de las dificultades que aparecen aa momento de entrenar redes profundas es que los gradientes tienden a desvanecerse o explotar a medida que atraviesan las capas desde la salida a la entrada.
- Lo anterior motiva el uso de tasas adaptativas por capa y a este punto por neurona e incluso por parámetro.
- Casi todas las buenas ideas para desarrollar algoritmos de este tipo, se basan en compensar la curvatura de a función objetivo, corrigiendo las direcciones de poca curvatura (gradiente cambia poco).
- Entre los métodos más populares tenemos RMS Prop, el momentum clásico y el momentum de Nesterov. Combinando estas ideas de distintas formas aparecen algoritmos como Adam y Nadam.
- Adam es esencialmente RMS Prop con momentum y Nadam es esencialmente RMS Prop con momentum de Nesterov.