DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Optimización

Fernando Berzal, berzal@acm.org

Optimización

- Convergencia del gradiente descendente
- Momentos
- Tasas de aprendizaje adaptativas
  - AdaGrad & AdaDelta
  - rprop & rmsprop
  - Adam
- Técnicas de optimización de segundo orden
  - Métodos quasi-Newton (p.ej. L-BFGS)
  - Gradientes conjugados

En la práctica…

Algoritmo de aprendizaje de redes multicapa

Aspectos que debemos considerar en su diseño:

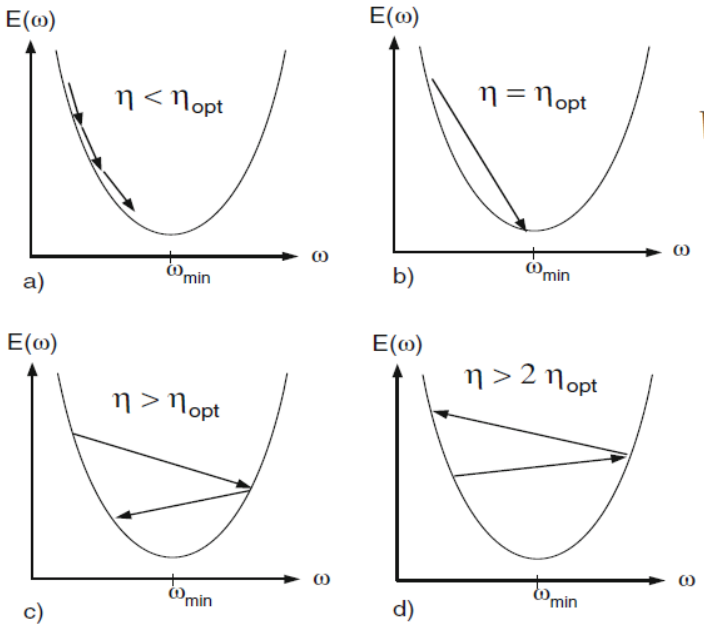
- **Parámetros:** ¿Qué topología de red utilizamos?...
- **Optimización:** ¿Cómo obtenemos los pesos?
- **Generalización:** ¿Cómo conseguimos que la red funcione bien con datos distintos a los del conjunto de entrenamiento?
- **Invarianza:** ¿Cómo conseguimos que la red sea robusta frente a transformaciones comunes en los datos?

DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Optimización
Gradiente descendente

Gradiente descendente

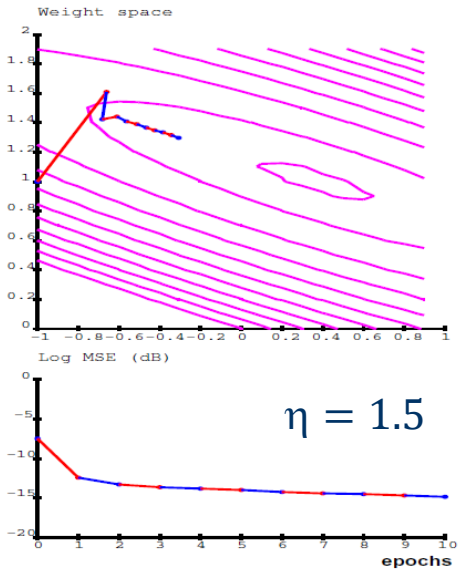
Teoría: Convergencia del gradiente descendente

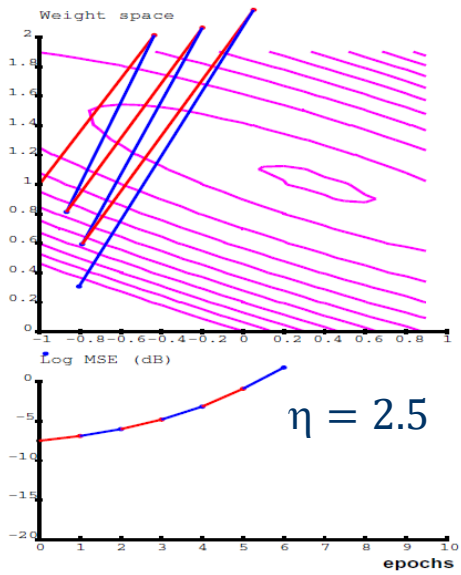


$$W(t+1) = W(t) - \eta \frac{dE(W)}{dW}$$

Gradiente descendente

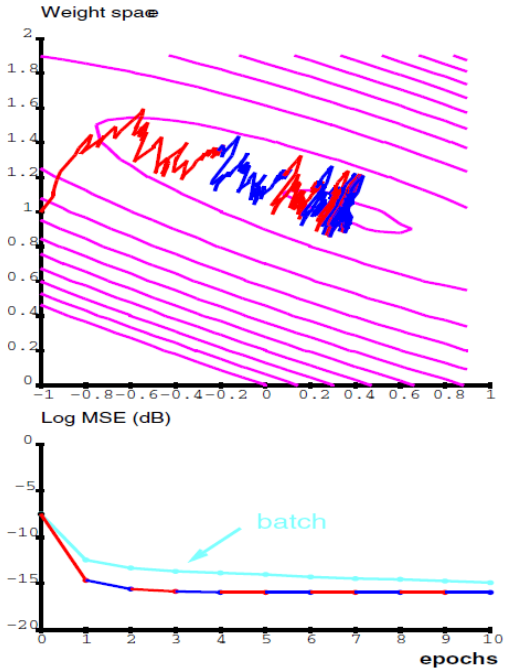
Ejemplo: Batch Learning



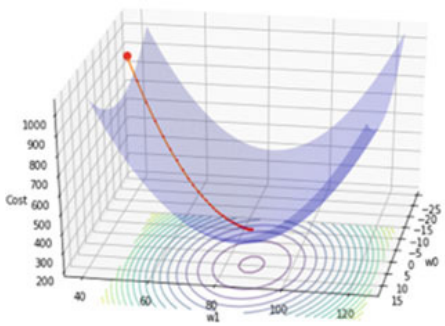


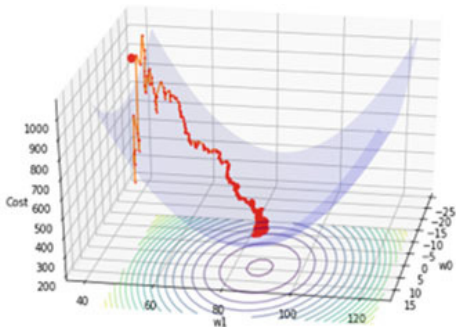
Gradiente descendente

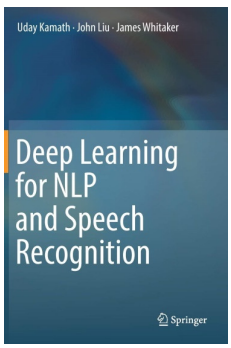
STOCHASTIC/ONLINE VS. BATCH LEARNING



Gradiente descendente







Gradiente descendente

IDEA: Aprendizaje por perturbación de los pesos
(p.ej. algoritmos evolutivos)

Modificamos aleatoriamente los pesos
y comprobamos si la perturbación mejora la red:
Demasiado ineficiente.

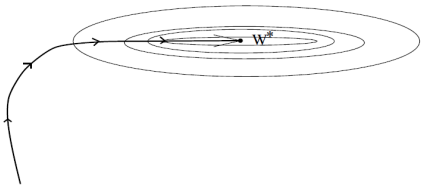
Una alternativa menos mala:
Modificamos las actividades de las neuronas ocultas (una vez que sabemos cómo queremos modificar la actividad de una neurona, podemos calcular cómo cambiar los pesos).

Hay menos actividades que pesos, pero backpropagation sigue siendo más eficiente.

Gradiente descendente

Implementación de backpropagation
Para acelerar la convergencia, se pueden usar:

- Momentos: $\Delta w(t+1) = \eta \frac{\partial E_{t+1}}{\partial w} + \mu \Delta w(t)$
- Tasas de aprendizaje adaptativas.
- rprop & rmsprop
- Técnicas de optimización que tengan en cuenta la curvatura del error (técnicas de segundo orden)



Optimización
Momentos

Momentos

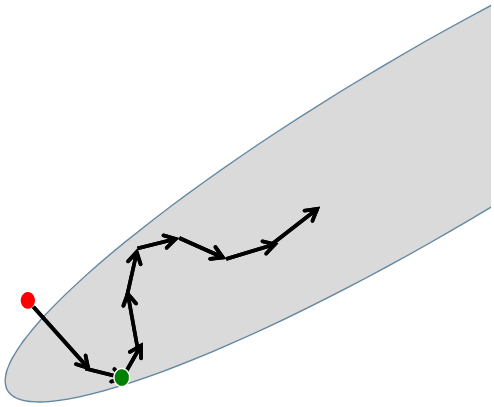
$$\Delta w(t+1) = \eta \frac{\partial E_{t+1}}{\partial w} + \mu \Delta w(t)$$

- En vez de utilizar el gradiente para modificar la "posición" del vector de pesos, lo utilizamos para modificar su "velocidad".
- Su inercia/momento μ hace que tienda a seguir moviéndose en la misma dirección.

Momentos

$$\Delta w(t+1) = \eta \frac{\partial E_{t+1}}{\partial w} + \mu \Delta w(t)$$

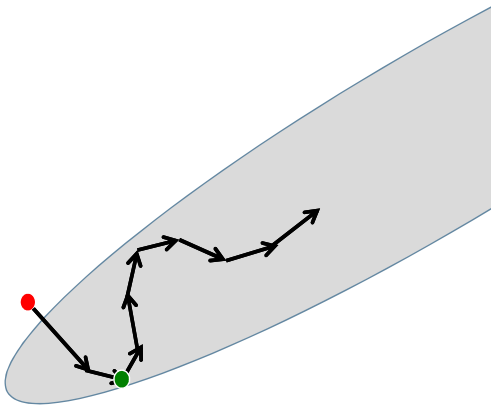
- Se amortiguan oscilaciones en direcciones de alta curvatura combinando gradientes de signo contrario.
- Se aumenta la velocidad en direcciones con un gradiente pequeño pero consistente.



Momentos

Velocidad: $v(t) = \mu v(t-1) - \eta \frac{\partial E(t)}{\partial w}$

- Momento $\mu \leq 1$



Cambio de los pesos = Velocidad:

$$\begin{aligned} \Delta w(t) &= v(t) \\ &= \mu v(t-1) - \eta \frac{\partial E(t)}{\partial w} \\ &= \mu \Delta w(t-1) - \eta \frac{\partial E(t)}{\partial w} \end{aligned}$$

Momentos

- Si la superficie de error es un plano inclinado, se alcanza una velocidad terminal (mucho más rápido que el gradiente descendente):

$$\mathbf{v}(\infty) = \frac{1}{1-\mu}\left(-\eta \frac{\partial E}{\partial \mathbf{w}}\right)$$

- Al comienzo, los gradientes pueden ser elevados, por lo que se empieza con un momento pequeño (0.5).
- Cuando desaparecen los grandes gradientes (y los pesos se estancan), podemos ir aumentando el momento hasta su valor final (0.9 o incluso 0.99).

Momentos

- El uso de momentos nos permite aprender con tasas que causarían oscilaciones divergentes en el gradiente descendente.
- El método estándar primero calcula el gradiente en la posición actual y luego da un salto en la dirección del gradiente acumulado.

Momentos

Método de Nesterov

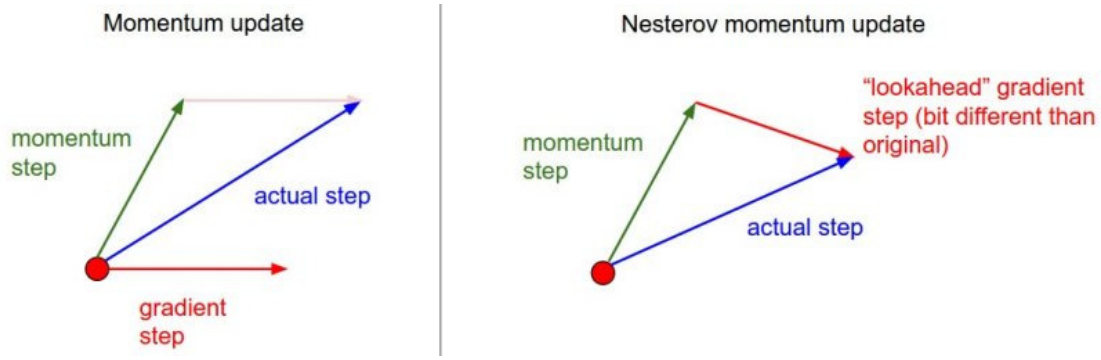
Versión mejorada basada en el método de optimización de funciones convexas de Nesterov:

- Primero se da un salto en la dirección del gradiente acumulado previamente.
- Luego se mide el gradiente en la posición a la que se llega tras el salto y se hace una corrección:

  ¡Mejor corregir el error después de cometerlo!

Momentos

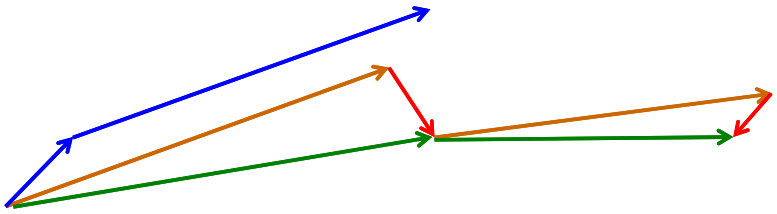
Método de Nesterov



Momento estándar vs. Método de Nesterov

Momentos

Método de Nesterov



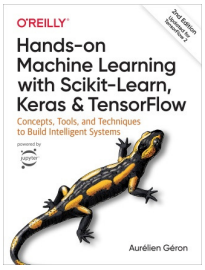
Momento estándar vs. Método de Nesterov

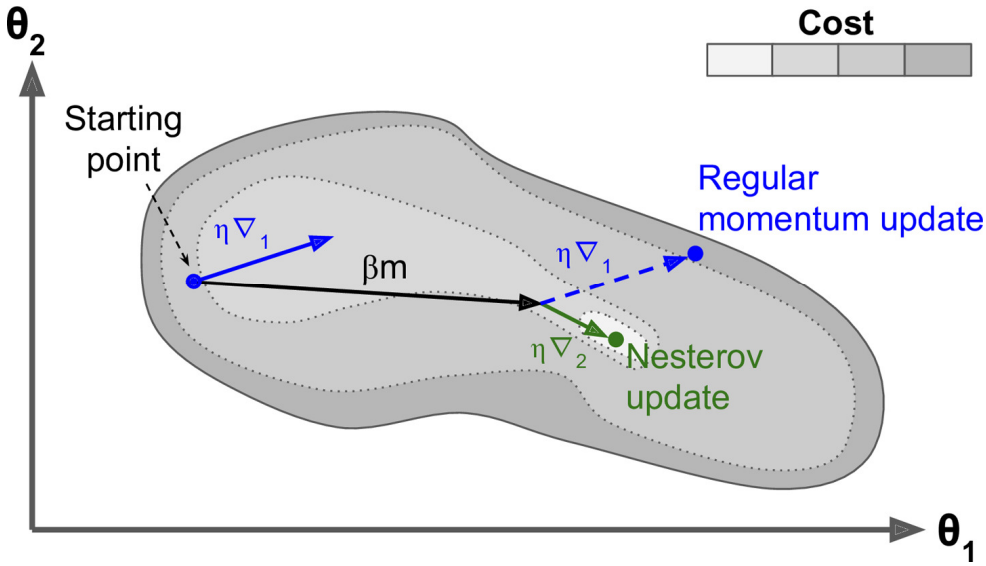
Vector marrón = salto
Vector rojo = corrección
Vector verde = gradiente acumulado

Vector azul = Momento estándar

Momentos

Método de Nesterov





Momento estándar vs. Método de Nesterov

Optimización
Tasas de aprendizaje adaptativas

Tasas de aprendizaje adaptativas

- Uso de tasas de aprendizaje distintas para cada uno de los parámetros de la red.
- Se puede ajustar la tasa de aprendizaje de cada parámetro de la red (peso) en función de la consistencia del gradiente para ese parámetro.

Tasas de aprendizaje adaptativas

En una red multicapa,
la tasa de aprendizaje más adecuada puede variar:

- Las magnitudes de los gradientes son muy diferentes para las distintas capas de la red.
- El "fan-in" de cada nodo determina el efecto causado por el cambio simultáneo de los pesos de entrada (para corregir un mismo error).

Solución: Uso de una tasa de aprendizaje global (fijada manualmente) multiplicada por una ganancia local que se determina empíricamente para cada peso.

Tasas de aprendizaje adaptativas

Una forma de hacerlo [Hinton et al.]

- Inicialmente, la ganancia local es 1 para todos los pesos.

$$\Delta w_{ij} = -\eta g_{ij} \frac{\partial E}{\partial w_{ij}}$$

- Se incrementa la ganancia local si el gradiente para ese peso no cambia de signo,
se disminuye si lo hace.

$$if \left(\frac{\partial E}{\partial w_{ij}}(t) \frac{\partial E}{\partial w_{ij}}(t-1) \right) > 0$$

$$then\ g_{ij}(t) = g_{ij}(t-1) + 0.05$$

$$else\ g_{ij}(t) = g_{ij}(t-1) * 0.95$$

Tasas de aprendizaje adaptativas

Una forma de hacerlo [Hinton et al.]

Aumentos aditivos,
descensos multiplicativos.

$$\Delta w_{ij} = -\eta g_{ij} \frac{\partial E}{\partial w_{ij}}$$

$$if \left(\frac{\partial E}{\partial w_{ij}}(t) \frac{\partial E}{\partial w_{ij}}(t-1) \right) > 0$$

$$then\ g_{ij}(t) = g_{ij}(t-1) + 0.05$$

$$else\ g_{ij}(t) = g_{ij}(t-1) * 0.95$$

- Las ganancias elevadas caen rápidamente cuando se producen oscilaciones.
- Si el gradiente es totalmente aleatorio, la ganancia se mantendrá en torno a 1 si sumamos $+\delta$ la mitad de las veces y multiplicamos por $(1-\delta)$ la otra mitad.

Tasas de aprendizaje adaptativas

Otros trucos para mejorar su funcionamiento:

- Limitar las ganancias para que siempre se mantengan en un rango razonable (p.ej. [0.1,10] ó [0.01,100]).
- Utilizar aprendizaje por lotes o mini-lotes (se reducen los cambios en el signo del gradiente debidos al error de muestreo propio del aprendizaje "online").
- Incorporar momentos a las tasas de aprendizaje adaptativas (p.ej. concordancia de signo entre el gradiente de un peso y su "velocidad").

Tasas de aprendizaje adaptativas

AdaGrad

En una superficie de error elíptica, sabemos que el gradiente no apunta directamente al mínimo, pero nos gustaría que el algoritmo de optimización corrigiese su dirección lo antes posible…

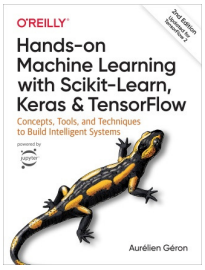
$$\mathbf{s} \leftarrow \mathbf{s} + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \oslash \sqrt{\mathbf{s} + \varepsilon}$$

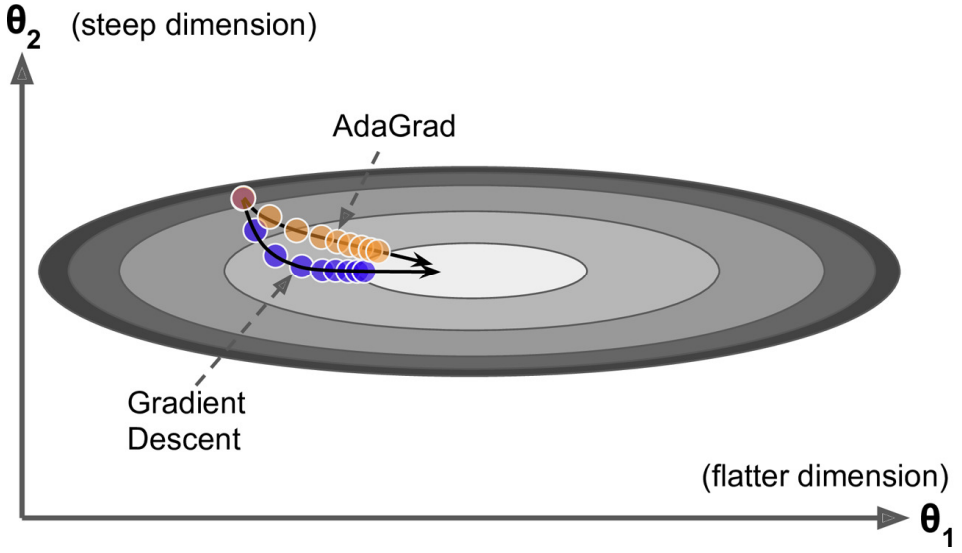
Si vamos acumulando los cuadrados de los gradientes, podemos hacer que las tasas de aprendizaje se vayan amortiguando más para las dimensiones en las que la variación del error es más brusca.

Tasas de aprendizaje adaptativas

AdaGrad

JMLR'2011





Tasas de aprendizaje adaptativas

AdaDelta

arXiv'2012

En lugar de ir acumulando todos los gradientes, que puede reducir de forma demasiado agresiva la tasa de aprendizaje, utilizamos una ventana de ancho limitado.

En lugar de guardar todos los valores del gradiente, se realiza una media móvil con suavizado exponencial.

NOTA:
Es un método incompatible con el uso de momentos…

Tasas de aprendizaje adaptativas

vSGD
[variance-based Stochastic Gradient Descent]

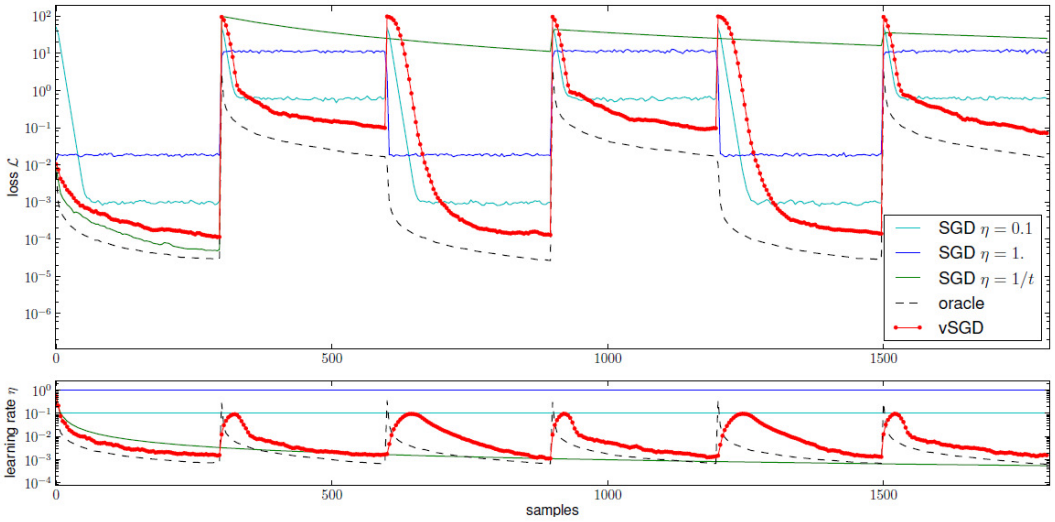
Yann LeCun et al.:
"No More Pesky Learning Rates"
ICML'2013

Algoritmo alternativo para ir ajustando las tasas de aprendizaje, un factor crítico en el rendimiento del gradiente descendente estocástico [SGD].

Tasas de aprendizaje adaptativas

"No More Pesky Learning Rates"

[LeCun et al., ICML'2013]



DECSAI
Departamento de Ciencias de la Computación e I.A.
Universidad de Granada

Optimización

rprop & rmsprop

rprop & rmsprop

rprop [resilient backpropagation]

Utiliza sólo el signo del gradiente.

¿Por qué? La magnitud del gradiente puede ser muy diferente para distintos pesos e ir variando a lo largo del aprendizaje, lo que hace difícil escoger una tasa de aprendizaje global.

En "full-batch learning", podemos eliminar esa variabilidad usando sólo el signo del gradiente:

- Todas las actualizaciones de pesos serán de la misma magnitud.
- Facilita escapar de mesetas con pequeños gradientes.

rprop & rmsprop

rprop [resilient backpropagation]

Utiliza sólo el signo del gradiente…
… y la idea de las tasas de aprendizaje adaptativas:

- Se multiplica por $\eta^+>1$ si no cambia el signo de los dos últimos gradientes (p.ej. 1.2).
- Se multiplica por $\eta^-<1$ si cambia el signo de los dos últimos gradientes (p.ej. 0.5).

RECOMENDACIÓN [Mike Shuster]:
Limitar el tamaño de los cambios, $10^{-6} < \Delta w < 50$.

rprop & rmsprop

rprop [resilient backpropagation]

no funciona con mini-lotes [mini-batches]

Con el gradiente descendente estocástico,
se promedian los gradientes entre distintos mini-lotes
(cuando la tasa de aprendizaje es pequeña):

- Si, para un peso, el gradiente es +0.1 en 9 mini-lotes y -0.9 en uno, queremos que el peso no varíe mucho.
- rprop aumentaría el peso nueve veces y lo disminuiría sólo una, de forma que el peso tendería a crecer :-(

rprop & rmsprop

rmsprop

Versión "mini-batch" de rprop

- Se divide la tasa de aprendizaje de cada peso por una media móvil de las magnitudes de los gradientes recientes para ese peso.
- Combina la robustez de rprop
con la eficiencia de los mini-lotes [mini-batches]
y promedia los gradientes de los distintos mini-lotes.

rprop & rmsprop

rmsprop

Versión "mini-batch" de rprop

- rprop es equivalente a usar el gradiente dividiendo ese gradiente por su magnitud.
- rmsprop mantiene una media móvil del gradiente al cuadrado para cada peso de la red neuronal:

$$MeanSquare(w,t) = 0.9\,MeanSquare(w,t-1) + 0.1\left(\partial E / \partial w(t)\right)^2$$

- rmsprop divide el gradiente por $\sqrt{MeanSquare(w,t)}$

rprop & rmsprop

rmsprop vs. AdaGrad

$$\mathbf{s} \leftarrow \beta\mathbf{s} + (1-\beta)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\,\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \oslash \sqrt{\mathbf{s}+\varepsilon}$$

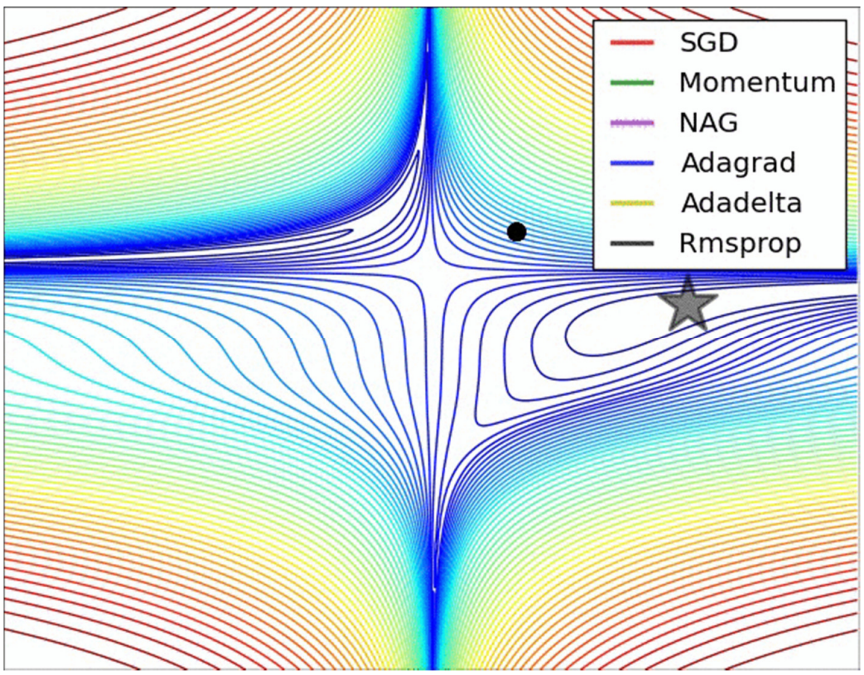
$$\mathbf{s} \leftarrow \mathbf{s} + \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\,\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \oslash \sqrt{\mathbf{s}+\varepsilon}$$

- AdaGrad puede ralentizarse demasiado pronto y no llegar a converger al mínimo (utiliza todos los gradientes obtenidos durante el entrenamiento).
- rmsprop evita este problema usando sólo los más recientes (con una media móvil, como AdaDelta).

Optimización
En la práctica...

En la práctica: Optimización

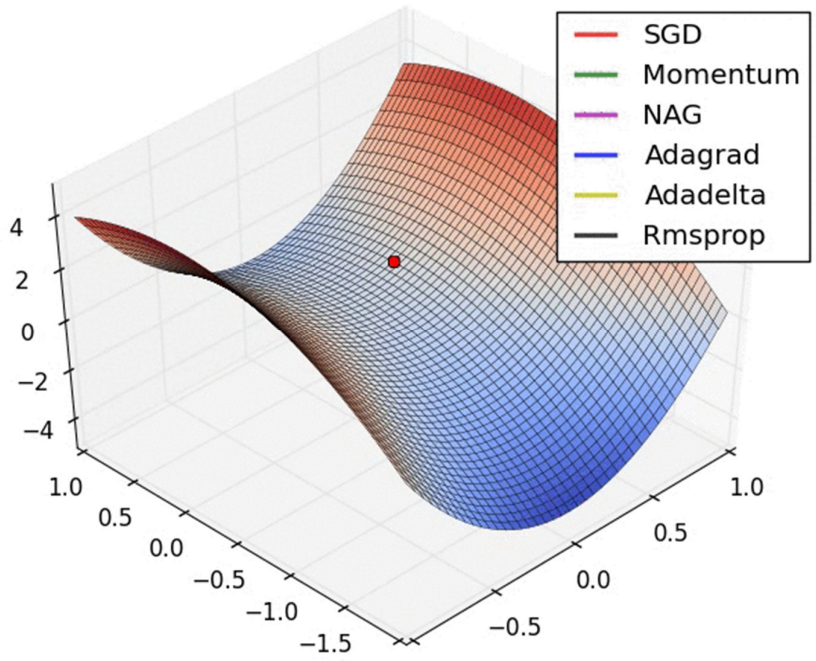
SGD [Stochastic Gradient Descent]



Alec Radford
https://twitter.com/alecrad

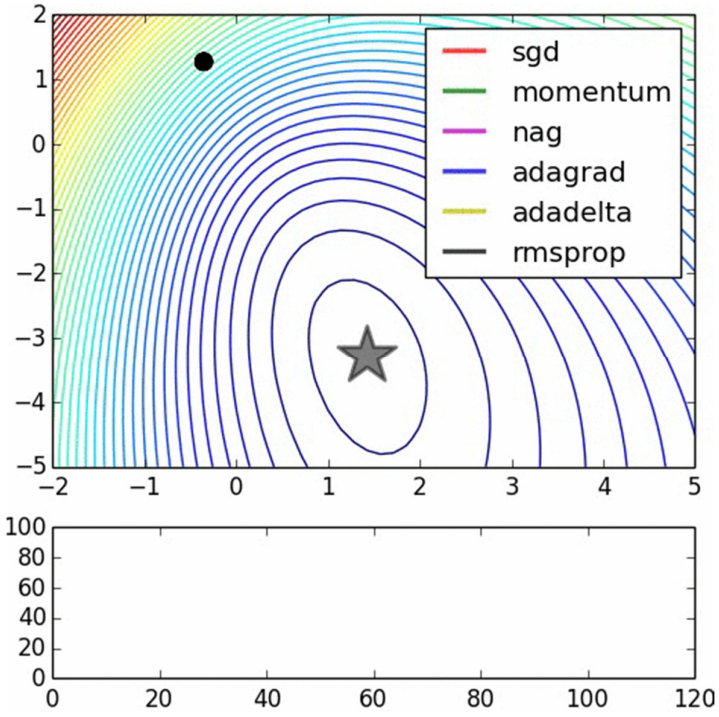
En la práctica: Optimización

SGD [Stochastic Gradient Descent] @ saddle point



Alec Radford
https://twitter.com/alecrad

En la práctica: Optimización



Alec Radford
https://twitter.com/alecrad

En la práctica: Optimización

Adam [Adaptive Moment estimation]

combina momentos con RMSProp

$$\mathbf{m} \leftarrow \beta_1 \mathbf{m} - (1 - \beta_1)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\mathbf{s} \leftarrow \beta_2 \mathbf{s} + (1 - \beta_2)\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \otimes \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

$$\widehat{\mathbf{m}} \leftarrow \frac{\mathbf{m}}{1 - \beta_1^{\,t}}$$

$$\widehat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \beta_2^{\,t}}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\, \widehat{\mathbf{m}} \oslash \sqrt{\widehat{\mathbf{s}} + \varepsilon}$$

NAdam utiliza momentos de Nesterov…

En la práctica: Optimización

Aviso

Aunque los métodos de optimización adaptativos (RMSProp, Adam & NAdam) convergen más rápidamente, en ocasiones no generalizan bien…

Un conjunto de datos puede ser "alérgico" a los gradientes adaptativos, en cuyo caso podemos recurrir a simples momentos [de Nesterov].

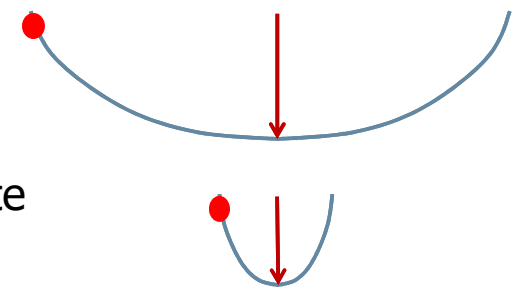
Ashia C. Wilson et al., "The Marginal Value of Adaptive Gradient Methods in Machine Learning," NIPS'2017 *Advances in Neural Information Processing Systems* 30 (2017): 4148–4158

Optimización
Técnicas de optimización de segundo orden

Idea

Técnicas de optimización de segundo orden

¿Cuánto podemos reducir el error moviéndonos en una dirección?

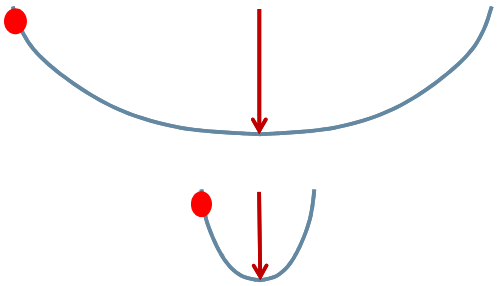


Asumiendo una curvatura constante en la superficie de error (i.e. superficie cuadrática), la reducción máxima del error dependerá de la relación entre el gradiente y la curvatura.

Idea

Técnicas de optimización de segundo orden

¿En qué dirección deberíamos movernos?



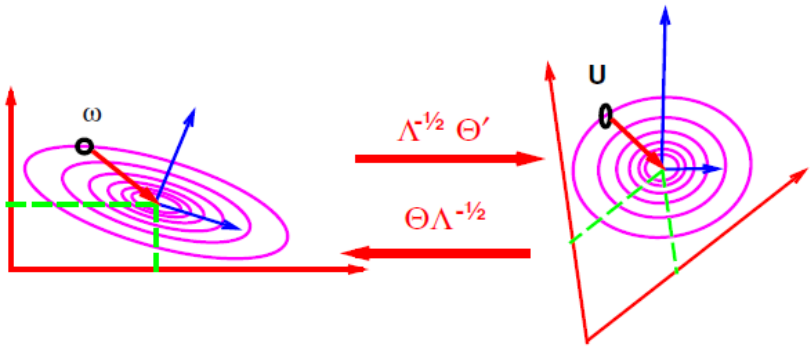
Una buena dirección puede ser aquélla en la que la relación gradiente/curvatura sea elevada, aunque el gradiente en sí sea pequeño.

Método de Newton

Técnicas de optimización de segundo orden

Método de Newton

$$\Delta w = \eta \left(\frac{\partial^2 E}{\partial w^2} \right)^{-1} \frac{\partial E}{\partial w} = \eta H(w)^{-1} \frac{\partial E}{\partial w}$$



Newton Algorithm here .....

....is like Gradient Descent there

Método de Newton

Técnicas de optimización de segundo orden

Método de Newton

$$\Delta w = \eta \left(\frac{\partial^2 E}{\partial w^2} \right)^{-1} \frac{\partial E}{\partial w} = \eta H(w)^{-1} \frac{\partial E}{\partial w}$$

El método de Newton multiplica el gradiente por la inversa de la matriz de curvatura / matriz Hessiana H^{-1}

Esta operación transforma elipsoides en esferas, por lo que si la superficie fuese realmente cuadrática, llegaríamos al mínimo en un solo paso.

Método de Newton

Técnicas de optimización de segundo orden

Método de Newton

$$\Delta w = \eta \left(\frac{\partial^2 E}{\partial w^2} \right)^{-1} \frac{\partial E}{\partial w} = \eta H(w)^{-1} \frac{\partial E}{\partial w}$$

Inconveniente: Hace falta almacenar e invertir una matriz Hessiana de tamaño NxN, lo que requiere $O(N^3)$ por iteración, lo que no resulta práctico.

Además, si la función de error no es cuadrática, ni siquiera tenemos garantías de convergencia :-(

Optimización "Hessian-free"

Técnicas de optimización de segundo orden

"Hessian-free optimization"

- Cada elemento de la matriz de curvatura especifica cómo cambia el gradiente en una dirección conforme nos movemos en otra dirección.

- La matriz de curvatura tiene demasiados términos para que la usemos en la práctica, pero podemos aproximarla de distintas formas,

  p.ej. Gradientes conjugados

  L-BFGS

Optimización "Hessian-free"

Técnicas de optimización de segundo orden

"Hessian-free optimization"

- Aproximamos la matriz de curvatura y, asumiendo que la aproximación es correcta, minimizamos el error utilizando una técnica eficiente (gradientes conjugados).

- Realizamos otra aproximación y volvemos a minimizar…

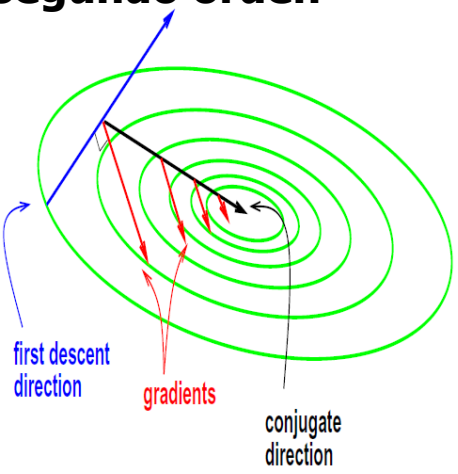
Gradientes conjugados

Técnicas de optimización de segundo orden

Gradientes conjugados

IDEA

Utilizamos una secuencia de pasos, cada uno de los cuales encuentra el mínimo en una dirección.

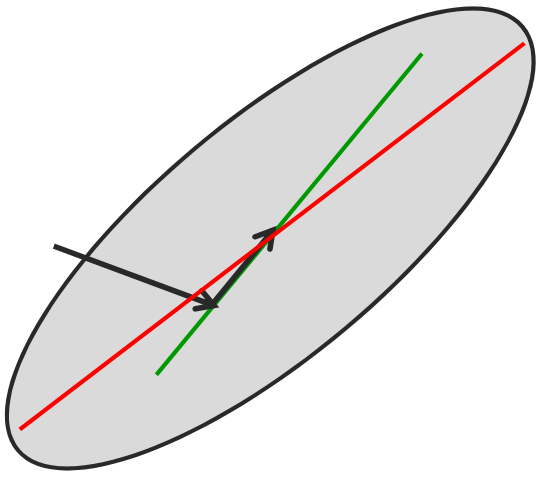


Para no deshacer la minimización que ya hayamos conseguido, nos aseguramos de que cada dirección sea "conjugada" con respecto a las direcciones previas.

Gradientes conjugados

Técnicas de optimización de segundo orden

Gradientes conjugados



El gradiente en la dirección del primer paso es 0 en todos los puntos de la línea verde, por lo que podemos movernos a lo largo de la línea verde sin afectar la minimización realizada en la primera dirección.

Gradientes conjugados

Técnicas de optimización de segundo orden

Gradientes conjugados

- Después de N pasos, el gradiente conjugado garantiza encontrar el óptimo en una superficie cuadrática N-dimensional.

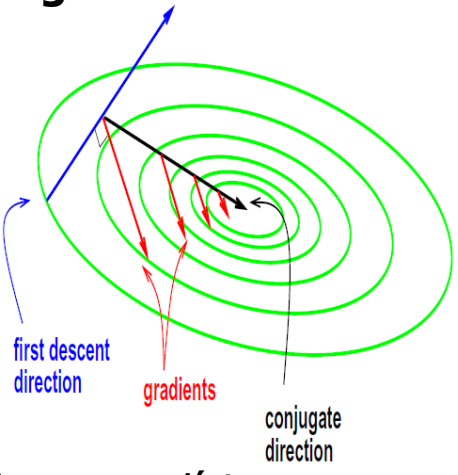
  NOTA: En muchos menos de N pasos, estaremos muy cerca del óptimo

- Los optimizadores "Hessian-free" realizan una aproximación cuadrática de la superficie de error y usan gradientes conjugados para minimizar.

Gradientes conjugados

Técnicas de optimización de segundo orden

Gradientes conjugados



- Algoritmo lineal, O(N).
- No usa la matriz Hessiana de forma explícita.

NOTA:

Sólo funciona con aprendizaje por lotes [batch learning].

Gradientes conjugados

Técnicas de optimización de segundo orden

Gradientes conjugados

TABLE 4.3 Summary of the Nonlinear Conjugate-Gradient Algorithm for the Supervised Training of a Multilayer Perceptron

Initialization

Unless prior knowledge on the weight vector $\mathbf{w}$ is available, choose the initial value $\mathbf{w}(0)$ by using a procedure similar to that described for the back-propagation algorithm.

Computation

1. For $\mathbf{w}(0)$, use back propagation to compute the gradient vector $\mathbf{g}(0)$.
2. Set $\mathbf{s}(0) = \mathbf{r}(0) = -\mathbf{g}(0)$.
3. At time-step n, use a line search to find $\eta(n)$ that minimizes $\mathscr{E}_{\text{av}}(\eta)$ sufficiently, representing the cost function $\mathscr{E}_{\text{av}}$ expressed as a function of η for fixed values of $\mathbf{w}$ and $\mathbf{s}$.
4. Test to determine whether the Euclidean norm of the residual $\mathbf{r}(n)$ has fallen below a specified value, that is, a small fraction of the initial value $\|\mathbf{r}(0)\|$.
5. Update the weight vector:
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta(n)\mathbf{s}(n)$$
6. For $\mathbf{w}(n+1)$, use back propagation to compute the updated gradient vector $\mathbf{g}(n+1)$.
7. Set $\mathbf{r}(n+1) = -\mathbf{g}(n+1)$.
8. Use the Polak–Ribière method to calculate:
$$\beta(n+1) = \max\left\{\frac{\mathbf{r}^T(n+1)(\mathbf{r}(n+1) - \mathbf{r}(n))}{\mathbf{r}^T(n)\mathbf{r}(n)}, 0\right\}$$
9. Update the direction vector:
$$\mathbf{s}(n+1) = \mathbf{r}(n+1) + \beta(n+1)\mathbf{s}(n)$$
10. Set $n = n + 1$, and go back to step 3.

Stopping criterion. Terminate the algorithm when the condition
$$\|\mathbf{r}(n)\| \leq \varepsilon\|\mathbf{r}(0)\|$$
is satisfied, where ε is a prescribed small number.

[Haykin: "Neural Networks and Learning Machines", 3rd edition]

Métodos quasi-Newton

Técnicas de optimización de segundo orden

Métodos quasi-Newton, p.ej. BFGS
[Broyden–Fletcher–Goldfarb–Shanno algorithm]

- Algoritmo cuadrático, $O(N^2)$.
- Estima iterativamente la inversa de la matriz Hessiana.

Limited-Memory BFGS [L-BFGS] aproxima el algoritmo BFGS utilizando una cantidad de memoria lineal, por lo que se utiliza a menudo (p.ej. MATLAB)

https://en.wikipedia.org/wiki/Limited-memory_BFGS

NOTA: Como los métodos anteriores, sólo se puede utilizar en aprendizaje por lotes [batch learning].

Resumen

No existe una receta simple, pero sí recomendaciones:

Para conjuntos de datos pequeños (~10,000 ejemplos):

- Aprendizaje por lotes [full-batch learning].
- Gradientes conjugados o L-BFGS.
- Tasas de aprendizaje adaptativas o rprop.

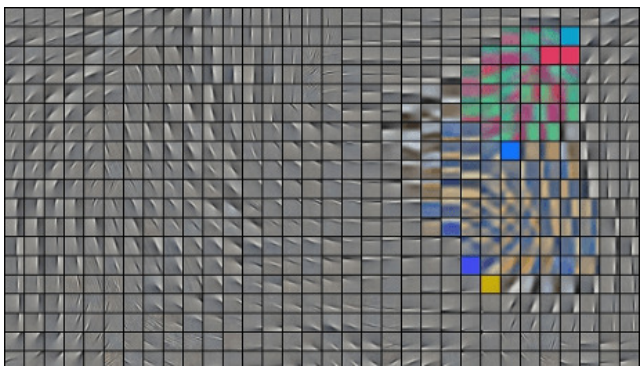
Para conjuntos de datos grandes:

- Aprendizaje por mini-lotes [mini-batch learning].
- rmsprop [Hinton], vSGD [LeCun], Adam…

Cursos

Neural Networks for Machine Learning
by Geoffrey Hinton
(University of Toronto & Google)
https://www.coursera.org/course/neuralnets

coursera

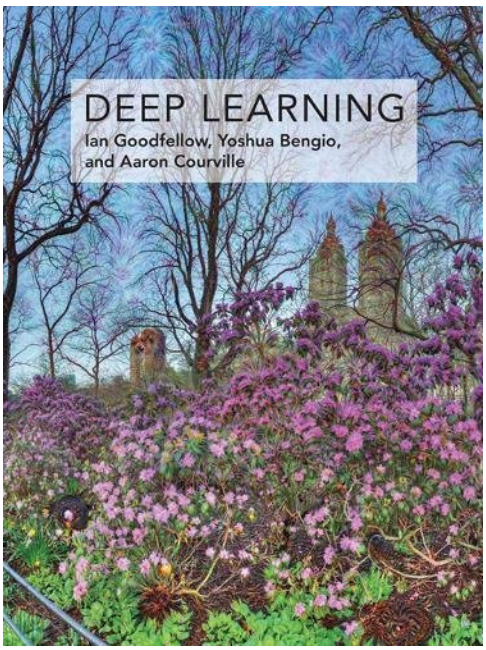


UNIVERSITY OF TORONTO

Bibliografía

Lecturas recomendadas

Ian Goodfellow,
Yoshua Bengio
& Aaron Courville:
Deep Learning
MIT Press, 2016
ISBN 0262035618



http://www.deeplearningbook.org

Bibliografía

Lecturas recomendadas

Fernando Berzal:
Redes Neuronales
& Deep Learning

CAPÍTULO 11
Optimización