Un plan de mejoras en 5 etapas parte 2

De las redes feed forward panditas a las profundas

3. Aprimoramos el descenso por el gradiente
Descenso por el gradiente estocástico

Introducimos aleatoriedad

☑ **Minibatch:** Este algoritmo consiste en dividir el conjunto de entrenamiento en mini grupos por lo que en cada época se hace una reasignación aleatoria de todos los ejemplos de manera que $Qm = p$ (se tienen p elementos del conjunto de entrenamiento divididos en m grupos de tamaño Q). Con esta técnica se pretende introducir aleatoriedad en el método del descenso por el gradiente al sortear época a época los elementos del conjunto de entrenamiento. En general se encuentra que la cantidad de minibatches debe ser lo suficientemente alta para lograr introducir la aleatoriedad deseada, sin embargo si la cantidad de particiones es demasiado alta también lo será el costo computacional pues serán muchas veces que se realice el descenso por el gradiente. Con respecto a esto Keskar, Mudigere, Nocedal, Smelyanskiy, & Tang (2016) han comentado que cuando los batch son muy amplios (con muchos elementos de entrenamiento) la función costo adquiere una forma tal que el método del descenso por el gradiente tiende a converger hacia mínimos más agudos (pendientes altas). Por su parte, dividir los ejemplos en conjuntos de elementos más pequeños para analizar produce una convergencia del sistema hacia mínimos más planos (pendientes menores). Esto quiere decir que los batch más largos tienden hacia mínimos de los cuales es difícil despegarse mientras que los batch más pequeños, al ser atraídos hacia mínimos con formas más planas tienen la capacidad de escapar de las regiones de mínimos locales.

☑ **<u>Dropout</u>:** Para reducir el overfitting y permitir que una red neuronal artificial responda correctamente cuando es testeada, se aplica el dropout. Este algoritmo implica suprimir un cierto número de neuronas de manera aleatoria por lo que cuando se calcula la salida no se considera toda la información. Desde este punto de vista, cada neurona presente en las capas ocultas son omitidas al azar con una probabilidad $p = 0{,}5$ tal que estas neuronas no dependen del resto. Con esto se pretende evitar el establecimiento de complejas adaptaciones donde cada característica a detectar es útil y posible sólo en el contexto planteado por todas las demás. Es decir, cada neurona aprende a detectar una característica que permite producir la respuesta correcta en una multiplicidad de contextos internos posibles y variados en el cual opera la red. En sentido gráfico, el método del descenso por el gradiente no es capaz de salir de mínimos locales y requiere tiempos muy largos para salir de un punto de inflexión, tiempo que crece exponencialmente a medida que aumenta el número de componentes en la cual es plano. Este problema se facilita con la aplicación del algoritmo dropout al suprimir una de las dimensiones donde podría estar ubicado un mínimo local.

Es posible encontrar un paralelismo entre este funcionamiento de la red artificial (dropout) con el funcionamiento del cerebro dado que las neuronas requieren de un tiempo posterior a la emisión de señales para recuperar la energía liberada cuando las transmiten a neuronas vecinas de manera que, por un tiempo, se podría decir que permanecen apagadas.

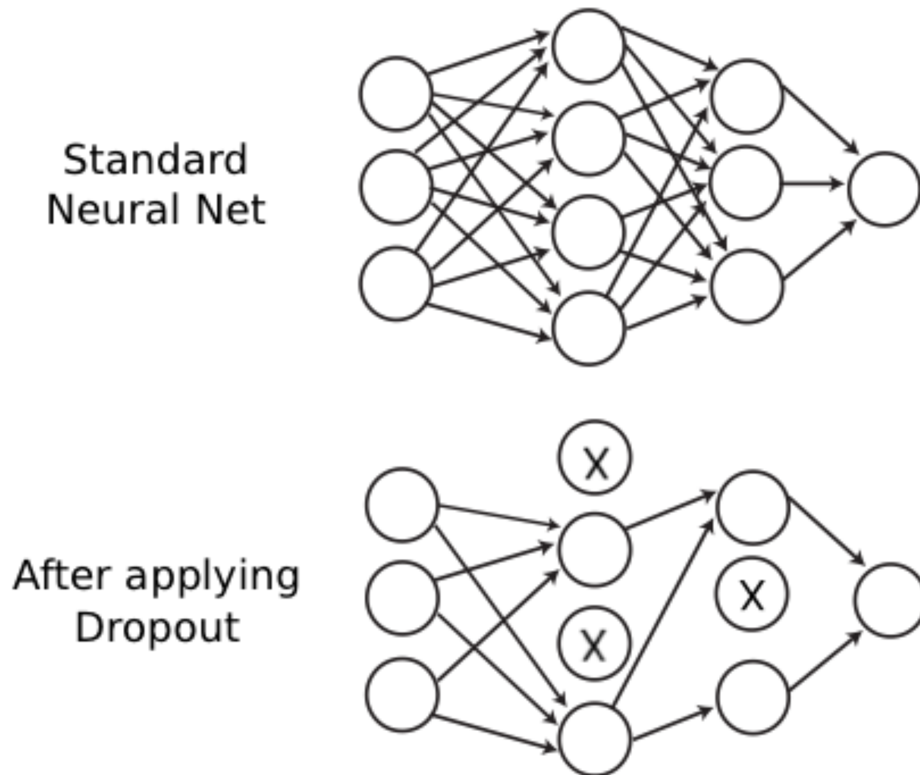


FIG. 39 **Dropout** During the training procedure neurons are randomly "dropped out" of the neural network with some probability p giving rise to a thinned network. This prevents overfitting by reducing correlations among neurons and reducing the variance in a method similar in spirit to ensemble methods.

*Improving neural networks by preventing co-adaptation of feature detectors by Geoffrey Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2012).

3. Aprimoramos el descenso por el gradiente
Descenso por el gradiente estocástico

④ Ponemos memoria y adaptación al descenso

4. Agregamos memoria y adaptación al descenso por el gradiente

Agregamos Momento

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$$

$$\Delta w_{ij}^{(t)} = \gamma \Delta w_{ij}^{(t-1)} - \eta \frac{\partial E}{\partial w_{ij}}$$

↑ término de momento

$$0 \leq \gamma \leq 1$$

Esto le permite al método ganar fuerza cuando el incremento es persistente pero pequeño

Si andamos por una zona "plana" de la función error, y el parámetro de momento es cercano a 1

$$\Delta w_{ij}^{(t)} \approx \Delta w_{ij}^{(t-1)}$$

$$(1-\gamma)\,\Delta w_{ij}^{(t)} = -\eta \frac{\partial E}{\partial w_{ij}}$$

$$\Delta w_{ij} = -\frac{\eta}{(1-\gamma)} \frac{\partial E}{\partial w_{ij}}$$

avanzamos con un paso más largo

Podemos comenzar con $\gamma = \frac{1}{2}$ y aumentarlo a medida que descendemos hasta llegar a $\gamma = 0.99$

Nesterov Accelerated Gradient (NAG)

Ilya Suskever 2012

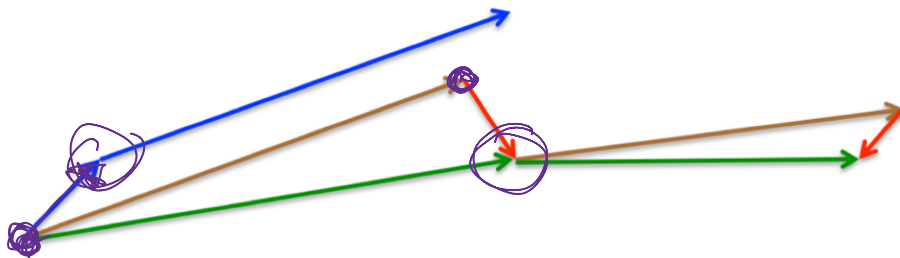
$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$$

$$\Delta w_{ij}^{(t)} = \gamma_t \Delta w_{ij}^{(t-1)} - \eta \frac{\partial E}{\partial w_{ij}}\left(w_{ij}^{(t-1)} + \gamma_t \Delta w_{ij}^{(t)}\right)$$

Primero damos un salto grande en la dirección del gradiente acumulado previo

Ahora nos paramos en este punto y desde allí calculamos el gradiente

Sumamos ambos



Los vectores azules son los saltos usuales.

Ahora queremos introducir un método "adaptativo" que considere conexiones "locales" para cada sinapsis

$$\Delta w_{ij}(t) = -\eta \, g_{ij}(t) \cdot \frac{\partial E}{\partial w_{ij}}$$

universal (η) — específico ($g_{ij}(t)$)

Hacemos inicialmente

$$g_{ij}(0) = 1 \quad \forall \, i,j$$

IF $\left[\frac{\partial E(t-1)}{\partial w_{ij}} \cdot \frac{\partial E(t)}{\partial w_{ij}}\right] > 0$ THEN

$$g_{ij}(t+1) = g_{ij}(t) + 0.5$$

ELSE

$$g(t+1) = 0.95 * g(t)$$

END

¡Esto lo hacemos por mini batch!

- Limitamos los valores de $g_{ij}(t)$

$$0.1 \leq g_{ij}(t) \leq 10 \quad \forall t, i, j$$

$$0.01 \leq g_{ij}(t) \leq 100 \quad \forall t, i, j$$

- usamos minibatches NO MUY PEQUEÑOS.
  Esto evita a que no haya cambios de signo en el gradiente solo porque tenemos "muestras" pequeños. Recuerden que el error es una "función aleatoria" (de muchas variables aleatorias).

- Mezclamos momento con pesos adaptativos.

- lo adaptativo se hace "por eje"

RPROP

Tenemos presente que el gradiente puede ser muy diferente para pesos diferentes y esto puede cambiar a lo largo del descenso por el gradiente

Combinamos la idea de adaptar solo por el signo con la idea de adaptar pesos por separado

IF $\left[\frac{\partial E(t-1)}{\partial w_{ij}} \cdot \frac{\partial E(t)}{\partial w_{ij}}\right] > 0$

$$g(t+1) = g(t) * 1.2$$

THEN

$$g(t+1) = g(t) * 0.5$$

END

Este método anda mal para minibatches

RMSPROP

Buscamos un método tipo RPROP pero que nos permite usar mini batches.

$$\text{Mean Square}(w_{ij}, t) = 0.9\,\text{Mean Square}(w_{ij}, t-1) + 0.1\left(\frac{\partial E}{\partial w_{ij}}\right)^2$$

$\downarrow$

S_t^{ij}

$$g_t = \frac{\partial E}{\partial w_{ij}^{(t)}}$$

$$S_t^{ij} = \beta\, S_{t-1}^{ij} + (1-\beta)\left(g_t^{ij}\right)^2$$

$$\Delta w_{ij}^{(t)} = -\eta\,\frac{g_t}{\sqrt{S_t^{ij} + \epsilon}}$$

$$w_{ij}^{(t+1)} = w_{ij}^{(t)} + \Delta w_{ij}^{(t)}$$

$\beta \approx 0.9$

$\epsilon \approx 10^{-8}$

$\eta \approx 10^{-3}$

ADAM (adaptive moment estimation)

$$g_t = \frac{\partial E}{\partial w_{ij}}$$

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

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

$$s_t = \beta_2 s_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{s}_t = \frac{s_t}{(1-\beta_2)}$$

$$E[m_t] = E[g_t]$$

$$E[s_t] = E[g_t^2]$$

$$w_{ij}^{t+1} = w_{ij}^{t} + \Delta w_{ij}^{t}$$

$$\Delta w_{ij}^{t} = -\eta_t \frac{\hat{m}_t}{\sqrt{\hat{s}_t + \epsilon}}$$

$$\beta_1 \approx 0.9$$

$$\beta_2 \approx 0.99$$

$$\epsilon \approx 10^{-8}$$

$$\eta \approx 10^{-3}$$

Regularizaciones

Agregamos términos adicionales a la función error (o costo, o loss) que "limiten" los valores de las sinapsis según nuestros deseos

$$E(\bar{w}, \xi) = \sum_{\mu=1}^{p} \sum_{i} e(\bar{w}, \xi_i^{\mu}) + \lambda R(f)$$

hyperparámetro (λ)

$R(f)$: función que tenemos que aprender

Regularización L_1 o Lasso (least shrinkage and selection operator)

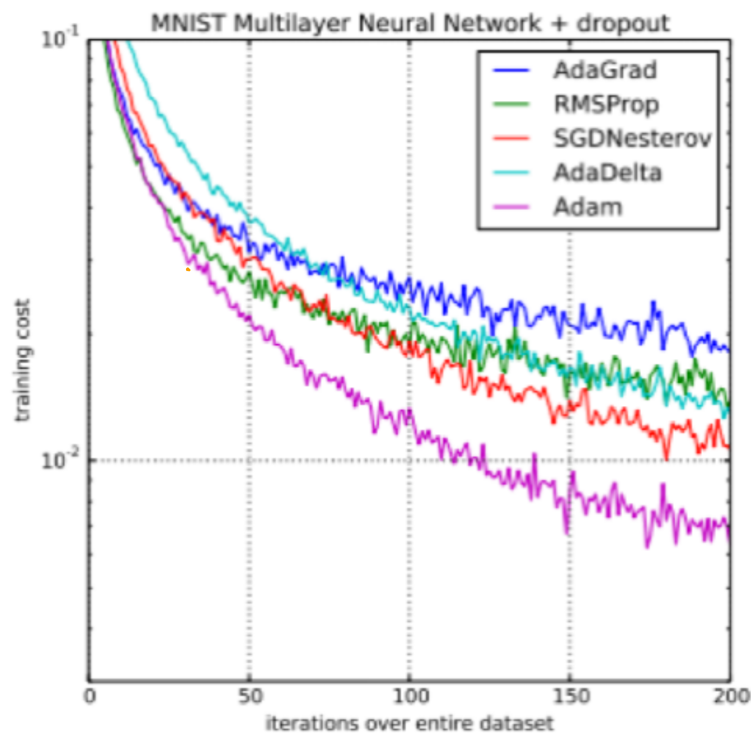
$$E_L(\{\bar{w}\}, \{\bar{\xi}\}) = \sum_{\mu=1}^{n} \sum_{i=1}^{M} (\xi_i^{\mu} - O_i^{\mu})^2 + \lambda \sum_{\alpha} |w_\alpha|$$

$\sum_{\alpha}$: suma sobre todos los acoplamientos y umbrales

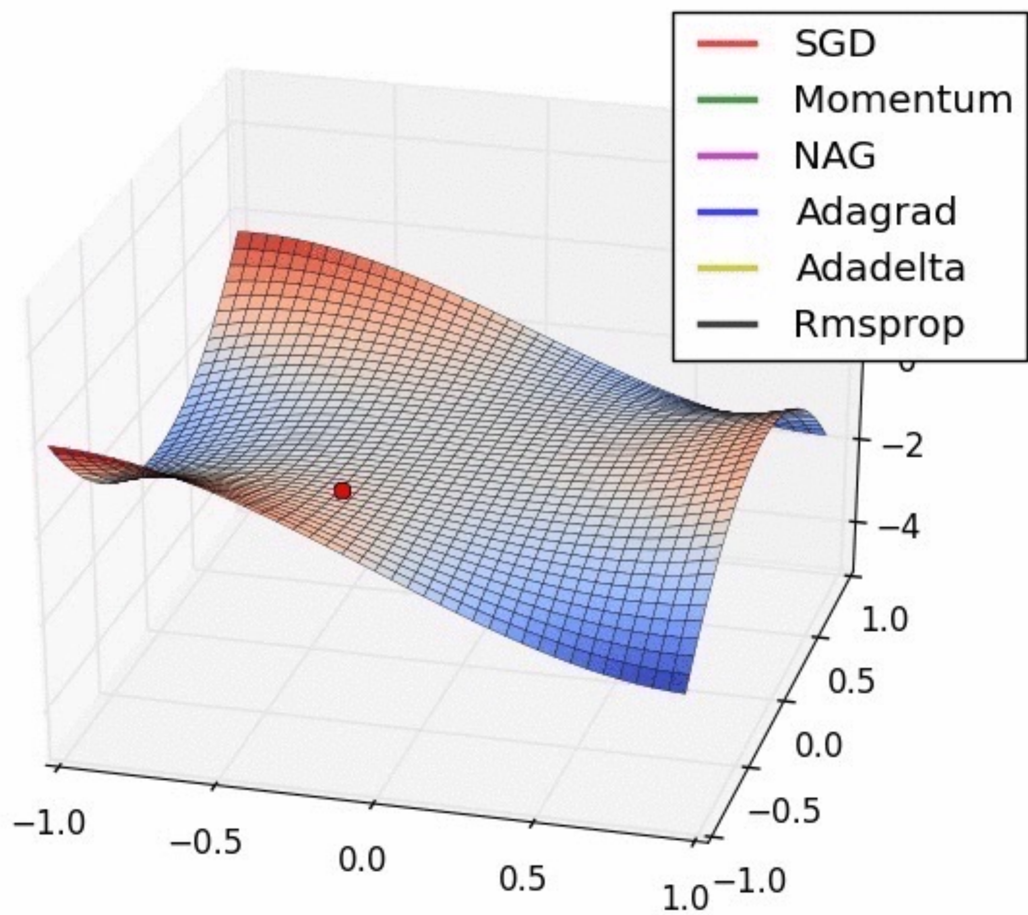
Regularización L_2 o Ridge

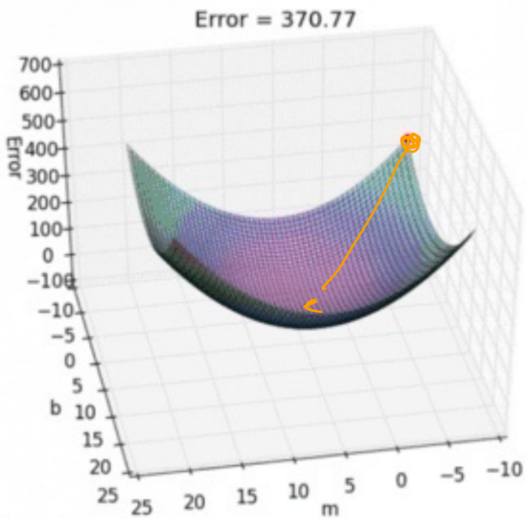
$$E_L(\{\bar{w}\}, \{\bar{\xi}\}) = \sum_{\mu=1}^{n} \sum_{i=1}^{M} (\xi_i^{\mu} - O_i^{\mu})^2 + \lambda \sum_{\alpha} w_\alpha^2$$

$\sum_{\alpha}$: suma sobre todos los acoplamientos y umbrales

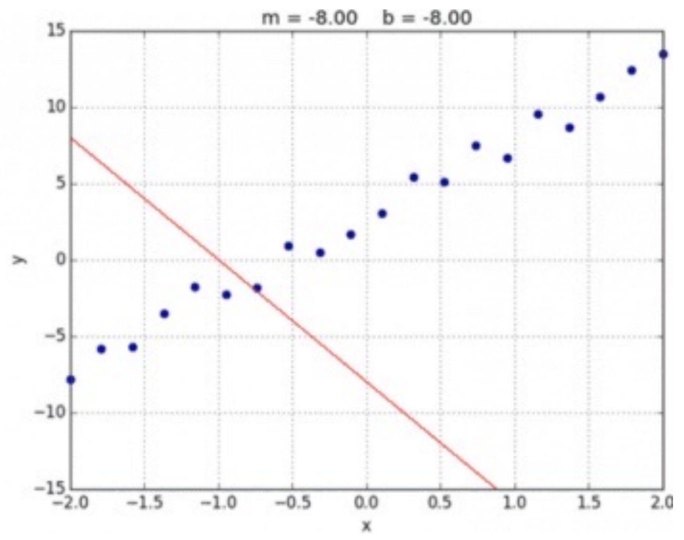


Diferentes métodos de optimizar el descenso por el gradiente (problema MNIST)

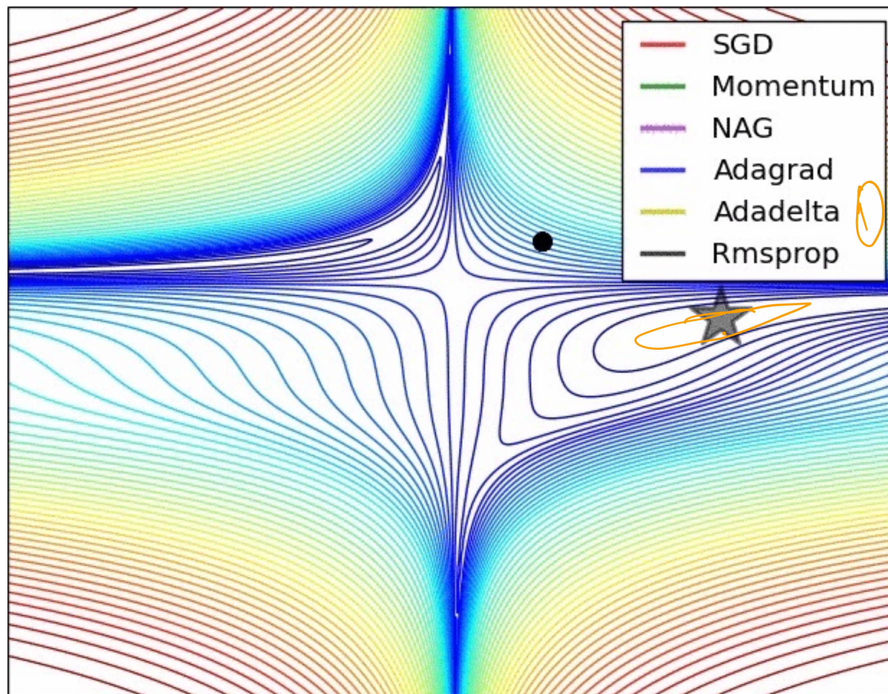




Error = 370.77
Error
m
b



m = -8.00 b = -8.00
x
y



SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop

4- Cambiamos la función error o costo

☑ **Cross Entropy:** La función error se ha definido como la suma de los cuadrados de la diferencia entre los valores deseados y la salida obtenida por la red. Por la naturaleza de esta función y dado que el método del descenso por el gradiente se definió proporcional al ${\partial E}/{\partial w}$, cuanto más próximos estemos al mínimo donde sea $E = C = 0$ menor será la velocidad (pendiente) con la que nos acerquemos. Si esta función costo fuera reemplazada por una función logarítmica, el descenso por el gradiente iría incrementando su velocidad de acercamiento al mínimo dado que el gradiente sería cada vez mayor. Esto se suele usar como una métrica para saber cuán cerca estamos de la solución correcta.

La cross entropy o entropía cruzada viene de la teoría de la información. La información es una medida de la cantidad de bits requeridos para codificar y transmitir un evento.

Eventos poco probables tienen mucha información
Eventos muy probables tienen poca información

La entropía es una medida del número de bit requeridos para codificar y enviar un evento obtenido de una distribución de probabilidades.

En teoría de información cuantifican la sorpresa.

Si un evento tiene poca probabilidad, es más "sorprendente"

Si la distribución de probabilidad de los bits tiene un pico crítico $\Rightarrow$ baja posibilidad de sorpresa

Si la distribución de probabilidad de los bits es balanceada $\Rightarrow$ alta probabilidad de sorpresa.

Para una variable aleatoria x obtenida de una distribución $p(x)$, la cantidad

$$H(x) = -\sum_{x \text{ de } P(x)} p(x) \ln(p(x))$$

La entropía cruzada se construye sobre la idea de entropía y calcula el número de bits requeridos para codificar o transmitir un evento construido a partir de una distribución comparada con otra.

En Inteligencia Artificial es el nº de bits necesarios para codificar datos que vienen de una fuente con una distribución p con un modelo q.

$$H = -\sum_{x \text{ de } P(x)} P(x) \log(Q(x))$$

Kollback 1959

Hopfield 1987

Sara Solla 1988

Para el perceptron para aprendizaje supervisado

Si $\quad O_i^\mu = \dfrac{1}{1+e^{-h_i^\mu}}$, entonces

$$E(\{w\},\{J\}) = -\sum_{i\mu} J_i^\mu \log_2(O_i^\mu) + (1-J_i^\mu)\log_2(1-O_i^\mu)$$

$E \geq 0$ para cualquier $\{w\}$ o $\{J\}$

$E = 0 \Rightarrow O_i^\mu = J_i^\mu \quad \forall i \wedge \mu$

Si $\quad O_i^\mu = \tanh(h_i^\mu)$, entonces

$$E(\{w\},\{J\}) = -\sum_{i\mu}\left\{(1+J_i^\mu)\log_2(1+O_i^\mu) + (1-J_i^\mu)\log_2(1-O_i^\mu)\right\}$$

$E \geq 0$ para cualquier $\{w\}$ o $\{J\}$

$E = 0 \Rightarrow O_i^\mu = J_i^\mu$

Observaciones Generales

- Si el conjunto de entrenamiento (data set en la jerga de IA) es muy redundante (tiene datos parecidos), el gradiente que obtenemos con una mitad de los datos es igual al gradiente que obtenemos con la otra mitad. Podemos entonces aprender con una mitad, y vamos cambiando "la mitad", eligiéndola al azar en cada época.

  No tiene que ser mitad. Podemos suprimir en cada época p neuronas y todas sus sinapsis.

- Si usamos todo el gradiente, podemos usar métodos de optimización global, sobre todo heurísticos, pero si el nº de sinapsis es muy alto y los datos son redundantes, es mejor usar dropout y minibatches.

- Cuanto mayores son los minibatches, más eficiente es el método computacionalmente

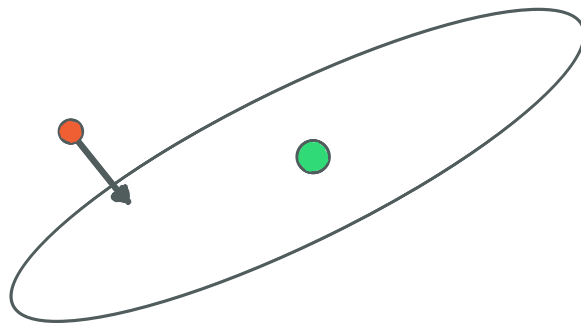
- Si dos capas ocultas de neuronas tienen los mismos umbrales y las mismas sinapsis entrando y saliendo, tendrán el mismo gradiente y seguirán siempre iguales. Por esto es importante evitar estas decisiones simétricas.

- Si una neurona oculta tiene muchas entradas (large fan-in en la jerga), pequeños cambios en los pesos pueden producir un redondeamiento en la actualización (overshooting). O sea, que no sigue bien la topología del gradiente. Por esto es importante normalizar los valores iniciales de forma tal que sean proporcionales a $\sqrt{N}$, donde N es el número de entradas

En el caso de neuronas lineales vimos que la función error es una función cuadrática en muchas direcciones y plana en otras

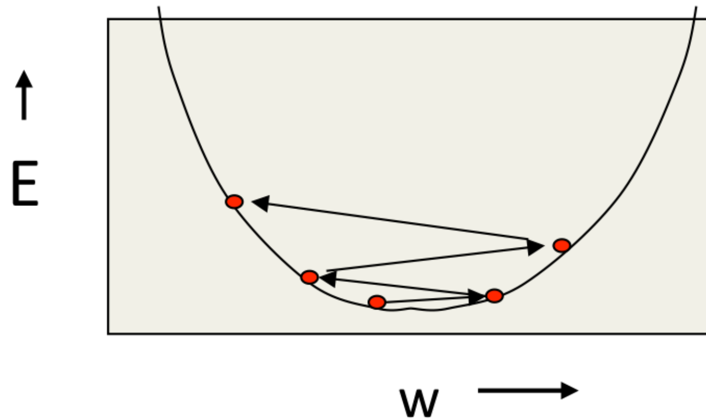
En el caso de funciones no lineales todo se complica pero localmente podemos pensar alrededor de cada mínimo como un bowl cuadrático con diferentes cuadraturas.

Bajar por el gradiente en un batch COMPLETO garantiza que el error disminuye, pero no apunta nunca hacia el mínimo



El gradiente es grande en la dirección en la cual deseamos "avanzar" poco, pero es pequeño en la dirección en la cual deseamos "avanzar" mucho.

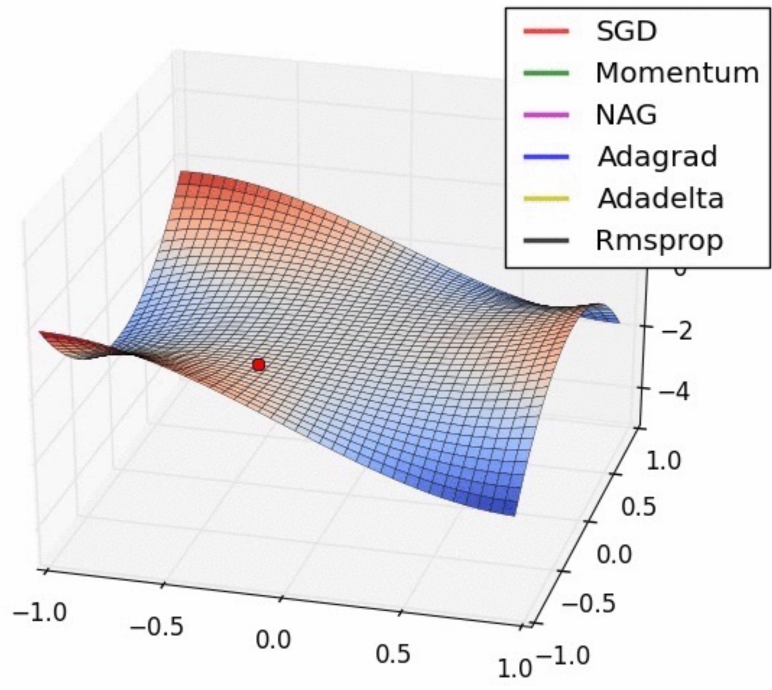
Si el learning rate es demasiado grande el descenso por el gradiente se queda "atrapado en un barranco", o "saltan el barranco"



Lo que deseamos es:

- movernos rápidamente en las direcciones con pequeños gradientes

- movernos lentamente en las direcciones con gradientes grandes.


SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0