Redes neuronales

Inicialización de pesos

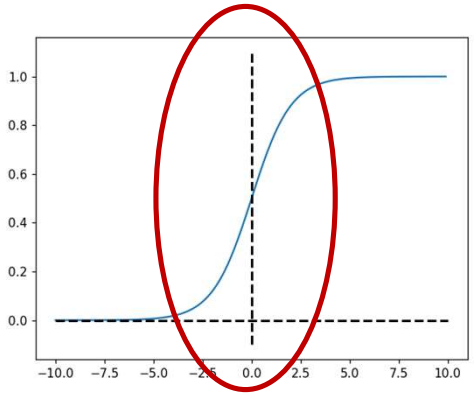
El rol de los parámetros

- Parámetros (pesos y biases) son el núcleo de las redes neuronales y algoritmos de aprendizaje.
- Buenos parámetros iniciales otorgan mejores propiedades a las redes neuronales.
- Cuáles son buenos parámetros iniciales?
  - Algunas opciones: zero parameters, uniform distribution, normal distribution

Análisis de distribución

Consideremos la función sigmoide

Función sigmoide



$$z_L = X_{L-1} \cdot W^L + b^L$$

$$y = \sigma(z_L) = \frac{1}{1 + e^{-z_L}}$$

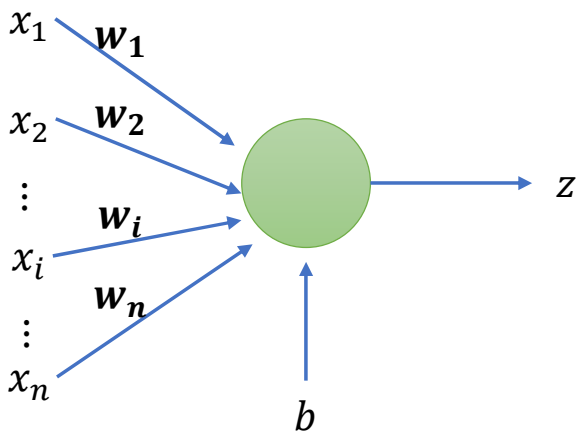
- Cuando z_L es muy grande, la neurona se satura
- Cuando z_L es muy pequeño (negativo), neurona se satura
- Nuestra meta es mantener valores alrededor de $z_L \approx 0$

Esta es la razón de porqué la distribución normal $\mathcal{N}(0, \sigma)$ es preferida para el análisis

- Media cero y desviación estándar acotada
- Este análisis es también correcto para otras funciones de activación

Inicialización Xavier

Hagamos el análisis para una neurona



Asumimos que las entradas tienen *zero mean*.
También asumimos que los pesos tienen zero mean. La pregunta es

Qué varianza de pesos necesitamos para preservar la variance entre entradas x y salida z?

$$Var(z) = Var(x)$$

Glorot and Bengio concluyeron que $Var(w) = \frac{1}{n}$. Significa que los pesos tienen que salir de una distribución $\mathcal{N}(0, {}^{1}/_{\sqrt{n}})$

Glorot and Bengio, Understanding the difficulty of training Deep feedforward neural networks. Int. Conf. Artif. Intell. and Stat. 2010.

Inicialización Kaiming He

Inicialización Xavier funciona bien para entradas con media cero. No obstante, ReLU no produce salidas con media cero.

Asumimos que los pesos tienen media cero. La pregunta es

Qué varianza de pesos necesitamos para preservar la variance entre entradas x y salida z?

$$Var(z) = Var(x)$$

He et al. concluyeron que $Var(w) = \frac{2}{n}$. Significa que los pesos tienen que salir de una distribución $\mathcal{N}(0, \sqrt{2/n})$

Análisis en material suplementario en U-Cursos

He et al., Delving deep into rectifiers. CVPR. 2015.

Resumen

- Initialización es importante para mantener valores de la computación acotados
- Previene la saturación de neuronas
- Ayuda a la optimización

Redes neuronales

Algoritmos de optimización

Gradiente descendente revisitado

Nuestro procedimiento de optimización consiste de aplicar la regla de actualización en los parámetros para reducir el error de la función de costo

$$w = w - \eta \frac{\partial C}{\partial w}$$

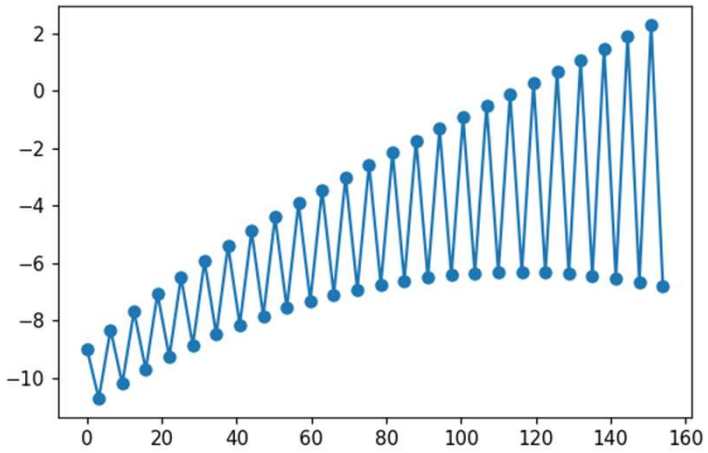
Problemas con algoritmo clásico:

- Dependencia de learning rate η
- Direcciones oscilantes en el mínimo
- Quedarse pegado en plateau o saddle points

Necesitamos mejores estrategias para actualizar parámetros

Exponentially weighted average

Supongamos que tenemos una secuencia de valores que varían en el tiempo y_t (time observations)



Queremos computar la tendencia de la secuencia aplicando promedios en una ventana de tiempo.

En lugar de usar un promedio simple en una ventana de tiempo, usamos pesos con decaimiento exponencial para el promedio y un estimado inicial v_0. Para estimar el t-ésimo elemento en la secuencia, usamos

$$v_t = \beta v_{t-1} + (1-\beta) y_t$$

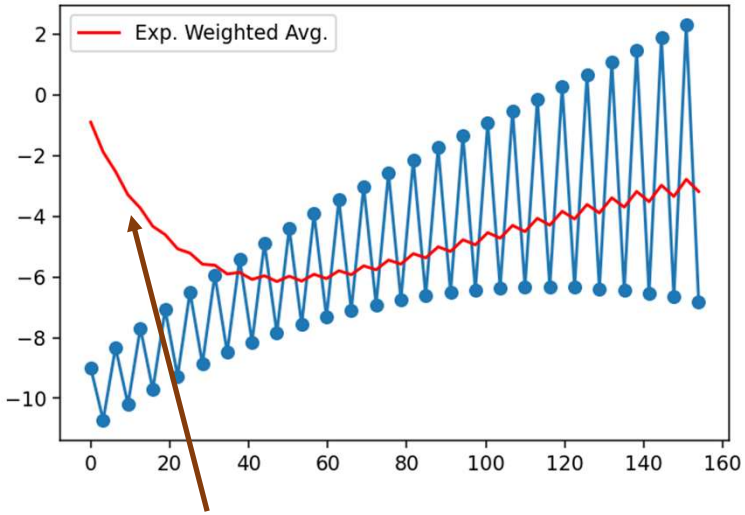
Promedio del tiempo previo

Observación actual

Parámetro $\beta < 1$ es el factor para el promedio ponderado

Exponentially weighted average

Let's suppose we have a sequence of time-varying values y_t(time observations)



First averages do not have enough information in the time window

We want to compute the trend of the sequence by applying averages in a time window

Instead of using a simple average on time window, we use an exponential decay weights for the average and an initial Estimate v_0. To estimate the t-th element in the estimated sequence, we use

$$v_t = \beta v_{t-1} + (1-\beta) y_t$$

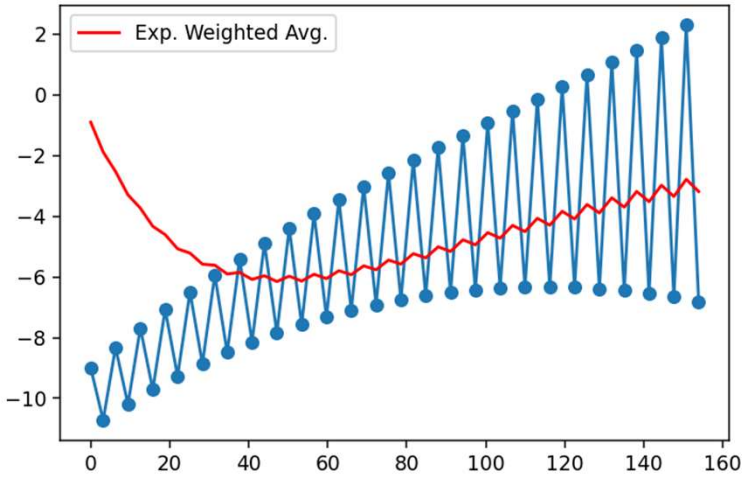
Average of previous time

Current observation

Parameter $\beta < 1$ is the factor for the weighted average

Exponentially weighted average

Let's suppose we have a sequence of time-varying values y^t (time observations)



We want to compute the trend of the sequence by applying averages in a time window

Instead of using a simple average on time window, we use an exponential decay weights for the average and an initial estimate v^0. To estimate the t-th element in the estimated sequence, we use

$$v_t = \beta v_{t-1} + (1-\beta) y_t$$

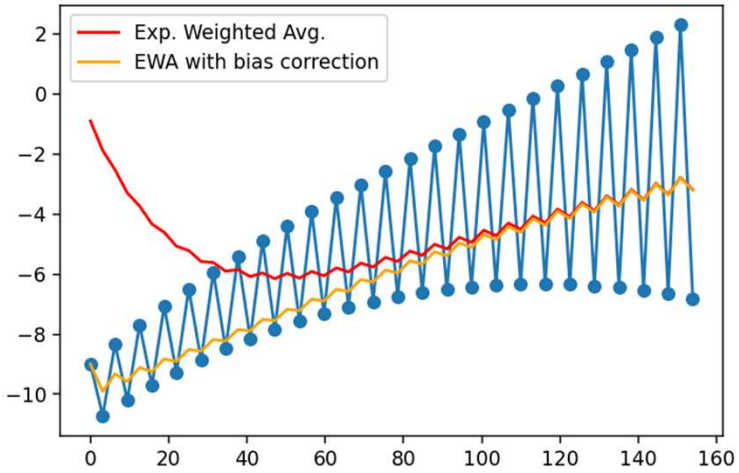
$$v_t = \frac{v_t}{1-\beta^t}$$

Bias correction

Parameter $\beta < 1$ is the factor for the weighted average

Exponentially weighted average

Let's suppose we have a sequence of time-varying values y^t (time observations)



We want to compute the trend of the sequence by applying averages in a time window

Instead of using a simple average on time window, we use an exponential decay weights for the average and an initial estimate v^0. To estimate the t-th element in the estimated sequence, we use

$$v_t = \beta v_{t-1} + (1-\beta) y_t$$

$$v_t = \frac{v_t}{1-\beta^t}$$

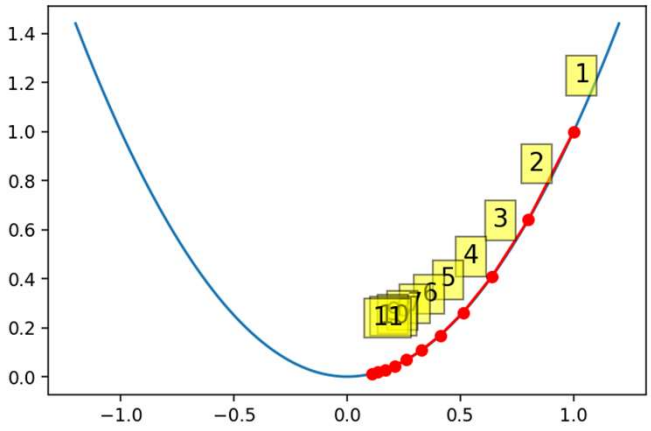
Bias correction

Parameter $\beta < 1$ is the factor for the weighted average

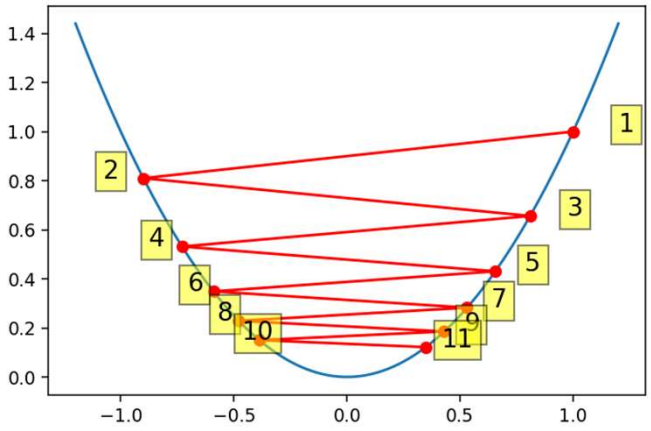
Gradient descent with momentum

When we are trying to optimize the cost function with gradient descent, the gradient vector is the direction to reach the minimum.
A big learning rate can produce oscillating directions.

$\eta = 0.1$



$\eta = 0.95$



Key observation: gradient vectors form a time-varying sequence

Momentum!

Use exponentially weighted average over gradient to smooth the transitions

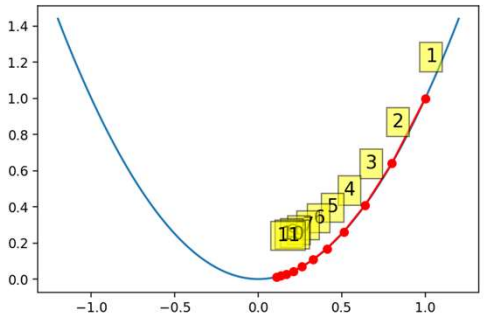
Gradient descent with momentum

The update rule for momentum is

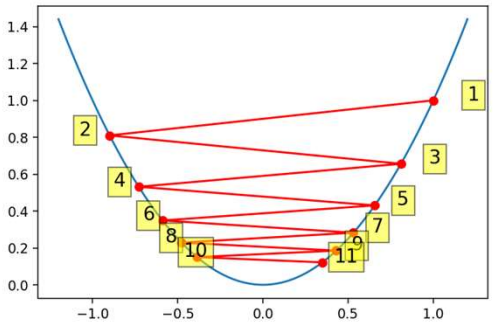
$$v_t = \beta v_{t-1} + (1-\beta)\frac{\partial C}{\partial w}$$

$$v_t = \frac{v_t}{1-\beta^t}$$

$$w = w - \eta v_t$$



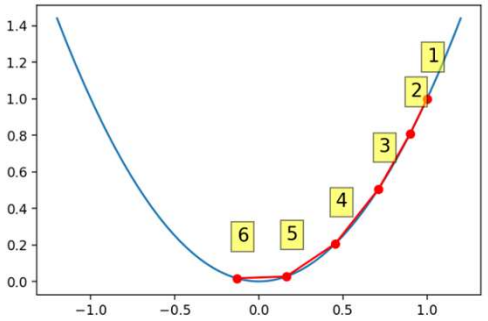
$\eta = 0.1$



$\eta = 0.95$

Momentum converges faster

In practice, momentum parameter β is commonly set to 0.9



$\eta = 0.95$

RMSProp

Idea: Perform an exponentially weighted average over the square of the gradients instead of gradient themselves

$$v_t = \beta v_{t-1} + (1-\beta)\left(\frac{\partial C}{\partial w}\right)^2$$

$$w = w - \frac{\eta}{\sqrt{v_t + \epsilon}}\frac{\partial C}{\partial w}$$

Each parameter has a cache variable that records the squared weighted average

Adam (Adaptive Moment Estimation)

Combination of Momentum + RMSProp

- Exponentially weighted average for the gradients
- Exponentially weighted average for the square gradients
- Bias correction

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial C}{\partial w}$$

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

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left(\frac{\partial C}{\partial w}\right)^2$$

$$\hat{v}_t = \frac{v_t}{1-{\beta_2}^t}$$

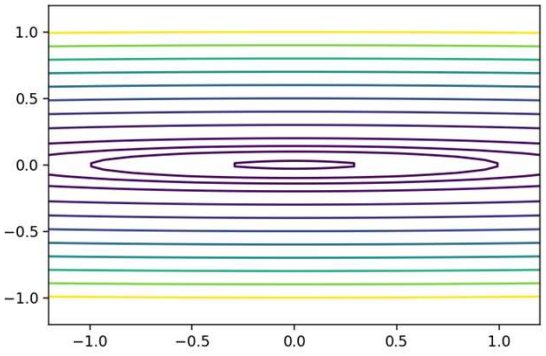
$$w = w - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}}\hat{m}_t$$

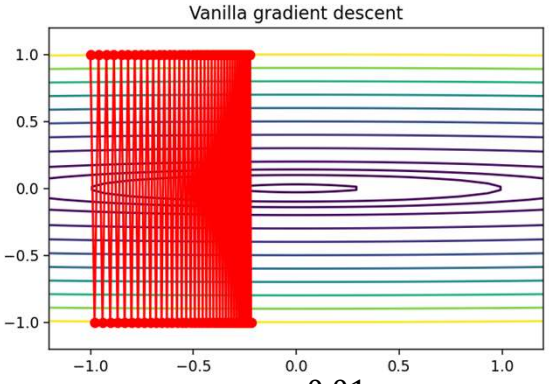
Typical hyper-parameters

- $\beta_1 = 0.9$
- $\beta_2 = 0.999$

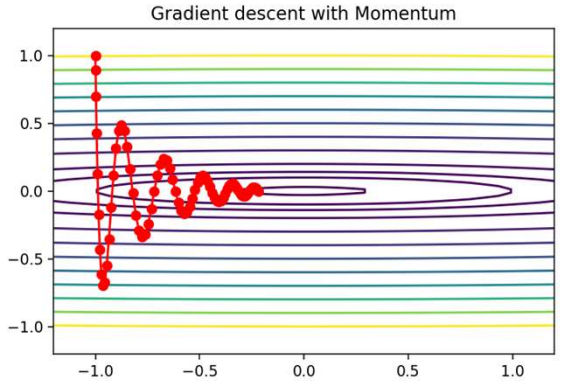
Comparison of optimization algorithms

We define a 2D optimization domain whose level contours are

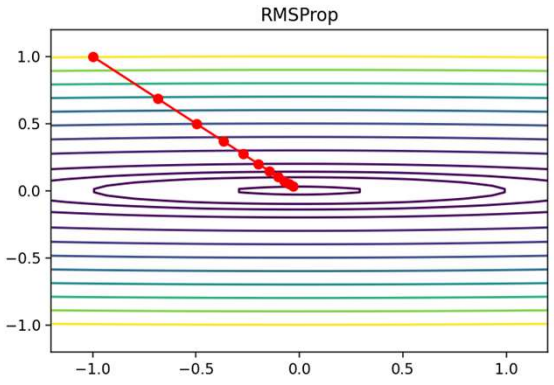




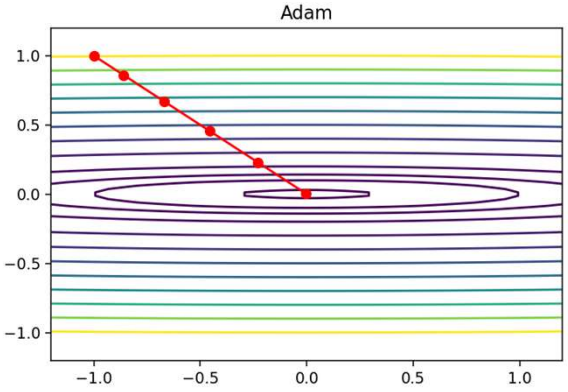
$\eta = 0.01$



$\eta = 0.01$



$\eta = 0.1$



$\eta = 0.1$

Summary

- Good convergence of exponentially weighted average gradients
- In practice, the best strategy is to combine first and second moments of gradients in the computation.