Tecnológico de Monterrey

Aprendizaje automático

Estrategias para el ajuste de redes neuronales

Dr. Omar Mendoza Montoya

04/09/2024

¿Qué problemas hay en el entrenamiento de una red neuronal multicapa?

Problemas en el entrenamiento de redes neuronales

- Diferenciabilidad de la función de costo
- Sobreajuste
- Escalado y normalización de los datos
- Inicialización del algoritmo
- Selección de hiperparámetros
- Problemas en el algoritmo de optimización
  - Convergencia prematura
  - Mínimos locales
  - Estabilidad
  - Desaparición y explosión del gradiente
- Razón de aprendizaje

Diferenciabilidad de la función de costo

- $L(D)$ debe ser una función continua y diferenciable respecto a cada uno de los pesos de la red.

- La selección de las funciones de activación como la función de costo debe permitir esta propiedad.
  - Para funciones de activación continuas y diferenciables, las funciones de pérdida típicas (con excepción de aquellas que involucran el valor absoluto), se cumple la continuidad y diferenciabilidad.

- La función ReLU es una excepción a esta regla, la cual en la práctica ha demostrado que no es un problema el hecho de que derivada esté indeterminada en el origen.

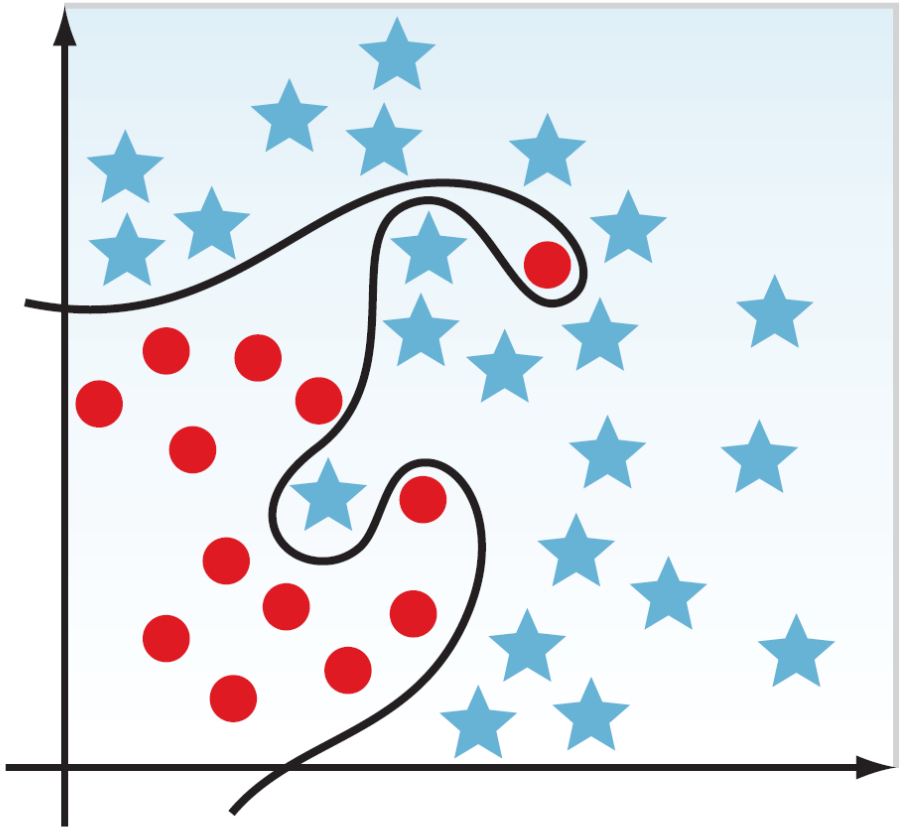
Sobreajuste

- Sobreajuste se refiere al hecho de entrenar un modelo para un conjunto particular de datos sin garantizar que se va a obtener un rendimiento adecuado al momento de predecir observaciones diferentes a las del conjunto de entrenamiento.

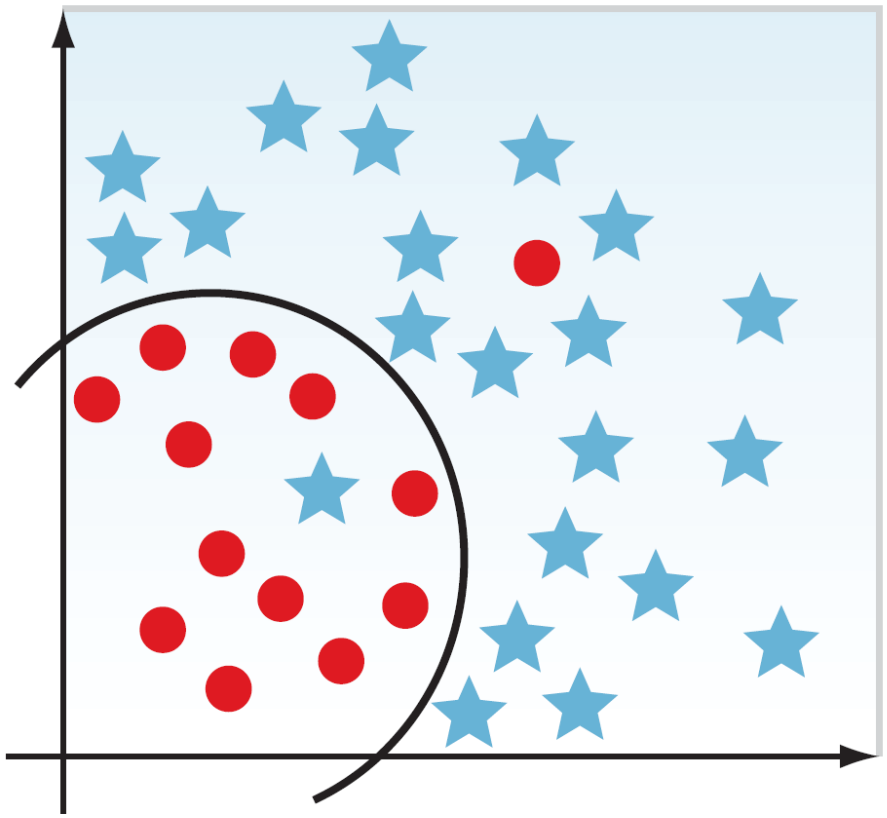
- Debido a que en una red neuronal es posible incrementar la cantidad de pesos del modelo aumentando el número de neuronas y capas, es posible estar en una situación en la que haya más parámetros que observaciones.

Cuando se tienen más parámetros que observaciones de entrenamiento, es posible que la red neuronal no tenga problemas en ajustarse perfectamente a los datos y tener un error de cero en el entrenamiento, aun cuando sea pésimo en predecir para nuevas observaciones.

Sobreajuste



Modelo con sobreajuste
Elgendy, 2020



Modelo con un ajuste correcto
Elgendy, 2020

Consenso de la comunidad: al menos entre 2 o 3 veces más observaciones que parámetros del modelo, aunque el número preciso de instancias de entrenamiento no se conoce de antemano.

Regularización

- Otra alternativa para el sobreajuste es agregar un término de regularización al modelo, de tal manera que los parámetros menos relevantes desaparezcan en la optimización de la función de costo:

$$L^*(D) = L(D) + \lambda \|\bar{W}\|^p$$

donde λ es el parámetro de regularización.

- Típicamente, $p = 2$ (regularización de Tykhonov), con lo que la regla de aprendizaje queda como sigue:

$$\bar{W} \Leftarrow \bar{W}(1 - \alpha\lambda) - \alpha\nabla L(D)$$

Dropout

- Una estrategia muy popular para combatir el sobreajuste en algunas arquitecturas profundas como las CNN es el Dropout.

- En esta estrategia, se utiliza un hiperparámetro p que indica la proporción de neuronas que se van a "apagar" en una capa de manera intencionada en una iteración del algoritmo de optimización.

- Las neuronas apagadas de una iteración no se consideran en la actualización de pesos sólo para dicha iteración. Una neurona que se haya apagado en un momento dado puede ser "encendida" en otra iteración del algoritmo de optimización.

- En el modelo final si se consideran todas las neuronas.

Dropout

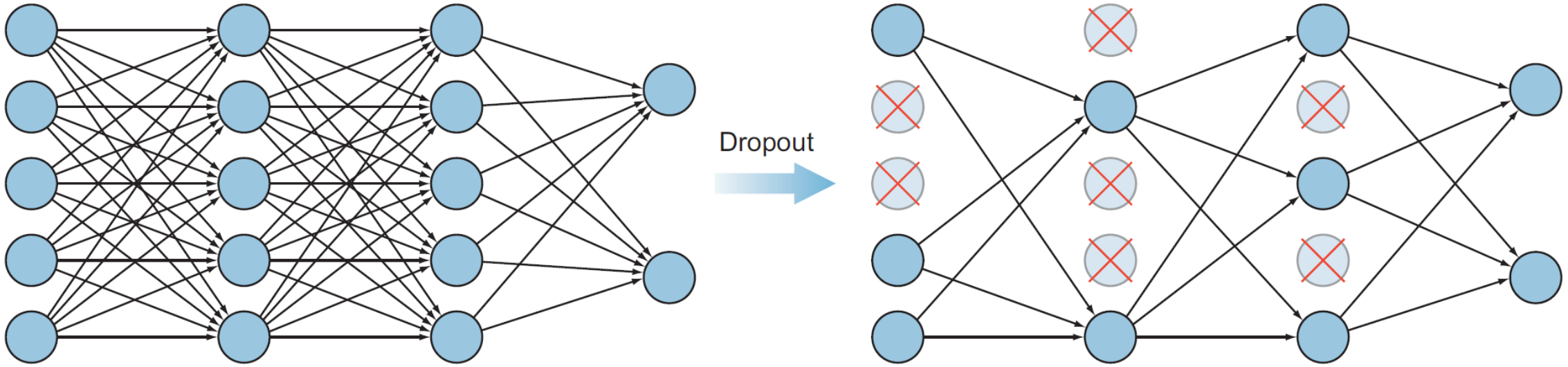


Figure 3.31 Dropout turns off a percentage of the neurons that make up a network layer.

Elgendy, 2020

Otras estrategias para el sobreajuste

- Compartir los parámetros de modelos ya pre-entrenados.

- Convergencia prematura. Detener el proceso de descenso de gradiente prematuramente cuando el error con un conjunto de prueba seleccionado aleatoriamente comience a aumentar.

- Aumentar la cantidad de capas y reducir el número de neuronas por capa. En teoría, un modelo con una sola capa oculta puede obtener el rendimiento de una configuración de múltiples capas, pero esto se logra aumentando considerablemente la cantidad de neuronas. Es decir, configuraciones profundas requieren menos neuronas.

Ajuste de hiperparámetros

- Las redes neuronales suelen tener un espacio de hiperparámetros más grande que otras técnicas de aprendizaje computacional (número de neuronas, capas, termino de regularización).

- Al igual que en otras técnicas de aprendizaje, se requiere de un conjunto de datos no visto por la etapa de entrenamiento (datos de validación).

- La manera como se encuentran dichos hiperparámetros es probando diferentes combinaciones de valores con el conjunto de validación, para determinar cuál nos proporciona el mejor rendimiento.

La estrategia de búsqueda de hiperparámetros más sencilla se conoce como Grid Search.

Los valores a probar se escogen dividiendo el espacio de búsqueda en una rejilla de manera uniforme, o con una escala logarítmica.

Preprocesado y normalización

- El rango y la escala de los datos puede influir en la optimización de los parámetros de la red neuronal.

- En ocasiones, es deseable ajustar todos los datos para que estén entre un rango fácil de manejar, por ejemplo, entre -1 y 1, o entre 0 y 1.

- También es importante detectar posibles variables predictoras que no tengan información relevante.

Preprocesado y normalización

- Estandarización de los datos. Se calcula la media y la desviación estándar para cada característica. Luego para cada dato que pase por la red se le resta dicha media y esta diferencia se divide entre la desviación estándar.

- Normalización min-max. Se determina el mínimo y máximo valor de cada característica y luego posteriormente a cada variable de un dato nuevo restar el valor mínimo y luego dividirlo entre la diferencia entre máximo y mínimo. Este proceso escala los datos en el rango de $[0, 1]$.

Este tipo de técnicas no garantizan un mejor rendimiento, pero es común que las características varíen en varios ordenes de magnitud. Con esto se evitan dichas variaciones y se evitan algunos problemas por darle más prioridad a ciertas características.

Preprocesado y normalización

- Blanqueado. En este caso el sistema de coordenados es rotado de tal forma que se reduce o elimina las correlaciones entre características. En este caso PCA puede ayudar para llevar a cabo esta operación.

Al tener variables sin correlación se garantiza que cada una de ellas aporte al modelo de aprendizaje. Si dos variables están altamente correlacionadas, el modelo sólo requiere de una de ellas, ya que la otra tiene información redundante.

Inicialización

- Debido a que no hay garantía que se llegue al óptimo global de la función de costo, los valores iniciales pueden afectar en la solución que se obtenga al final del proceso de entrenamiento.

- Una manera sencilla de inicializar los pesos es muestrearlos de una distribución Gaussiana con media 0 y una desviación estándar pequeña (por ejemplo 0.01).

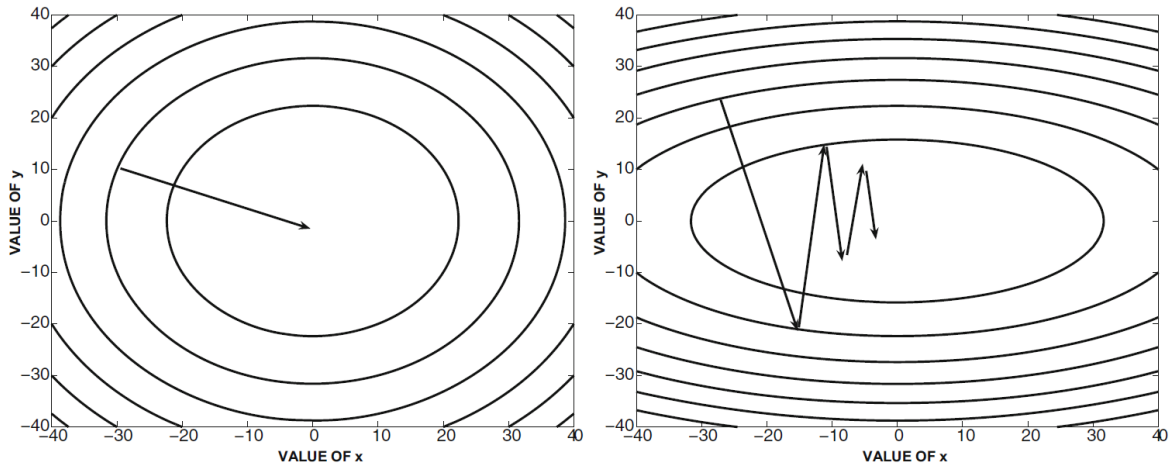
- Sin embargo, este enfoque no considera que hay neuronas con un mayor número de entradas, lo que originaría que la salida pudiera estar fuera del rango que se esperaría.

Una estrategia práctica para inicializar los pesos de la red consiste en variar la desviación estándar de la distribución Gaussiana de la que se muestrean los pesos iniciales para que se adapte a la cantidad de entradas ($1/\sqrt{r}$, donde r es el número de entradas.

Estabilidad del algoritmo de optimización

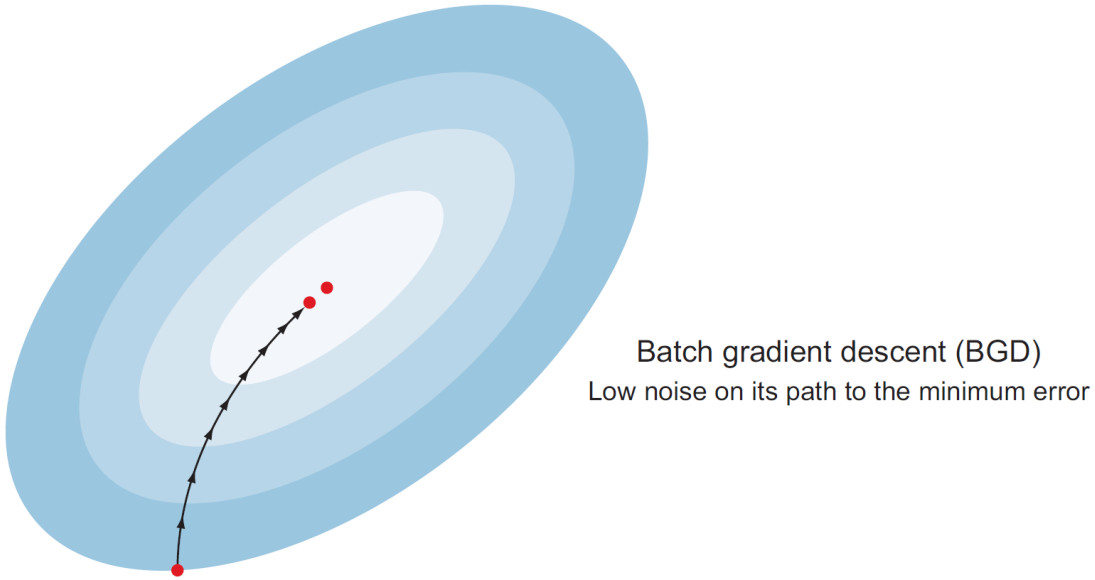
- El algoritmo suele tener problemas de estabilidad en su versión de descenso de gradiente estocástico. Esto se alivia con la versión mini lote del algoritmo, en el cual, para cada actualización, se suman varios gradientes individuales:

$$\bar{W} \Leftarrow \bar{W} - \alpha \sum_{i \in B} \nabla L_i$$



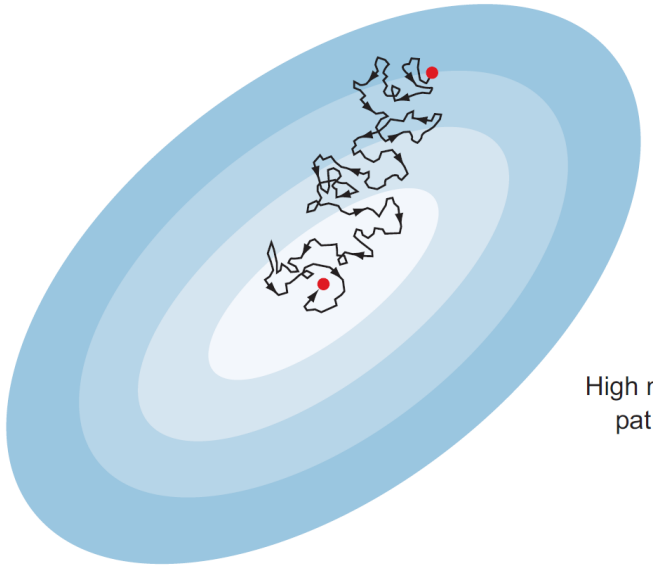
Aggarwal, 2023

Estabilidad del algoritmo de optimización



Batch gradient descent (BGD)
Low noise on its path to the minimum error

Figure 4.21 Batch GD with low noise on its path to the minimum error



Stochastic (GD)
High noise and oscillates on its path to the minimum error

Figure 4.22 Stochastic GD with high noise that oscillates on its path to the minimum error

Elgendy, 2020

La desaparición y la explosión del gradiente

- Entrenar redes neuronales profundas puede representar un problema en la práctica, sobre todo por la forma como se calculan las derivadas para los pesos en las primeras y últimas capas.

- Las derivadas parciales dependen de los valores de las derivadas de las funciones de activación. Dichas derivadas se multiplican una y otra vez conforme se va de adelante hacia atrás en la red.

Si la derivada de la función de activación está acotada entre -1 y 1, al multiplicarse una y otra vez el producto se va a cero. Con ello, las derivadas para los pesos de las primeras capas se vuelven cercanas a cero, por lo que la actualización de los pesos en dichas capas es mínima (desaparición del gradiente).

Si se intenta corregir la desaparición del gradiente modificando las funciones de activación para que tengan valores más grandes, puede ocurrir lo opuesto y el gradiente aumentar en magnitud sin límite (explosión del gradiente).

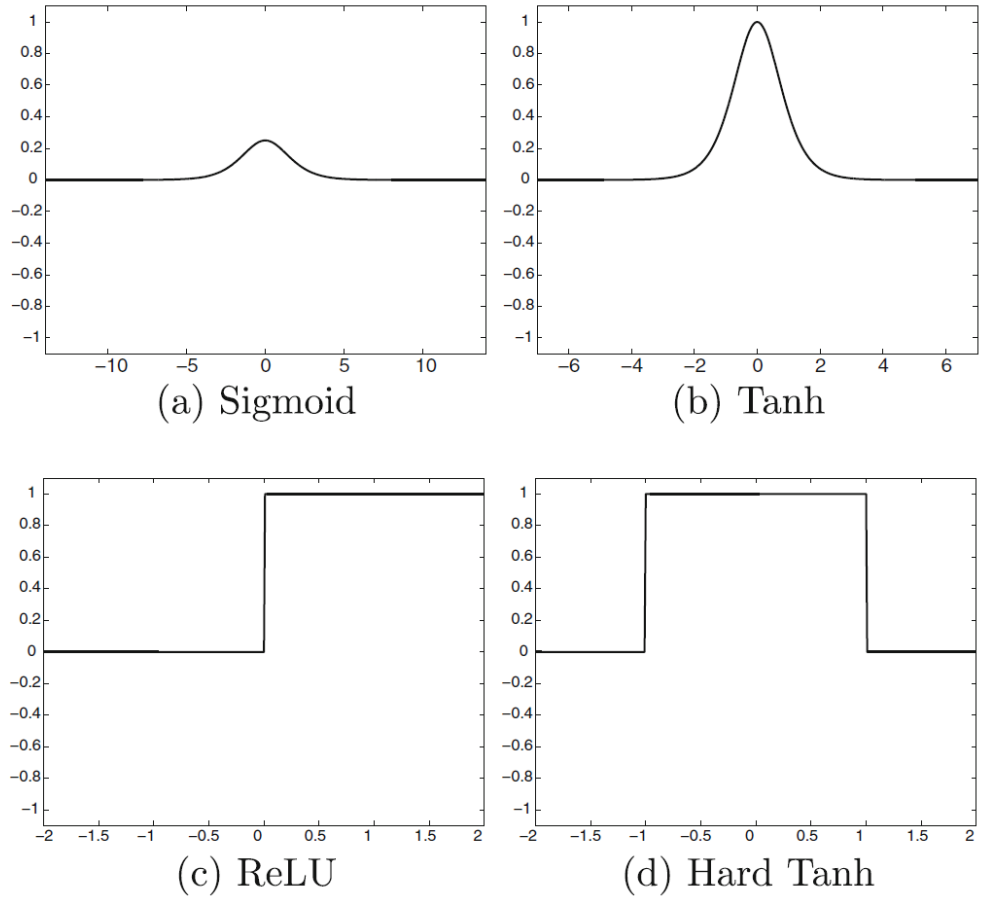
Desaparición y la explosión del gradiente

- El problema de desaparición del gradiente es típico en métodos de optimización basados en el gradiente.

- Si el gradiente es pequeño, significa o que ya se está en el óptimo, o que para llegar al óptimo es necesario realizar una cantidad enorme de pequeños pasos llevar a cabo constantes correcciones en la dirección de optimización. Esto depende de la función de costo a optimizar.

- Funciones como la ReLU y la tangente hiperbólica dura se han vuelto populares para arquitecturas profundas debido a que sus derivadas son 0 o 1. Esto alivia un poco el problema del desvanecimiento del gradiente, aunque muchos problemas de inestabilidad siguen aun presentes.

Desaparición y la explosión del gradiente



Aggarwal, 2023

Figure 3.10: The derivatives of different activation functions are shown. Piecewise linear activation functions have local gradient values of 1.

Neuronas muertas

- Funciones como la ReLU pueden provocar el problema de las neuronas muertas.

- Para la función ReLU, si la entrada de una neurona es negativa, la neurona siempre responderá con $\mathbf{0}$. A su vez, sus derivadas pueden saltar a 0, por lo que nunca se actualizarían sus pesos.

- Una alternativa para la función ReLU es la función leaky ReLU.

$$g(x) = \begin{cases} \beta x & x \leq 0 \\ x & \text{otros casos} \end{cases} \qquad \beta \in (0,1)$$

Razón de aprendizaje

- Taza de aprendizaje variable. Una taza de aprendizaje α pequeña para la actualización de los pesos ocasiona que se requiera una cantidad considerable de pasos para alcanzar la solución. Por otro lado, un valor alto de α origina problemas de estabilidad, sobre todo alrededor del óptimo, aunque haya actualizaciones importantes al principio.

  - Una opción es utilizar una taza de aprendizaje variable, como la taza con decaimiento exponencial o el decaimiento inverso, de tal forma que conforme se avanza en la optimización, se disminuya el valor de α.

$$\alpha_t = \alpha_0 e^{-kt} \quad \text{Decaimiento exponencial}$$

$$\alpha_t = \frac{\alpha_0}{1 + kt} \quad \text{Decaimiento inverso}$$

Razón de aprendizaje

- Taza de aprendizaje con momentum. Otra alternativa para el problema de la razón de aprendizaje es determinar si hay un zigzagueo en el gradiente, lo que indicaría inestabilidad.

  - Esto se puede corregir evitando que haya cambios bruscos en la dirección del gradiente de acuerdo a su momentum.

$$\bar{V} \Leftarrow -\alpha \nabla L(D, \bar{W}) \qquad \bar{W} \Leftarrow \bar{W} + \bar{V} \qquad \text{Sin momentum}$$

$$\bar{V} \Leftarrow \beta \bar{V} - \alpha \nabla L(D, \bar{W}) \qquad \bar{W} \Leftarrow \bar{W} + \bar{V} \qquad \text{Con momentum} \quad (\beta = \text{parámetro de fricción})$$

$$\bar{V} \Leftarrow \beta \bar{V} - \alpha \nabla L(D, \bar{W} + \beta \bar{V}) \qquad \bar{W} \Leftarrow \bar{W} + \bar{V} \qquad \text{Momentum de Nesterov} \quad (\beta = \text{parámetro de fricción})$$

Razón de aprendizaje

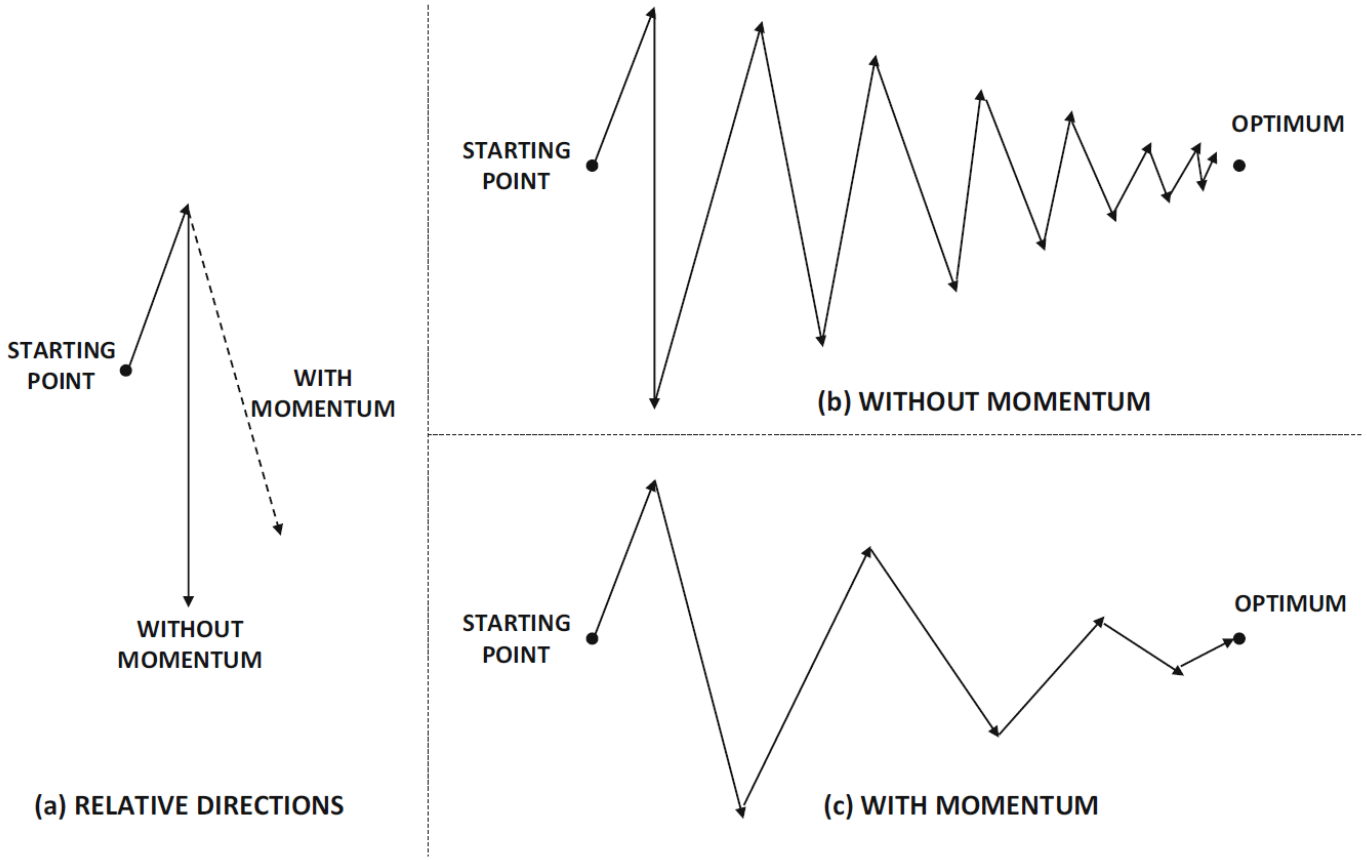


Figure 3.11: Effect of momentum in smoothing zigzag updates

Aggarwal, 2023

Razón de aprendizaje por parámetro

- Se puede tener una razón de aprendizaje para cada parámetro y que ese se adapte durante el proceso de optimización.
- AdaGrad

$$A_i \Leftarrow A_i + \frac{\partial L}{\partial w_i} \qquad w_i \Leftarrow w_i - \frac{\alpha}{\sqrt{A_i}} \frac{\partial L}{\partial w_i}$$

- RMSProp con momentum de Nesterov

$$v_i \Leftarrow \beta v_i - \frac{\alpha}{\sqrt{A_i}} \frac{\partial L(\bar{W} + \beta \bar{V})}{\partial w_i} \qquad w_i \Leftarrow w_i + v_i$$

- AdaDelta
- Adam

Bibliografía

- Aggarwal, C. C. (2023). *Neural networks and deep learning* (2da ed.). Springer.
  - Capítulo 2
  - Capítulo 4
  - Capítulo 5

- Elgendy, M. (2020). *Deep learning for vision systems* (1ra ed.). Manning Publications Co.
  - Capítulo 4