Deep Learning

Episodio 2: Redes Neuronales Completas

Fernando Gama

Escuela de Graduados en Ingeniería Informática y Sistemas, Facultad de Ingeniería, UBA

7 de Julio de 2022

Escenas del capítulo anterior...

- Programación tradicional $\Rightarrow$ Especificar cada instrucción a la computadora
- Aprendizaje automático $\Rightarrow$ Determinar las operaciones posibles y dejar que la computadora elija
  $\Rightarrow$ La computadora elige mirando datos

- Describir los datos de manera numérica $\Rightarrow$ Matemática $\Rightarrow$ Álgebra y Probabilidad
  $\Rightarrow$ Determinan las operaciones posibles sobre los datos

- Algoritmos que funcionen bien sobre datos no vistos $\Rightarrow$ Generalización

¿Qué es aprender? ¿Cómo describimos el aprendizaje?

- Necesitamos definir, de alguna manera, qué quiere decir aprender
  - ⇒ Realizamos una observación. Inferimos alguna conclusión. La contrastamos.
  - ⇒ La próxima vez que nos topemos con una observación similar, queremos la inferencia adecuada

- Sabemos que para definir algo de manera unívoca, necesitamos usar matemática
  - ⇒ Una observación es un dato que denotamos como $\mathbf{x}$
  - ⇒ La inferencia la llamamos información y la denotamos con $\mathbf{y}$
  - ⇒ El proceso de inferencia tiene que ser una función $\mathsf{f} : \mathbf{x} \mapsto \mathbf{y}$
  - ⇒ El contraste con la realidad lo tenemos que *medir*, con otra función J

Similitudes

- La próxima vez que nos topemos con una observación similar, queremos la inferencia adecuada
  - $\Rightarrow$ Determinar qué quiere decir que dos observaciones sean similares

- Adoptamos una concepción probabilística
  - $\Rightarrow$ Dos muestras son similares si vienen de la misma distribución de probabilidades

$$\boldsymbol{X} \sim \mathrm{f}_x(\mathbf{x})$$

  - $\Rightarrow$ Las observaciones son realizaciones de la variable aleatoria

$$\boldsymbol{X} \rightsquigarrow \{\mathbf{x}_1, \mathbf{x}_2, \ldots\}$$

- Si las observaciones futuras no vienen de la misma distribución
  - $\Rightarrow$ No podemos esperar que nuestro aprendizaje funcione
  - $\Rightarrow$ No existe un único algoritmo de aprendizaje que funcione para todo

- Tenemos un conjunto de datos que provienen de una misma distribución $\boldsymbol{X}$
- Tenemos un algoritmo de aprendizaje f que proporciona una inferencia $\mathbf{y}$
- Tenemos una medida de qué tan errónea es la inferencia J

- Con todo esto podemos definir la noción del **riesgo** R del algoritmo f

$$\mathsf{R}\{\mathsf{f}\} = \mathbb{E}_{\boldsymbol{X} \sim \mathsf{f}_x}\left[\mathsf{J}(\mathsf{f}(\boldsymbol{X}))\right]$$

  $\Rightarrow$ Obtenemos la inferencia $\boldsymbol{Y} = \mathsf{f}(\boldsymbol{X})$ resultante de aplicar el algoritmo de aprendizaje f

  $\Rightarrow$ Calculamos cuán errónea es esa inferencia $\mathsf{J}(\mathsf{f}(\boldsymbol{X}))$

  $\Rightarrow$ Obtenemos el error medio sobre todos los datos posibles a través de su distribución

Minimización del Riesgo

- El riesgo es una medida para saber qué tan bueno es nuestro algoritmo de aprendizaje
- Podemos, entonces, buscar el algoritmo de aprendizaje que minimiza el riesgo

$$\min_{\mathsf{f}} \mathsf{R}\{f\} = \min_{\mathsf{f}} \mathbb{E}_{\boldsymbol{X} \sim \mathsf{f}_x} \left[\mathsf{J}(\mathsf{f}(\boldsymbol{X})) \right]$$

- Esta formulación es razonable, pero de difícil implementación práctica (por no decir imposible)
  - $\Rightarrow$ ¿Cómo puedo conocer la distribución f_x de los datos?
  - $\Rightarrow$ ¿Cómo puedo resolver el problema de minimización?

Ley de los Grandes Números

- No conocemos la distribución f_x de los datos
- Pero tenemos acceso a un gran conjunto de datos

$$\boldsymbol{X} \rightsquigarrow \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$$

$\Rightarrow$ Las muestras $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ son independientes, idénticamente distribuidas (i.i.d.)

- Si las muestras $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ son i.i.d.
- Y si la cantidad de muestras M es lo suficientemente grande

$$\mathbb{E}_{\boldsymbol{X} \sim \mathsf{f}_x}\left[\mathsf{g}(\boldsymbol{X})\right] \approx \frac{1}{M} \sum_{m=1}^{M} \mathsf{g}(\mathbf{x}_m)$$

$\Rightarrow$ Este resultado se conoce como la Ley de los Grandes Números

$\Rightarrow$ El promedio se parece a la media si la cantidad de muestras es grande y son i.i.d.

Minimización del Riesgo Empírico

- Tenemos acceso a una gran cantidad de datos $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$
- Asumimos que son i.i.d. y usamos la Ley de los Grandes Números
- Proponemos, entonces, resolver el problema de minimización del riesgo empírico

Minimización del Riesgo Empírico

$$\min_{\mathsf{f}} \frac{1}{M} \sum_{m=1}^{M} \mathsf{J}(\mathsf{f}(\mathbf{x}_m))$$

- La función f que encontremos va a ser un algoritmo de aprendizaje
- Pero también queremos que funcione en datos no observados $\Rightarrow$ Generalización
  $\Rightarrow$ Esto será cierto si los datos no observados tienen la misma distribución
- Esto resuelve el problema de no conocer la distribución de los datos
  $\Rightarrow$ Simplemente juntamos muchos datos y usamos la Ley de los Grandes Números
- Pero la otra pregunta persiste: ¿Cómo minimizamos el riesgo empírico?

- El problema de minimización del riesgo empírico es el siguiente

$$\min_{\mathsf{f}} \hat{\mathsf{R}}(\mathsf{f}) = \min_{\mathsf{f}} \frac{1}{M} \sum_{m=1}^{M} \mathsf{J}(\mathsf{f}(\mathbf{x}_m))$$

- En este contexto, el riesgo empírico es una función $\hat{\mathsf{R}}(\mathsf{f})$ de una función f
  $\Rightarrow$ En matemática, esto se conoce como un funcional (asigna una función a un número)

- Encontrar la *función* que minimiza otra función es un problema difícil en extremo
  $\Rightarrow$ ¿Cómo diseñar un algoritmo que actúe sobre funcionales? (Álgebra y Análisis Matemático)
  $\Rightarrow$ Optimización funcional $\Rightarrow$ Cálculo de variaciones, métodos de kernels, etc.
- Es mucho más fácil encontrar vectores (o matrices, o tensores) que minimizan una determinada función
  $\Rightarrow$ Transformar un problema de optimización funcional en uno de optimización paramétrica

- Elegir una familia paramétrica $\Rightarrow$ La función f depende de ciertos parámetros $\boldsymbol{\theta}$

$$\mathsf{f}(\mathbf{x}) = \mathsf{f}(\mathbf{x}; \boldsymbol{\theta})$$

  $\Rightarrow$ Ahora el riesgo empírico es una función de un vector $\hat{\mathsf{R}}(\mathsf{f}) = \hat{\mathsf{R}}(\boldsymbol{\theta})$ $\Rightarrow$ Una función

  $\Rightarrow$ Y pensar en un algoritmo que opere sobre vectores es mucho más factible

- Ciertamente, ahora no se minimiza sobre *todas* las funciones posibles

  $\Rightarrow$ Se minimiza sobre todos los parámetros que caracterizan a la función f

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^D} \frac{1}{M} \sum_{m=1}^{M} \mathsf{J}\big(\mathsf{f}(\mathbf{x}_m; \boldsymbol{\theta})\big)$$

  $\Rightarrow$ Pero ahora es un problema que se puede resolver una forma mucho más fácil

Ejemplo: Parametrización Lineal

- Ahora la pregunta pasa a ser: ¿Qué parametrización elijo?
  - $\Rightarrow$ Este es el problema fundamental en los problemas de aprendizaje
  - $\Rightarrow$ Elegir una parametrización adecuada determina el éxito del algoritmo de aprendizaje
- La elección más elemental posible es una parametrización lineal

$$\mathsf{f}(\mathbf{x}; \mathbf{W}, \mathbf{b}) = \mathbf{W}\mathbf{x} + \mathbf{b}$$

- Ejemplo del episodio 1: Dados datos $\{(\mathbf{x}_m, y_m)\}$ y función de costo $\mathsf{J}(y, \mathsf{f}(\mathbf{x})) = (y - \mathsf{f}(\mathbf{x}))^2$
- Se adopta una parametrización lineal $\mathsf{f}(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{x}$ $(\mathbf{W} \Rightarrow \mathbf{w}, \mathbf{b} \Rightarrow 0)$
- Resulta fácil encontrar cuál es la solución al problema de minimización del riesgo empírico

$$\min_{\mathbf{w}\in\mathbb{R}^N} \frac{1}{M} \sum_{m=1}^{M} \left(y_m - \mathbf{w}^\mathsf{T}\mathbf{x}_m\right)^2 \quad \Rightarrow \quad \mathbf{w}^\star = \Big[\sum_{m=1}^{M} \mathbf{x}_m \mathbf{x}_m^\mathsf{T} \Big]^{-1} \Big[\sum_{m=1}^{M} y_m \mathbf{x}_m \Big]$$

Ejemplo: Parametrización Lineal

- En vez de resolver el problema sobre todas las posibles funciones f

$$\min_{\mathbf{w}\in\mathbb{R}^N} \frac{1}{M}\sum_{m=1}^{M}\left(y_m - \mathbf{w}^\mathsf{T}\mathbf{x}_m\right) \quad \Rightarrow \quad \mathbf{w}^\star = \Big[\sum_{m=1}^{M}\mathbf{x}_m\mathbf{x}_m^\mathsf{T}\Big]^{-1}\Big[\sum_{m=1}^{M} y_m\mathbf{x}_m\Big]$$

  $\Rightarrow$ Lo resolvimos sobre todas las posibles funciones lineales $\mathsf{f}(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{x}$

- ¿Es esta una buena parametrización para el problema en cuestión?
  $\Rightarrow$ Lo va a ser si es cierto que la relación entre $\mathbf{x}$ e $\mathbf{y}$ es lineal $\boldsymbol{Y} = \mathbf{w}^\mathsf{T}\boldsymbol{X}$
- Las parametrizaciones lineales tienen varias ventajas
  $\Rightarrow$ Trazabailidad matemática $\Rightarrow$ Otorga garantías de funcionamiento
  $\Rightarrow$ Ofrece expresiones 'cerradas' (que no siempre son buenas para la computación)

- ¿Qué hacemos si la relación entre entrada y salida del algoritmo es no lineal?

Limitaciones del Modelo Lineal

- Muchas veces, asumimos que un modelo lineal es suficiente $\Rightarrow$ Pero a veces realmente no alcanza

- Consideremos el ejemplo de querer aprender una XOR $\Rightarrow$ Una función binaria $\mathsf{f} : \{0,1\}^2 \to \{0,1\}$

(x_1, x_2)	$\mathsf{f}(x_1, x_2)$
$(0,0)$	0
$(0,1)$	1
$(1,0)$	1
$(1,1)$	0

- Asumimos que tenemos acceso a las cuatro muestras $\{((0,0),0),((0,1),1),((1,0),1),((1,1),0)\}$
- Proponemos una parametrización lineal para la función $\mathsf{f}(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{x} + b$ con $\mathbf{w} \in \mathbb{R}^2$ y $b \in \mathbb{R}$
- Adoptamos un costo cuadrático $\mathsf{J}(y, \mathsf{f}(\mathbf{x})) = (y - \mathsf{f}(\mathbf{x}))^2$

Modelo Lineal No Aprende XOR

- Resolvemos el problema de minimización del riesgo empírico para la familia paramétrica

$$\min_{\mathbf{w}\in\mathbb{R}^2, b\in\mathbb{R}} \frac{1}{4}\sum_{m=1}^{4}\left(y_m - \mathbf{w}^\mathsf{T}\mathbf{x}_m - b\right)^2$$

- La solución viene dada por $\mathbf{w} = \mathbf{0}$ y $b = 1/2$ de tal forma que $\mathsf{f}(\mathbf{x}) = 0{,}5$

(x_1, x_2)	$\mathsf{f}(x_1, x_2)$
$(0, 0)$	0,5
$(0, 1)$	0,5
$(1, 0)$	0,5
$(1, 1)$	0,5

- Hasta en ejemplos fáciles, mirar sólo algoritmos lineales puede ser un problema
  - $\Rightarrow$ Resigno la trazabailidad matemática (y en muchos casos las garantías)
  - $\Rightarrow$ Mejoramos en el desempeño empírico de los algoritmos

- ¿Cuál es la forma más sencilla de conseguir un algoritmo no-lineal?
  - ⇒ Le agrego una función no lineal a la salida del algoritmo lineal

Perceptrón

$$\mathbf{f}(\mathbf{x}) = \sigma\left(\mathbf{W}\mathbf{x} + \mathbf{b}\right)$$

  - ⇒ $\sigma : \mathbb{R} \to \mathbb{R}$ es una función no lineal aplicada a cada elemento del vector
  - ⇒ La elección de σ es muy importante y es área activa de investigación
- Por razones neurobiológicas esta ecuación se conoce como un **perceptrón**

- El agregado de una función no lineal puntual alcanza para aprender la función XOR

$$\mathsf{f}(\mathbf{x};\boldsymbol{\Theta}) = \mathbf{w}_2{}^{\mathsf{T}}\sigma(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1)+b_2 \quad , \quad \Theta = \{\mathbf{W}_1, \mathbf{b}_1, \mathbf{w}_2, b_2\}$$

  $\Rightarrow \sigma(x) = \text{máx}\{0, x\}$ es una ReLU (*rectified linear unit*), $\mathbf{w}_2 \in \mathbb{R}^2$, $\mathbf{W} \in \mathbb{R}^{2\times 2}$

  $\Rightarrow$ Un perceptrón seguido de una función lineal es suficiente para aprender la XOR

- Los parámetros para que $\mathsf{f}(\mathbf{x};\boldsymbol{\Theta})$ sea igual a la XOR son

$$\mathbf{W}_1 = \begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \ , \ \mathbf{b}_1 = \begin{bmatrix}0\\ -1\end{bmatrix} \ , \ \mathbf{w}_2 = \begin{bmatrix}1\\ -2\end{bmatrix} \ , \ b_2 = 0$$

- Comprobamos que efectivamente lo aprende

$$\begin{bmatrix}1 & -2\end{bmatrix}\sigma\left(\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}\begin{bmatrix}0 & 0 & 1 & 1\\ 0 & 1 & 0 & 1\end{bmatrix} + \begin{bmatrix}0\\ -1\end{bmatrix}\right) = \begin{bmatrix}1 & -2\end{bmatrix}\sigma\left(\begin{bmatrix}0 & 1 & 1 & 2\\ 0 & 1 & 1 & 2\end{bmatrix} + \begin{bmatrix}0\\ -1\end{bmatrix}\right)$$

$$= \begin{bmatrix}1 & -2\end{bmatrix}\begin{bmatrix}\text{máx}\{0,\ 0\} & \text{máx}\{0,\ 1\} & \text{máx}\{0,\ 1\} & \text{máx}\{0,\ 2\}\\ \text{máx}\{0,\ -1\} & \text{máx}\{0,\ 0\} & \text{máx}\{0,\ 0\} & \text{máx}\{0,\ 1\}\end{bmatrix} = \begin{bmatrix}1 & -2\end{bmatrix}\begin{bmatrix}0 & 1 & 1 & 2\\ 0 & 0 & 0 & 1\end{bmatrix}$$

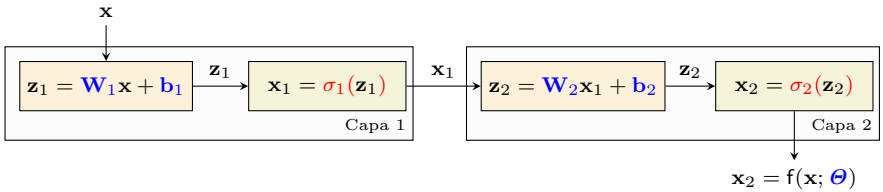
  $\Rightarrow$ Y la última multiplicación de matrices arroja $\begin{bmatrix}0 & 1 & 1 & 0\end{bmatrix}$ que es lo que queríamos

- Un perceptrón seguido de una función lineal es capaz de aprender una XOR
- Una red neuronal se define como una cascada de perceptrones

Perceptrón Multicapa

$$\mathsf{f}(\mathbf{x};\boldsymbol{\Theta}) = \mathbf{x}_L \quad \text{con} \quad \mathbf{x}_\ell = \sigma_\ell\left(\mathbf{W}_\ell \mathbf{x}_{\ell-1} + \mathbf{b}_\ell\right),\ \ell = 1,\dots,L \quad \text{y} \quad \mathbf{x}_0 = \mathbf{x}$$

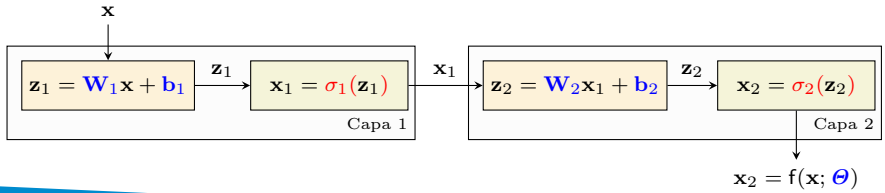
- El perceptrón multicapa es una parametrización $\Rightarrow$ Aprendemos $\boldsymbol{\Theta} = \{(\mathbf{W}_\ell, \mathbf{b}_\ell)\}_{\ell=1,\dots,L}$



Perceptrón Multicapa

$$\mathsf{f}(\mathbf{x};\boldsymbol{\Theta}) = \mathbf{x}_L \quad \text{con} \quad \mathbf{x}_\ell = \sigma_\ell\left(\mathbf{W}_\ell \mathbf{x}_{\ell-1} + \mathbf{b}_\ell\right),\ \ell = 1,\dots,L \quad \text{y} \quad \mathbf{x}_0 = \mathbf{x}$$

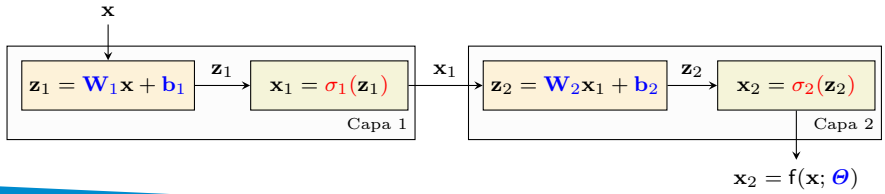
- L es la cantidad de capas del perceptrón $\Rightarrow$ Se elige por diseño $\Rightarrow$ Profundidad
- $\mathbf{x}_\ell \in \mathbb{R}^{N_\ell}$ es la salida de la capa ℓ, los parámetros son $\mathbf{W}_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ y $\mathbf{b}_\ell \in \mathbb{R}^{N_\ell}$
- N_ℓ es la dimensión del vector de salida (*hidden units*) $\Rightarrow$ Se elige por diseño $\Rightarrow$ Ancho
- σ_ℓ es una no-linealidad puntual conocida como función de activación



Perceptrón Multicapa

$$\mathsf{f}(\mathbf{x}; \boldsymbol{\Theta}) = \mathbf{x}_L \quad \text{con} \quad \mathbf{x}_\ell = \sigma_\ell\left(\mathbf{W}_\ell \mathbf{x}_{\ell-1} + \mathbf{b}_\ell\right) \;,\; \ell = 1, \ldots, L \quad \text{y} \quad \mathbf{x}_0 = \mathbf{x}$$

- El perceptrón multicapa (*multilayer perceptron*; MLP por sus siglas en inglés) recibe muchos nombres
  - $\Rightarrow$ Redes neuronales *feedforward* porque la información sólo fluye hacia adelante
  - $\Rightarrow$ Redes neuronales completas (*fully connected*) porque las matrices $\mathbf{W}_\ell$ pueden tomar cualquier valor
- El término red neuronal surge por los primeros modelos matemáticos de la actividad neuronal



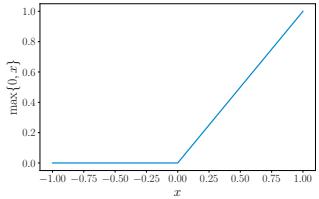
Funciones de Activación

- La inclusión de una función de activación no lineal σ_ℓ es clave $\Rightarrow$ ¿Cómo elegir σ_ℓ?
  - $\Rightarrow$ Hay muchos tipos de activaciones disponibles $\Rightarrow$ Es una elección del diseñador
  - $\Rightarrow$ Suele ser difícil determinar cuál es la apropiada
  - $\Rightarrow$ Conocer cuáles son las opciones y sus características principales ayuda
  - $\Rightarrow$ Saber de antemano cuál es la mejor de todas es prácticamente imposible
  - $\Rightarrow$ Se eligen por prueba y error observando el desempeño del algoritmo en las muestras de prueba

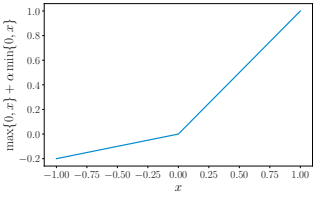
- Vamos a presentar varias de las opciones y discutir sus características principales

ReLU: Rectified Linear Units

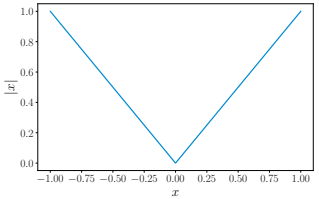
- $\text{ReLU}(x) = \max\{0, x\}$
  - $\Rightarrow$ Elección por default $\Rightarrow$ Fáciles de optimizar cuando el comportamiento es casi lineal
  - $\Rightarrow$ La unidad se considera *activa* cuando está en los positivos
  - $\Rightarrow$ Al inicializar $\mathbf{b}_\ell$ es bueno que sean positivos para asegurarse que todas las unidades están activas
- Limitaciones: Las ReLU no pueden aprender cuando están inactivas
- Extensiones: $\text{LeakyReLU}(x) = \max\{0, x\} + \alpha \min\{0, x\}$ para algún valor α a elección, valor absoluto $|x|$



ReLU



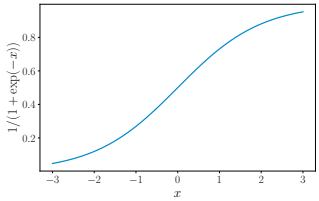
LeakyReLU



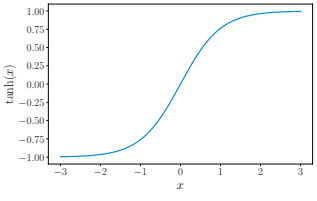
Valor absoluto

Sigmoideas y Tangente Hiperbólico

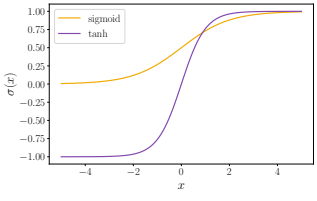
- Sigmoidea $\sigma(x) = 1/(1 + \exp(-x))$, tangente hiperbólico $\sigma(x) = \tanh(x)$
  - $\Rightarrow$ Saturan la entrada $\Rightarrow$ Es importante cuando sabemos que queremos valores limitados
  - $\Rightarrow$ Cuando saturan, no aprenden $\Rightarrow$ Son particularmente útiles si los valores están cerca de cero
  - $\Rightarrow$ Se dice que han caído en desuso $\Rightarrow$ Su utilidad depende de la función de costo J



Sigmoidea



Tangente hiperbólico



Ambas

Hiperparámetros

- Un perceptrón multicapa tiene varios valores que dependen del diseñador
  - $\Rightarrow$ La elección de la función de activación σ_ℓ
  - $\Rightarrow$ La cantidad de unidades en cada capa N_ℓ
  - $\Rightarrow$ La cantidad de capas L
- Estos valores se conocen como hiperparámetros (los parámetros son los que se aprenden)
- Los mejores valores de los hiperparámetros se encuentran mediante prueba y error
  - $\Rightarrow$ Existen métodos para automatizar la búsqueda de hiperparámetros (pero es costoso)

- El objetivo de una red neuronal es aproximar una función $\mathsf{f}(\mathbf{x}; \boldsymbol{\Theta}) \approx \mathsf{f}(\mathbf{x})$
- Sabemos que un algoritmo lineal sirve para describir modelos lineales
  - $\Rightarrow$ Son fáciles de aprender y tienen trazabilidad matemática para dar garantías
  - $\Rightarrow$ Muchas veces los modelos lineales son suficientes $\Rightarrow$ Pero muchas veces, no
- Por lo que, en general, queremos algoritmos de aprendizaje para modelos más complejos

- Si un algoritmo lineal aprende modelos lineales, ¿qué clase de modelos aprende una red neuronal?

Teorema Universal de Aproximación (Cybenko, 1989; Hornik, 1991)

Sea $\sigma : \mathbb{R} \to \mathbb{R}$ una función continua, no-constante, acotada y monótonamente creciente. Sea $\mathsf{f} \in \mathbb{C}([0,1]^N)$ una función continua $\mathsf{f} : [0,1]^N \to \mathbb{R}$.

Dadas una función $\mathsf{f} \in \mathbb{C}([0,1]^N)$ y un $\varepsilon > 0$, entonces existen: un entero N_1 y matrices $\mathbf{W}_1 \in \mathbb{R}^{N_1 \times N}$, y vectores $\mathbf{b}_1, \mathbf{w}_2 \in \mathbb{R}^{N_1}$, tal que la red neuronal

$$\mathsf{f}(\mathbf{x}; \boldsymbol{\Theta}) = \mathbf{w}_2{}^{\mathsf{T}} \sigma\big(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_2\big)$$

con $\boldsymbol{\Theta} = \{\mathbf{W}_1, \mathbf{b}_1, \mathbf{w}_2\}$ satisface

$$\big|\mathsf{f}(\mathbf{x}) - \mathsf{f}(\mathbf{x}; \boldsymbol{\Theta})\big| \leq \varepsilon$$

para todo $\mathbf{x} \in \mathbb{R}^N$.

Implicancias y Extensiones del Teorema

Teorema Universal de Aproximación (Cybenko, 1989; Hornik, 1991)

$$\left|\mathbf{f}(\mathbf{x}) - \mathbf{w}_2{}^{\mathsf{T}}\sigma\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_2\right)\right| \leq \varepsilon$$

- Una red neuronal con una sóla capa es suficiente para aproximar $\mathbf{f}(\mathbf{x})$
  $\Rightarrow$ Siempre y cuando sea lo suficientemente ancha $\Rightarrow$ N_1 tiene que ser grande
- Podemos aprender funciones no lineales con una red neuronal de una capa
  $\Rightarrow$ No es necesario diseñar algoritmos complejos específicos para cada tarea

Implicancias y Extensiones del Teorema

Teorema Universal de Aproximación (Cybenko, 1989; Hornik, 1991)

$$\left|\mathbf{f}(\mathbf{x}) - \mathbf{w}_2{}^{\mathsf{T}}\sigma\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_2\right)\right| \leq \varepsilon$$

- El teorema es válido para funciones definidas en cualquier subconjunto compacto de $\mathbb{R}^N$
- El teorema también es válido para funciones que mapean entre espacios discretos
- Las derivadas de la red neuronal también aproximan las derivadas de la función
- La formulación original del teorema requiere que σ sea tanh o similar

  $\Rightarrow$ Se pueden probar resultados similares para $\sigma(x) = \mathsf{ReLU}(x)$ y sus extensiones

Implicancias y Extensiones del Teorema

Teorema Universal de Aproximación (Cybenko, 1989; Hornik, 1991)

$$\left|\mathbf{f}(\mathbf{x}) - \mathbf{w}_2{}^{\mathsf{T}}\sigma\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_2\right)\right| \leq \varepsilon$$

- El teorema dice que podemos aprender cualquier modelo con una red neuronal lo suficientemente ancha
  $\Rightarrow$ No nos aclara qué tan ancha tiene que ser (puede ser necesario una cant. exponencial)
- Tampoco nos garantiza que el entrenamiento va a ser suficiente para aprender el modelo
- Incluso si la red neuronal puede representar la función, el aprendizaje puede fallar
  $\Rightarrow$ El algoritmo de optimización puede no encontrar los valores correctos
  $\Rightarrow$ Si las muestras de entrenamiento no son representativas, podemos incurrir en *overfitting*
- Nada es Gratis $\Rightarrow$ No hay un procedimiento universal de aprendizaje que permita generalizar

Implicancias y Extensiones del Teorema

Teorema Universal de Aproximación (Cybenko, 1989; Hornik, 1991)

$$\left|\mathbf{f}(\mathbf{x}) - \mathbf{w}_2{}^{\mathsf{T}}\sigma\left(\mathbf{W}_1\mathbf{x} + \mathbf{b}_2\right)\right| \leq \varepsilon$$

- Si bien en teoría podemos aprender cualquier función usando una red neuronal
  - $\Rightarrow$ Puede que tenga que ser imprácticamente ancha
  - $\Rightarrow$ Puede que la optimización falle y/o la red entrenada no generalice
- Muchas veces es más conveniente usar redes neuronales profundas
  - $\Rightarrow$ Menos anchas y con mejores propiedades de generalización
  - $\Rightarrow$ Más difíciles de entrenar
- Hay un *tradeoff* entre el ancho y la profundidad de las redes neuronales

Minimización del Riesgo Empírico

Redes Neuronales

Optimización

Minimización del Riesgo Empírico

- Aprendizaje $\Rightarrow$ Encontrar el algoritmo $\mathsf{f} : \mathbb{R}^N \to \mathbb{R}^{N'}$ que minimice el riesgo $\mathsf{R}(\mathsf{f})$

$$\min_{\mathsf{f}} \mathbb{E}_{\boldsymbol{X} \sim \mathsf{f}_x} \left[\mathsf{J}(\mathsf{f}(\boldsymbol{X})) \right]$$

- No conocemos $\boldsymbol{X} \sim \mathsf{f}_x$ la distribución de los datos
  $\Rightarrow$ Tenemos acceso a una muestra $\boldsymbol{X} \rightsquigarrow \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ $\Rightarrow$ Minimización del Riesgo Empírico

$$\min_{\mathsf{f}} \frac{1}{M} \sum_{m=1}^{M} \mathsf{J}(\mathsf{f}(\mathbf{x}_m))$$

- Minimizar sobre funcionales es sumamente difícil $\Rightarrow$ Parametrización

$$\min_{\boldsymbol{\theta}} \frac{1}{M} \sum_{m=1}^{M} \mathsf{J}(\mathsf{f}(\mathbf{x}_m; \boldsymbol{\theta}))$$

- ¿Cómo resolvemos el problema de minimización?

- ¿Cómo resolvemos el problema de minimización?

- En el caso de aprender la XOR lo hicimos “a ojo”, probando con criterio
- En el caso de regresión lineal con costo cuadrático encontramos el punto crítico
  $\Rightarrow$ Tomamos la derivada de la función, la igualamos a cero, y resolvimos

- Y en un caso general, ¿cómo hacemos?

¿Se puede automatizar el problema de optimización?

Optimización de Gradiente Descendente

- Usemos lo que sabemos del análisis matemático para diseñar un método automático
- Queremos minimizar la función $\hat{\mathsf{R}}(\boldsymbol{\theta}) = \frac{1}{M}\sum_{m=1}^{M} \mathsf{J}(\mathsf{f}(\mathbf{x}_m;\boldsymbol{\theta}))$ numéricamente y automáticamente
  - $\Rightarrow$ Supongamos que lo queremos hacer de manera iterativa $\boldsymbol{\theta}_{k+1} \leftarrow \texttt{algoritmo}(\boldsymbol{\theta}_k)$
  - $\Rightarrow$ Empezamos con un punto $\boldsymbol{\theta}_0$ $\Rightarrow$ ¿A dónde nos movemos luego?
- Un poco de inspiración: En una dimensión $\hat{\mathsf{R}}(\theta+\varepsilon) \approx \hat{\mathsf{R}}(\theta) + \varepsilon\hat{\mathsf{R}}'(\theta)$ para $\varepsilon \approx 0$, $\varepsilon > 0$
  - $\Rightarrow$ Si yo muevo el valor de θ a $\theta+\varepsilon$, la función se va a mover en $\varepsilon\hat{\mathsf{R}}'(\theta)$
  - $\Rightarrow$ Es evidente que si $\hat{\mathsf{R}}'(\theta) < 0$ la función va a ser más chica en $\theta+\varepsilon$
  - $\Rightarrow$ Pero si $\hat{\mathsf{R}}'(\theta) > 0$ la función va a ser más grande en $\theta+\varepsilon$
- ¿Cómo podemos elegir ε, entonces? Si $\varepsilon = -\eta\hat{\mathsf{R}}'(\theta)$ obtenemos que $\hat{\mathsf{R}}(\theta+\varepsilon) \approx \hat{\mathsf{R}}(\theta) - \eta(\hat{\mathsf{R}}'(\theta))^2$
  - $\Rightarrow$ Es decir, la función siempre es más chica (η es tal que $\eta\hat{\mathsf{R}}'(\theta) = \varepsilon \ll 0$)

$$\theta_{k+1} \leftarrow \theta_k - \eta\hat{\mathsf{R}}'(\theta_k)$$

- La derivada contiene información de los lugares donde la función (de)crece
- Para $\hat{\mathsf{R}} : \mathbb{R}^M \to \mathbb{R}$ una función de múltiples dimensiones $\Rightarrow$ Gradiente $\nabla_{\boldsymbol{\theta}}$

$$\nabla_{\boldsymbol{\theta}} \hat{\mathsf{R}} = \left[\frac{\partial\, \hat{\mathsf{R}}}{\partial [\boldsymbol{\theta}]_1} \quad \frac{\partial\, \hat{\mathsf{R}}}{\partial [\boldsymbol{\theta}]_2} \quad \cdots \quad \frac{\partial\, \hat{\mathsf{R}}}{\partial [\boldsymbol{\theta}]_M} \right] \in \mathbb{R}^M$$

- Derivada direccional $\Rightarrow$ Pendiente de la función $\hat{\mathsf{R}}$ en la dirección $\check{\mathbf{u}}$

$$\frac{\partial}{\partial \alpha} \{ \hat{\mathsf{R}}(\boldsymbol{\theta} + \alpha \check{\mathbf{u}}) \} \Big|_{\alpha=0} = \check{\mathbf{u}}^{\mathsf{T}} \nabla_{\boldsymbol{\theta}} \hat{\mathsf{R}}$$

  $\Rightarrow$ La contribución a la pendiente en la dirección $\check{\mathbf{u}}$ es $\langle \check{\mathbf{u}}, \nabla_{\boldsymbol{\theta}} \hat{\mathsf{R}} \rangle$
- ¿Cuál es la dirección donde la función decrece lo más posible?

  $\Rightarrow \langle \check{\mathbf{u}}, \nabla_{\boldsymbol{\theta}} \hat{\mathsf{R}} \rangle$ es un producto interno $\Rightarrow$ El mínimo es cuando $\check{\mathbf{u}}$ va en la dirección de $-\nabla_{\boldsymbol{\theta}} \hat{\mathsf{R}}$

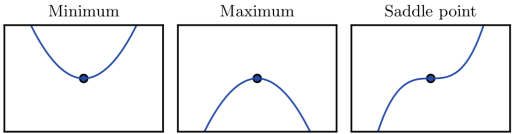
Gradiente Descendente

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta \nabla_{\boldsymbol{\theta}} \hat{\mathsf{R}}(\boldsymbol{\theta}_k)$$

Gradiente Descendente

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta \nabla_{\boldsymbol{\theta}} \hat{\mathsf{R}}(\boldsymbol{\theta}_k)$$

- El valor de η se conoce como tasa de aprendizaje (*learning rate*) $\Rightarrow$ Determina el tamaño del salto
  $\Rightarrow$ Es un parámetro fundamental para que la optimización de gradiente descendente sea exitosa
- ¿Qué pasa cuando $\nabla_{\boldsymbol{\theta}} \hat{\mathsf{R}}(\boldsymbol{\theta}_k) = 0$? El algoritmo no cambia $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k$
  $\Rightarrow$ Los puntos donde $\nabla_{\boldsymbol{\theta}} \hat{\mathsf{R}}(\boldsymbol{\theta}) = 0$ se conocen como puntos críticos
  $\Rightarrow$ Pueden ser mínimos, máximos o puntos de ensilladura



Gradiente Descendente para un Perceptrón Multicapa

- Es fundamental calcular el gradiente del riesgo, aunque sea numéricamente

$$\hat{\mathsf{R}}(\boldsymbol{\theta}) = \frac{1}{M} \sum_{m=1}^{M} \mathsf{J}(\mathsf{f}(\mathbf{x}_m; \boldsymbol{\theta}))$$

- Para nosotros, $\mathsf{f}(\mathbf{x}_m; \boldsymbol{\theta})$ tiene una forma muy particular $\Rightarrow$ Perceptrón multicapa
  $\Rightarrow$ ¿Cómo queda la derivada cuando $\mathsf{f}(\mathbf{x}_m; \boldsymbol{\theta}) = \mathbf{x}_L$ para $\mathbf{x}_\ell = \sigma(\mathbf{W}_\ell \mathbf{x}_{\ell-1} + \mathbf{b}_\ell)$?
- En el caso del perceptrón multicapa, $\boldsymbol{\Theta} = \{(\mathbf{W}_1, \mathbf{b}_1), \ldots, (\mathbf{W}_L, \mathbf{b}_L)\}$

- ¿Cómo calculamos la derivada? $\Rightarrow$ Usando la regla de la cadena

Gradiente Descendente para un Perceptrón

- Empecemos fácil $\Rightarrow$ Una sóla capa $\Rightarrow$ Calculemos el gradiente para un perceptrón

$$\mathsf{f}(\mathbf{x}_m; \mathbf{W}_1, \mathbf{b}_1) = \sigma(\mathbf{W}_1 \mathbf{x}_m + \mathbf{b}_1)$$

- Recordemos la regla de la cadena: $\big(\mathsf{f}(\mathsf{g}(x))\big)' = \mathsf{f}'(y)\mathsf{g}'(x)$ con $y = \mathsf{g}(x)$

- Lo función $\hat{\mathsf{R}}(\mathbf{W}_1, \mathbf{b}_1)$ es $\hat{\mathsf{R}} : \mathbb{R}^{N_1 \times N} \times \mathbb{R}^{N_1} \to \mathbb{R}$, lo que implica que las derivadas que queremos

$$\frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{W}_1} \in \mathbb{R}^{N_1 \times N} : \left[\frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{W}_1}\right]_{ij} = \frac{\partial\, \hat{\mathsf{R}}}{\partial [\mathbf{W}_1]_{ij}} \quad \text{y} \quad \frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{b}_1} \in \mathbb{R}^{N_1} : \left[\frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{b}_1}\right]_{i} = \frac{\partial\, \hat{\mathsf{R}}}{\partial [\mathbf{b}_1]_{i}}$$

Gradiente Descendente para un Perceptrón

- Empecemos a calcular $\partial \hat{\mathsf{R}}/\partial \mathbf{W}_1$

$$\frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{W}_1} = \frac{\partial}{\partial \mathbf{W}_1} \left\{ \frac{1}{M} \sum_{m=1}^{M} \mathsf{J}\big(\mathsf{f}(\mathbf{x}_m; \mathbf{W}_1, \mathbf{b}_1)\big) \right\} = \frac{1}{M} \sum_{m=1}^{M} \frac{\partial}{\partial \mathbf{W}_1} \left\{ \mathsf{J}\big(\mathsf{f}(\mathbf{x}_m; \mathbf{W}_1, \mathbf{b}_1)\big) \right\}$$

- El paso siguiente es calcular $\partial \mathsf{J}/\partial \mathbf{W}_1 \Rightarrow$ Para eso definimos $\mathbf{x}_{1m} = \mathsf{f}(\mathbf{x}_m; \mathbf{W}_1, \mathbf{b}_1) \in \mathbb{R}^{N_1}$

$$\frac{\partial\, \mathsf{J}}{\partial \mathbf{W}_1} = \frac{\partial}{\partial \mathbf{W}_1} \left\{ \mathsf{J}\big(\mathsf{f}(\mathbf{x}_m; \mathbf{W}_1, \mathbf{b}_1)\big) \right\} = \frac{\partial\, \mathsf{J}}{\partial \mathbf{x}_{1m}} \frac{\partial}{\partial \mathbf{W}_1} \left\{ \mathsf{f}(\mathbf{x}_m; \mathbf{W}_1, \mathbf{b}_1) \right\}$$

$\Rightarrow$ La función $\mathsf{J}(\mathbf{y})$ como función de su argumento va de $\mathbb{R}^{N_1} \to \mathbb{R}$, con lo que $\partial \mathsf{J}/\partial \mathbf{x}_{1m} \in \mathbb{R}^{N_1}$ implica

$$\frac{\partial\, \mathsf{J}}{\partial \mathbf{x}_{1m}} = \frac{\partial\, \mathsf{J}}{\partial \mathbf{y}}\bigg|_{\mathbf{y}=\mathbf{x}_{1m}=\mathsf{f}(\mathbf{x}_m;\mathbf{W}_1,\mathbf{b}_1)} \Rightarrow \left[\frac{\partial\, \mathsf{J}}{\partial \mathbf{x}_{1m}}\right]_i = \frac{\partial\, \mathsf{J}}{\partial [\mathbf{y}]_i}\bigg|_{\mathbf{y}=\mathbf{x}_{1m}=\mathsf{f}(\mathbf{x}_m;\mathbf{W}_1,\mathbf{b}_1)}$$

$\Rightarrow$ Para simplificar la notación, denotamos $\mathbf{g}_{\mathsf{J}'(\mathbf{x}_{1m})} = \partial \mathsf{J}/\partial \mathbf{x}_{1m}$ como un vector en $\mathbb{R}^{N_1}$

Gradiente Descendente para un Perceptrón

- Ahora viene la parte de $\partial \mathsf{f}/\partial \mathbf{W}_1$ donde tenemos que usar que $\mathsf{f}(\mathbf{x}_m; \mathbf{W}_1, \mathbf{b}_1) = \sigma(\mathbf{W}_1 \mathbf{x}_m + \mathbf{b}_1)$

$$\frac{\partial\, \mathsf{f}}{\partial \mathbf{W}_1} = \frac{\partial}{\partial \mathbf{W}_1} \{\sigma(\mathbf{W}_1 \mathbf{x}_m + \mathbf{b}_1)\} = \frac{\partial \sigma}{\partial \mathbf{z}_{1m}} \frac{\partial}{\partial \mathbf{W}_1} \{\mathbf{W}_1 \mathbf{x}_m + \mathbf{b}_1\} \text{ donde } \mathbf{z}_{1m} = \mathbf{W}_1 \mathbf{x}_m + \mathbf{b}_1$$

$\Rightarrow$ La función $\sigma(\mathbf{z})$ como función de su argumento va de $\mathbb{R}^{N_1} \to \mathbb{R}^{N_1}$, con lo que $\partial \sigma / \partial \mathbf{z}_{1m} \in \mathbb{R}^{N_1 \times N_1}$

$$\frac{\partial\, \sigma}{\partial \mathbf{z}_{1m}} = \frac{\partial\, \sigma}{\partial \mathbf{y}}\bigg|_{\mathbf{y}=\mathbf{z}_{1m}=\mathbf{W}_1\mathbf{x}_m+\mathbf{b}_1} \Rightarrow \left[\frac{\partial\, \sigma}{\partial \mathbf{z}_{1m}}\right]_{ij} = \frac{\partial\, \sigma_i}{\partial [\mathbf{z}_{1m}]_j} = \sigma'([\mathbf{z}_{1m}]_j)$$

$\Rightarrow$ Pero σ es una función puntual, con lo que cada fila de la matriz $\partial \sigma / \partial \mathbf{z}_{1m}$ es igual

$\Rightarrow$ Para simplificar la notación, denotamos $\mathbf{G}_{\sigma'(\mathbf{z}_{1m})} = \partial \sigma / \partial \mathbf{z}_{1m}$ una matriz en $\mathbb{R}^{N_1 \times N_1}$

- El último paso: $\partial\{\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1\}/\partial\mathbf{W}_1$ $\Rightarrow$ Como función de $\mathbf{W}_1$, va de $\mathbb{R}^{N_1 \times N} \rightarrow \mathbb{R}^{N_1}$
  $\Rightarrow$ La derivada es un tensor de tamaño $N_1 \times N_1 \times N$ (pero es una función lineal)

$$\left[\frac{\partial}{\partial\mathbf{W}_1}\{\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1\}\right]_{ijk} = \frac{\partial[\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1]_i}{\partial[\mathbf{W}]_{jk}}$$

  $\Rightarrow$ La derivada del elemento i de $\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1$ respecto del elemento (j,k) de la matriz $\mathbf{W}_1$
- Calculamos el elemento i de $\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1$ como

$$\left[\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1\right]_i = \sum_{k'=1}^{N} [\mathbf{W}_1]_{ik'}[\mathbf{x}_m]_{k'} + [\mathbf{b}]_i$$

  $\Rightarrow$ Primera observación: $\partial[\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1]_i/\partial[\mathbf{W}]_{jk} = 0$ si $i \neq j$ $\Rightarrow$ La fila j de $\mathbf{W}_1$ no participa

$$\frac{\partial[\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1]_i}{\partial[\mathbf{W}]_{ik}} = \frac{\partial}{\partial[\mathbf{W}]_{ik}}\left\{\sum_{k'=1}^{N} [\mathbf{W}_1]_{ik'}[\mathbf{x}_m]_{k'} + [\mathbf{b}]_i\right\} = \sum_{k'=1}^{N} \frac{\partial[\mathbf{W}_1]_{ik'}[\mathbf{x}_m]_{k'}}{\partial[\mathbf{W}]_{ik}}$$

  $\Rightarrow$ Esto es no-nulo únicamente cuando $k = k'$ con lo cual la derivada es $[\mathbf{x}_m]_k$

Gradiente Descendente para un Perceptrón

- En resumen, tenemos que el tensor de derivadas $\partial\{\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1\}/\partial\mathbf{W}_1 \in \mathbb{R}^{N_1 \times N_1 \times N}$ satisface

$$\left[\frac{\partial}{\partial \mathbf{W}_1}\{\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1\}\right]_{ijk} = \begin{cases} [\mathbf{x}_m]_k & \text{si } i = j \\ 0 & \text{en caso contrario} \end{cases}$$

- Si pensamos los tensores como un conjunto de N_1 matrices de tamaño $N_1 \times N$ tenemos algo así

$$\frac{\partial}{\partial \mathbf{W}_1}\{\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1\} = \left\{ \begin{bmatrix} [\mathbf{x}_m]_1 & [\mathbf{x}_m]_2 & \cdots & [\mathbf{x}_m]_N \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}, \cdots, \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ [\mathbf{x}_m]_1 & [\mathbf{x}_m]_2 & \cdots & [\mathbf{x}_m]_N \end{bmatrix}, \right\}$$

$\Rightarrow$ Se repite el dato en las filas $\Rightarrow$ Se puede escribir de manera compacta como

$$\frac{\partial}{\partial \mathbf{W}_1}\{\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1\} = \left\{ \begin{bmatrix} \mathbf{x}_m^\mathsf{T} \\ \mathbf{0}^\mathsf{T} \\ \vdots \\ \mathbf{0}^\mathsf{T} \end{bmatrix}, \begin{bmatrix} \mathbf{0}^\mathsf{T} \\ \mathbf{x}_m^\mathsf{T} \\ \vdots \\ \mathbf{0}^\mathsf{T} \end{bmatrix}, \cdots \begin{bmatrix} \mathbf{0}^\mathsf{T} \\ \mathbf{0}^\mathsf{T} \\ \vdots \\ \mathbf{x}_m^\mathsf{T} \end{bmatrix} \right\}$$

- Ahora juntamos todo para obtener la derivada

$$\frac{\partial\,\hat{\mathsf{R}}}{\partial \mathbf{W}_1} = \frac{1}{M}\sum_{m=1}^{M} \underbrace{\mathbf{g}^{\mathsf{T}}_{J'(\mathbf{x}_{1m})}}_{1\times N_1} \underbrace{\mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{1m})}}_{N_1\times N_1} \underbrace{\frac{\partial}{\partial \mathbf{W}_1}\{\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1\}}_{N_1\times N_1\times N}$$

- Recordamos multiplicación con tensores: pensarlo como un conjunto de N_1 matrices de tamaño $N_1 \times N$
  - $\Rightarrow$ El resultado es N_1 multiplicaciones de $(1 \times N_1)(N_1 \times N_1)(N_1 \times N) = 1 \times N$
  - $\Rightarrow$ Da por resultado N_1 vectores de tamaño N $\Rightarrow$ Matriz $N_1 \times N$ $\Rightarrow$ Lo que estábamos buscando
- Para obtener $\partial\hat{\mathsf{R}}/\partial\mathbf{b}_1$ sólo el último término cambia (función $\mathbb{R}^{N_1} \to \mathbb{R}^{N_1}$)

$$\left[\frac{\partial}{\partial \mathbf{b}_1}\{\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1\}\right]_{ij} = \frac{\partial}{\partial[\mathbf{b}_j]}\{[\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1]_i\} = \frac{\partial}{\partial[\mathbf{b}_j]}\left\{\sum_{k=1}^{N}[\mathbf{W}_1]_{ik}[\mathbf{x}_m]_k + [\mathbf{b}]_i\right\}$$

  - $\Rightarrow$ Esto es igual a 1 si $i = j$ y 0 en otro caso $\Rightarrow$ $\partial\{\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1\}/\partial\mathbf{b}_1 = \mathbf{I}_{N_1\times N_1}$

- Hasta ahora, para el caso del perceptrón, tenemos

$$\frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{W}_1} = \frac{1}{M}\sum_{m=1}^{M} \underbrace{\mathbf{g}^{\mathsf{T}}_{J'(\mathbf{x}_{1m})}}_{\text{costo}} \underbrace{\mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{1m})}}_{\text{activación}} \underbrace{\frac{\partial}{\partial \mathbf{W}_1}\{\mathbf{W}_1\mathbf{x}_m + \mathbf{b}_1\}}_{\text{operación lineal}}$$

- Supongamos ahora que tenemos un perceptrón multicapa $\mathbf{x}_L$ tal que $\mathbf{x}_\ell = \sigma_\ell(\mathbf{W}_\ell \mathbf{x}_{\ell-1}\mathbf{b}_\ell)$
  $\Rightarrow$ Definimos $\mathbf{x}_{\ell m} = \sigma(\mathbf{z}_{\ell m})$ y $\mathbf{z}_{\ell m} = \mathbf{W}_\ell \mathbf{x}_{(\ell-1)m}\mathbf{b}_\ell$
- Ahora podemos calcular la derivada con respecto a los parámetros de la capa ℓ

$$\frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{W}_\ell} = \frac{1}{M}\sum_{m=1}^{M} \mathbf{g}^{\mathsf{T}}_{J'(\mathbf{x}_{Lm})}\, \mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{Lm})} \frac{\partial}{\partial \mathbf{W}_\ell}\{\mathbf{W}_L\mathbf{x}_{(L-1)m} + \mathbf{b}_L\}$$

- Lo que nos falta ahora es $\partial\{\mathbf{W}_L\mathbf{x}_{(L-1)m} + \mathbf{b}_L\}/\partial \mathbf{W}_\ell$ $\Rightarrow$ Recordar que $\mathbf{x}_{(L-1)m}$ es función de $\mathbf{W}_\ell$

$$\frac{\partial}{\partial \mathbf{W}_\ell}\{\mathbf{W}_L\mathbf{x}_{(L-1)m} + \mathbf{b}_L\} = \mathbf{W}_L \frac{\partial\, \mathbf{x}_{(L-1)m}}{\partial \mathbf{W}_\ell}$$

- Pero ya conocemos la derivada $\partial \mathbf{x}_{(L-1)m}/\partial \mathbf{W}_\ell$ porque es la derivada del perceptrón

$$\begin{aligned}\frac{\partial\, \mathbf{x}_{(L-1)m}}{\partial \mathbf{W}_\ell} &= \frac{\partial}{\partial \mathbf{W}_\ell}\left\{\sigma(\mathbf{z}_{(L-1)m})\right\} = \mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{(L-1)m})}\frac{\partial}{\partial \mathbf{W}_\ell}\left\{\mathbf{W}_{L-1}\mathbf{x}_{(L-2)m} + \mathbf{b}_{L-1}\right\} \\ &= \mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{(L-1)m})}\mathbf{W}_{L-1}\frac{\partial\, \mathbf{x}_{(L-2)m}}{\partial \mathbf{W}_\ell}\end{aligned}$$

  $\Rightarrow$ Y se repite recursivamente hasta llegar a la capa ℓ

- Juntando todo, ahora tenemos que

$$\frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{W}_\ell} = \frac{1}{M}\sum_{m=1}^{M} \underbrace{\mathbf{g}^{\mathsf{T}}_{J'(\mathbf{x}_{Lm})}}_{\text{costo}} \underbrace{\mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{Lm})}\mathbf{W}_L}_{\text{capa } L} \underbrace{\mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{(L-1)m})}\mathbf{W}_{L-1}}_{\text{capa } L-1} \cdots \mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{\ell m})}\frac{\partial}{\partial \mathbf{W}_\ell}\left\{\mathbf{W}_\ell \mathbf{x}_{(\ell-1)m} + \mathbf{b}_\ell\right\}$$

  $\Rightarrow$ Acá tenemos que $\partial\{\mathbf{W}_\ell \mathbf{x}_{(\ell-1)m} + \mathbf{b}_\ell\}/\partial \mathbf{W}_\ell$ el tensor $N_\ell \times N_\ell \times N_{\ell-1}$

  $\Rightarrow$ Si queremos $\partial\{\mathbf{W}_\ell \mathbf{x}_{(\ell-1)m} + \mathbf{b}_\ell\}/\partial \mathbf{W}_\ell$ simplemente reemplazamos el último término por $\mathbf{I}_{N_\ell \times N_\ell}$

Implementación Numérica: Backpropagation

- El algoritmo de *backpropagation* refiere a un procedimiento para calcular la derivada [Rumelhart, 1986]
- Recordemos que estamos haciendo gradiente descendente para actualizar $\mathbf{W}_\ell$ y $\mathbf{b}_\ell$ para todo ℓ

$$\mathbf{W}_{\ell(k+1)} = \mathbf{W}_{\ell k} - \eta \frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{W}_{\ell k}} \quad , \quad \mathbf{b}_{\ell(k+1)} = \mathbf{b}_{\ell k} - \eta \frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{b}_{\ell k}}$$

- En la iteración k conocemos $\mathbf{W}_{\ell k}$ y $\mathbf{b}_{\ell k}$ para todo ℓ; y además tenemos $\mathbf{x}_m$

$$\mathbf{x}_m \to \big(\mathbf{z}_{1m}, \mathbf{x}_{1m}\big) \to \big(\mathbf{z}_{2m}, \mathbf{x}_{2m}\big) \to \cdots \to \big(\mathbf{z}_{Lm}, \mathbf{x}_{Lm}\big) \quad \text{con} \quad \mathbf{z}_{\ell m} = \mathbf{W}_{\ell k}\mathbf{x}_{(\ell-1)m} + \mathbf{b}_{\ell k}\ ,\ \mathbf{x}_{\ell m} = \sigma(\mathbf{z}_{\ell m})$$

  $\Rightarrow$ Y de esta manera hacemos una pasada 'hacia adelante' para calcular la salida
- Pero para calcular el gradiente necesitamos

$$\frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{W}_{\ell k}} = \frac{1}{M} \sum_{m=1}^{M} \underbrace{\mathbf{g}^{\mathsf{T}}_{\mathsf{J}'(\mathbf{x}_{Lm})}}_{\text{costo}} \underbrace{\mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{Lm})} \mathbf{W}_{Lk}}_{\text{capa } L} \underbrace{\mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{(L-1)m})} \mathbf{W}_{(L-1)k}}_{\text{capa } L-1} \cdots \mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{\ell m})} \frac{\partial}{\partial \mathbf{W}_{\ell k}} \big\{ \mathbf{W}_{\ell k}\mathbf{x}_{(\ell-1)m} + \mathbf{b}_{\ell k} \big\}$$

- Pero para calcular el gradiente necesitamos

$$\frac{\partial\,\hat{\mathsf{R}}}{\partial \mathbf{W}_{\ell k}} = \frac{1}{M}\sum_{m=1}^{M} \underbrace{\mathbf{g}_{\mathsf{J}'(\mathbf{x}_{Lm})}^{\mathsf{T}}}_{\text{costo}} \underbrace{\mathbf{G}_{\sigma'(\mathbf{z}_{Lm})}^{\mathsf{T}} \mathbf{W}_{Lk}}_{\text{capa } L} \underbrace{\mathbf{G}_{\sigma'(\mathbf{z}_{(L-1)m})}^{\mathsf{T}} \mathbf{W}_{(L-1)k}}_{\text{capa } L-1} \cdots \mathbf{G}_{\sigma'(\mathbf{z}_{\ell m})}^{\mathsf{T}} \frac{\partial}{\partial \mathbf{W}_{\ell k}} \{\mathbf{W}_{\ell k}\mathbf{x}_{(\ell-1)m} + \mathbf{b}_{\ell k}\}$$

- Se observa que cada capa necesita el valor de la capa anterior

  $\Rightarrow$ En la pasada hacia delante, también podemos calcular lo que necesitamos para el gradiente

$$\mathbf{x}_m \rightarrow \left(\mathbf{z}_{1m}, \mathbf{x}_{1m}, \mathbf{G}_{\sigma'(\mathbf{z}_{1m})}, \frac{\partial}{\partial \mathbf{W}_{1k}}\{\mathbf{W}_{1k}\mathbf{x}_m + \mathbf{b}_{1k}\}\right) \rightarrow \left(\mathbf{z}_{2m}, \mathbf{x}_{2m}, \mathbf{G}_{\sigma'(\mathbf{z}_{2m})}, \frac{\partial}{\partial \mathbf{W}_{2k}}\{\mathbf{W}_{2k}\mathbf{x}_{1m} + \mathbf{b}_{2k}\}\right)$$
$$\rightarrow \cdots \rightarrow \left(\mathbf{z}_{Lm}, \mathbf{x}_{Lm}, \mathbf{G}_{\sigma'(\mathbf{z}_{Lm})}, \frac{\partial}{\partial \mathbf{W}_{Lk}}\{\mathbf{W}_{Lk}\mathbf{x}_{(L-1)m} + \mathbf{b}_{Lk}\}\right) \rightarrow \left(\mathbf{J}(\mathbf{x}_{Lm}), \mathbf{g}_{\mathsf{J}'(\mathbf{x}_{Lm})}\right)$$

Implementación Numérica: Backpropagation

- Y ahora que calculamos todo lo que necesitamos en la pasada hacia adelante

$$\mathbf{x}_m \rightarrow \left(\mathbf{z}_{1m}, \mathbf{x}_{1m}, \mathbf{G}_{\sigma'(\mathbf{z}_{1m})}, \frac{\partial}{\partial \mathbf{W}_{1k}}\{\mathbf{W}_{1k}\mathbf{x}_m + \mathbf{b}_{1k}\}\right) \rightarrow \left(\mathbf{z}_{2m}, \mathbf{x}_{2m}, \mathbf{G}_{\sigma'(\mathbf{z}_{2m})}, \frac{\partial}{\partial \mathbf{W}_{2k}}\{\mathbf{W}_{2k}\mathbf{x}_{1m} + \mathbf{b}_{2k}\}\right)$$
$$\rightarrow \cdots \rightarrow \left(\mathbf{z}_{Lm}, \mathbf{x}_{Lm}, \mathbf{G}_{\sigma'(\mathbf{z}_{Lm})}, \frac{\partial}{\partial \mathbf{W}_{Lk}}\{\mathbf{W}_{Lk}\mathbf{x}_{(L-1)m} + \mathbf{b}_{Lk}\}\right) \rightarrow \left(\mathbf{J}(\mathbf{x}_{Lm}), \mathbf{g}_{J'(\mathbf{x}_{Lm})}\right)$$

$\Rightarrow$ Podemos volver hacia atrás, calculando las derivadas, de la última a la primera

$$\frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{W}_{Lk}} = \frac{1}{M}\sum_{m=1}^{M} \mathbf{g}^{\mathsf{T}}_{J'(\mathbf{x}_{Lm})}\mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{Lm})}\frac{\partial}{\partial \mathbf{W}_{Lk}}\{\mathbf{W}_{Lk}\mathbf{x}_{(L-1)m} + \mathbf{b}_{Lk}\}$$

$$\frac{\partial\, \hat{\mathsf{R}}}{\partial \mathbf{W}_{(L-1)k}} = \frac{1}{M}\sum_{m=1}^{M} \mathbf{g}^{\mathsf{T}}_{J'(\mathbf{x}_{Lm})}\mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{Lm})}\mathbf{W}_{Lk}\mathbf{G}^{\mathsf{T}}_{\sigma'(\mathbf{z}_{(L-1)m})}\frac{\partial}{\partial \mathbf{W}_{(L-1)k}}\{\mathbf{W}_{(L-1)k}\mathbf{x}_{(L-2)m} + \mathbf{b}_{(L-1)k}\}$$

$$\vdots$$

Implementación Numérica: Backpropagation

- El algoritmo de backpropagation sirve para calcular el gradiente y consiste en dos fases
  - ⇒ 'Hacia adelante': calcula la salida y guarda valores relevantes para el gradiente
  - ⇒ 'Hacia atrás': multiplica esos valores para calcular el gradiente en cada capa

Algoritmo de Backpropagation

Dados $\{\mathbf{W}_{\ell k}, \mathbf{b}_{\ell k}\}$ para todo ℓ; dado $\mathbf{x}_m$
Para cada $\ell = 1, \ldots, L$
- Calcular $\mathbf{z}_{\ell m} = \mathbf{W}_{\ell k}\mathbf{x}_{(\ell-1)m} + \mathbf{b}_{\ell k}$
- Calcular $\mathbf{x}_{\ell m} = \sigma(\mathbf{z}_{\ell m})$
- Calcular $\mathbf{G}_{\sigma'(\mathbf{z}_{\ell m})}$
- Calcular $\partial \mathbf{z}_{\ell m} / \partial \mathbf{W}_{\ell k}$
- Calcular $\partial \mathbf{z}_{\ell m} / \partial \mathbf{b}_{\ell k}$

Para cada $\ell = L, \ldots, 1$
- Calcular $\partial \hat{\mathrm{R}} / \partial \mathbf{W}_{\ell k}$
- Calcular $\partial \hat{\mathrm{R}} / \partial \mathbf{b}_{\ell k}$

Gradiente Descendente Estocástico (SGD)

- El algoritmo de gradiente descendente es sencillo $\Rightarrow \boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta \, \partial \hat{\mathsf{R}} / \partial \boldsymbol{\theta}_k$
- La clave está en calcular el gradiente del riesgo empírico $\hat{\mathsf{R}}(\boldsymbol{\theta})$

$$\frac{\partial \, \hat{\mathsf{R}}}{\partial \boldsymbol{\theta}} = \frac{\partial}{\partial \boldsymbol{\theta}} \left\{ \frac{1}{M} \sum_{m=1}^{M} \mathsf{J}(\mathsf{f}(\mathbf{x}_m; \boldsymbol{\theta})) \right\} = \frac{1}{M} \sum_{m=1}^{M} \frac{\partial}{\partial \boldsymbol{\theta}} \Big\{ \mathsf{J}\big(\mathsf{f}(\mathbf{x}_m; \boldsymbol{\theta})\big) \Big\}$$

  $\Rightarrow$ Para $\mathsf{f}(\mathbf{x}; \boldsymbol{\theta})$ un perceptrón multicapa $\Rightarrow$ Usamos backpropagation para calcular el gradiente

- Ahora bien, el uso de backpropagation parece razonable, pero abre dos interrogantes
  $\Rightarrow$ Teórico $\Rightarrow$ ¿Hacer gradiente descendente sobre el riesgo empírico $\hat{\mathsf{R}}$ sirve para el riesgo R?
  $\Rightarrow$ Práctico $\Rightarrow$ Cada pasada del backpropagation se realiza para una sola muestra

Gradiente Descendente Estocástico (SGD)

- ¿Hacer gradiente descendente sobre el riesgo empírico sirve para minimizar el riesgo? **
  - $\Rightarrow$ Recordemos que el riesgo es $\mathsf{R}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{X}\sim\mathsf{f}_x}[\mathsf{J}(\mathsf{f}(\boldsymbol{X};\boldsymbol{\theta}))]$ $\Rightarrow$ Su derivada es $\partial\mathsf{R}/\partial\boldsymbol{\theta} = \mathbb{E}_{\boldsymbol{X}\sim\mathsf{f}_x}[\partial\mathsf{J}/\partial\boldsymbol{\theta}]$
  - $\Rightarrow$ No tenemos acceso a $\mathbb{E}_{\boldsymbol{X}\sim\mathsf{f}_x}[\partial\mathsf{J}(\mathsf{f}(\boldsymbol{X};\boldsymbol{\theta}))/\partial\boldsymbol{\theta}]$, sólo tenemos acceso a $\partial\mathsf{J}(\mathsf{f}(\mathbf{x}_m;\boldsymbol{\theta}))/\partial\boldsymbol{\theta}$
  - $\Rightarrow$ Ley de los grandes números $\Rightarrow$ $M^{-1}\sum_{m=1}^{M}\partial\mathsf{J}(\mathsf{f}(\mathbf{x}_m;\boldsymbol{\theta}))/\partial\boldsymbol{\theta} \approx \mathbb{E}_{\boldsymbol{X}\sim\mathsf{f}_x}[\partial\mathsf{J}(\mathsf{f}(\boldsymbol{X};\boldsymbol{\theta}))/\partial\boldsymbol{\theta}]$, $M \gg 1$
  - $\Rightarrow$ Gradiente Descendente Estocástico (SGD: *stochastic gradient descent*) (muestras, no esperanza)

- Cada pasada del backpropagation se realiza para una sola muestra
  - $\Rightarrow$ Necesito muchas muestras para estimar bien la esperanza
  - $\Rightarrow$ Pero cada muestra demanda una pasada "hacia adelante y hacia atrás" del algoritmo
  - $\Rightarrow$ Cuantas más muestras, más tardo en actualizar los parámetros

- Tenemos una muestra de tamaño M: $\{\mathbf{x}_1, \ldots, \mathbf{x}_M\}$ $\Rightarrow$ M debería ser grande
- Tomamos un subconjunto de muestras $M' \ll M$ de manera aleatoria $\{\mathbf{x}_{m'}\}_{m'=1}^{M'}$
- Hacemos una pasada del backpropagation, sólo sobre las M' muestras $\Rightarrow$ Más rápido
- El estimador del gradiente ahora es $M'^{-1} \sum_{m'=1}^{M'} \partial \mathsf{J}(\mathsf{f}(\mathbf{x}_{m'}; \boldsymbol{\theta}))/\partial \boldsymbol{\theta}$ $\Rightarrow$ Un estimador un poco peor
- Actualizamos: $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta \, \partial \hat{\mathsf{R}}/\partial \boldsymbol{\theta}_k$ $\Rightarrow$ Pero usando $\partial \hat{\mathsf{R}}/\partial \boldsymbol{\theta}_k$ estimado sobre las M' muestras
- Tomamos un nuevo subconjunto de M' muestras $\Rightarrow$ Repetimos el proceso actualizando los parámetros

- Cada subconjunto de M' muestras se conoce como *batch* (*minibatch*)
- Si queremos seguir actualizando los parámetros, podemos volver a pasar por la muestra (en otro orden)
  $\Rightarrow$ El estimador del gradiente deja de consistir en muestras i.i.d. $\Rightarrow$ Sesgo
- Cada pasada por el conjunto completo de muestras se conoce como *epoch*

- Implementación práctica del uso de batches y epochs
  - $\Rightarrow$ Para cada epoch, reordenamos los índices $\{1, ..., M\}$ de manera aleatoria
  - $\Rightarrow$ Elegimos M' índices de manera consecutiva del conjunto reordenado $\Rightarrow$ Un batch
  - $\Rightarrow$ Una vez que llegamos a la última muestras, termina el epoch
  - $\Rightarrow$ Realizamos un nuevo reordenamiento aleatorio de los índices y repetimos $\Rightarrow$ Nueva epoch

- Valores de M' más grandes mejoran el estimador, pero no de manera lineal
- Procesamiento en paralelo $\Rightarrow$ Cada muestra va a un procesador $\Rightarrow$ Memoria escala con M'
- Si los batches son muy pequeños pueden quedar procesadores sin utilizar
- Es común utilizar una cantidad de batches que sean potencias de 2 (especialmente en GPUs)
- La aleatoriedad del SGD parece tener un efecto regularizador que ayuda a la generalización

- Cuando usamos SGD estamos estimando el gradiente $\Rightarrow$ La estimación no es perfecta
  - $\Rightarrow$ El algoritmo va a seguir saltando alrededor del mínimo por cambios en cada estimación
  - $\Rightarrow$ Es necesario reducir la tasa de aprendizaje a medida que nos acercamos al mínimo

Convergencia de SGD

Cuando usamos SGD, cambiamos la tasa de aprendizaje a cada iteración:

$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta_k \, \partial \hat{\mathrm{R}} / \partial \theta_k$$

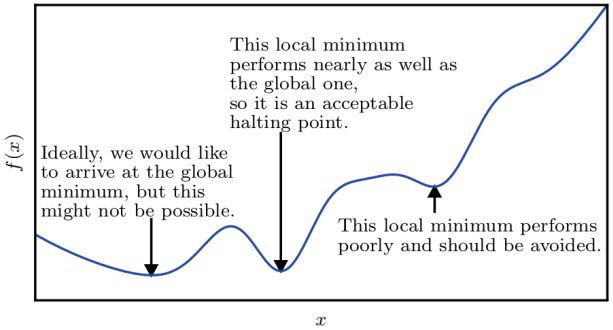
Si la tasa de aprendizaje satisface que

$$\sum_{k=1}^{\infty} \eta_k = \infty \quad \text{y} \quad \sum_{k=1}^{\infty} \eta_k^2 < \infty$$

entonces el algoritmo de SGD converge a un mínimo.**

  - $\Rightarrow$ Típicamente $\eta_k = (1 - k/\tau)\eta_0 + k e_\tau / \tau$ para algún horizonte τ

- En general, los algoritmos de optimización no ofrecen garantías de optimalidad
  - ⇒ A menos que se trate de un problema de optimización convexa
  - ⇒ En ese caso, siempre se garantiza que se llega al punto óptimo
- En casos no-convexos, como lo es la mayoría de los problemas de deep learning, hay que tener cuidado



Mejorando SGD: Momentum

- Los problemas con mínimos locales parecen ser menos relevantes en la práctica
- Los problemas con puntos de ensilladura son un poco más críticos $\Rightarrow$ Diseñar métodos para superarlos
  $\Rightarrow$ Todos los métodos construyen sobre las nociones básicas de SGD (gradiente descendente)
- Momentum: Acelerar la convergencia utilizando una media móvil de los últimos valores
  $\Rightarrow$ El algoritmo tiene tendencia a moverse en la misma dirección en la que venía

SGD con Momentum

Dado: tasa de aprendizaje η, parámetro α, valor inicial $\boldsymbol{\theta}_0$, velocidad inicial $\mathbf{v}_0$

Para cada $k = 0, 1, \ldots$

Obtener un batch de muestras $\{\mathbf{x}_1, \ldots, \mathbf{x}_{M'}\}$

Estimar el gradiente: $\mathbf{g}_k \leftarrow M'^{-1} \sum_{m'} \partial \mathsf{J}(\mathsf{f}(\mathbf{x}_{m'}; \boldsymbol{\theta}_k))/\partial \boldsymbol{\theta}_k$

Calcular la velocidad $\mathbf{v}_{k+1} \leftarrow \alpha \mathbf{v}_k - \eta \mathbf{g}_k$

Actualizar los parámetros $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k + \mathbf{v}_{k+1}$

Mejorando SGD: Tasa de Aprendizaje

- Es evidente para cualquier usuario de deep learning que la tasa de aprendizaje es fundamental
  $\Rightarrow$ Y es uno de los hiperparámetros más difíciles de determinar correctamente

AdaGrad (Duchi et al , 2011)

Dado: tasa de aprendizaje η, valor inicial $\boldsymbol{\theta}_0$, constante $\delta \approx 10^{-7}$

Inicializar $\mathbf{r}_0 = \mathbf{0}$
Para cada $k = 0, 1, \ldots$
  Obtener un batch de muestras $\{\mathbf{x}_1, \ldots, \mathbf{x}_{M'}\}$
  Estimar el gradiente: $\mathbf{g}_k \leftarrow M'^{-1} \sum_{m'} \partial \mathsf{J}(\mathsf{f}(\mathbf{x}_{m'}; \boldsymbol{\theta}_k))/\partial \boldsymbol{\theta}_k$
  Calcular el cuadrado del gradiente $\mathbf{r}_{k+1} \leftarrow \mathbf{r}_k + \mathbf{g}_k^2$
  Actualizar los parámetros $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \frac{\eta}{\delta + \sqrt{\mathbf{r}_{k+1}}} \odot \mathbf{g}_k$

- Parámetros con gradiente elevado decrecen más rápido $\Rightarrow$ Avanza en direcciones no muy inclinadas
- Acumular gradientes desde el principio del entrenamiento ocasiona un excesivo decrecimiento de η

RMSProp (Hinton , 2012)

Dado: tasa de aprendizaje η, decaimiento ρ, valor inicial $\boldsymbol{\theta}_0$, constante $\delta \approx 10^{-6}$

Inicializar $\mathbf{r}_0 = \mathbf{0}$
Para cada $k = 0, 1, \ldots$
- Obtener un batch de muestras $\{\mathbf{x}_1, \ldots, \mathbf{x}_{M'}\}$
- Estimar el gradiente: $\mathbf{g}_k \leftarrow M'^{-1} \sum_{m'} \partial \mathsf{J}(\mathsf{f}(\mathbf{x}_{m'}; \boldsymbol{\theta}_k))/\partial \boldsymbol{\theta}_k$
- Calcular el cuadrado del gradiente $\mathbf{r}_{k+1} \leftarrow \rho \mathbf{r}_k + (1-\rho)\mathbf{g}_k^2$
- Actualizar los parámetros $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \frac{\eta}{\delta + \sqrt{\mathbf{r}_{k+1}}} \odot \mathbf{g}_k$

- Usa una media móvil para el cálculo del cuadrado de los gradientes $\Rightarrow$ Decae exponencialmente

  $\Rightarrow$ Descarta los cuadrados de los gradientes del pasado

ADAM (Kingma y Ba , 2014)

Dado: tasa de aprendizaje η, decaimientos ρ_1, ρ_2, valor inicial $\boldsymbol{\theta}_0$, constante $\delta \approx 10^{-8}$

Inicializar $\mathbf{r}_0 = \mathbf{0}$ y $\mathbf{s}_0 = \mathbf{0}$
Para cada $k = 0, 1, \ldots$
- Obtener un batch de muestras $\{\mathbf{x}_1, \ldots, \mathbf{x}_{M'}\}$
- Estimar el gradiente: $\mathbf{g}_k \leftarrow M'^{-1} \sum_{m'} \partial \mathsf{J}(\mathsf{f}(\mathbf{x}_{m'}; \boldsymbol{\theta}_k))/\partial \boldsymbol{\theta}_k$
- Estimar el primer momento $\mathbf{s}_{k+1} \leftarrow \rho_1 \mathbf{s}_k + (1 - \rho_1)\mathbf{g}_k$
- Estimar el segundo momento $\mathbf{r}_{k+1} \leftarrow \rho_2 \mathbf{r}_k + (1 - \rho_2)\mathbf{g}_k^2$
- Corregir el sesgo $\mathbf{s}_{k+1} \leftarrow \mathbf{s}_{k+1}/(1 - \rho_1^{k+1})$ y $\mathbf{r}_{k+1} \leftarrow \mathbf{r}_{k+1}/(1 - \rho_2^{k+1})$
- Actualizar los parámetros $\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \eta \mathbf{s}_{k+1}/(\delta + \sqrt{\mathbf{r}_{k+1}})$

- Se utiliza momentum para estimar el gradiente (ver vector $\mathbf{s}$)
- Se corrigen los estimadores de primer y segundo orden

Inicialización de los Parámetros $\boldsymbol{\theta}_0$

- Algunos algoritmos no son iterativos sino que proponen una solución para el punto de interés
- Otros algoritmos son iterativos y convergen independientemente de la inicialización
- En Deep Learning, los algoritmos son iterativos y son sensibles a la elección de $\boldsymbol{\theta}_0$
  $\Rightarrow$ Los puntos de inicialización determinan el tipo de solución, la convergencia y la velocidad

- Las estrategias de inicialización son mayormente heurísticas (dificultad del problema)
- Objetivo primordial de la inicialización: quebrar la simetría de las distintas hidden units
- Solución típica: elegir los pesos $\mathbf{W}_0$ de manera aleatoria (normal o uniforme)
  $\Rightarrow$ La distribución no parece afectar mucho, pero la escala sí
  $\Rightarrow$ Valores grandes de inicialización van a quebrar mejor la simetría
  $\Rightarrow$ Si son muy grandes puede que desestabilicen numéricamente la optimización
- Elegir los vectores de offset $\mathbf{b}_0$ a una determinada constante (cercana a cero)

Resumen del Capítulo

- Qué quiere decir aprender $\Rightarrow$ Resolver el problema de minimización del riesgo empírico
  - $\Rightarrow$ Usar los datos porque no contamos con la distribución de los mismos
  - $\Rightarrow$ Problema que persiste: encontrar la mejor función
- Optimizar sobre el espacio de todas las funciones posibles es prácticamente imposible
  - $\Rightarrow$ Elegir una parametrización de la función de aprendizaje $\Rightarrow$ Redes Neuronales
  - $\Rightarrow$ Las redes neuronales son cascadas de capas $\Rightarrow$ Transformación lineal + activación no lineal
  - $\Rightarrow$ Las redes neuronales tienen hiperparámetros a cargo del diseñador: profundidad, ancho, activación
- Optimizar sobre parámetros se puede resolver $\Rightarrow$ Pero necesitamos automatizar la optimización
  - $\Rightarrow$ Algoritmo de gradiente (estocástico) descendente
  - $\Rightarrow$ Algoritmo de backpropagation (calcular automáticamente la derivada $\Rightarrow$ regla de la cadena)
  - $\Rightarrow$ Más hiperparámetros para el diseñador: tasa de aprendizaje, tamaño de batch, cantidad de epochs

Escenas del Próximo Capítulo...

- El algoritmo de gradiente descendente requiere un riesgo derivable
  $\Rightarrow$ ¿Qué hacemos en problemas que no ofrecen riesgos derivables de manera natural?
  $\Rightarrow$ Elección de la función de costo

- ¿Alcanza sólo con minimizar la función de costo? ¿Se puede incorporar información externa?
  $\Rightarrow$ Regularización: Técnicas para mejorar el entrenamiento y aprendizaje