Proyecto: Esquema de Aprendizaje Basado en la Ecuación HJB para redes Neuronales Artificiales

Optimización I

Maestría en Computación

Centro de Investigación en Matemáticas

Axel Báez Ochoa axel.baez@cimat.mx

Esteban Reyes Saldaña esteban.reyes@cimat.mx

26 de marzo de 2021

Resumen

En este trabajo se plantea el problema de la actualización instantánea de los pesos de una Red Neuronal Alimentada Hacia Adelante (FFNN) como un problema de control inspirado en el algoritmo de Retropropagación (mejor conocido como Backpropagation). Se utiliza la ecuación de Hamilton-Jacobi-Bellman (HJB) para encontrar una regla de actualización de pesos óptima. La contribución principal de este trabajo es que, utilizando la ecuación HJB, pueden obtenerse soluciones tanto para el costo óptimo como para las actualizaciones de los pesos en cualquier Red Neuronal Alimentada Hacia Adelante. Se compara el enfoque propuesto con algunos de los algoritmos de aprendizaje más conocidos. Se ha encontrado que con esta propuesta la convergencia es más rápida en términos de tiempo computacional.

1. Introducción

Alrededor de 1960, Frank Rosenblatt [3] introdujo el modelo del perceptrón inspirado en el comportamiento de las neuronas humanas las cuales reciben varios estímulos y son capaces de producir una salida. Para calcular la salida, Rosenblatt introdujo el concepto de "peso" como un número real que expresa la importancia de los estímulos recibidos en la neurona. Se observó que un perceptrón era capaz de aprender a emitir una respuesta acertada dados ciertos estímulos recibidos. El problema de aprendizaje de un perceptrón radica en encontrar un conjunto de pesos adecuado, sujeto a un conjunto de aprendizaje.

El cómputo neuronal disfrutó de un auge impresionante a principios de los 60's, teniendo como personajes principales a Marvin Minsky y David Rumelhart. Sin embargo, en 1969, en el libro "Perceptrones: Una Introducción a la Geometría Computacional" [1], Minsky habla sobre las limitaciones del modelo del perceptrón al demostrar la incapacidad de un perceptrón de aprender de la puerta lógica XOR y desechó la posibilidad de unir varios perceptrones para formar una red, pues el problema de encontrar un conjunto de pesos adecuado se volvía más complicado.

Como Minsky tenía gran influencia en la política científica de entonces, todo el cómputo neuronal se detuvo por casi 20 años, a este periodo se le conoce como "El Invierno de la Inteligencia Artificial". En 1986, Rumelhart publicó el libro "Procesos Paralelos Distribuídos", donde por primera vez se habla de una red de neuronas conectadas cuya

capacidad colectiva de aprendizaje no es igual a la suma de las capacidades de aprendizaje individuales. Además, exhibe un método de aprendizaje para una red neuronal llamado "Backpropagation".

Desde la publicación del algoritmo de Backpropagation, se han entrenado a las redes neuronales para reconocer patrones de conjuntos de aprendizaje. Algunos de los problemas que se resolvieron con este método incluyen reconocomiento computacional de dígitos escritos a mano, lectura automática de textos, caminatas robóticas autónomas, aprobación de crédito bancario, entre otras. Sin embargo, debido a las limitaciones de procesamiento computacional, se abandonó hasta finales de los noventas.

La capacidad de procesamiento en los últimos años ha mostrado un crecimiento exponencial y las redes neuronales han mostrado la capacidad de aprender de problemas complejos. Para conjuntos de aprendizaje suficientemente grandes, Backpropagation ha mostrado ser ineficiente en términos de tiempo computacional.

El algoritmo Backpropagation hace uso del descenso gradiente. A través de la última década, se han propuesto varios esquemas de aprendizaje para redes neuronales que mejoran el procesamiento y la tasa de convergencia de dicho algoritmo: el descenso gradiente hace uso de la primera derivada y el método de Newton hace uso de la segunda derivada para mejorar el descenso gradiente, sin embargo, esto es computacionalmente intensivo. El método de Gauss-Newton aproxima la segunda derivada con ayuda de la primera derivada para así, dar una optimización más simple.

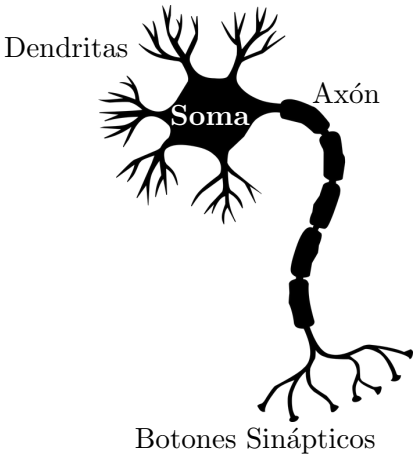
Por ejemplo, el algoritmo basado en la Función de Liapunov, interpola entre el descenso gradiente y el método de Gauss-Newton para mejorar la tasa de convergencia. A partir de este algoritmo, se han propuesto varios métodos que hacen uso de la teoría de estabilidad de Liapunov para mejorar el tiempo de convergencia.

Por otro lado, un problema de control consiste de una función llamada "estado" que está controlada por funciones llamadas "controles" que se mueve a través del tiempo y de la que se conoce su dinámica (que es una ecuación diferencial del estado y de los controles con una condición incial). Se mide el costo de moverse a lo largo de un intervalo de tiempo en la curva estado-tiempo y se trata de minimizar esta función de costo sujeta a las condiciones anteriores. Se demuestra que un problema de control, con las condiciones adecuadas, se puede llevar a la ecuación de Hamilton-Jacobi-Bellman.

En este trabajo se utiliza un algoritmo que planea el problema de actualización de los pesos en una red neuronal como un problema de control que se resuelve utilizando la ecuación de Hamilton-Jacobi-Bellman (HJB). Para la experimentación numérica, se utilizaron dos conjuntos de datos y los resultados del aprendizaje obtenidos se compararon con algunos de los algoritmos más populares.Para estos ejemplos, se encontró que el algoritmo HJB converge más rápido en términos de tiempo computacional.

2. Redes Neuronales Artificiales

2.1. Inspiración Biológica



Actualmente se sabe que las neuronas en el cerebro humano se encargan de hacer posible el aprendizaje conectándose entre ellas para aprender de los estímulos que reciben. Lo que básicamente ocurre es que por medio de las **dendritas**, una neurona es estimulada o excitada mediante impulsos eléctricos procesados en el cuerpo de la neurona o **soma**. Cuando en el soma se alcanza un cierto

umbral, se dispara o activa, pasando la señal al **axón** que envía la información del soma para luego comunicarse con otras neuronas por medio de los **botones sinápticos**.

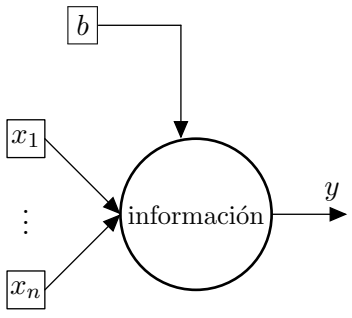
Se ha observado que las neuronas aprenden de la experiencia para responder a nueva información. Además, trabajan en conjunto como un sistema cuyas partes no tienen las mismas propiedades que el sistema completo.

3. Perceptrón como Modelo Matemático

Definición 3.1. Un **perceptrón** o neurona $\mathcal{N}$ es un programa que recibe una o más entradas, externas o de otras neuronas, para luego combinarlas y producir una salida.

Matemáticamente, un perceptrón que consta de n entradas, recibe información de vectores $n-$dimensionales $(x_1, x_2, \ldots, x_p)$ a los que se les denomina **vectores de entrada** para producir vectores de salida (uno por cada vector de entrada). En este momento se considerarán neuronas cuya salida sea un número real.

Para un vector de entrada fijo $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ se produce una salida $y \in \mathbb{R}$, como se ilustra en el siguiente diagrama.



Observación 1. Se incorpora un término b denominado **sesgo o bias** $b \in \{0, 1\}$, que representa una carga eléctrica acumulada de un proceso realizado con anterioridad. En este sentido, $b = 0$ representa que la neurona no se activó y $b = 1$ diría que se activó. Más adelante se verá que este término sirve de "soporte" para la información que recibe el cuerpo de la neurona.

En el modelo biológico, la intensidad depende del impulso eléctrico asociado a las conexiones entre neuronas. En este modelo matemático, lo anterior quiere decir que cada uno de los componentes de un vector de entrada dado, tiene asociado un número real que pondera esta información al que sen le llamará **peso**.

Definición 3.2. Se toma una neurona $\mathcal{N}$ con $n \in \mathbb{N}$ entradas. El **vector de pesos** o vector de nivel de influencia $w \in \mathbb{R}^n$, asociado a $\mathcal{N}$, es el vector que pondera a cada uno de los vectores de entrada.

Esto quiere decir que, para un vector de pesos $w = (w_1, \ldots, w_n)$ fijo y un vector de entrada dado $x = (x_1, \ldots, x_n)$, la información de x que llega a $\mathcal{N}$ está dada por

$$x \cdot w = \sum_{i=1}^{n} w_i x_i.$$

Nota. A cada una de las entradas de un vector de pesos se le llama peso.

Al término bias también se le asocia un peso. Así que la información total que llega a una neurona $\mathcal{N}$, de un vector de entrada $x = (x_1, \ldots, x_n)$ y un término b, está dada por

$$h = x \cdot w + w_b b = \sum_{i=1}^{n} w_i x_i + w_b b. \qquad (1)$$

Observación 2. Por convención w_i se toma en $[-1, 1]$. Dado $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$, tal que $x_i \geq 0$, para todo $1 \leq i \leq n$,

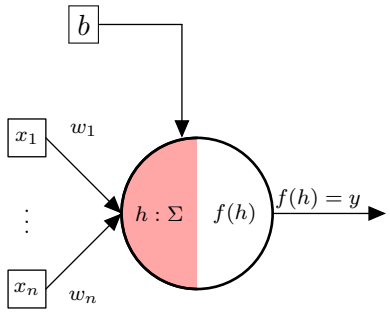
1. Si $w_i = 0$, la entrada x_i no llega a la neurona.
2. Si $w_i = 1$, entrada x_i llega completa.
3. Si $w_i \in (0, 1)$, la entrada x_i llega a la neurona pero con menor intensidad.
4. Si $w_i \in (-1, 0)$, hay un robo de información.

Definición 3.3. Una **función de activación** es una función f que calcula la salida de una neurona a partir de los valores de entrada. Es decir, el argumento de f es (1).

Observación 3. Cuando recién se inventó el modelo de perceptrón, las funciones de activación se escogían binarias. Es decir, $f(x) \in \{0, 1\}$, donde si $f(x) = 0$, la neurona no se activa y si $f(x) = 1$, la neurona se activa.

De esta manera se obtiene la salida de la neurona $y \in \mathbb{R}$.

El siguiente esquema resume las etapas del transito de la información en una neurona matemática.



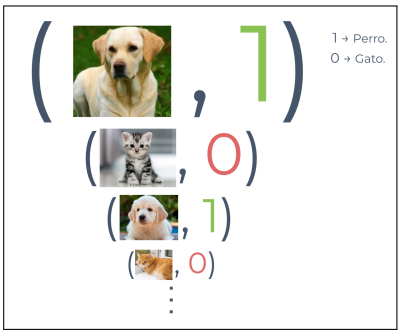
3.1. Aprendizaje de una Neurona

Definición 3.4. Sea $\mathcal{N}$ una neurona con n entradas. Dados un conjunto de vectores de entrada $x_1, \ldots, x_k \in \mathbb{R}^n$ y un conjunto de salidas $y_1, \ldots, y_k \in \mathbb{R}$, se define un **conjunto de entrenamiento** o **conjunto de patrones** como el conjunto de todos los pares (x_i, y_i) para $1 \leq i \leq k$. Es decir, un conjunto de aprendizaje es un conjunto de vectores de entrada de los que ya se conoce su salida.

$$\begin{aligned} x_1 = (x_{1_1}, \ldots, x_{n_1}) &\rightarrow y_1^d, \\ x_2 = (x_{1_2}, \ldots, x_{n_2}) &\rightarrow y_2^d, \\ x_3 = (x_{1_3}, \ldots, x_{n_3}) &\rightarrow y_3^d, \\ &\vdots \\ x_k = (x_{1_k}, \ldots, x_{n_k}) &\rightarrow y_k^d, \end{aligned}$$

donde el superíndice d denota que tal salida es la "deseada".

Por ejemplo, un conjunto de aprendizaje podría estar dado por vectores de entrada que representen imágenes de perros o gatos y la salida a cada uno de estos vectores de entrada será uno si corresponde a un perro o cero si corresponde a un gato.



Nótese que, en general, dada una neurona $\mathcal{N}$ y un conjunto de aprendizaje, se desconoce el vector de pesos tal que, al pasar la información por $\mathcal{N}$, dé la salida y_i del conjunto de aprendizaje. Por lo tanto, la meta es encontrar el vector de pesos $w = (w_1, \ldots, w_n)$ y el peso w_b para que, dado un Conjunto de Aprendizaje, cuando cada uno de los vectores de entrada $x_i \in \mathbb{R}^n$ entre a $\mathcal{N}$, dé la salida deseada y_i.

Si inicialmente se da un vector de pesos aleatorios, éste se podría iterar para que aproxime mejor las salidas deseadas.

Definición 3.5. Sea $T \in \mathbb{N}$ un número finito de iteraciones de pesos y t tal que $1 \leq t \leq T$. Dado un patrón (x_p, y_p^d) de un conjunto de aprendizaje y un vector de w_t en la iteración t, se denotará a la **salida producida** por x_p, en la iteración t, como y_p^t.

Definición 3.6. Se dirá que una neurona **está aprendiendo** si, dadas las suposiciones de la Definición 3.5 y una función de error $e_t(y_p^d, y_p^t) > 0$, se cumple que

$$e_{t+1} \leq e_t.$$

Una neurona es capaz de modelar un conjunto de aprendizaje dado variando los pesos, como se verá a continuación.

Nota. Hasta ahora, no se ha dado un método que diga cómo ajustar los pesos iniciales.

3.2. Ejemplos con una neurona

Ejemplo 3.1. PUERTA AND

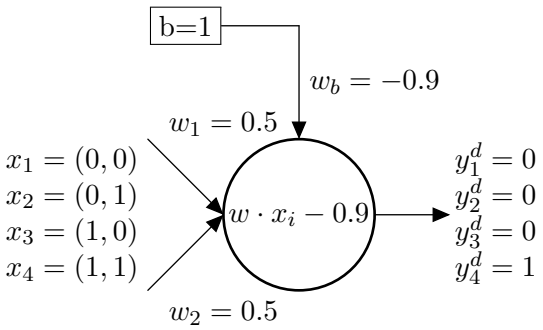
Se quiere que una neurona modele la tabla de verdad de la conjunción, dada por

$$\begin{aligned} x_1 = (0,0) &\rightarrow y_1^d = 0, \\ x_2 = (0,1) &\rightarrow y_2^d = 0, \\ x_3 = (1,0) &\rightarrow y_3^d = 0, \\ x_4 = (1,1) &\rightarrow y_4^d = 1. \end{aligned}$$

Donde 0 representa falso y 1 verdadero. Se toma $b = 1$ y se dan los siguientes pesos iniciales

$$\begin{aligned} w &= (1,1), \\ w_b &= 1. \end{aligned}$$

Se procede a ajustar al tanteo. Una combinación de pesos que hacen posible este resultado es



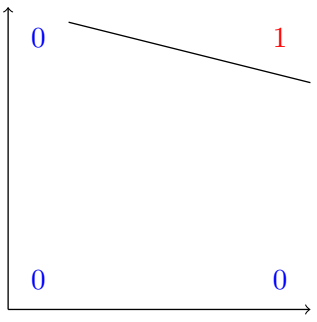
con la función de activación

$$f(t) = \begin{cases} 0, & \text{si } t < 0, \\ 1, & \text{si } t \geq 0 \end{cases}$$

En realidad, para este ejemplo,

$$0.5x_{i_1} + 0.5x_{i_2} - 0.9 = 0 \qquad (2)$$

es una recta que separa en dos clases a las salidas deseadas en el plano $x_{i_1}x_{i_2}$, como se observa en la siguiente figura



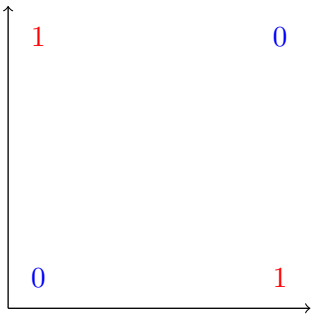
Así que, para una neurona con solo dos pesos y salidas binarias, el problema de encontrar una combinación de pesos se convierte en encontrar un separador lineal del conjunto de puntos en el espacio de patrones en dos clases.

Observación 4. La combinación de pesos que hace posible la separabilidad lineal, no es única. Además, w_b funciona como el término independiente en (2), el cual permite desplazar la recta de manera vertical.

Ejemplo 3.2. PUERTA XOR. El siguiente conjunto de aprendizaje representa la tabla de verdad de la disyunción exclusiva. Nuevamente, 0 representa falso y 1 verdadero.

$$\begin{aligned} x_1 = (0,0) &\rightarrow y_1^d = 0 \\ x_2 = (0,1) &\rightarrow y_2^d = 1 \\ x_3 = (1,0) &\rightarrow y_3^d = 1 \\ x_4 = (1,1) &\rightarrow y_4^d = 0. \end{aligned}$$

Así que se quiere encontrar una recta que separe en dos clases al diagrama



Lo cual no es posible con una sola recta pero sí con dos. De manera intuitiva, esto dice que si se agrega otra neurona al programa, se podría separar linealmente a este conjunto de patrones. Marvin Minksy en su libro Perceptrons [1] en 1969 escribió sobre las limitaciones de una neurona, pues

es incapaz de trascender a la separalibilidad lineal. Aunque Minsky habló sobre la posibilidad de agregar más neuronas para resolver el problema de la separabilidad lineal, descalificó este método, argumentando que no se encontraría un algoritmo para encontrar un vector de pesos adecuado que hiciera que las neuronas aprendieran. Como Minsky tenía un gran prestigio en la política científica, todas las investigaciones en cómputo neuronal se detuvieron alrededor de 1970.

Sin embargo, en 1987, David Rumelhart y Francis Crick, en su libro Procesos Paralelos Distribuídos [2] escribieron sobre redes de neuronas cuya capacidad colectiva no es igual a la suma de las capacidades individuales. Además, mostraron un algoritmo para entrenar una red neuronal a partir de un conjunto de aprendizaje, que ajustó los pesos de manera adecuada.

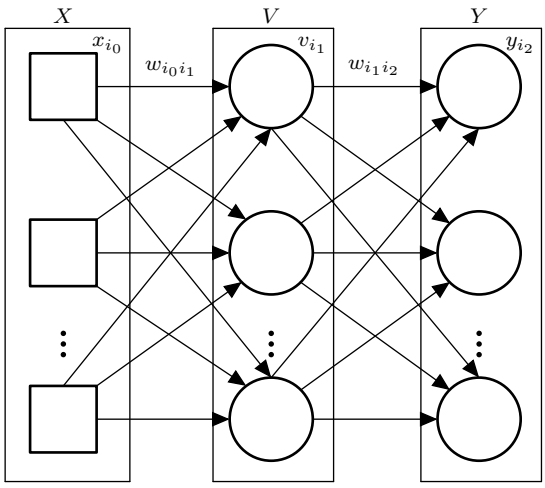
Dicho algoritmo se llama Retropropagación, del cual se hablará en la Sección 5.

4. Redes Neuronales Artificiales

Definición 4.1. Una **Red Neuronal Preali-mentada** consiste de una capa de entrada (X con $N_0 \in \mathbb{N}$ entradas ordenadas) y dos capas de neuronas ordenadas (V con $N_1 \in \mathbb{N}$ neuronas, Y con $N_2 \in \mathbb{N}$ neuronas) llamadas capa oculta y capa de salida, respectivamente, tales que sólo existen conexiones hacia adelante entre neuronas de capas adyacentes.

Nota. La definición anterior indica que, en una red neuronal artificial de este tipo, no se permiten ciclos ni conexiones entre neuronas de la misma capa.

La siguiente figura representa una red neuronal como en la Definición 4.1.

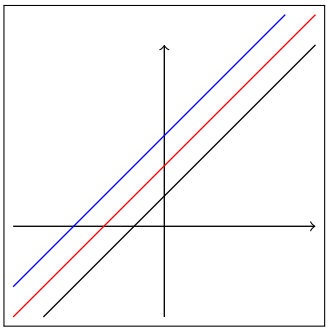


Nótese que el segundo subíndice indica la capa a la que pertenece la neurona en cuestión. **Funciones de Activación**

La intensidad de la información que pasa de la capa de entrada a la neurona i_1 de la capa oculta está dada por

$$h_{i_1} = \sum_{i_0} x_{i_0} w_{i_1} + \theta_{i_1}. \tag{3}$$

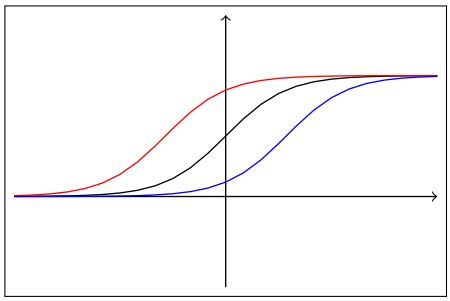
Donde θ_{i_1} es el bias correspondiente a la neurona con la etiqueta i_1. En términos geométricos, este término permite desplazar verticalmente al hiperplano definido por los pesos.



Intuitivamente (3) dice que la información recibida en una neurona de la capa oculta es la suma de la información de cada neurona de la capa anterior con la intensidad que está dada por el respectivo peso. Una vez que se tiene (3), este término pasa a través de una función de activación, análogo al modelo de una sola neurona.

$$f\left(\sum_{i_0} x_{i_0} w_{i_1} + \theta_{i_1}\right) = f(h_{i_1}).$$

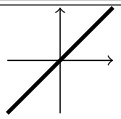
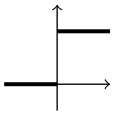
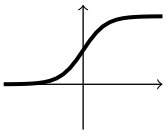
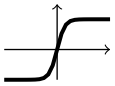
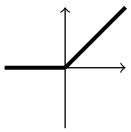
Para la función de activación, el término θ_{j_1} permite trasladar la función de activación horizontalmente.



En resumen,

$$h_{i_1} = \sum_{i_0} w_{i_0 i_1} x_{i_0}, \quad f(h_{i_1}) = v_{i_1},$$
$$h_{i_2} = \sum_{i_1} w_{i_1 i_2} v_{i_1}, \quad f(h_{i_2}) = y_{i_2}.$$

Algunos ejemplos clásicos de funciones de activación son

Función	Gráfica
Lineal $f(x) = x$	
Límite Duro $f(x) = \begin{cases} 0, & si \quad x < 0 \\ 1, & si \quad x \geq 0 \end{cases}$	
Sigmoide $f(x) = \dfrac{1}{1+e^{-x}}$	
Tangente Hiperbólica $f(x) = \dfrac{e^x - e^{-x}}{e^x + e^{-x}}$	
Lineal Positiva $f(x) = \begin{cases} 0, & si \quad x < 0 \\ x, & si \quad x \geq 0 \end{cases}$	

Además, se pueden introducir constantes α, β a una función de activación tales que

$$\alpha + \beta f\left(\sum_{i_0} x_{i_0} w_{i_1} + \theta_{i_1}\right),$$

donde α traslada a la función de activación y β la dilata verticalmente.

Observación 5. Como se vió en la primera sección, los pesos iniciales se dan de manera aleatoria y se requieren ajustarlos para que converjan a los valores esperados del conjunto de entrenamiento. Uno de los primeros métodos para hacer que una red neuronal converja es el de Backpropagation, que usa el descenso gradiente para aproximarse a un mínimo local de una función de error dada.

5. Backpropagation

5.1. Descenso Gradiente

Definición 5.1. Sea $f : \mathbb{R}^n \to \mathbb{R}$ una función derivable. El **gradiente** de f en p es

$$\nabla f(p) = \left(\frac{\partial f(p)}{\partial x_1}, \frac{\partial f(p)}{\partial x_2}, \ldots, \frac{\partial f(p)}{\partial x_n}\right).$$

Para una función de n variables, el gradiente es el vector normal a la curva de nivel en un punto dado.

Teorema 1. Sea $f : \mathbb{R}^n \to \mathbb{R}$ una función derivable. La dirección donde f crece más rápido es la dirección de ∇f (el gradiente de f).

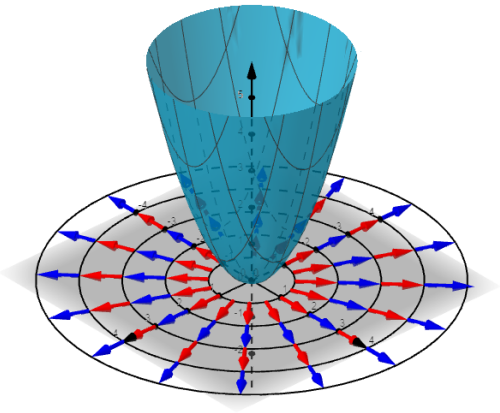
Corolario 1. Bajo las condiciones del teorema anterior, f decrece más rápido en la dirección de $-\nabla f$.

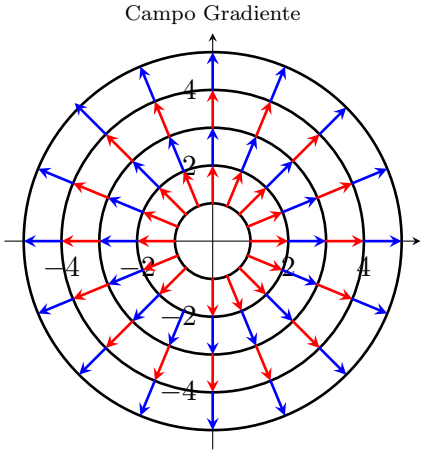
Observación 6. Para el esquema de aprendizaje de Backpropagation se considerará la función de error (Error Cuadrático) dada por

$$E = \frac{1}{2}\sum_{i_2}(y_{i_2} - y_{i_2}^d)^2,$$

donde y_{i_2} es la salida producida por la red neuronal en la neurona con etiqueta i_2 en un momento dado y $y_{i_2}^d$ es la salida deseada. La suma en esta función de error está multiplicada por un medio, ya que más adelante se requerirá derivar esta función con respecto a los pesos.

Ejemplo 5.1. Para la función $f(x, y) = x^2 + y^2$ cuyo gradiente es $\nabla f(x, y) = (2x, 2y)$ se tiene el siguiente campo gradiente





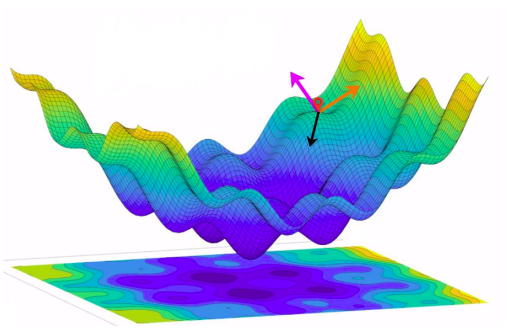
Es de interés saber cómo se cambian los pesos usando el descenso del gradiente, para esto, el Corolario 1 dice que para acercarse a un mínimo local en la función de error cuadrático $E(\hat{w})$, se debe actualizar el vector de pesos en la dirección opuesta al vector gradiente.

Observaciones. Sea $\hat{w}$ un vector de pesos arbitrario.

1. Para encontrar un error mínimo local, se requiere moverse en dirección $-\nabla E$.
2. Se deben tomar pequeñas porciones del gradiente de $E(\hat{w})$ para acercarnos a un mínimo, es decir

$$\hat{w} - \eta \nabla E(\hat{w}),$$

donde $0 < \eta \leq 1$.



5.2. La Función de Activación Sigmoide

Dicha función se define como

$$\sigma(x) = \frac{1}{1+e^{-x}}. \tag{4}$$

Cuya derivada es

$$\begin{aligned}
\sigma'(x) &= \frac{d}{dx}(1+e^{-x})^{-1} \\
&= -(1+e^{-x})^{-2}(-e^{-x}) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1+e^{-x}-1}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2} \\
&= \sigma(x) - \sigma^2(x) \\
&= \sigma(x)(1-\sigma(x)).
\end{aligned}$$

La derivada de la función sigmoide queda en términos de ella misma, para la capa oculta el argumento de la función sigmoide es

$$h_{i_1} = \sum_{i_0} x_{i_0} w_{i_1} + \theta_{i_1}.$$

Entonces, para la derivada, se tiene

$$\begin{aligned}
\sigma'(h_{i_1}) =& \sigma(h_{i_1})(1-\sigma(h_{i_1})) \\
=& v_{i_1}(1-v_{i_1}).
\end{aligned} \tag{5}$$

Por lo tanto, la derivada de la función sigmoide en h_{i_1} queda en términos de la salida de la neurona con etiqueta i_1. Análogamente para h_{i_2} se tiene

$$\begin{aligned}
\sigma'(h_{i_2}) &= \sigma(h_{i_2})(1-\sigma(h_{i_2})) \\
&= y_{i_2}(1-y_{i_2}).
\end{aligned} \tag{6}$$

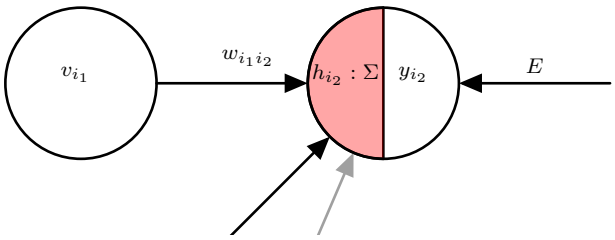
La función $E(\hat{w}) = \frac{1}{2}\sum_{i_2}(y_{i_2} - y^d_{i_2})^2$ depende de los pesos. El objetivo es encontrar el vector gradiente de la función de error cuadrático, por lo que se debe calcular la derivada parcial de E con respecto a cada peso.

5.3. De la Capa de Salida a la Capa Oculta

Para calcular la derivada de E con respecto a un peso fijo que está entre la capa oculta y la capa de salida, se tiene

$$\frac{\partial E}{\partial w_{i_1 i_2}} = \frac{\partial E}{\partial y_{i_2}}\frac{\partial y_{i_2}}{\partial h_{i_2}}\frac{\partial h_{i_2}}{\partial w_{i_1 i_2}}, \tag{7}$$

que representa retroceder de la última capa hacia la capa oculta. Se procede a realizar los cálculos.



El error con respecto a la salida es

$$\begin{aligned}
\frac{\partial E}{\partial y_{i_2}} &= \frac{\partial}{y_{i_2}}\left(\frac{1}{2}\sum_{i_2}(y_{i_2} - y^d_{i_2})^2\right) \\
&= \frac{1}{2}\frac{\partial}{\partial y_{i_2}}\left(\sum_{i_2}(y_{i_2} - y^d_{i_2})^2\right) \\
&= \frac{1}{2}2(y_{i_2} - y^d_{i_2})\frac{\partial}{\partial y_{i_2}}(y_{i_2} - y^d_{i_2}) \qquad (8) \\
&= \frac{1}{2}2(y_{i_2} - y^d_{i_2})\left(\frac{\partial y_{i_2}}{\partial y_{i_2}} - \frac{\partial y^d_{i_2}}{\partial y_{i_2}}\right) \\
&= \frac{1}{2}2(y_{i_2} - y^d_{i_2})\,(1 - 0) \\
&= y_{i_2} - y^d_{i_2}.
\end{aligned}$$

La salida con respecto a la suma es

$$\begin{aligned}
\frac{\partial y_{i_2}}{\partial h_{i_2}} &= \frac{\partial}{\partial h_{i_2}}\sigma(h_{i_2}) \\
&= \underbrace{\sigma(h_{i_2})(1 - \sigma(h_{i_2}))}_{6} \qquad (9) \\
&= y_{i_2}(1 - y_{i_2}).
\end{aligned}$$

La suma con respecto al peso es

$$\begin{aligned}
\frac{\partial h_{i_2}}{\partial w_{i_1 i_2}} &= \frac{\partial}{\partial w_{i_1 i_2}}\left(\sum_{i_1} w_{i_1 i_2} v_{i_1} + \theta_{i_1}\right) \qquad (10) \\
&= v_{i_1}.
\end{aligned}$$

De lo anterior se tiene que

$$\begin{aligned}
\frac{\partial E}{\partial w_{i_1 i_2}} &= \frac{\partial E}{\partial y_{i_2}}\frac{\partial y_{i_2}}{\partial h_k}\frac{\partial h_{i_2}}{\partial w_{i_1 i_2}} \\
&= \underbrace{(y_{i_2} - y^d_{i_2})}_{(8)}\underbrace{y_{i_2}(1 - y_{i_2})}_{(9)} v_{i_1}.
\end{aligned}$$

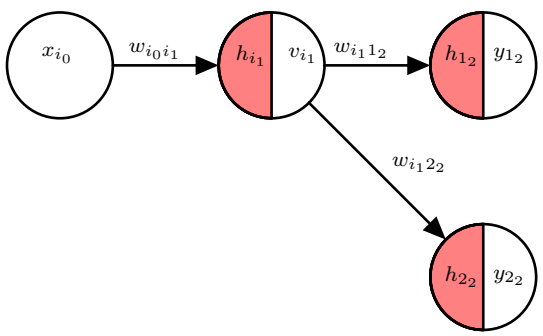
Las primeras dos componentes de la última expresión dependen de la neurona con etiqueta i_2, por motivos de notación se nombrará como

$$\delta_{i_2} = y_{i_2}(1 - y_{i_2})(y_{i_2} - y^d_{i_2}). \tag{11}$$

Por lo que

$$\frac{\partial E}{\partial y_{i_2}} = v_{i_1}\delta_{i_2}.$$

5.4. De la Capa Oculta a la Capa de Entrada



El error para $w_{i_0 i_1}$ depende del error de las conexiones de la capa anterior.

$$\begin{aligned}
\frac{\partial E}{\partial w_{i_0 i_1}} &= \frac{\partial}{\partial w_{i_0 i_1}} \frac{1}{2} \sum_{i_2} (y_{i_2} - y_{i_2}^d)^2 \\
&= \frac{1}{2} \cdot 2 \sum_{i_2} (y_{i_2} - y_{i_2}^d) \frac{\partial}{\partial w_{i_0 i_1}} (y_{i_2} - y_{i_2}^d) \\
&= \sum_{i_2} (y_{i_2} - y_{i_2}^d) \left(\frac{\partial y_{i_2}}{\partial w_{i_0 i_1}} - \frac{\partial y_{i_2}^d}{\partial w_{i_0 i_1}} \right) \\
&= \sum_{i_2} (y_{i_2} - y_{i_2}^d) \left(\frac{\partial y_{i_2}}{\partial w_{i_0 i_1}} - 0 \right) \\
&= \sum_{i_2} (y_{i_2} - y_{i_2}^d) \frac{\partial y_{i_2}}{\partial w_{i_0 i_1}} \\
&= \sum_{i_2} (y_{i_2} - y_{i_2}^d) \frac{\partial y_{i_2}}{\partial h_{i_2}} \frac{\partial h_{i_2}}{\partial v_{i_1}} \frac{\partial v_{i_1}}{\partial w_{i_0 i_1}} \\
&= \underbrace{\frac{\partial v_{i_1}}{\partial w_{i_0 i_1}}}_{12} \sum_{i_2} (y_{i_2} - y_{i_2}^d) \underbrace{\frac{\partial y_{i_2}}{\partial h_{i_2}}}_{6} \frac{\partial h_{i_2}}{\partial v_{i_1}}.
\end{aligned}$$

Calculando las derivadas

$$\begin{aligned}
\frac{\partial v_{i_1}}{\partial w_{i_0 i_1}} &= \frac{\partial v_{i_1}}{\partial h_{i_1}} \frac{\partial h_{i_1}}{\partial \partial w_{i_0 i_1}} \\
&= \underbrace{\frac{\partial}{\partial h_{i_1}} \sigma(h_{i_1})}_{5} \frac{\partial}{\partial w_{i_0 i_1}} \sum_{i_1} (w_{i_0 i_1} x_{i_0} + \theta_{i_1}) \\
&= v_{i_1}(1 - v_{i_1}) x_{i_0}.
\end{aligned} \tag{12}$$

$$\begin{aligned}
\frac{\partial h_{i_2}}{\partial v_{i_1}} &= \frac{\partial}{\partial v_{i_1}} \sum_{i_2} (w_{i_0 i_1} v_{i_1} + \theta_{i_2}) \\
&= w_{i_0 i_1}.
\end{aligned} \tag{13}$$

Así que para la capa oculta se obtiene

$$\begin{aligned}
\frac{\partial E}{\partial w_{i_0 i_1}} &= \frac{\partial v_{i_1}}{\partial w_{i_0 i_1}} \sum_{i_2} (y_{i_2} - y_{i_2}^d) \frac{\partial y_{i_2}}{\partial h_{i_2}} \frac{\partial h_{i_2}}{\partial v_{i_1}} \\
&= \underbrace{v_{i_1}(1 - v_{i_1}) x_{i_0}}_{(12)} \sum_{i_2} \underbrace{(y_{i_2} - y_{i_2}^d) \overbrace{y_{i_2}(1 - y_{i_2})}^{(9)}}_{(11)} w_{i_1 i_2} \\
&= v_{i_1}(1 - v_{i_1}) x_{i_0} \sum_{i_2} \delta_{i_2} w_{i_1 i_2} \\
&= x_{i_0} v_{i_1}(1 - v_{i_1}) \sum_{i_2} \delta_{i_2} w_{i_1 i_2}.
\end{aligned}$$

Si se define

$$\delta_{i_1} = v_{i_1}(1 - v_{i_1}) \sum_{i_2} \delta_{i_2} w_{i_1 i_2},$$

entonces

$$\frac{\partial E}{\partial w_{i_0 i_1}} = x_{i_0} \delta_{i_1}.$$

Incorporando el Bias

Para las neuronas de la última capa con sesgo (θ_{i_2}), se tiene que

$$\begin{aligned}
\frac{\partial E}{\partial \theta_{i_2}} &= \frac{\partial E}{\partial y_{i_2}} \frac{\partial y_{i_2}}{\partial h_{i_2}} \frac{\partial h_{i_2}}{\partial \theta_{i_2}} \\
&= \delta_{i_2} \frac{\partial h_{i_2}}{\partial \theta_{i_2}} \\
&= \delta_{i_2} \frac{\partial}{\partial \theta_{i_2}} \sum_{i_2} (w_{i_1 i_2} v_{i_1} + \theta_{i_2}) \\
&= \delta_{i_2}.
\end{aligned}$$

5.5. El Algoritmo

1. Crear una red neuronal y cargar los datos de entrenamiento.

   Nota. La red neuronal que se crea tiene un número fijo de neuronas para la capa de entrada y la capa de salida (dependen del conjunto de aprendizaje). Sin embargo, el número de neuronas en la capa oculta queda libre.

2. Generar pesos aleatorios tanto para las conexiones entre neuronas como para los bias.

3. Para cada nodo en la capa de salida, calcular

   $$\delta_{i_2} = y_{i_2}(1 - y_{i_2})(y_{i_2} - y_{i_2}^d).$$

4. Para cada nodo en la capa oculta, calcular

   $$\delta_{i_1} = v_{i_1}(1 - v_{i_1}) \sum_{i_2} \delta_{i_2} w_{i_1 i_2}.$$

5. Actualizar los pesos como sigue

   $$\begin{aligned}
   \Delta w_{i_1 i_2} &= -\eta \delta_{i_2} v_{i_1}, \\
   \Delta w_{i_0 i_1} &= -\eta \delta_{i_1} x_{i_0}, \\
   \Delta \theta_{i_2} &= -\eta \delta_{i_2}, \\
   \Delta \theta_{i_1} &= -\eta \delta_{i_1},
   \end{aligned}$$

para un $0 < \eta \leq 1$ (este número también se desconoce y debe ajustarse según la experimentación). Luego aplicar

$$\begin{aligned} w + \Delta w &\rightarrow w \\ \theta + \Delta\theta &\rightarrow \theta. \end{aligned}$$

Aunque este algoritmo ha mostrado eficiencia, los recursos computacionales son limitados y para grandes conjuntos de datos, requiere un tiempo computacional exhaustivo. Por otro lado, como los pesos se dan de manera aleatoria se podría comenzar en un mínimo local del espacio pesos-función de error que nunca cumpla con los parámetros elegidos.

De aquí la importancia de buscar nuevos algoritmos para entrenar a redes neuronales. En la siguiente sección se muestra un algoritmo que translada el problema de encontrar los pesos de una red neuronal a un problema de control cuya solución corresponde a la solución de una ecuación diferencial parcial conocida.

Referencias

[1] Minsky, M., and Papert, S. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA, USA, 1969.

[2] Rumelhart, D. E., McClelland, J. L., and PDP Research Group, Eds. *Parallel Distributed Processing. Volume 1: Foundations.* MIT Press, Cambridge, MA, 1986.

[3] Van Der Malsburg, C. Frank rosenblatt: Principles of neurodynamics: Perceptrons and the theory of brain mechanisms. In *Brain Theory* (Berlin, Heidelberg, 1986), G. Palm and A. Aertsen, Eds., Springer Berlin Heidelberg, pp. 245–248.