Introducción a las Redes Neuronales

Club de Machine Learning

E. C.

September 7, 2023

Introducción a las Redes Neuronales
Club de Machine Learning
E. C.
September 7, 2023

Outline

Outline

Aprendizaje estadístico

Optimización

Redes Neuronales

Backprop

Implementación

Introducción

El *aprendizaje de máquina/automático/autónomo/estadístico* es un área de estudio que busca brindar a las computadoras la capacidad de *aprender* sin ser explícitamente programadas.

Introducción

El *aprendizaje de máquina/automático/autónomo/estadístico* es un área de estudio que busca brindar a las computadoras la capacidad de *aprender* sin ser explícitamente programadas.

1. ¿Quién lo dijo? No recuerdo.

Introducción

El *aprendizaje de máquina/automático/autónomo/estadístico* es un área de estudio que busca brindar a las computadoras la capacidad de *aprender* sin ser explícitamente programadas.

"Aprender" más o menos significa mejorar el rendimiento en algún trabajo, bajo alguna medida adecuada, conforme el programa obtiene más "experiencia".

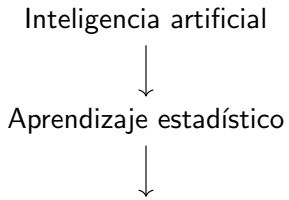
Introducción

El *aprendizaje de máquina/automático/autónomo/estadístico* es un área de estudio que busca brindar a las computadoras la capacidad de *aprender* sin ser explícitamente programadas.

"Aprender" más o menos significa mejorar el rendimiento en algún trabajo, bajo alguna medida adecuada, conforme el programa obtiene más "experiencia".

1. ¿Quién lo dijo? No recuerdo.

Introducción



Introducción
Inteligencia artificial
Aprendizaje estadístico
Redes neuronales / DL

Principios básicos del ML

Un algorítmo de aprendizaje estadístico es un algorítmo que es capaz de aprender a partir de un conjunto de datos.

Principios básicos del ML

Un algorítmo de aprendizaje estadístico es un algorítmo que es capaz de aprender a partir de un conjunto de datos.

Principios básicos del ML

Un algorítmo de aprendizaje estadístico es un algorítmo que es capaz de aprender a partir de un conjunto de datos.

El conjunto de datos consiste de *ejemplos* de características de algún objeto. El algorítmo debe describir/predecir alguna otra característica o valor de cada ejemplo. Matemáticamente, el algoritmo representa un mapeo entre los elementos de los datos y el valor a predecir.

Principios básicos del ML

Un algorítmo de aprendizaje estadístico es un algorítmo que es capaz de aprender a partir de un conjunto de datos.

El conjunto de datos consiste de ejemplos de características de algún objeto. El algorítmo debe describir/predecir alguna otra característica o valor de cada ejemplo. Matemáticamente, el algoritmo representa un mapeo entre los elementos de los datos y el valor a predecir.

Principios básicos del ML

Un algorítmo de aprendizaje estadístico es un algorítmo que es capaz de aprender a partir de un conjunto de datos.

El conjunto de datos consiste de *ejemplos* de características de algún objeto. El algorítmo debe describir/predecir alguna otra característica o valor de cada ejemplo. Matemáticamente, el algoritmo representa un mapeo entre los elementos de los datos y el valor a predecir.

Ejemplo

Los pixeles de una imágen, o las propiedades de una casa, secuencias ordenadas de palabras, etc. . .

Principios básicos del ML

Un algorítmo de aprendizaje estadístico es un algorítmo que es capaz de aprender a partir de un conjunto de datos.

El conjunto de datos consiste de ejemplos de características de algún objeto. El algorítmo debe describir/predecir alguna otra característica o valor de cada ejemplo. Matemáticamente, el algoritmo representa un mapeo entre los elementos de los datos y el valor a predecir.

Ejemplo

Los pixeles de una imágen, o las propiedades de una casa, secuencias ordenadas de palabras, etc. . .

Paradigmas de aprendizaje

Dependiendo del problema por resolver, existen distintas paradigmas de aprendizaje estadístico.

1. No supervisado
   - 1.1 Clustering
   - 1.2 Autocodificadores
   - 1.3 Modelos generativos (GAN, transformers, etc. . .)
2. Supervisado
   - 2.1 Clasificación
   - 2.2 Regresión
3. Refuerzo (control)

Paradigmas de aprendizaje

Dependiendo del problema por resolver, existen distintas paradigmas de aprendizaje estadístico.

1. No supervisado
   - 1.1 Clustering
   - 1.2 Autocodificadores
   - 1.3 Modelos generativos (GAN, transformers, etc. . .)
2. Supervisado
   - 2.1 Clasificación
   - 2.2 Regresión
3. Refuerzo (control)

Paradigmas de aprendizaje

Dependiendo del problema por resolver, existen distintas paradigmas de aprendizaje estadístico.

1. No supervisado
   - 1.1 Clustering
   - 1.2 Autocodificadores
   - 1.3 Modelos generativos (GAN, transformers, etc. . .)
2. Supervisado
   - 2.1 Clasificación
   - 2.2 Regresión
3. Refuerzo (control)

Nos enfocamos en el aprendizaje supervisado.

Paradigmas de aprendizaje

Dependiendo del problema por resolver, existen distintas paradigmas de aprendizaje estadístico.

1. No supervisado
   - 1.1 Clustering
   - 1.2 Autocodificadores
   - 1.3 Modelos generativos (GAN, transformers, etc. . .)
2. Supervisado
   - 2.1 Clasificación
   - 2.2 Regresión
3. Refuerzo (control)

Nos enfocamos en el aprendizaje supervisado.

Aprendizaje estadístico

Modelamos un elemento del conjunto de datos como un elemento de $\mathbb{R}^d$, donde la dimensión d es la cantidad de características del objeto.

Problema

Asumiendo que nuestros datos "provienen" de una distribución de probabilidad conjunta p_{datos}, deseamos encontrar una función f_θ que describe la relación entre el vector de características $\mathbf{x} \in \mathbb{R}^m$ y el vector a predicir $\mathbf{y} \in \mathbb{R}^p$.

Aprendizaje estadístico

Modelamos un elemento del conjunto de datos como un elemento de $\mathbb{R}^d$, donde la dimensión d es la cantidad de características del objeto.

Problema

Asumiendo que nuestros datos "provienen" de una distribución de probabilidad conjunta p_{datos}, deseamos encontrar una función f_θ que describe la relación entre el vector de características $\mathbf{x} \in \mathbb{R}^m$ y el vector a predicir $\mathbf{y} \in \mathbb{R}^p$.

1. Notemos que $d = m + p$ en ésta notación.
2. La distribución de los datos es de la forma

$$p(\mathbf{x}, \mathbf{y}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x}),$$

donde el "supervisor" nos da la probabilidad condicional $p(\mathbf{y}|\mathbf{x})$.

Aprendizaje estadístico

¿Cómo encontramos a tal función f_θ entre tantas?

Aprendizaje estadístico
¿Cómo encontramos a tal función f_θ entre tantas?

1. Vladimir Vapnik es el padre de la teoría del aprendizaje estadístico y creador de las MSV.
2. Su libro *Statistical Learning Theory* (1998) es muy recomendado pero no lo he leído.

Aprendizaje estadístico

¿Cómo encontramos a tal función f_θ entre tantas? Siguiendo los principios del aprendizaje estadístico (Vapnik), definimos una **función de error**

$$L : \mathbb{R}^{2p} \to \mathbb{R} \tag{1}$$
$$(f_\theta(\mathbf{x}), \mathbf{y}) \mapsto L(f_\theta(\mathbf{x}), \mathbf{y}), \tag{2}$$

que nos brinda una noción de las diferencias entre nuestra predicción $\hat{\mathbf{y}} = f_\theta(\mathbf{x})$ y el valor verdadero $\mathbf{y}$.

Aprendizaje estadístico

¿Cómo encontramos a tal función f_θ entre tantas? Siguiendo los principios del aprendizaje estadístico (Vapnik), definimos una **función de error**

$$L : \mathbb{R}^{2p} \to \mathbb{R} \tag{1}$$
$$(f_\theta(\mathbf{x}), \mathbf{y}) \mapsto L(f_\theta(\mathbf{x}), \mathbf{y}), \tag{2}$$

que nos brinda una noción de las diferencias entre nuestra predicción $\hat{\mathbf{y}} = f_\theta(\mathbf{x})$ y el valor verdadero $\mathbf{y}$.

1. Vladimir Vapnik es el padre de la teoría del aprendizaje estadístico y creador de las MSV.
2. Su libro *Statistical Learning Theory* (1998) es muy recomendado pero no lo he leído.

Riesgo

La teoría del aprendizaje estadístico utiliza a dicha función de error para definir el concepto de **riesgo**. El riesgo es un funcional respecto a los parámetros del modelo θ, dado por el valor esperado de la función de error L respecto a la distribución generadora de los datos:

$$R(\theta) = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{datos}}}\left[L(f_\theta(\mathbf{x}), \mathbf{y})\right] \tag{3}$$

$$= \int L(f_\theta(\mathbf{x}), \mathbf{y})\, dp_{\text{datos}}(\mathbf{x}, \mathbf{y}). \tag{4}$$

Riesgo
La teoría del aprendizaje estadístico utiliza a dicha función de error para definir el concepto de **riesgo**. El riesgo es un funcional respecto a los parámetros del modelo θ, dado por el valor esperado de la función de error L respecto a la distribución generadora de los datos:
$R(\theta) = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{datos}}}\left[L(f_\theta(\mathbf{x}), \mathbf{y})\right]$ (3)
$= \int L(f_\theta(\mathbf{x}), \mathbf{y})\, dp_{\text{datos}}(\mathbf{x}, \mathbf{y}).$ (4)

Riesgo

La teoría del aprendizaje estadístico utiliza a dicha función de error para definir el concepto de **riesgo**. El riesgo es un funcional respecto a los parámetros del modelo θ, dado por el valor esperado de la función de error L respecto a la distribución generadora de los datos:

$$R(\theta) = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{datos}}}\left[L(f_\theta(\mathbf{x}), \mathbf{y})\right] \tag{3}$$

$$= \int L(f_\theta(\mathbf{x}), \mathbf{y})\, dp_{\text{datos}}(\mathbf{x}, \mathbf{y}). \tag{4}$$

El objetivo es *minimizar* el riesgo.

Riesgo
La teoría del aprendizaje estadístico utiliza a dicha función de error para definir el concepto de **riesgo**. El riesgo es un funcional respecto a los parámetros del modelo θ, dado por el valor esperado de la función de error L respecto a la distribución generadora de los datos:
$R(\theta) = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{datos}}}\left[L(f_\theta(\mathbf{x}), \mathbf{y})\right]$ (3)
$= \int L(f_\theta(\mathbf{x}), \mathbf{y})\, dp_{\text{datos}}(\mathbf{x}, \mathbf{y}).$ (4)
El objetivo es *minimizar* el riesgo.

Riesgo

La teoría del aprendizaje estadístico utiliza a dicha función de error para definir el concepto de **riesgo**. El riesgo es un funcional respecto a los parámetros del modelo θ, dado por el valor esperado de la función de error L respecto a la distribución generadora de los datos:

$$R(\theta) = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{datos}}}\left[L(f_\theta(\mathbf{x}), \mathbf{y})\right] \tag{3}$$

$$= \int L(f_\theta(\mathbf{x}), \mathbf{y})\, dp_{\text{datos}}(\mathbf{x}, \mathbf{y}). \tag{4}$$

El objetivo es *minimizar* el riesgo.

Pero tenemos un **problema**: ¡no conocemos a p_{datos}! Solo tenemos acceso a una muestra de datos

$$\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) \sim p_{\text{datos}} \,|\, i = 0, \ldots, n\}. \tag{5}$$

Riesgo
La teoría del aprendizaje estadístico utiliza a dicha función de error para definir el concepto de **riesgo**. El riesgo es un funcional respecto a los parámetros del modelo θ, dado por el valor esperado de la función de error L respecto a la distribución generadora de los datos:
$R(\theta) = \mathbb{E}_{\mathbf{x},\mathbf{y}\sim p_{\text{datos}}}\left[L(f_\theta(\mathbf{x}), \mathbf{y})\right]$ (3)
$= \int L(f_\theta(\mathbf{x}), \mathbf{y})\, dp_{\text{datos}}(\mathbf{x}, \mathbf{y})$. (4)
El objetivo es *minimizar* el riesgo.
Pero tenemos un **problema**: ¡no conocemos a p_{datos}! Solo tenemos acceso a una muestra de datos
$\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) \sim p_{\text{datos}} \,|\, i = 0, \ldots, n\}$. (5)

Riesgo empírico

Lo que hacemos es aproximar el riesgo definiendo el **riesgo empírico** como

$$R_{\text{emp}}(\theta) = \frac{1}{n} \sum_{i=0}^{n} L(f_\theta(\mathbf{x}_i), \mathbf{y}_i). \tag{6}$$

Ahora buscamos minimizar a $J(\theta) := R_{\text{emp}}(\theta)$ respecto a los parámetros del modelo, i.e., obtenemos el siguiente problema:

Problema de optimización

$$\theta^* = \arg\min_\theta J(\theta), \quad \theta \in \mathbb{R}^q. \tag{7}$$

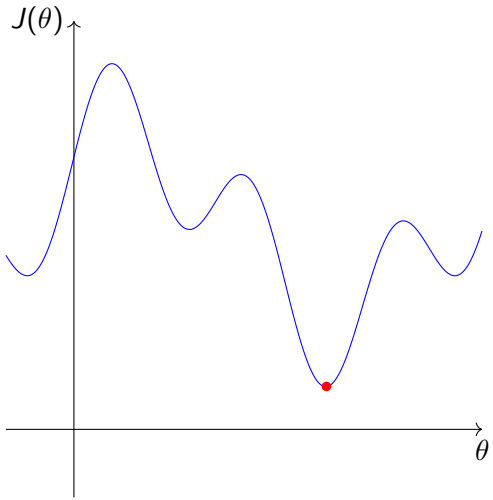
Riesgo empírico
Lo que hacemos es aproximar el riesgo definiendo el **riesgo empírico** como
$R_{emp}(\theta) = \frac{1}{n}\sum_{i=0}^{n} L(f_\theta(\mathbf{x}_i), \mathbf{y}_i).$ (6)
Ahora buscamos minimizar a $J(\theta) := R_{emp}(\theta)$ respecto a los parámetros del modelo, i.e., obtenemos el siguiente problema:
Problema de optimización
$\theta^* = \arg\min_\theta J(\theta), \quad \theta \in \mathbb{R}^q.$ (7)

1. El principio de inducción de la minimización del riesgo empírico asume que la función $f_{\theta*}$ que minimiza a $R(\theta)$ sobre el espacio de parámetros, nos proveé un riesgo empírico $R_{\text{emp}}(\theta)$ que es cercano al mínimo.
2. Los ejemplos de mínimos cuadrados o de máxima versomilitud son realizaciones de éste principio.
3. ¿Pero qué tan bueno es éste principio?
   - 3.1 ¿El principio es consistente? Es decir, ¿$R_{\text{emp}}(\theta)$ converge a al minimo cuando $n \to \infty$? Ésto es equivalente a preguntar si el riesgo empírico converge *uniformemente* al riesgo:

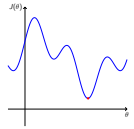
$$\text{Prob}\left\{\sup_\theta |R(\theta) - R_{\text{emp}}(\theta)| > \varepsilon\right\} \to 0, \quad \text{cuando} \quad n \to \infty.$$

   - 3.2 ¿Cómo es la tasa de convergencia?

Optimización



Optimización



1. ¿Qué pasa si la función es altamente no convexa?
2. Los mínimos locales son comunes en las redes neuronales?

Método de optimización

Los métodos análiticos del cálculo no son de gran ayuda, debido prinicipalmente a que:

1. Generalmente no existe una forma cerrada de la solución.
2. El tamaño de la dimensión del espacio de parámetros es muy grande.
3. El costo computacional es demasiado elevado.

Así que debemos utilizar métodos numéricos para obtener θ^*.

Método de optimización

Los métodos análiticos del cálculo no son de gran ayuda, debido prinicipalmente a que:

1. Generalmente no existe una forma cerrada de la solución.
2. El tamaño de la dimensión del espacio de parámetros es muy grande.
3. El costo computacional es demasiado elevado.

Así que debemos utilizar métodos numéricos para obtener θ^*.

1. Considere el caso de la regresión lineal simple. Utilizando el cálculo podemos obtener la solución al problema:

$$\beta^* = \arg\min_{\beta} \|X\beta - \mathbf{y}\|_2,$$

conocido como la ecuación normal:

$$\beta^* = \left(X^\top X\right)^{-1} X^\top Y.$$

Si la matriz tiene 100,000 columnas, requerimos guardar 10^{10} números, a 8 bytes por número ésto es 80gb de memoria, que además debe ser invertida!

Busqueda en linea

La clase de algoritmos de optimización más comunes son los de *busqueda en linea*. Para ésto es necesario elegir una dirección d_k en el espacio de parámetros y luego movernos de acuerdo a un *tamaño de paso* α_k. Éstos métodos son iterativos y siguen la regla básica de actualización:

$$\theta_{k+1} = \theta_k + \alpha_k \mathbf{p}_k, \quad k = 0, 1, 2, \ldots \tag{8}$$

El método más sencillo y más utilizado es el método del **descenso por gradiente**. Utilizamos la información de la derivada para obtener la dirección de la busqueda.

Busqueda en linea

La clase de algoritmos de optimización más comunes son los de *busqueda en linea*. Para ésto es necesario elegir una dirección d_k en el espacio de parámetros y luego movernos de acuerdo a un *tamaño de paso* α_k. Éstos métodos son iterativos y siguen la regla básica de actualización:

$$\theta_{k+1} = \theta_k + \alpha_k \mathbf{p}_k, \quad k = 0, 1, 2, \ldots \tag{8}$$

El método más sencillo y más utilizado es el método del **descenso por gradiente**. Utilizamos la información de la derivada para obtener la dirección de la busqueda.

Descenso por gradiente

Algorithm 1 Descenso por gradiente

$k \leftarrow 0$
while $J(\theta_k) >$ tol **do**
  Calcular $\nabla J(\theta_k)$ y α_k
  $\theta_{k+1} \leftarrow \theta_k - \alpha_k \nabla J(\theta_k)$
  $k \leftarrow k + 1$
end while

Nota

El uso de derivadas de orden mayor puede servir, pero requiere un costo computacional mayor.

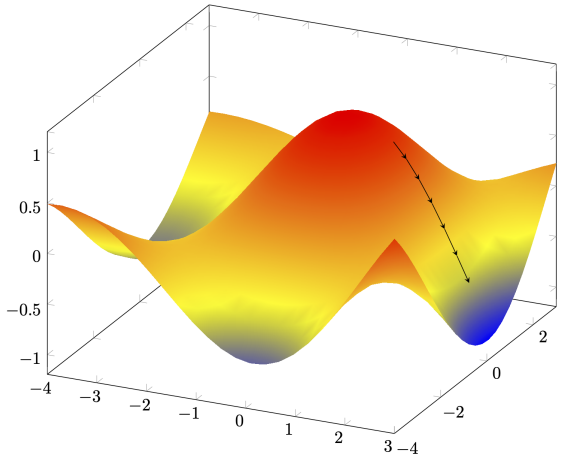
Descenso por gradiente

Algorithm 1 Descenso por gradiente
$k \leftarrow 0$
while $J(\theta_k) >$ tol **do**
Calcular $\nabla J(\theta_k)$ y α_k
$\theta_{k+1} \leftarrow \theta_k - \alpha_k \nabla J(\theta_k)$
$k \leftarrow k + 1$
end while

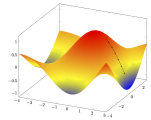
Nota
El uso de derivadas de orden mayor puede servir, pero requiere un costo computacional mayor.

1. El uso del Hessiano H es utilizado en ciertos problemas, pero en el deep learning no es muy común por la complejidad computacional incurrida al manipular a H.
2. Las condiciones de Wolfe nos permiten encontrar, de manera eficiente, al tamaño de paso α_k aceptable, que reduce la función a minimizar "suficientemente". Las condiciones son:
   2.1 $f(\mathbf{x}_k + \alpha \mathbf{p}_k) \leq f(\mathbf{x}_k) + c_1 \alpha_k \mathbf{p}_k^\top \nabla f(\mathbf{x}_k)$,
   2.2 $-\mathbf{p}_k^\top \nabla f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \leq -c_2 \mathbf{p}_k^\top \nabla f(\mathbf{x}_k)$,
   donde $0 < c_1 < c_2 < 1$. No es muy común usarlas por el costo computacional.

Descenso por gradiente



Descenso por gradiente



1. La función de costo de las redes neuronales son altamente no convexas.

Optimización con derivadas

Existen muchos métodos de optimización que utilizan las derivadas, el uso de alguno en particular depende del problema en cuestión.

1. SGD (primer orden)
2. ADAM (primer orden)
3. L-BFGS (segundo orden)
4. etc. . .

Optimización con derivadas

Existen muchos métodos de optimización que utilizan las derivadas, el uso de alguno en particular depende del problema en cuestión.
1. SGD (primer orden)
2. ADAM (primer orden)
3. L-BFGS (segundo orden)
4. etc. . .

1. Para evitar el problema de los minimos locales, generalmente se utiliza alguna variante del descenso estocástico. Ésto involucra simplemente calcular el gradiente con una pequeña muestra de los datos, elegidos de manera aleatoria.

Redes neuronales

Las redes neuronales son modelos ligeramente inspiradas en la biología. El modelo básico, **feed forward network**, consiste de una composición de muchas funciones más básicas, que llamaremos **neuronas** $f_j^{(i)}$, de la forma

$$f_\theta(\mathbf{x}) = f^{(L)}(f^{(L-1)}(\cdots f^{(2)}(f^{(1)}(\mathbf{x}))\cdots)), \tag{9}$$

donde a las funciones $f^{(i)}$ se les conoce como **capas**.

Por convención denotemos al índice de la última capa por L, y no consideramos al vector de características como una capa.

Redes neuronales

Las redes neuronales son modelos ligeramente inspiradas en la biología. El modelo básico, **feed forward network**, consiste de una composición de muchas funciones más básicas, que llamaremos **neuronas** $f_j^{(i)}$, de la forma

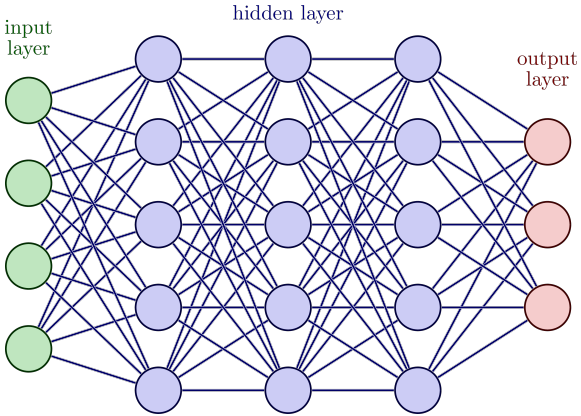
$$f_\theta(\mathbf{x}) = f^{(L)}(f^{(L-1)}(\cdots f^{(2)}(f^{(1)}(\mathbf{x}))\cdots)), \tag{9}$$

donde a las funciones $f^{(i)}$ se les conoce como **capas**.

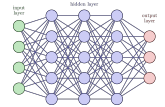
Por convención denotemos al índice de la última capa por L, y no consideramos al vector de características como una capa.

Redes neuronales

Se les dice *redes* porque generalmente se les asocia un gráfo que dicta la composición.



Redes neuronales
Se les dice *redes* porque generalmente se les asocia un gráfo que dicta la composición.



Representaciones internas

¿Cuál es la razón del éxito de las redes neuronales?

Representaciones internas
¿Cuál es la razón del éxito de las redes neuronales?

1. El término parace venir del artículo que reinició la investigación en redes neuronales, *Learning representations by back-propagating errors* (1986) por Rumelhart y Hinton.
2. La idea de las representaciones aprendidas es que transforman los datos en formas más refinadas que son útiles para resolver el problema subyacente.
3. No basta que las capas $f^{(i)}$ sean transformaciones lineales, es necesario introducir funciones no lineales para capturar la complejidad de los datos.

Representaciones internas

¿Cuál es la razón del éxito de las redes neuronales? Se cree que se debe al aprendizaje de representaciones distintas (internas) de los datos.

$$\underbrace{\mathbb{R}^m}_{\text{entrada}} \to \underbrace{\mathbb{R}^{k_1} \to \mathbb{R}^{k_2} \to \cdots}_{\text{internas}} \to \underbrace{\mathbb{R}^{k_n}}_{\text{salida}}. \tag{10}$$

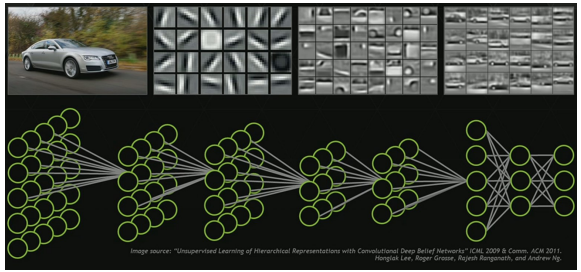
Representaciones internas
¿Cuál es la razón del éxito de las redes neuronales? Se cree que se debe al aprendizaje de representaciones distintas (internas) de los datos.
$\underbrace{\mathbb{R}^m}_{\text{entrada}} \to \underbrace{\mathbb{R}^{k_1} \to \mathbb{R}^{k_2} \to \cdots}_{\text{internas}} \to \underbrace{\mathbb{R}^{k_n}}_{\text{salida}}$ (10)

1. El término parace venir del artículo que reinició la investigación en redes neuronales, *Learning representations by back-propagating errors* (1986) por Rumelhart y Hinton.
2. La idea de las representaciones aprendidas es que transforman los datos en formas más refinadas que son útiles para resolver el problema subyacente.
3. No basta que las capas $f^{(i)}$ sean transformaciones lineales, es necesario introducir funciones no lineales para capturar la complejidad de los datos.

Representaciones internas

¿Cuál es la razón del éxito de las redes neuronales? Se cree que se debe al aprendizaje de representaciones distintas (internas) de los datos.

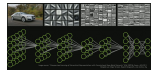
$$\underbrace{\mathbb{R}^m}_{\text{entrada}} \to \underbrace{\mathbb{R}^{k_1} \to \mathbb{R}^{k_2} \to \cdots}_{\text{internas}} \to \underbrace{\mathbb{R}^{k_n}}_{\text{salida}}. \tag{10}$$



Representaciones internas

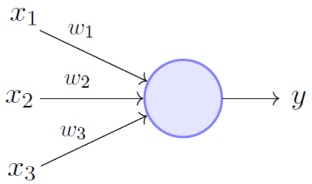
¿Cuál es la razón del éxito de las redes neuronales? Se cree que se debe al aprendizaje de representaciones distintas (internas) de los datos.

$$\underbrace{\mathbb{R}^m}_{\text{entrada}} \to \underbrace{\mathbb{R}^{k_1} \to \mathbb{R}^{k_2} \to \cdots}_{\text{internas}} \to \underbrace{\mathbb{R}^{k_n}}_{\text{salida}} \tag{10}$$



1. El término parace venir del artículo que reinició la investigación en redes neuronales, *Learning representations by back-propagating errors* (1986) por Rumelhart y Hinton.
2. La idea de las representaciones aprendidas es que transforman los datos en formas más refinadas que son útiles para resolver el problema subyacente.
3. No basta que las capas $f^{(i)}$ sean transformaciones lineales, es necesario introducir funciones no lineales para capturar la complejidad de los datos.

Neurona



Perceptron Model (Minsky-Papert in 1969)

Una neurona $a_j^{(i)}$ es un *nodo* en la i-ésima capa. Cada neurona está ligada con las neuronas de la capa anterior $\mathbf{a}^{(i-1)}$ por medio de un producto interno con un vector de conexiones $\mathbf{w}_j^{(i)}$, seguido por una función no lineal.

Neurona
Perceptron Model (Minsky-Papert in 1969)
Una neurona $a_j^{(i)}$ es un *nodo* en la i-ésima capa. Cada neurona está ligada con las neuronas de la capa anterior $\mathbf{a}^{(i-1)}$ por medio de un producto interno con un vector de conexiones $\mathbf{w}_j^{(i)}$, seguido por una función no lineal.

Neurona

El valor de la neurona se calcula como:

$$a_j^{(i)} = f_j^{(i)}\left(\mathbf{a}^{(i-1)}\right) = \sigma\left(\sum_r w_{jr}^{(i)} a_r^{(i-1)} + b_j^{(i)}\right), \qquad (11)$$

donde $\mathbf{b}^{(i)}$ es una constante adicional llamada *sesgo* y σ es la función no lineal.

Neurona
El valor de la neurona se calcula como:
$$a_j^{(i)} = f_j^{(i)}\left(\mathbf{a}^{(i-1)}\right) = \sigma\left(\sum_r w_{jr}^{(i)} a_r^{(i-1)} + b_j^{(i)}\right), \qquad (11)$$
donde $\mathbf{b}^{(i)}$ es una constante adicional llamada sesgo y σ es la función no lineal.

1. La función ReLU no es diferenciable en 0, entonces ¿como la podemos utilizar con un algoritmo de optimización que utiliza la derivada?

   La respuesta es que por cuestiones numéricas, casi nunca es necesario evaluar la derivada en *exactamente* 0. De hecho, maquinas como `autograd` de *PyTorch* utilizan sub ó sup derivadas, en éstos casos extraños.

Neurona

El valor de la neurona se calcula como:

$$a_j^{(i)} = f_j^{(i)}\left(\mathbf{a}^{(i-1)}\right) = \sigma\left(\sum_r w_{jr}^{(i)} a_r^{(i-1)} + b_j^{(i)}\right), \qquad (11)$$

donde $\mathbf{b}^{(i)}$ es una constante adicional llamada *sesgo* y σ es la función no lineal.

Necesitamos calcular la derivada de $J(\theta)$, por lo tanto la función σ debe ser diferenciable (no es cierto). Generalmente se consideran funciones sigmoidales, e.g. la función tanh ó

$$\sigma(t) = \frac{1}{1+e^{-t}}. \qquad (12)$$

Neurona
El valor de la neurona se calcula como:
$$a_j^{(i)} = f_j^{(i)}\left(\mathbf{a}^{(i-1)}\right) = \sigma\left(\sum_r w_{jr}^{(i)} a_r^{(i-1)} + b_j^{(i)}\right), \qquad (11)$$
donde $\mathbf{b}^{(i)}$ es una constante adicional llamada sesgo y σ es la función no lineal.
Necesitamos calcular la derivada de $J(\theta)$, por lo tanto la función σ debe ser diferenciable (no es cierto). Generalmente se consideran funciones sigmoidales, e.g. la función tanh ó
$$\sigma(t) = \frac{1}{1+e^{-t}}. \qquad (12)$$

1. La función ReLU no es diferenciable en 0, entonces ¿como la podemos utilizar con un algoritmo de optimización que utiliza la derivada?

   La respuesta es que por cuestiones numéricas, casi nunca es necesario evaluar la derivada en *exactamente* 0. De hecho, maquinas como `autograd` de *PyTorch* utilizan sub ó sup derivadas, en éstos casos extraños.

Capa interna

Vectorialmente, podemos expresar el cálculo de todas las neuronas de una capa de la siguiente manera:

$$\mathbf{a}^{(i)} = f^{(i)}(\mathbf{a}^{(i-1)}) = \sigma\left(W^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)}\right), \tag{13}$$

donde W es la matriz de conexiones, $\mathbf{b}^{(i)}$ es el vector de sesgos y σ es la función de activación que se aplica elemento por elemento.

La red neuronal simplemente consiste de la composición de muchas de éstas capas:

$$f_\theta(\mathbf{x}) = \sigma\left(W^{(L)}\sigma\left(W^{(L-1)}\sigma\left(\cdots\right) + b^{(L-1)}\right) + \mathbf{b}^{(L)}\right). \tag{14}$$

Capa interna

Vectorialmente, podemos expresar el cálculo de todas las neuronas de una capa de la siguiente manera:

$$\mathbf{a}^{(i)} = f^{(i)}(\mathbf{a}^{(i-1)}) = \sigma\left(W^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)}\right), \quad (13)$$

donde W es la matriz de conexiones, $\mathbf{b}^{(i)}$ es el vector de sesgos y σ es la función de activación que se aplica elemento por elemento.

La red neuronal simplemente consiste de la composición de muchas de éstas capas:

$$f_\theta(\mathbf{x}) = \sigma\left(W^{(L)}\sigma\left(W^{(L-1)}\sigma\left(\cdots\right) + b^{(L-1)}\right) + \mathbf{b}^{(L)}\right). \quad (14)$$

1. La matriz de conexiones W consiste simplemente de los vectores de conexión $\mathbf{w}_j^{(i)}$ para la j-ésima neurona apilados como filas:

$$W^{(i)} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1\ell} \\ \vdots & \vdots & \ddots & \vdots \\ w_{j1} & w_{j2} & \cdots & w_{j\ell} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k1} & w_{k2} & \cdots & w_{k\ell} \end{pmatrix},$$

donde k es el número de neuronas de la capa i y ℓ es la cantidad de neuronas de la capa anterior $i-1$.

Capa de salida

La capa de salida $f^{(L)}$ no necesariamente termina con una aplicación de una función de activación.

Depende totalmente del problema, por ejemplo:

1. en el caso de la clasificación multi-clase, generalmente se utiliza la función *softmax*,
2. en el caso de regresión generalmente se utiliza la identidad.

Capa de salida

La capa de salida $f^{(L)}$ no necesariamente termina con una aplicación de una función de activación.

Depende totalmente del problema, por ejemplo:
1. en el caso de la clasificación multi-clase, generalmente se utiliza la función softmax,
2. en el caso de regresión generalmente se utiliza la identidad.

1. La función softmax es una función que toma un vector y lo normaliza de tal modo que las clases pueden ser interpretadas como probabilidades:

$$\mathcal{S} : \mathbb{R}^u \to (0,1)^u,$$

donde u es el número de clases y

$$\mathcal{S}\left(\mathbf{a}^{(L)}\right)_i = \frac{e^{a_i^{(L)}}}{\sum_{j=1}^{u} e^{a_j^{(L)}}}.$$

DL

El deep learning no es nada más que el uso de redes neuronales con muchas capas/neuronas, i.e., con un espacio de parametros de dimensión muy grande.

1. Redes densas / MLP,
2. Redes convolucionales,
3. Transformadores, etc. . .

DL

El deep learning no es nada más que el uso de redes neuronales con muchas capas/neuronas, i.e., con un espacio de parametros de dimensión muy grande.

1. Redes densas / MLP,
2. Redes convolucionales,
3. Transformadores, etc. . .

1. Se estima que OpenAI (Sam Altman lo dijo) gastó más de $100 millones de dólares en el entrenamiento de GTP-4!

   Por eso hay que apoyar a los jugadores pequeños como TinyCorp, a pesar de que Hotz esté loquito.

DL

El deep learning no es nada más que el uso de redes neuronales con muchas capas/neuronas, i.e., con un espacio de parametros de dimensión muy grande.

1. Redes densas / MLP,
2. Redes convolucionales,
3. Transformadores, etc. . .

¿Qué significa *muy grande*?

DL

El deep learning no es nada más que el uso de redes neuronales con muchas capas/neuronas, i.e., con un espacio de parametros de dimensión muy grande.

1. Redes densas / MLP,
2. Redes convolucionales,
3. Transformadores, etc. . .

¿Qué significa *muy grande*?

1. Se estima que OpenAI (Sam Altman lo dijo) gastó más de $100 millones de dólares en el entrenamiento de GTP-4!

   Por eso hay que apoyar a los jugadores pequeños como TinyCorp, a pesar de que Hotz esté loquito.

DL

El deep learning no es nada más que el uso de redes neuronales con muchas capas/neuronas, i.e., con un espacio de parametros de dimensión muy grande.

1. Redes densas / MLP,
2. Redes convolucionales,
3. Transformadores, etc. . .

¿Qué significa *muy grande*?

1. AlexNet (2012) $\sim$ 62 millones
2. LLaMA 2 $\sim$ 70 mil millones
3. GPT-4 $\sim$ 1.76 billones

DL

El deep learning no es nada más que el uso de redes neuronales con muchas capas/neuronas, i.e., con un espacio de parametros de dimensión muy grande.

1. Redes densas / MLP,
2. Redes convolucionales,
3. Transformadores, etc. . .

¿Qué significa *muy grande*?

1. AlexNet (2012) $\sim$ 62 millones
2. LLaMA 2 $\sim$ 70 mil millones
3. GPT-4 $\sim$ 1.76 billones

1. Se estima que OpenAI (Sam Altman lo dijo) gastó más de \$100 millones de dólares en el entrenamiento de GTP-4!

   Por eso hay que apoyar a los jugadores pequeños como TinyCorp, a pesar de que Hotz esté loquito.

Función de error

Recordando los principios del aprendizaje estadístico, necesitamos una manera de medir el rendimiento de la red neuronal, i.e., debemos elegir una función de error L.

1. Para problemas de regresión es muy común usar los *errores cuadrados* (MSE):

$$L(f_\theta(\mathbf{x}), \mathbf{y}) = \frac{1}{2}\|f_\theta(\mathbf{x}) - \mathbf{y}\|^2. \tag{15}$$

2. Para problemas de clasificación es común usar la *entropía cruzada*:

$$H(p, q) = -\sum_i p_i \log q_i. \tag{16}$$

Función de error

Recordando los principios del aprendizaje estadístico, necesitamos una manera de medir el rendimiento de la red neuronal, i.e., debemos elegir una función de error L.

1. Para problemas de regresión es muy común usar los *errores cuadrados* (MSE):

$$L(f_\theta(\mathbf{x}), \mathbf{y}) = \frac{1}{2}\|f_\theta(\mathbf{x}) - \mathbf{y}\|^2. \tag{15}$$

2. Para problemas de clasificación es común usar la *entropía cruzada*:

$$H(p, q) = -\sum_i p_i \log q_i. \tag{16}$$

1. La entropía cruzada de una distribución q relativa a una distribución p sobre un conjunto se define formalmente como:

$$H(p, q) = -\mathbb{E}_p[\log q].$$

2. Con ésto concluímos lo que Edgar y sus colegas llaman el *problema directo*. El nombre de feed forward net se debe a que la información de los datos y las representaciones siguientes *fluyen hacia adelante* de manera secuencial en el grafo de la red.

   El problema siguiente es la optimización de la red, en donde nos enfocamos en la noción del flujo *al revés*.

Optimización de la red

Con la función de error elegida, de nuevo buscamos minimizar el riesgo empírico. En éste caso los parámetros del modelo constituyen el conjunto de las conexiones y de los sesgos:

$$\theta = \{W^{(0)}, W^{(1)}, \ldots, W^{(L)}\} \cup \{\mathbf{b}^{(0)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}. \qquad (17)$$

Para minimizar a $J(\theta)$ utilizamos algún variante del descenso por gradiente pertinente al problema.

Optimización de la red

Con la función de error elegida, de nuevo buscamos minimizar el riesgo empírico. En éste caso los parámetros del modelo constituyen el conjunto de las conexiones y de los sesgos:

$$\theta = \{W^{(0)}, W^{(1)}, \ldots, W^{(L)}\} \cup \{\mathbf{b}^{(0)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}. \qquad (17)$$

Para minimizar a $J(\theta)$ utilizamos algún variante del descenso por gradiente pertinente al problema.

Optimización de la red

Con la función de error elegida, de nuevo buscamos minimizar el riesgo empírico. En éste caso los parámetros del modelo constituyen el conjunto de las conexiones y de los sesgos:

$$\theta = \{W^{(0)}, W^{(1)}, \ldots, W^{(L)}\} \cup \{\mathbf{b}^{(0)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}. \qquad (17)$$

Para minimizar a $J(\theta)$ utilizamos algún variante del descenso por gradiente pertinente al problema.

Problema:

Optimización de la red

Con la función de error elegida, de nuevo buscamos minimizar el riesgo empírico. En éste caso los parámetros del modelo constituyen el conjunto de las conexiones y de los sesgos:

$$\theta = \{W^{(0)}, W^{(1)}, \ldots, W^{(L)}\} \cup \{\mathbf{b}^{(0)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}. \qquad (17)$$

Para minimizar a $J(\theta)$ utilizamos algún variante del descenso por gradiente pertinente al problema.

Problema:

Optimización de la red

Con la función de error elegida, de nuevo buscamos minimizar el riesgo empírico. En éste caso los parámetros del modelo constituyen el conjunto de las conexiones y de los sesgos:

$$\theta = \{W^{(0)}, W^{(1)}, \ldots, W^{(L)}\} \cup \{\mathbf{b}^{(0)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}. \qquad (17)$$

Para minimizar a $J(\theta)$ utilizamos algún variante del descenso por gradiente pertinente al problema.

Problema: ¿Cómo calculamos las derivadas parciales de f_θ respecto a los parámetros?

Optimización de la red

Con la función de error elegida, de nuevo buscamos minimizar el riesgo empírico. En éste caso los parámetros del modelo constituyen el conjunto de las conexiones y de los sesgos:

$$\theta = \{W^{(0)}, W^{(1)}, \ldots, W^{(L)}\} \cup \{\mathbf{b}^{(0)}, \mathbf{b}^{(1)}, \ldots, \mathbf{b}^{(L)}\}. \qquad (17)$$

Para minimizar a $J(\theta)$ utilizamos algún variante del descenso por gradiente pertinente al problema.

Problema: ¿Cómo calculamos las derivadas parciales de f_θ respecto a los parámetros?

Optimización de la red

No es factible calcular el gradiente simbolicamente por la cantidad de parámetros y las relaciones altamente no lineales, ni con un CAS.

Optimización de la red

No es factible calcular el gradiente simbolicamente por la cantidad de parámetros y las relaciones altamente no lineales, ni con un CAS.

1. Yo personalmente nunca he verificado que los CAS modernos realmente no puedan con el cálculo simbólico del gradiente de una red pequeña.
2. Probablemente sí se puede entrenar con derivadas numéricas, pero no se hace porque es más eficiente usar la diferenciación automática.

Optimización de la red

No es factible calcular el gradiente simbolicamente por la cantidad de parámetros y las relaciones altamente no lineales, ni con un CAS.

Tampoco podemos calcular el gradiente numéricamente por la inestabilidad numérica de los métodos de optimización.

Optimización de la red

No es factible calcular el gradiente simbolicamente por la cantidad de parámetros y las relaciones altamente no lineales, ni con un CAS.

Tampoco podemos calcular el gradiente numéricamente por la inestabilidad numérica de los métodos de optimización.

1. Yo personalmente nunca he verificado que los CAS modernos realmente no puedan con el cálculo simbólico del gradiente de una red pequeña.
2. Probablemente sí se puede entrenar con derivadas numéricas, pero no se hace porque es más eficiente usar la diferenciación automática.

Optimización de la red

No es factible calcular el gradiente simbolicamente por la cantidad de parámetros y las relaciones altamente no lineales, ni con un CAS.

Tampoco podemos calcular el gradiente numéricamente por la inestabilidad numérica de los métodos de optimización.

Solución: existe un método de calcular todas las derivadas parciales necesarias de manera directa y eficiente.

Optimización de la red

No es factible calcular el gradiente simbolicamente por la cantidad de parámetros y las relaciones altamente no lineales, ni con un CAS.

Tampoco podemos calcular el gradiente numéricamente por la inestabilidad numérica de los métodos de optimización.

Solución: existe un método de calcular todas las derivadas parciales necesarias de manera directa y eficiente.

1. Yo personalmente nunca he verificado que los CAS modernos realmente no puedan con el cálculo simbólico del gradiente de una red pequeña.
2. Probablemente sí se puede entrenar con derivadas numéricas, pero no se hace porque es más eficiente usar la diferenciación automática.

Backpropagation

El algoritmo de la **propagación hacia atrás** es un método que calcula las derivadas parciales de la función de costo $J(\theta)$ de manera iterativa y eficiente.

Es un caso particular de la **diferenciación automática**, el cual es un método que cálcula la derivada de un programa, aplicando la regla de la cadena a una secuencia de operaciones básicas.

Backpropagation

El algoritmo de la **propagación hacia atrás** es un método que calcula las derivadas parciales de la función de costo $J(\theta)$ de manera iterativa y eficiente.

Es un caso particular de la **diferenciación automática**, el cual es un método que cálcula la derivada de un programa, aplicando la regla de la cadena a una secuencia de operaciones básicas.

1. El caso particular de la diferenciación automática es el *modo en reverso*.

Backpropagation

En el contexto de las redes neuronales, la idea básica de la propagación hacia atrás es la transimisión de los errores de f_θ a todos los nodos de la red.

La parte interesante del algoritmo es como propagar la información de los errores (y esto tiene implicaciones prácticas), pero matematicamente lo único que hacemos es usar la regla de la cadena para obtener las derivadas parciales $\nabla_{W^{(k)}} J$ y $\nabla_{\mathbf{b}^{(k)}} J$.

Backpropagation

En el contexto de las redes neuronales, la idea básica de la propagación hacia atrás es la transimisión de los errores de f_θ a todos los nodos de la red.

La parte interesante del algoritmo es como propagar la información de los errores (y esto tiene implicaciones prácticas), pero matematicamente lo único que hacemos es usar la regla de la cadena para obtener las derivadas parciales $\nabla_{W^{(k)}} J$ y $\nabla_{\mathbf{b}^{(k)}} J$.

1. Intuitivamente tenemos:
   - 1.1 Calculamos el error de la capa de salida, δ_n.
   - 1.2 Calculamos el error respecto a un capa interna, δ_k utilizando el error δ_{k+1}.
   - 1.3 Obtenemos la derivada parcial respecto a cualquier parámetro utilizando el error correspondiente.
   - 1.4 Repetimos éste proceso propagando los "errores" hacia todas las capas hasta llegar a la primera capa interna.

Backpropogation

Utilizando la definición de la red y la regla de la cadena podemos obtener las siguientes ecuaciones explícitas y vectoriales.

Ecuaciones de backprop

Sean $\mathbf{z}^{(i)} = W^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)}$, de manera que $\mathbf{a}^{(i)} = \sigma\left(\mathbf{z}^{(i)}\right)$, entonces

1. $\delta_L = \sigma'(\mathbf{z}^{(n)}) \odot (\mathbf{a}^{(L)} - \mathbf{y})$,
2. $\delta_k = \sigma'(\mathbf{z}^{(k)}) \odot (W^{(k+1)})^\top \delta_{k+1}$,
3. $\nabla_{\mathbf{b}^{(k)}} J(\theta) = \delta_k$,
4. $\nabla_{W^{(k)}} J(\theta) = \delta_k (\mathbf{a}^{(k)})^\top$.

Backpropogation

Utilizando la definición de la red y la regla de la cadena podemos obtener las siguientes ecuaciones explícitas y vectoriales.

Ecuaciones de backprop

Sean $\mathbf{z}^{(i)} = W^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)}$, de manera que $\mathbf{a}^{(i)} = \sigma\left(\mathbf{z}^{(i)}\right)$, entonces

1. $\delta_L = \sigma'(\mathbf{z}^{(n)}) \odot (\mathbf{a}^{(L)} - \mathbf{y})$,
2. $\delta_k = \sigma'(\mathbf{z}^{(k)}) \odot (W^{(k+1)})^\top \delta_{k+1}$,
3. $\nabla_{\mathbf{b}^{(k)}} J(\theta) = \delta_k$,
4. $\nabla_{W^{(k)}} J(\theta) = \delta_k (\mathbf{a}^{(k)})^\top$.

1. Ésta versión es particular para el caso de la regresión con la función de error MSE.
2. La forma de las ecuaciones nos permite una programación particularmente útil del algoritmo.
3. El producto $\odot$ se conoce como el producto de Hadamard, pero no es nada más que el producto de $\mathbb{R}$ elemento por elemento.

Backpropogation

Utilizando la definición de la red y la regla de la cadena podemos obtener las siguientes ecuaciones explícitas y vectoriales.

Ecuaciones de backprop

Sean $\mathbf{z}^{(i)} = W^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)}$, de manera que $\mathbf{a}^{(i)} = \sigma\left(\mathbf{z}^{(i)}\right)$, entonces

1. $\delta_L = \sigma'(\mathbf{z}^{(n)}) \odot (\mathbf{a}^{(L)} - \mathbf{y})$,
2. $\delta_k = \sigma'(\mathbf{z}^{(k)}) \odot (W^{(k+1)})^\top \delta_{k+1}$,
3. $\nabla_{\mathbf{b}^{(k)}} J(\theta) = \delta_k$,
4. $\nabla_{W^{(k)}} J(\theta) = \delta_k (\mathbf{a}^{(k)})^\top$.

En la vida real no se programa así...

Backpropogation

Utilizando la definición de la red y la regla de la cadena podemos obtener las siguientes ecuaciones explícitas y vectoriales.

Ecuaciones de backprop

Sean $\mathbf{z}^{(i)} = W^{(i)}\mathbf{a}^{(i-1)} + \mathbf{b}^{(i)}$, de manera que $\mathbf{a}^{(i)} = \sigma\left(\mathbf{z}^{(i)}\right)$, entonces

1. $\delta_L = \sigma'(\mathbf{z}^{(n)}) \odot (\mathbf{a}^{(L)} - \mathbf{y})$,
2. $\delta_k = \sigma'(\mathbf{z}^{(k)}) \odot (W^{(k+1)})^\top \delta_{k+1}$,
3. $\nabla_{\mathbf{b}^{(k)}} J(\theta) = \delta_k$,
4. $\nabla_{W^{(k)}} J(\theta) = \delta_k (\mathbf{a}^{(k)})^\top$.

En la vida real no se programa así...

1. Ésta versión es particular para el caso de la regresión con la función de error MSE.
2. La forma de las ecuaciones nos permite una programación particularmente útil del algoritmo.
3. El producto $\odot$ se conoce como el producto de Hadamard, pero no es nada más que el producto de $\mathbb{R}$ elemento por elemento.

Diferenciación automática

blah blah

Diferenciación automática

blah blah

Implementación

blah blah

Implementación

blah blah

Bibliografía

Es difícil aprender sin programar un modelo, pero...

1. Deep Learning. Goodfellow, Bengio, Courville
2. Pattern Recognition. Bishop
3. Los videos de 3Blue1Brown
4. Los videos de Andrej Karpathy

Bibliografía

Es difícil aprender sin programar un modelo, pero...

1. Deep Learning. Goodfellow, Bengio, Courville
2. Pattern Recognition. Bishop
3. Los videos de 3Blue1Brown
4. Los videos de Andrej Karpathy

Ideas para presentar en el club

1. El concepto de generalización y el paradigma del train/test.
2. Explicar un método de optimización.
3. ¿Por qué funciona tan bien el SGD?
4. ¿El éxito de las grandes redes invalida la teoría estadística del aprendizaje?
5. ¿Cómo funciona una red convolucional?
6. ¿Cómo funciona un autocodificador?
7. ¿Cómo funciona un transformador?
8. Programa un GPT chiquito.
9. ¿Cómo funcionan las GAN?
10. Tensorflow, Jax, etc...
11. Otros modelos de ML como kNN, SVM, random forests...
12. Participa en un challenge de Kaggle y muestra tu experiencia y resultados.

Ideas para presentar en el club
1. El concepto de generalización y el paradigma del train/test.
2. Explicar un método de optimización.
3. ¿Por qué funciona tan bien el SGD?
4. ¿El éxito de las grandes redes invalida la teoría estadística del aprendizaje?
5. ¿Cómo funciona una red convolucional?
6. ¿Cómo funciona un autocodificador?
7. ¿Cómo funciona un transformador?
8. Programa un GPT chiquito.
9. ¿Cómo funcionan las GAN?
10. Tensorflow, Jax, etc...
11. Otros modelos de ML como kNN, SVM, random forests...
12. Participa en un challenge de Kaggle y muestra tu experiencia y resultados.

1. Lean *La última pregunta* de Asimov y diganme que Multivac no les recuerda a chatGPT jaja.