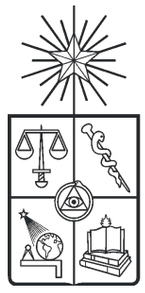


dcc
CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Regresiones

Modelos Lineales y Redes Neuronales

Felipe Bravo

Modelos de Regresión

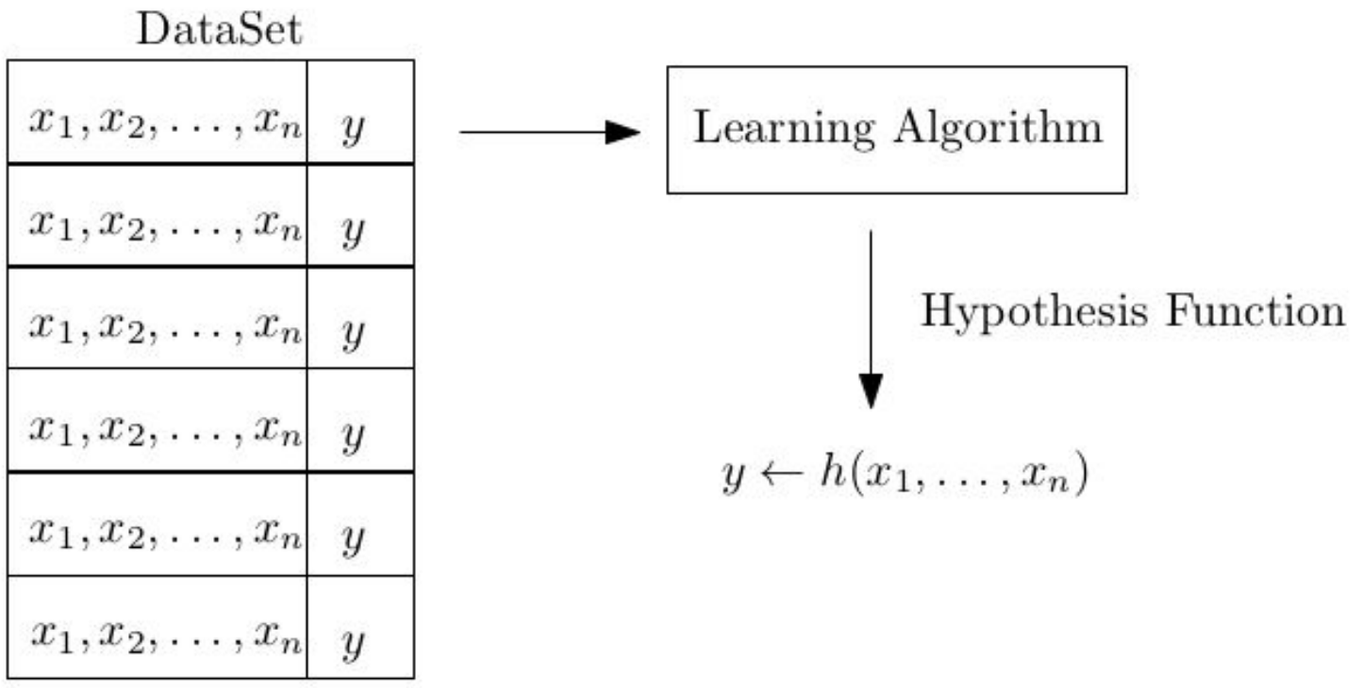
- Un modelo de regresión se usa para modelar la relación de una variable dependiente **y** numérica con n variables independientes x_1, x_2, ..., x_n .
- A grandes rasgos queremos conocer el valor **esperado** de **y** a partir los valores de **x**:

$$\mathbb{E}(y|x_1, x_2, \ldots, x_n)$$

- Usamos estos modelos cuando creemos que la variable de respuesta **y** puede ser modelada por otras variables independientes también conocidas como covariables o atributos.
- Para realizar este tipo de análisis necesitamos un dataset formado por **m** observaciones que incluyan tanto a la variable de respuesta como a cada uno de los atributos.
- Nos referimos al proceso de **ajustar** una función de regresión al proceso en que a partir de los datos inferimos una función de hipótesis **h** que nos permite predecir valores de **y** desconocidos usando los valores de los atributos.

Modelos de Regresión

- A este proceso de ajustar una función a partir de los datos se le llama en machine learning como **entrenamiento**.
- Se entiende que las funciones **aprenden** a partir de los datos.
- Como necesitamos observaciones donde el valor de **y** sea conocido para aprender la función, se le llama a este tipo de técnicas como técnicas de **aprendizaje supervisado**.
- Cuando **y** es una variable categórica hablamos de un problema de **clasificación**.



Modelos de Regresión

- En la regresión lineal simple se tiene una única variable independiente **x** para modelar la variable dependiente **y**.
- Se asume la siguiente relación lineal entre la variables:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \quad \forall i$$

- El parámetro β_0 representa el intercepto de la recta (el valor de **y** cuando **x** vale cero).
- El parámetro β_1 es la pendiente y representa el cambio de **y** cuando variamos el valor de **x**. Entre mayor sea la magnitud de este parámetro mayor será la relación lineal entre las variables.
- Los valores $\boldsymbol{\varepsilon_i}$ corresponden a los errores asociados al modelo.
- Tenemos que encontrar una función lineal o recta $\mathbf{h_\beta}$ que nos permita encontrar una estimación de **y**, $\hat{\mathbf{y}}$ para cualquier valor de **x** con el mínimo error esperado.

$$h(x) = \beta_0 + \beta_1 x$$

Mínimos Cuadrados

- El método de mínimos cuadrados ordinarios se usa para estimar $\hat{\beta}_0$ y $\hat{\beta}_1$ minimizando la suma de los errores cuadráticos (SSE) de los datos observados.
- Supongamos que tenemos m observaciones de **y** y de **x**, calculamos la suma de los errores cuadráticos (SSE) o E de error de la siguiente forma:

$$E = \sum_{i=1}^{m} (y_i - h(x_i))^2 = \sum_{i=1}^{m} (y_i - \beta_0 - \beta_1 x_i)^2 \qquad (1)$$

- Para encontrar los parámetros que minimizan el error calculamos las derivadas parciales de SSE respecto a β_0 y β_1 . Luego igualamos las derivadas a cero y resolvemos la ecuación para despejar los parámetros.

$$\frac{\partial E}{\partial \beta_0} = -2 \sum_{i=1}^{m} (y_i - \beta_0 - \beta_1 x_i) = 0 \qquad (2)$$

$$\frac{\partial E}{\partial \beta_1} = -2 \sum_{i=1}^{m} (y_i - \beta_0 - \beta_1 x_i) x_i = 0 \qquad (3)$$

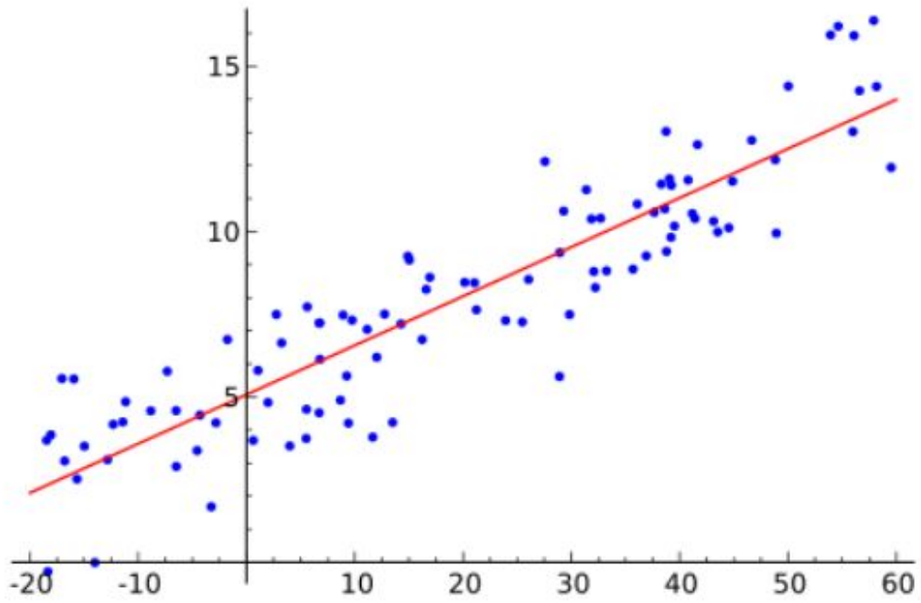
Mínimos Cuadrados (2)

- Del sistema de ecuaciones anterior se obtienen las soluciones normales:

$$\hat{\beta}_1 = \frac{\sum_i^m (x_i - \overline{x})(y_i - \overline{y})}{\sum_i^m (x_i - \overline{x})^2} \qquad (4)$$

$$\hat{\beta}_0 = \overline{y} - \hat{\beta}_1 \overline{x} \qquad (5)$$

- El modelo ajustado representa la recta de mínimo error cuadrático.



Coeficiente de Determinación R^2

- Una vez ajustado nuestro modelo lineal debemos evaluar la calidad del modelo.
- Una medida muy común es el coeficiente de determinación R^2.
- Para calcularlo debo calcular otros errores distintos a los errores cuadráticos SSE.
- Se define a la suma cuadrática total (**SST**) como el error predictivo cuando usamos la media y para predecir la variable de respuesta y (es muy similar a la varianza de la variable):

$$\text{SST} = \sum_{i}^{m} (y_i - \overline{y})^2$$

- Luego tenemos a la suma de los cuadrados explicada por el modelo (SSM) que nos indica la variabilidad de los valores predichos por el modelo respecto a la media.

$$\text{SSM} = \sum_{i}^{m} (\hat{y}_i - \overline{y})^2$$

Coeficiente de Determinación R^2

- Se define el coeficiente de determinación para un modelo lineal R^2 como:

$$R^2 = \frac{\text{SSM}}{\text{SST}} = \frac{\sum_i^m (\hat{y}_i - \overline{y})^2}{\sum_i^m (y_i - \overline{y})^2} \qquad (6)$$

- El coeficiente adquiere valores entre 0 a 1 y mientras más cercano a 1 sea su valor mayor será la calidad del modelo.

- El valor de R^2 es equivalente a la correlación lineal (Pearsons) entre y e ŷ al cuadrado.

$$R^2 = \text{cor}(y, \hat{y})^2$$

Regresión Lineal Múltiple

- Supongamos que tenemos n variables independientes: x_1, x_2, ..., x_n.
- Intuitivamente, estas variables en conjunto podrían explicar de mejor manera la variabilidad de la variable de respuesta **y** que un modelo simple.
- Se define un modelo lineal multivariado de la siguiente manera:

$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_n x_{i,n} + \epsilon_i \quad \forall i \in \{1, m\}$$

- En el modelo multivariado se extienden todas las propiedades del modelo lineal simple.
- Se puede representar el problema de manera matricial:

$$Y = X\beta + \epsilon$$

- Donde Y es un vector de m × 1 de variables de respuesta: $Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{pmatrix}$

Regresión Lineal Múltiple

- X es una matriz de $m \times (n + 1)$ con las variables explicativas.
- Tenemos m observaciones de las n variables.
- La primera columna es constante igual a 1 ($x_{i,0} = 1 \ \forall i$) para incluir la variable de intercepto β_0 de manera limpia.

$$X = \begin{pmatrix} x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m,0} & x_{m,1} & x_{m,2} & \cdots & x_{m,n} \end{pmatrix}$$

- Luego, β es un vector de parámetros de $(n + 1) \times 1$

$$\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}$$

Regresión Lineal Múltiple

- Finalmente, ε es un vector con los errores del modelo de dimensiones m × 1.

$$\epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_m \end{pmatrix}$$

- Usando la notación matricial, podemos ver que la suma de los errores cuadráticos (SSE) se puede expresar como:

$$\text{SSE} = (Y - X\beta)^T (Y - X\beta)$$

- Minimizando esta expresión derivando el error en función de β e igualando a cero se llega a las ecuaciones normales:

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

Supuestos del Modelo Lineal

Cada vez que ajustamos un modelo lineal estamos asumiendo implícitamente ciertos supuestos sobre los datos.

01	**Linealidad**	La variable de respuesta se relaciona linealmente con los atributos.
02	**Normalidad**	Los errores tienen distribución normal de media cero: $\varepsilon_i \sim N(0, \sigma^2)$
03	**Homocedasticidad**	Los errores tienen varianza constante (mismo valor de σ^2).
04	**Independencia**	Los errores son independientes entre sí.

Interpretación Probabilística

- Considerando los supuestos anteriores podemos ver que la densidad de probabilidad (PDF) de los errores ε se definen por una normal de media cero y varianza constante:

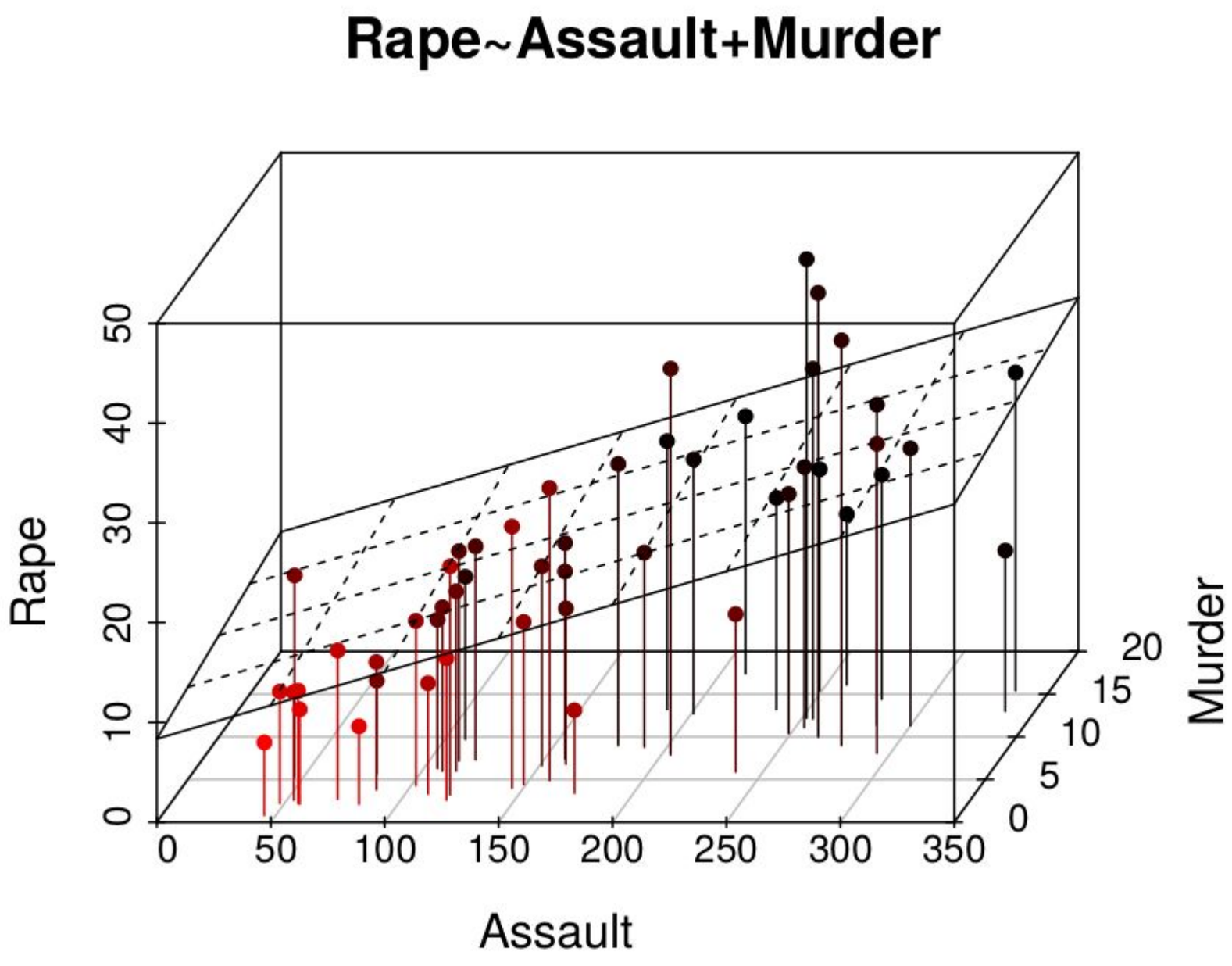
$$\text{PDF}(\epsilon_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\epsilon_i^2}{2\sigma^2}\right)$$

- Esto implica que:

$$\text{PDF}(y_i|x_i;\beta) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - h_\beta(x_i))^2}{2\sigma^2}\right)$$

- Lo que implica que la distribución de **y** dada los valores de **x** y parametrizada por β sigue una distribución normal.
- Luego si uno estima los parámetros de β usando una técnica de estimación llamada **máxima verosimilitud** llega a los mismos resultados que haciendo estimación por mínimos cuadrados.
- Esto nos dice que cuando estimamos los parámetros del modelo usando mínimos cuadrados estamos realizando las mismas hipótesis probabilísticas mencionados anteriormente.

Ejemplo


Rape~Assault+Murder
Rape
0
10
20
30
40
50
Assault
0
50
100
150
200
250
300
350
Murder
0
5
10
15
20

Métricas de Evaluación

Algunas métricas usadas para evaluar modelos de regresión:

Error cuadrático medio (RMSE)

$$\mathrm{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

- Error absoluto medio (MAE)

$$\mathrm{MAE}(\mathbf{X}, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right|$$

Entrenando un modelo lineal

- Una forma alternativa a ver el problema de regresión es definiendo una función de pérdida **L(ŷ, y)**, indicando la **pérdida** o **error** de la predicción de ŷ cuando la salida verdadera es y.
- Una función de pérdida calcula un valor escalar a partir de ŷ e y.
- Una función de pérdida a usar para regresión es el error cuadrático medio (MSE), que es el SSE normalizado por la cantidad de ejemplos.

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - h(x_i))^2 \qquad (7)$$

- El objetivo del entrenamiento es minimizar la pérdida en los datos de entrenamiento.

Entrenando un modelo lineal

- La regresión lineal es un caso particular de modelo de regresión donde los parámetros tienen solución **exacta** (ecuaciones normales).
- Alternativamente, una regresión se puede entrenar usando métodos iterativos basados en **gradientes**.
- Se calculan los gradientes de los parámetros con respecto a la pérdida L, y se mueven los parámetros en las direcciones opuestas del gradiente.
- Diferentes métodos de **optimización** difieren en cómo se calcula la estimación del error y cómo se define el movimiento en la dirección opuesta al gradiente.

Descenso del Gradiente

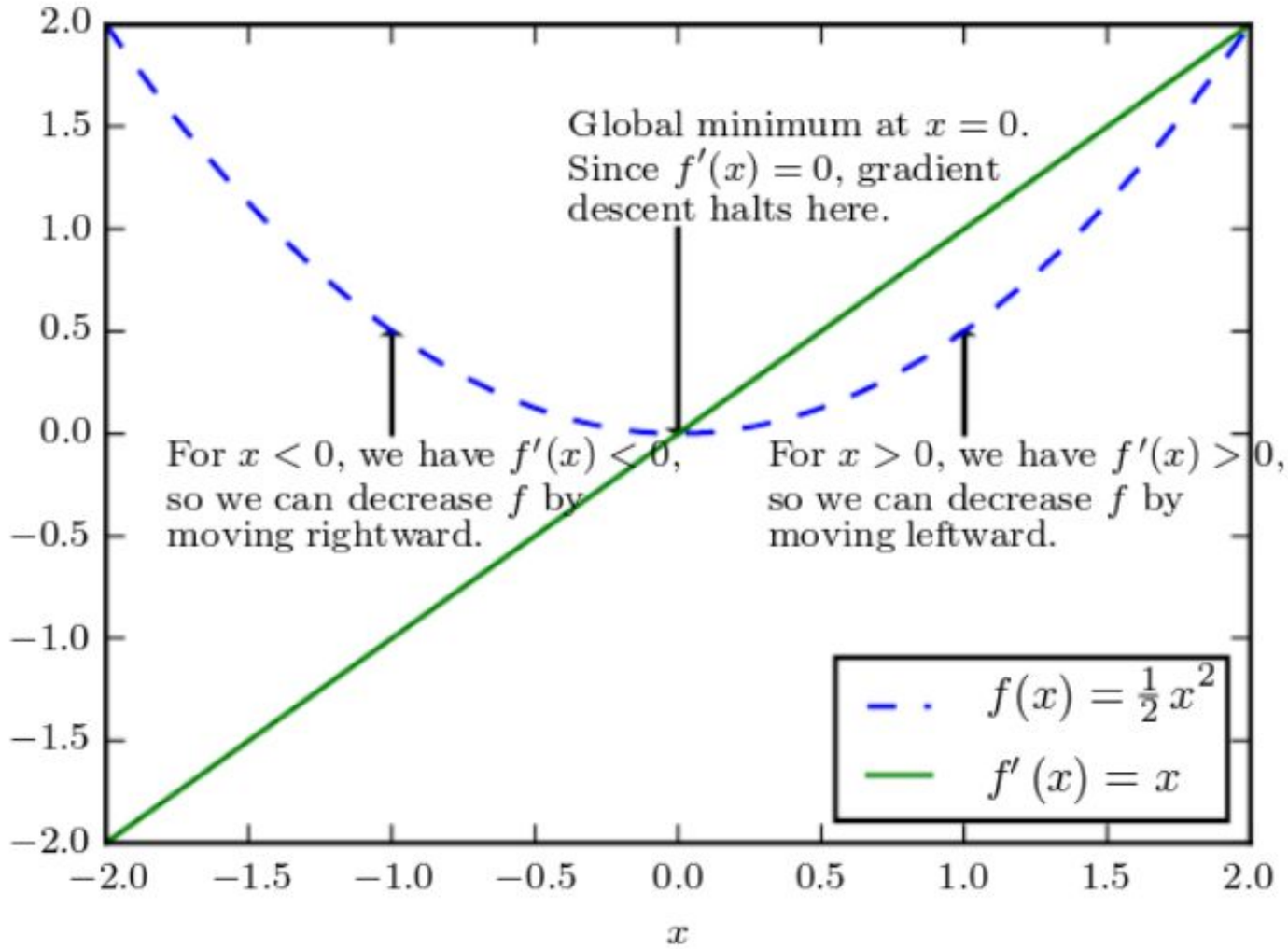
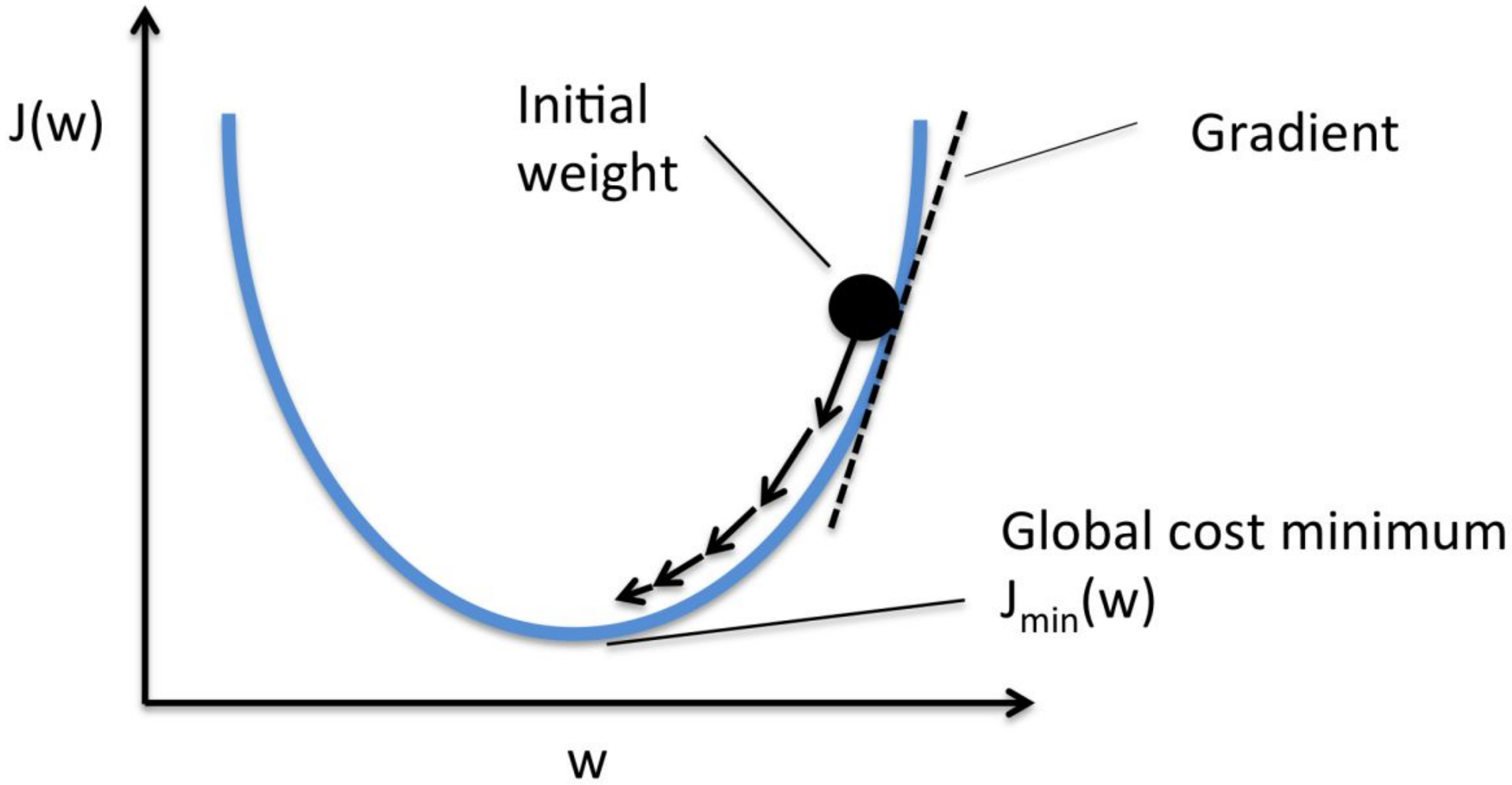


Figure 4.1: Gradient descent. An illustration of how the gradient descent algorithm uses the derivatives of a function to follow the function downhill to a minimum.

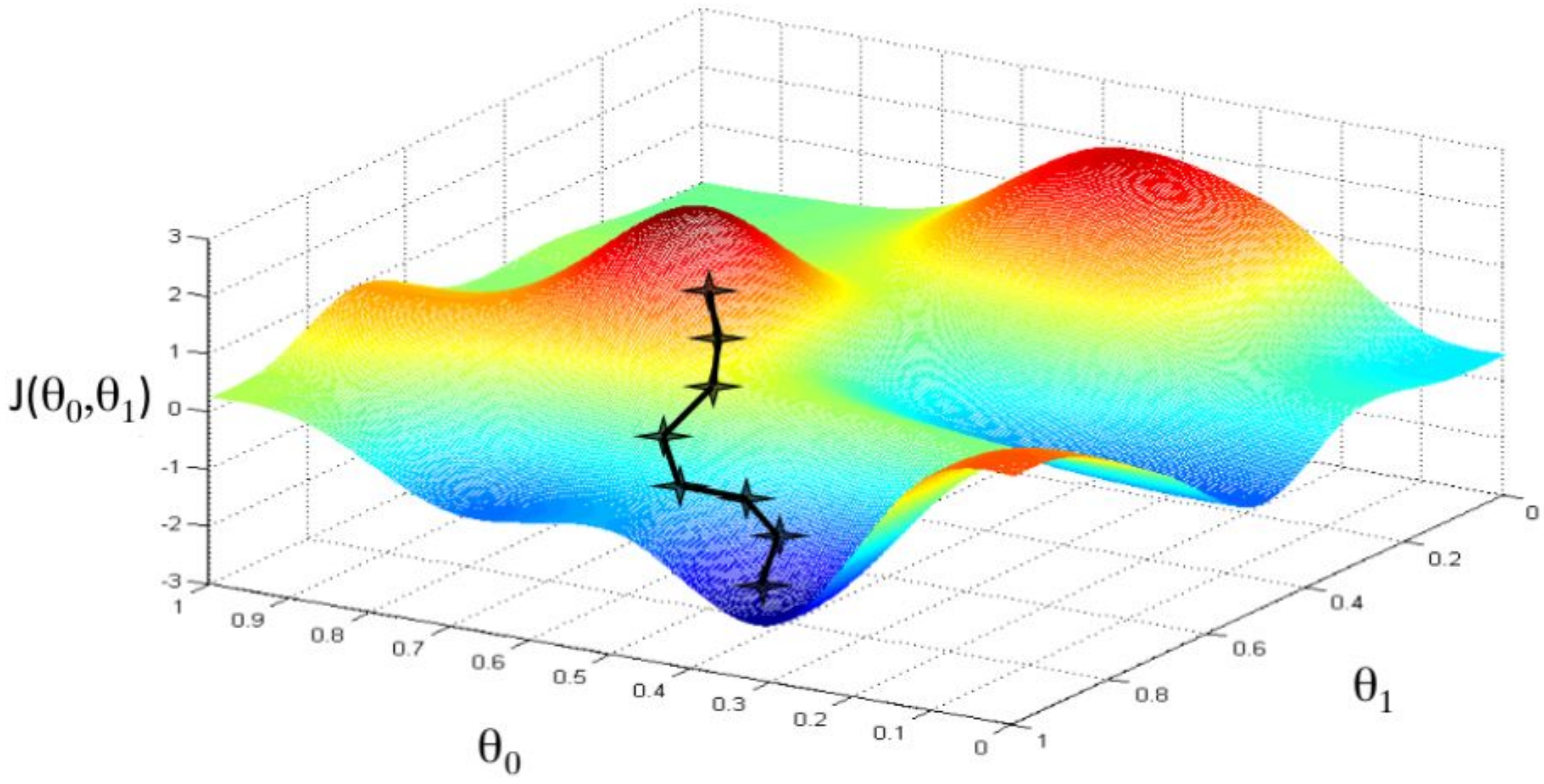
- Imagen sacada de [1].

Descenso del Gradiente



- Imagen sacada de: https://sebastianraschka.com/images/faq/closed-form-vs-gd/ball.png

Descenso del Gradiente



- Imagen sacada de: https://www.coursera.org/learn/machine-learning

Descenso del Gradiente Online Estocástico (SGD)

- Se inicializan los parámetros **w** con valores iniciales (o pesos) aleatorios.
- Por cada dato de entrenamiento (x, y) calculo L con el valor actual de **w** y actualizo los parámetros usando la siguiente regla hasta converger:

$$w_i \leftarrow w_i - \eta \frac{\partial L}{w_i}(x, y)$$ (Para todos los parámetros w_i)

Algorithm 2.1 Online stochastic gradient descent training.

Input:
- Function $f(x; \Theta)$ parameterized with parameters Θ.
- Training set of inputs $x_1, \ldots, x_n$ and desired outputs $y_1, \ldots, y_n$.
- Loss function L.

1: **while** stopping criteria not met **do**
2: Sample a training example x_i , y_i
3: Compute the loss $L(f(x_i; \Theta), y_i)$
4: $\hat{g} \leftarrow$ gradients of $L(f(x_i; \Theta), y_i)$ w.r.t Θ
5: $\Theta \leftarrow \Theta - \eta_t \hat{g}$
6: **return** Θ

Descenso del Gradiente Online Estocástico (SGD)

- La tasa de aprendizaje **η** puede ser fija a lo largo del proceso de entrenamiento o se puede decaer en función del paso de tiempo t.
- El error calculado en la línea 3 se basa en **un solo dato** de entrenamiento y, por lo tanto, es solo una estimación **aproximada** de la pérdida total L que queremos minimizar.
- El ruido en el cálculo de la pérdida puede dar lugar a gradientes inexactos (un solo dato puede proporcionar **información ruidosa**).

Funciones de Pérdida para clasificación

El MSE es una función de pérdida para entrenar modelos de regresión. También se puede tener funciones de pérdida para entrenar modelos de **clasificación**.

- Hinge (función bisagra): para problemas de clasificación binaria, la salida del clasificador es un escalar $\tilde{y}$ y la salida deseada y está en {+1, −1}.
- La regla de clasificación es $\hat{y} = \text{sign}(\tilde{y})$, y la clasificación se considera correcta cuando $y \cdot \tilde{y} > 0$.

$$L_{\text{hinge(binary)}}(\tilde{y}, y) = \text{máx}(0, 1 - y \cdot \tilde{y})$$

- Esta es la función de pérdida de la SVM (hiperplano de máximo margen).
- ¡Podemos entrenar una SVM lineal usando SGD!
- ¿Es máx una función derivable? Para aplicar SGD a máx(0, x), el valor del gradiente es 1 cuando x > 0 y 0 para el caso contrario.

Regresión Logística

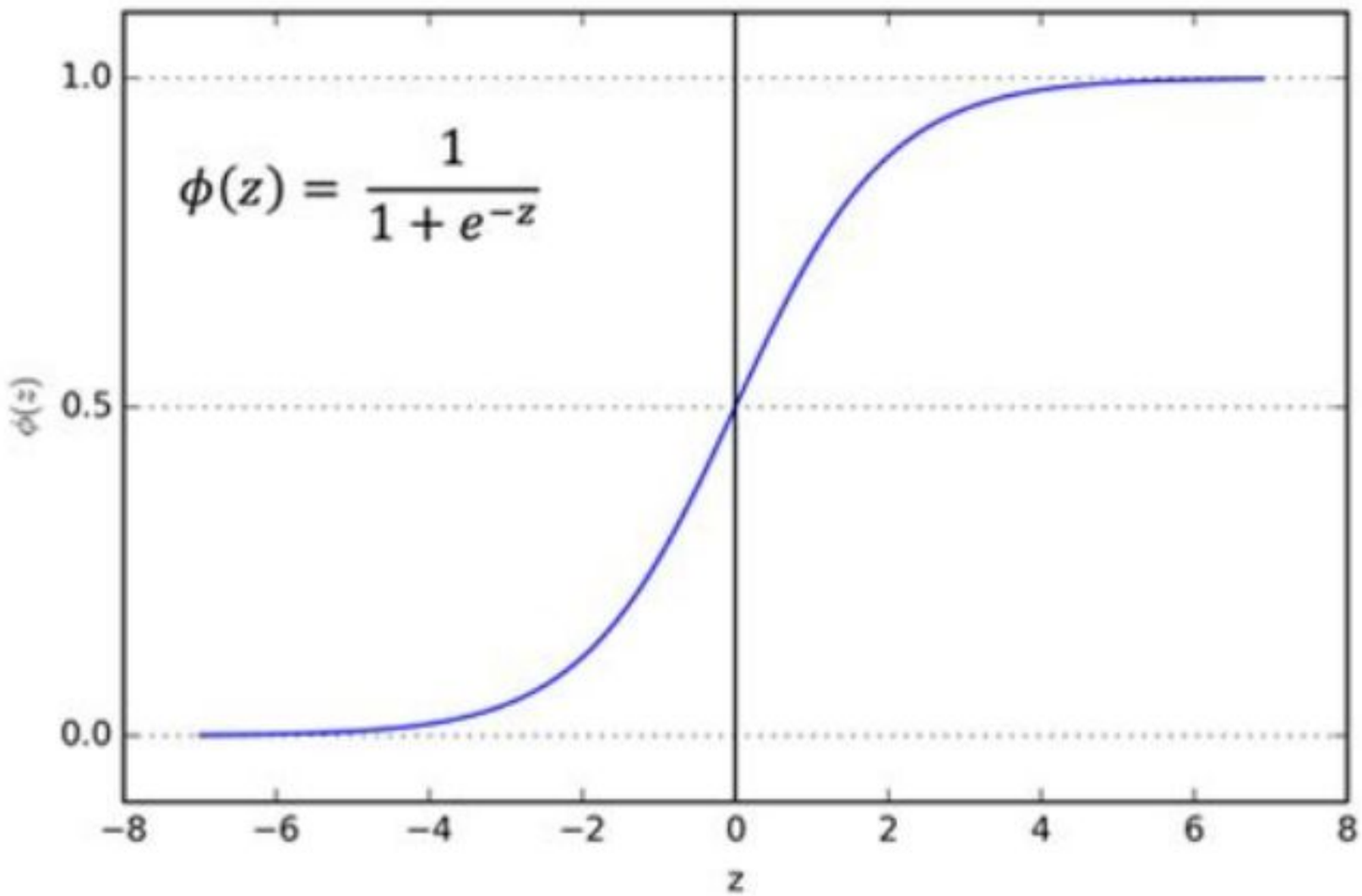
- Una regresión logística es un modelo de **clasificación** que estima la probabilidad posterior P(y|x) de una variable binaria **y** dado los datos observados **x** ajustando un modelo lineal a los datos.
- Los parámetros del modelo son un vector de parámetros **w**.
- Si asumimos el término de intercepto como 1 $x_0 = 1$, tenemos una función lineal de la siguiente forma:

$$\tilde{y} = \sum_{i=0}^{n} w_i x_i = w^T x \tag{8}$$

- Para darle una interpretación probabilística a la salida, transformamos $\tilde{y}$ al intervalo [0, 1] usando una función sigmoidal:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{9}$$

Función Sigmoidal



Regresión Logística

- Esto se puede resumir en la función de pérdida logística:

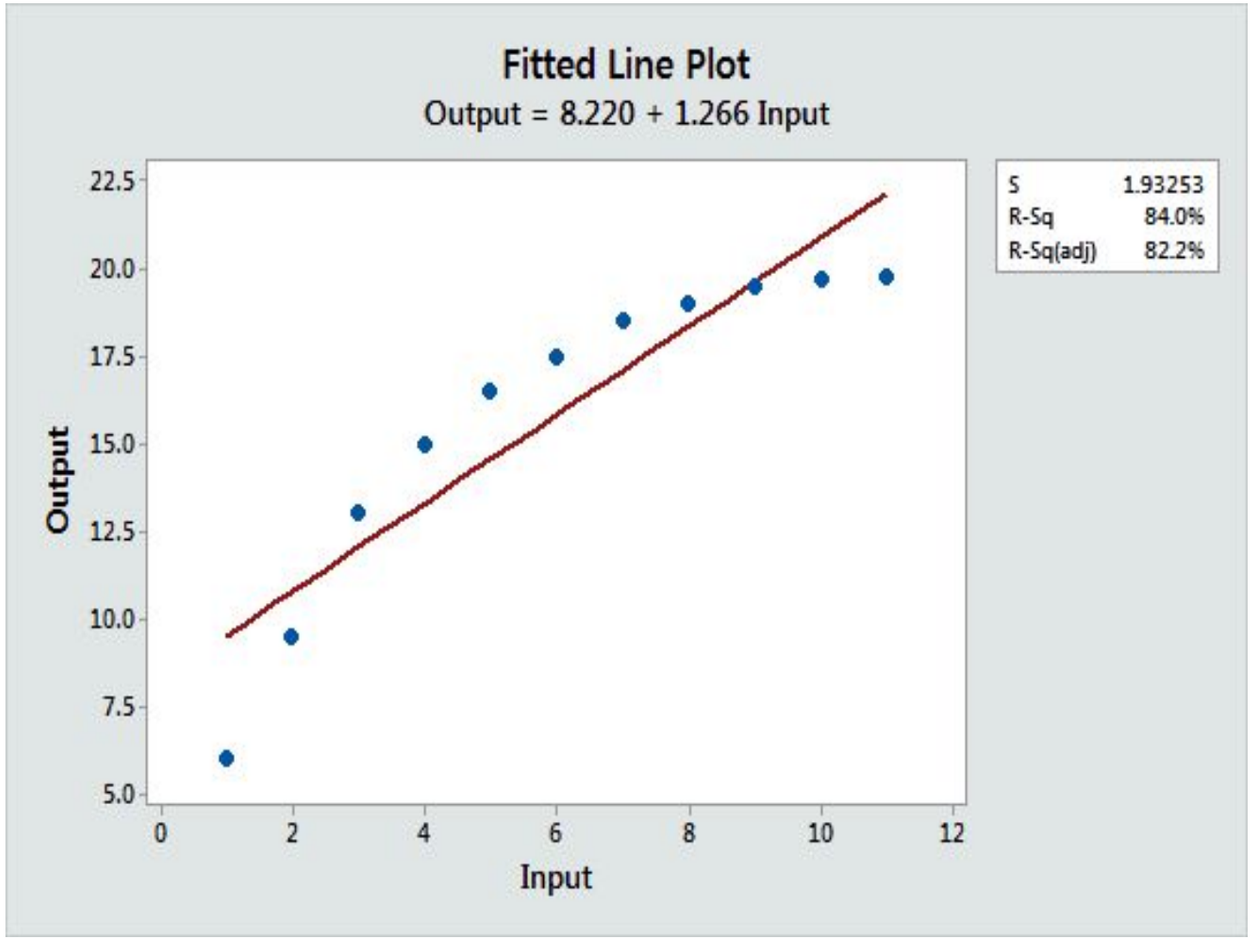
$$L_{\text{logistic}}(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

- Esta función de pérdida es entonces el negativo del log-likelihood de un modelo probabilístico donde P(y|x) sigue una distribución de Bernoulli.
- Muchas funciones de pérdidas son el negativo de una función de verosimilitud.
- Entonces, minimizar la pérdida equivale en esos casos a realizar estimación por **máxima verosimilitud**.

  ¡Podemos entrenar una regresión logística usando SGD!

Introducción a las redes neuronales

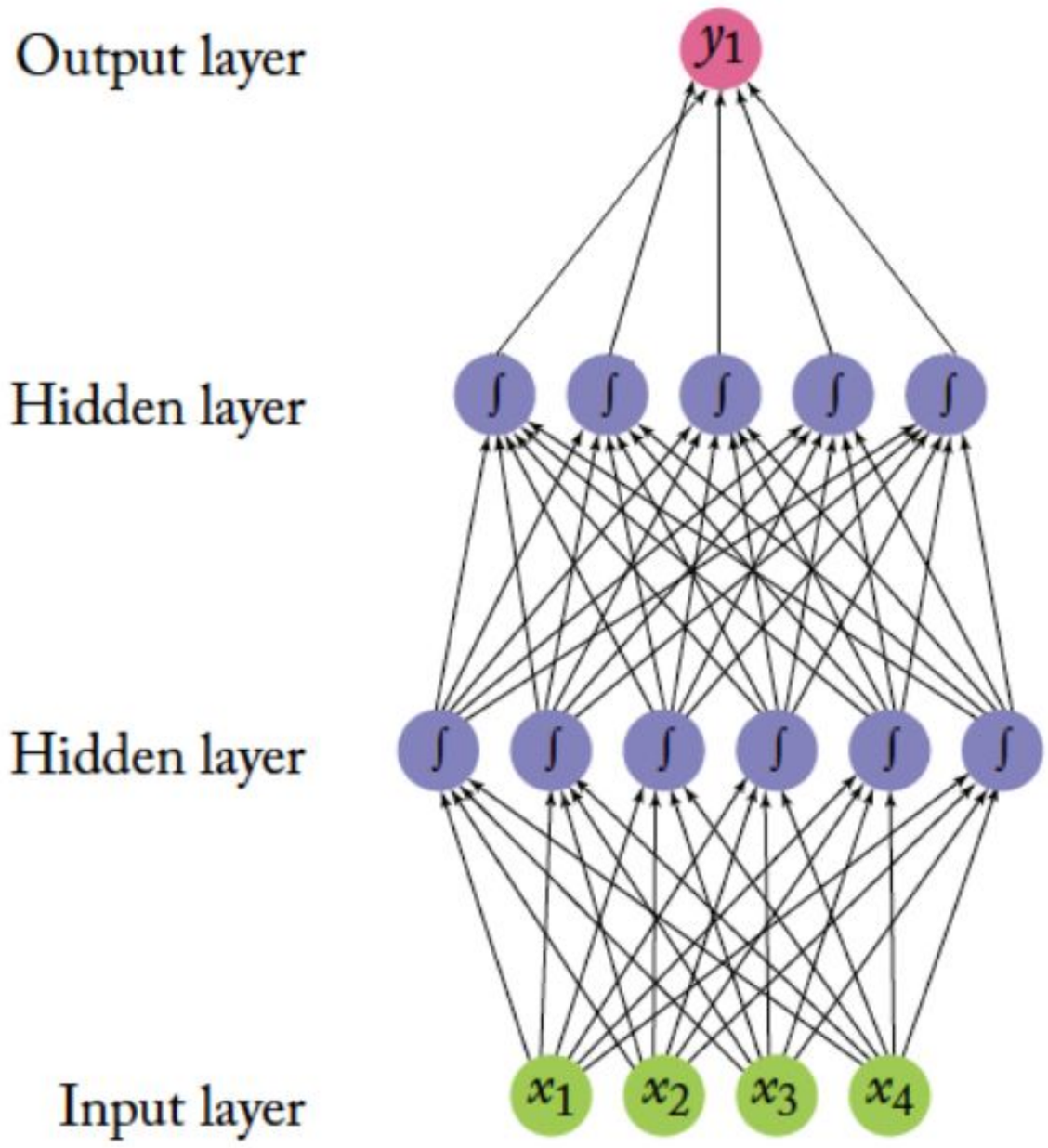
- Una gran limitación de los modelos lineales que sólo pueden encontrar relaciones **lineales** entre la entrada y la salida.
- Las **redes neuronales** son modelos de aprendizaje automático muy populares formados por unidades llamadas **neuronas**.
- Son capaces de aprender relaciones **no-lineales** entre x e y.
- También se pueden entrenar con métodos de **gradiente**.



Introducción a las redes neuronales

- Una **neurona** es una unidad computacional que tiene entradas y salidas escalares.
- Cada entrada tiene un peso asociado **w**.
- La neurona multiplica cada entrada por su peso y luego las suma (también se pueden usar otras funciones de agregación como **max**).
- Luego aplica una función de activación **g** (generalmente no lineal) al resultado, y la pasa a su salida.
- Varias neuronas pueden agruparse en una **capa** de neuronas.
- La salida de una capa puede pasarse como input a otra capa de forma de **cascada**.
- A este tipo de redes se les conoce como **Feedforward Network**s o **Multi-Layer Perceptron**.

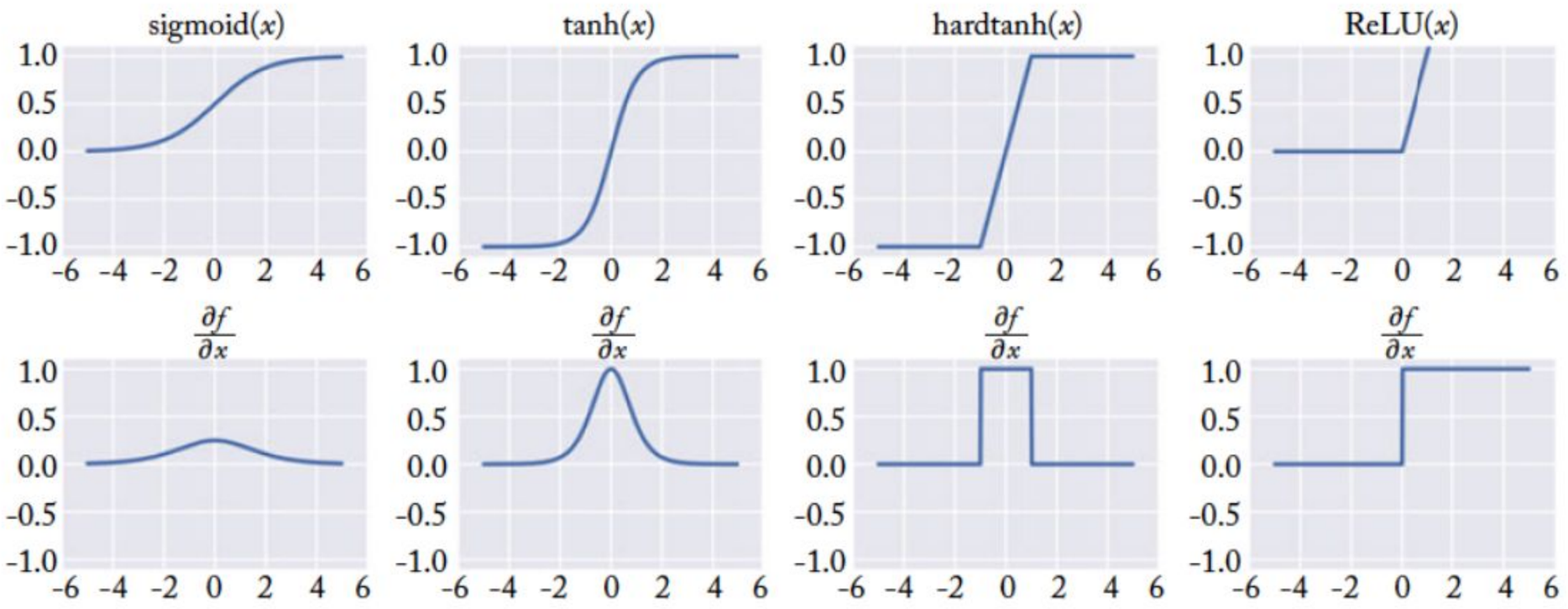
Feedforward Network de dos capas



Fuente: [2]

Funciones de activación

- La función de activación no lineal **g** tiene un papel crucial en la capacidad de la red para representar funciones complejas.
- Si quitamos la no-linealidad aportada por **g**, la red neuronal sólo podría representar transformaciones lineales de la entrada.



Fuente: [2]

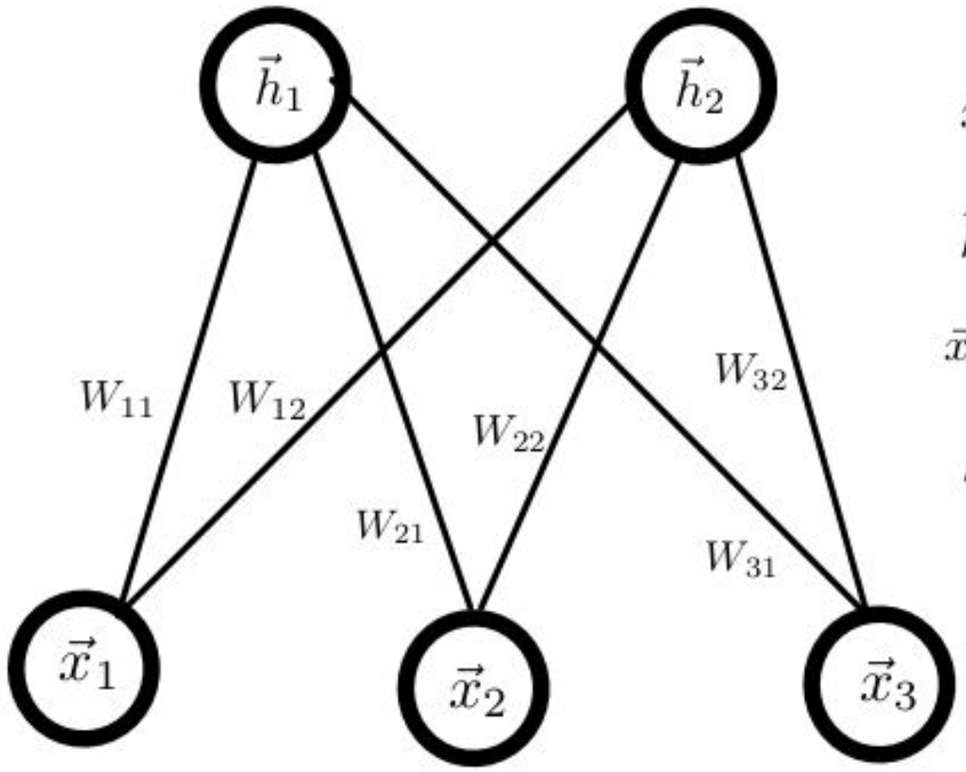
Redes Feedforward

- La red feedforward de la imagen es una pila de modelos lineales separados por funciones no lineales.
- Los valores de cada fila de neuronas en la red se pueden considerar como un vector.
- La capa de entrada es un vector de 4 dimensiones $\vec{x}$ y la capa de arriba es un vector de 6 dimensiones $\vec{h}$
- En esta notación asumimos que los vectores son filas y los superíndices corresponden a capas de red.
- Esta capa de conexión completa se puede ver una transformación lineal de 4 a 6 dimensiones.
- Una capa de conexión completa implementa una multiplicación vector-matriz:

$$\vec{h} = \vec{x}W$$

- El peso de la conexión desde la neurona i en la fila de entrada hasta la neurona j en la fila de salida es $W_{[i,j]}$.
- Los valores de $\vec{h}$ se transforman usando una función no lineal **g** que se aplica a cada elemento antes de pasar como entrada a la siguiente capa.

Capa complementamente conectado como una multiplicación vector por matriz



$$\vec{x} = [\vec{x}_1, \vec{x}_2, \vec{x}_3] \qquad W = \begin{pmatrix} W_{1,1} & W_{1,2} \\ W_{2,1} & W_{2,2} \\ W_{3,1} & W_{3,2} \end{pmatrix}$$

$$\vec{h} = \vec{x}W$$

$$\vec{x}W = [\vec{x}_1 * W_{11} + \vec{x}_2 * W_{21} + x_3 * W_{31}, \vec{x}_1 * W_{12} + \vec{x}_2 * W_{22} + x_3 * W_{32}]$$

$$\vec{h} = [\vec{h}_1, \vec{h}_2]$$

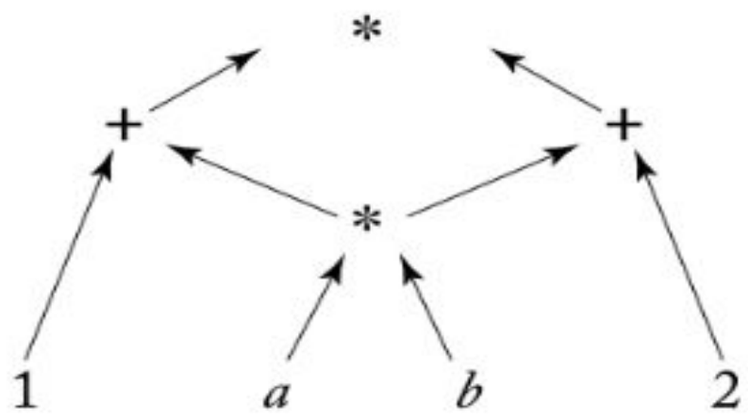
La red como una función

- La red feedforward de la imagen es una pila de modelos lineales separados por funciones no lineales.

$$
\begin{aligned}
NN_{MLP2}(\vec{x}) &= \vec{y} \\
\vec{h}^1 &= g^1(\vec{x}W^1 + \vec{b}^1) \\
\vec{h}^2 &= g^2(\vec{h}^1 W^2 + \vec{b}^2) \\
\vec{y} &= \vec{h}^2 W^3 \\
\vec{y} &= (g^2(g^1(\vec{x}W^1 + \vec{b}^1)W^2 + \vec{b}^2))W^3.
\end{aligned}
\tag{10}
$$

El Grafo de Cómputo

- Las redes neuronales se entrenan usando **descenso por gradiente.**
- En teoría, se podrían calcular los gradientes de los diversos parámetros de una red a mano e implementarlos en código.
- Este procedimiento es engorroso y propenso a errores.
- Es preferible usar herramientas de derivación automática [3].
- Un **grafo de cómputo** (computation graph) es un grafo capaz de representar cualquier proceso de cómputo matemático (ej: evaluar una red neuronal).
- Considere, por ejemplo el grafo computacional para (a ∗ b + 1) ∗ (a ∗ b + 2):



- El cálculo de a ∗ b es compartido.
- La estructura del grafo define el orden del cálculo en términos de las dependencias entre los diferentes componentes.

El Grafo de Cómputo

El grafo de cómputo nos permite:

1. Construir fácilmente redes neuronales arbitrarias.
2. Evaluar sus predicciones para una entrada dada (forward pass)

Algorithm 5.3 Computation graph forward pass.

1: **for** i = 1 to N **do**
2: Let $a_1, \ldots, a_m = \pi^{-1}(i)$
3: $v(i) \leftarrow f_i(v(a_1), \ldots, v(a_m))$

3. Calcular los gradientes para sus parámetros con respecto a funciones de pérdida arbitrarias (backward pass o **backpropagation**).

Algorithm 5.4 Computation graph backward pass (backpropagation).

1: $d(N) \leftarrow 1$ ▷ $\frac{\partial N}{\partial N} = 1$
2: **for** i = N-1 to 1 **do**
3: $d(i) \leftarrow \sum_{j \in \pi(i)} d(j) \cdot \frac{\partial f_j}{\partial i}$ ▷ $\frac{\partial N}{\partial i} = \sum_{j \in \pi(i)} \frac{\partial N}{\partial j} \frac{\partial j}{\partial i}$

- El algoritmo de backpropagation (backward pass) esencialmente sigue la regla de la cadena de derivación.

- Un muy buen tutorial del algoritmo backpropagation usando la abstracción del grafo de cómputo: https://colah.github.io/posts/2015-08-Backprop/

SGD por Mini-batches

- SGD es susceptible al ruido inducido por un único ejemplo (un outlier puede mover mucho el gradiente).
- Una forma común para reducir este ruido es estimar el error y los gradientes sobre muestras de **m** ejemplos.
- Esto se llama **minibatch** SGD.
- Valores grandes para **m** dan una mejor estimación del gradiente en base al dataset completo, mientras que valores más pequeños permiten realizar más actualizaciones y **converger más rápido**.
- Para tamaños razonables de m, algunas arquitecturas computacionales (i.e., **GPUs, TPUs**) permiten paralelizar SGD eficientemente (es la única forma de entrenar redes de muchos parámetros en tiempo razonable).

Descenso de Gradiente Estocástico por Mini-batches

Algorithm 2.2 Minibatch stochastic gradient descent training.

Input:
- Function $f(\boldsymbol{x}; \Theta)$ parameterized with parameters Θ.
- Training set of inputs $\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}$ and desired outputs $\boldsymbol{y_1}, \ldots, \boldsymbol{y_n}$.
- Loss function L.

while stopping criteria not met **do**
  Sample a minibatch of m examples $\{(\boldsymbol{x_1}, \boldsymbol{y_1}), \ldots, (\boldsymbol{x_m}, \boldsymbol{y_m})\}$
  $\hat{\boldsymbol{g}} \leftarrow 0$
  for $i = 1$ to m **do**
    Compute the loss $L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$
    $\hat{\boldsymbol{g}} \leftarrow \hat{\boldsymbol{g}} +$ gradients of $\frac{1}{m} L(f(\boldsymbol{x_i}; \Theta), \boldsymbol{y_i})$ w.r.t Θ
  $\Theta \leftarrow \Theta - \eta_t \hat{\boldsymbol{g}}$
return Θ

Fuente: [2]

Comentarios

- Las redes neuronales son modelos muy poderosos para **regresión y clasificación**.
- El uso de redes con varias capas se llama popularmente como **Deep Learning**.
- Existen arquitecturas de red que son buenas para aprender **representaciones** sobre datos complejos: texto, imágenes, audio, video.
- Arquitecturas famosas: redes neuronales **convolucionales**, redes **recurrentes** y redes de **atención**.
- La alta capacidad de estas redes las hace muy proclives al overfitting.
- Hay varias técnicas para mitigarlo: regularización, drop-out, batch normalization.

Frameworks para Deep Learning

Varios paquetes de software implementan el modelo de grafo de cómputo. Estos paquetes implementan los componentes esenciales (tipos de nodo) para definir una amplia gama de **arquitecturas** de redes neuronales.

1. **TensorFlow** (https://www.tensorflow.org/): una biblioteca de software de código abierto para cálculos numéricos utilizando data-flow graphs desarrollados originalmente por Google Brain Team.
2. **Keras**: API de redes neuronales de alto nivel que corre sobre Tensorflow y otros backends (https://keras.io/).
3. **PyTorch**: biblioteca de código abierto de aprendizaje automático para Python, basada en Torch, desarrollada por el grupo de investigación de inteligencia artificial de Facebook. Es compatible con la construcción de grafos de cómputo dinámicos, se crea un grafo de cómputo diferente desde cero para cada muestra de entrenamiento. (https://pytorch.org/).

Bilbiografía

1. L. Wasserman All of Statistics: A Concise Course in Statistical Inference, Springer Texts in Statistics, 2005.
2. Goldberg, Y. (2016). A primer on neural network models for natural language processing. J. Artif. Intell. Res.(JAIR), 57:345–420.
3. Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.

dcc

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

www.dcc.uchile.cl

/ DCCUCHILE