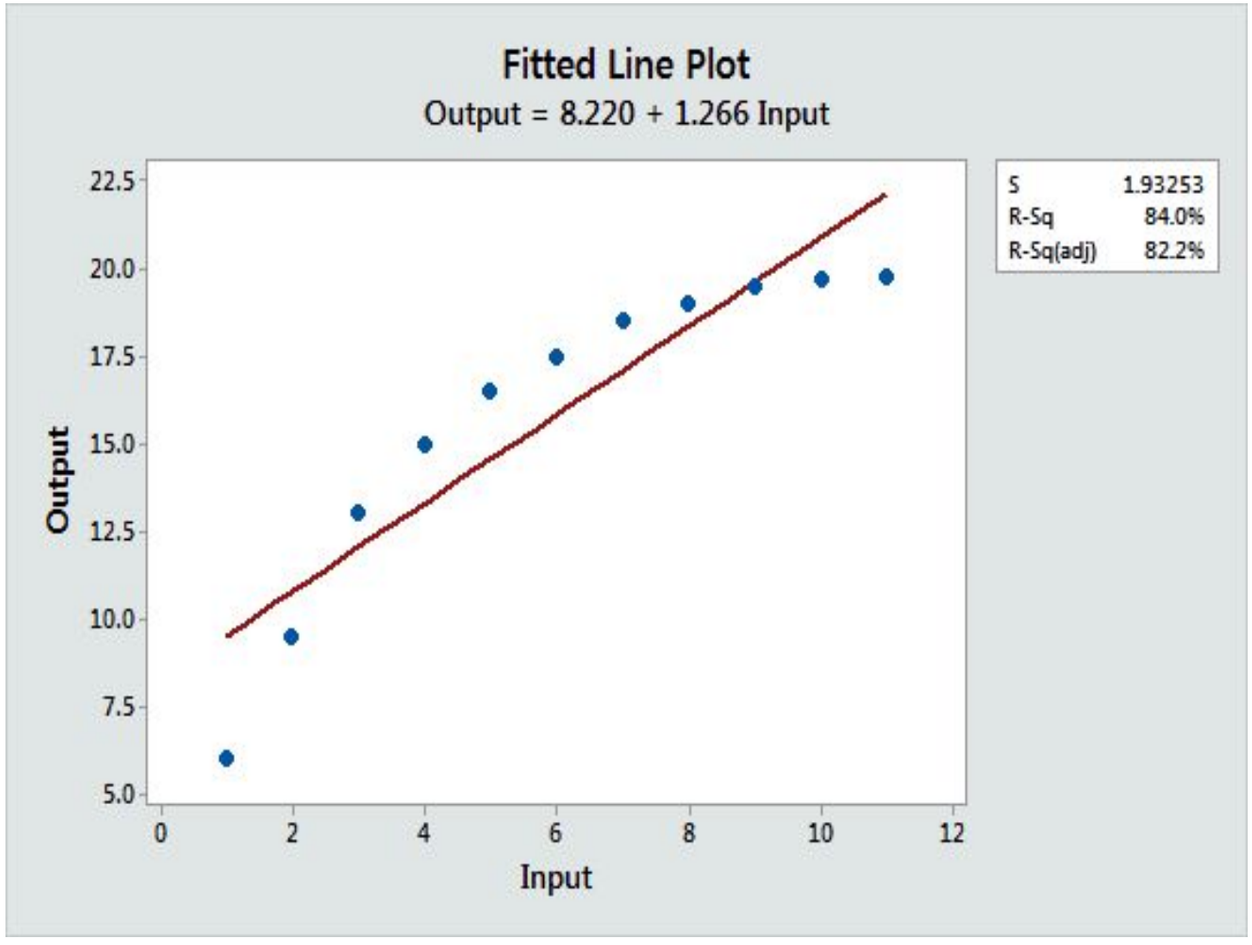
CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Introducción a las Redes Neuronales

Introducción a las redes neuronales

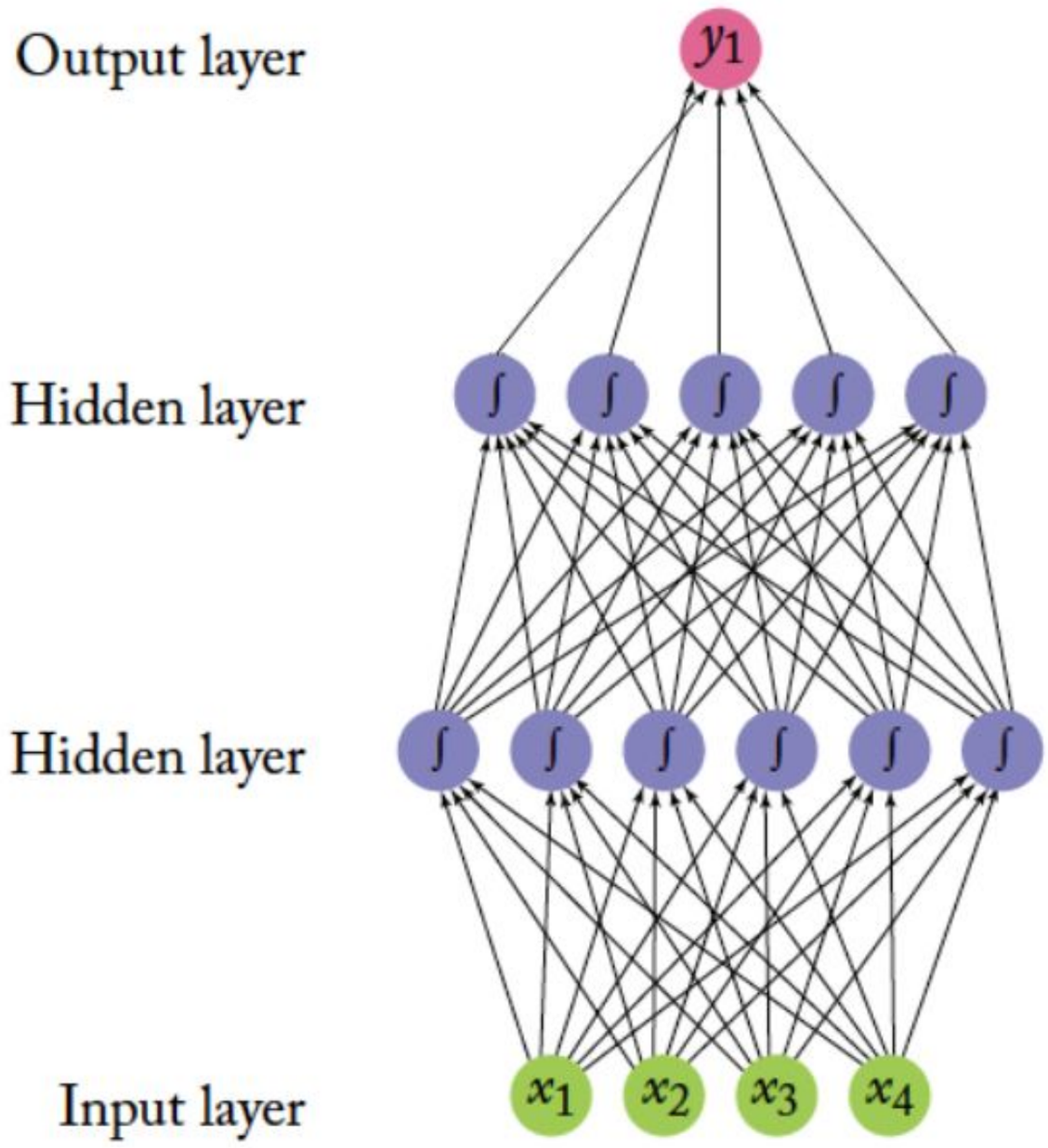
- Una gran limitación de los modelos lineales que sólo pueden encontrar relaciones **lineales** entre la entrada y la salida.
- Las **redes neuronales** son modelos de aprendizaje automático muy populares formados por unidades llamadas **neuronas**.
- Son capaces de aprender relaciones **no-lineales**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

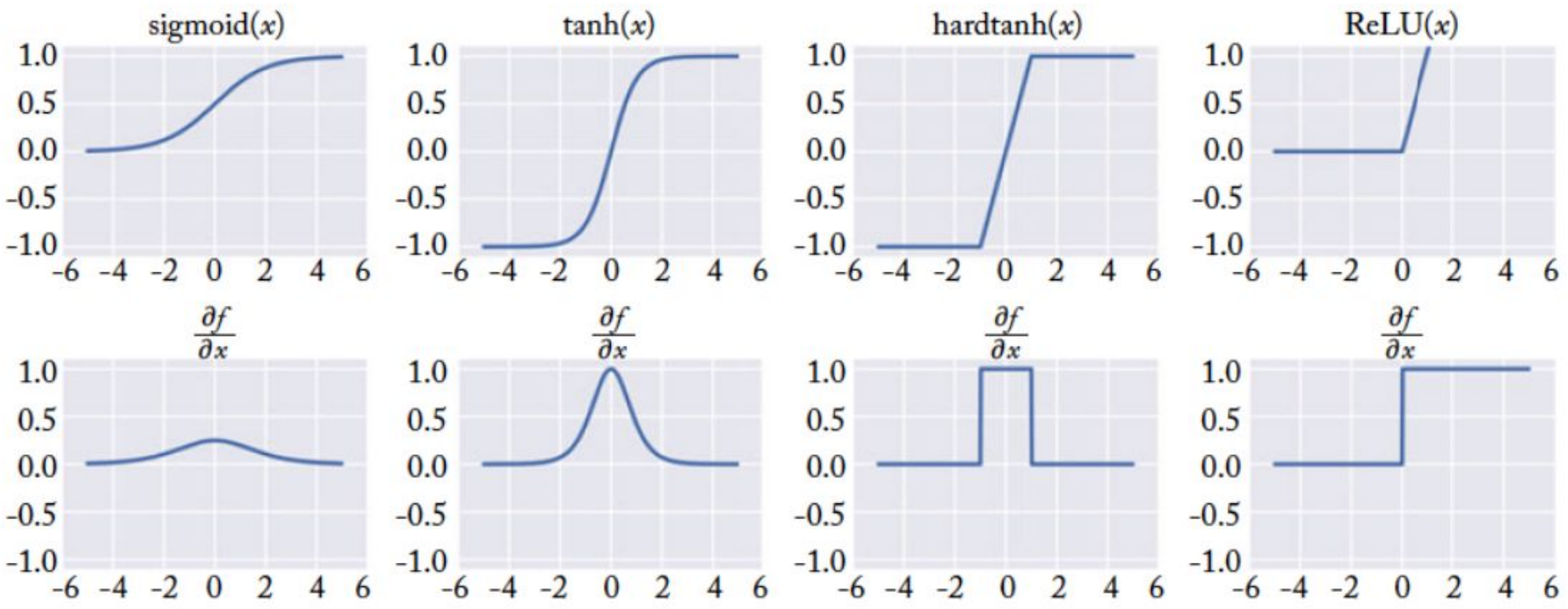
Feedforward Network de dos capas



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]

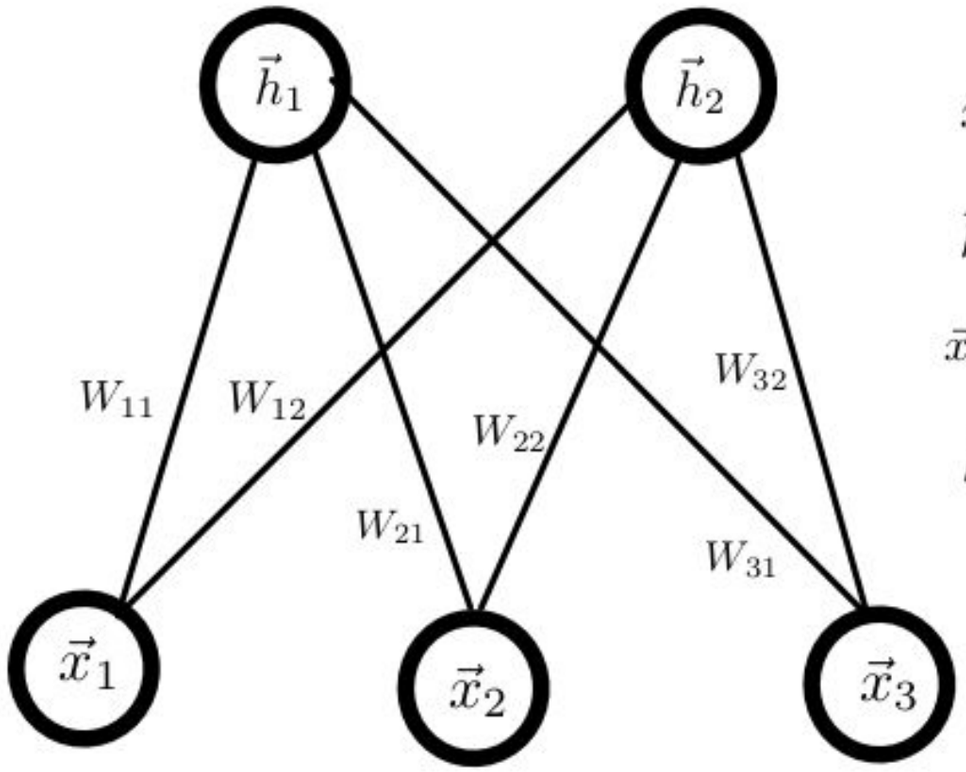
Redes Feedforward

- 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



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

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

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

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

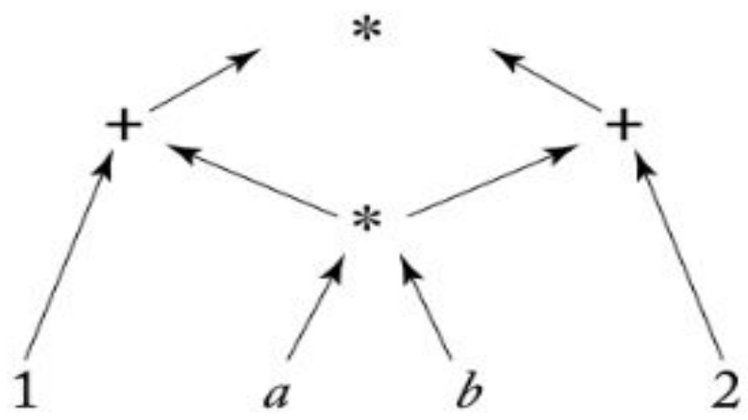
La red como una función

- 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. Goldberg, Y. (2016). A primer on neural network models for natural language processing. J. Artif. Intell. Res.(JAIR), 57:345–420.
2. 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