

Introducción al Deep Learning

Jorge Linde Díaz

CUNEF



Introducción al Deep Learning

1 Introducción

- Motivación y características principales de las Redes Neuronales
- Funcionamiento y estructura
- Áreas de aplicación

2 Introducción a las Redes Neuronales

- Desde el Perceptrón hasta las Redes Neuronales
- Gradient descent
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

Índice

1 Introducción

- Motivación y características principales de las Redes Neuronales
- Funcionamiento y estructura
- Áreas de aplicación

2 Introducción a las Redes Neuronales

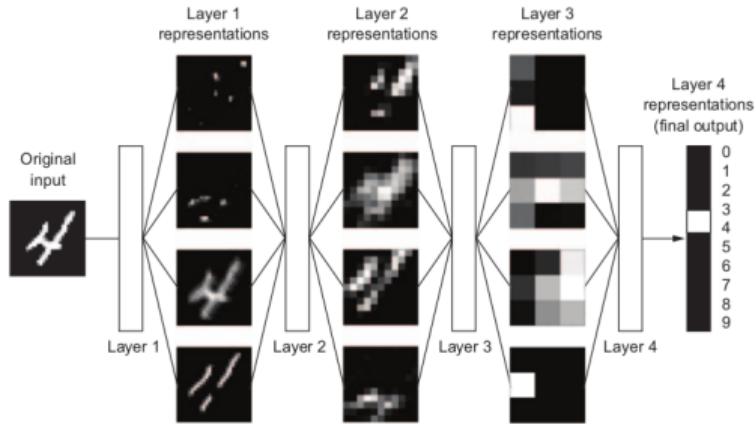
- Desde el Perceptrón hasta las Redes Neuronales
- Gradient descent
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

Introducción

El Deep Learning o aprendizaje profundo se basa en las Redes Neuronales (profundas) para resolver problemas en el ámbito de la Inteligencia Artificial.



Red neuronal es una función que transforma un input en un output. Esta formada por capas, transforma imágenes hasta llegar a un output. Es necesario entrenar a las capas para mejorar la red neuronal.

Figure: Red Neuronal

IA vs ML vs DL

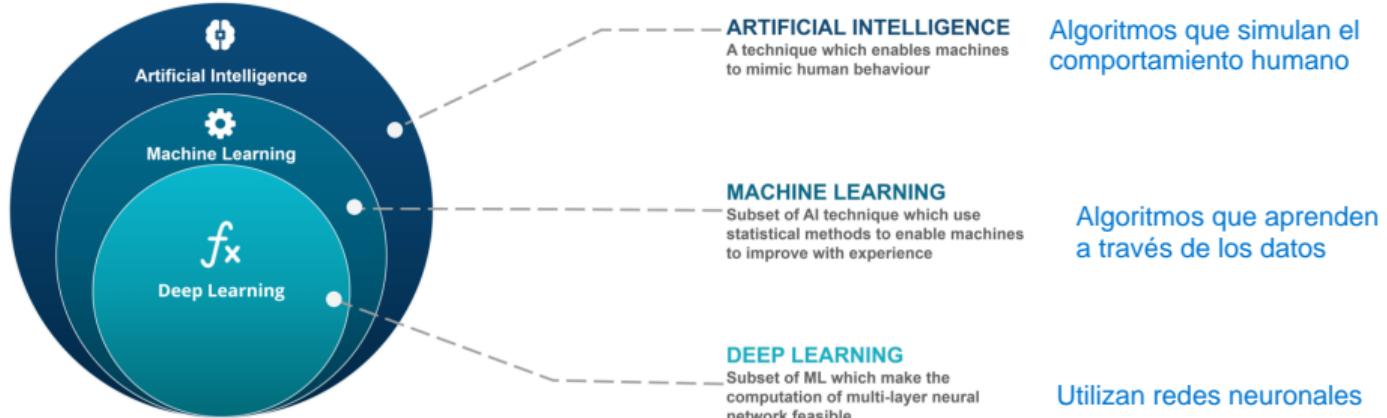


Figure: IA vs ML vs DL

Machine Learning

nos vamos a centrar en algoritmos supervisados durante esta asignatura

APRENDEN SACAR
INFORMACIÓN DE
DATOS SIN
PEDIRSELOS



Introducción

Actualmente hay un gran interés en la Inteligencia Artificial, en particular en el Deep Learning, el cual podemos observar en conferencias y anuncios. Esto ha sido debido gracias a que el Deep Learning ha conseguido superar ciertos hitos hasta ahora no logrados:

Introducción

Actualmente hay un gran interés en la Inteligencia Artificial, en particular en el Deep Learning, el cual podemos observar en conferencias y anuncios. Esto ha sido debido gracias a que el Deep Learning ha conseguido superar ciertos hitos hasta ahora no logrados:

Example

- Detección de objetos Asimila objetos con imágenes, ha ayudado a la conducción autónoma
- Vencer al campeón del mundo de GO
- Conducción autónoma
- Procesamiento del lenguaje natural

Introducción

Actualmente hay un gran interés en la Inteligencia Artificial, en particular en el Deep Learning, el cual podemos observar en conferencias y anuncios. Esto ha sido debido gracias a que el Deep Learning ha conseguido superar ciertos hitos hasta ahora no logrados:

Example

- Detección de objetos
- Vencer al campeón del mundo de GO
- Conducción autónoma
- Procesamiento del lenguaje natural

Introducción

Actualmente hay un gran interés en la Inteligencia Artificial, en particular en el Deep Learning, el cual podemos observar en conferencias y anuncios. Esto ha sido debido gracias a que el Deep Learning ha conseguido superar ciertos hitos hasta ahora no logrados:

Example

- Detección de objetos
- Vencer al campeón del mundo de GO
- Conducción autónoma
- Procesamiento del lenguaje natural

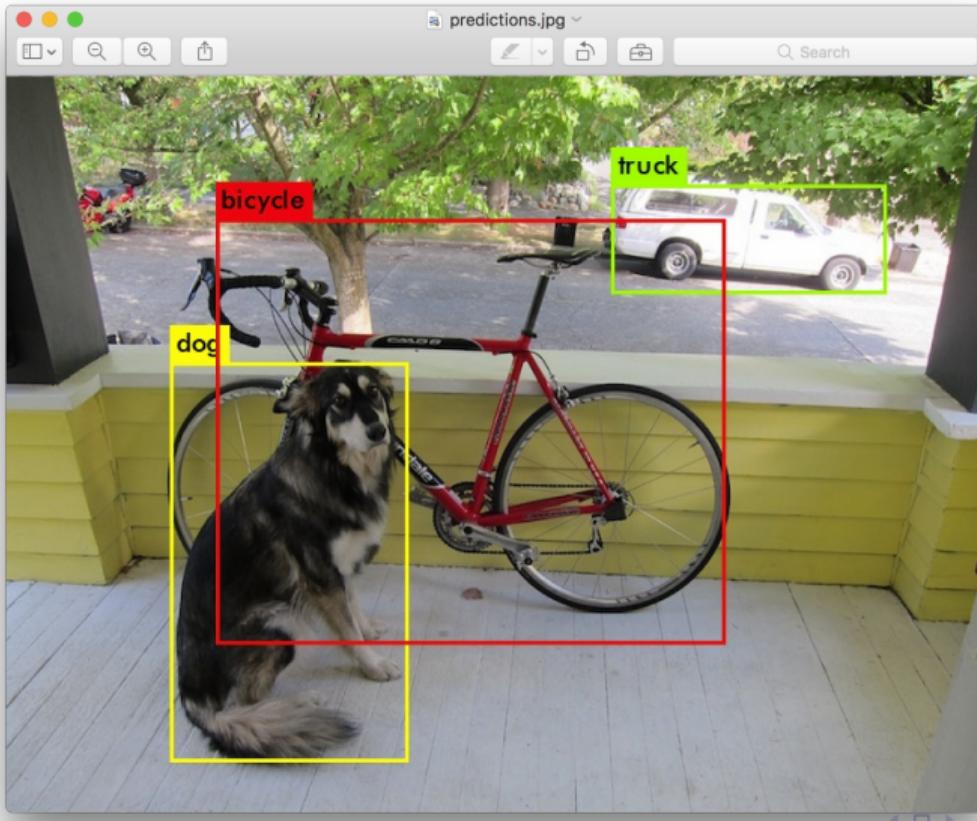
Introducción

Actualmente hay un gran interés en la Inteligencia Artificial, en particular en el Deep Learning, el cual podemos observar en conferencias y anuncios. Esto ha sido debido gracias a que el Deep Learning ha conseguido superar ciertos hitos hasta ahora no logrados:

Example

- Detección de objetos
- Vencer al campeón del mundo de GO
- Conducción autónoma
- Procesamiento del lenguaje natural

Detección de objetos



Vencer al campeón del mundo de GO



Figure: YouTube: AlphaGo - The movie

Conducción autónoma

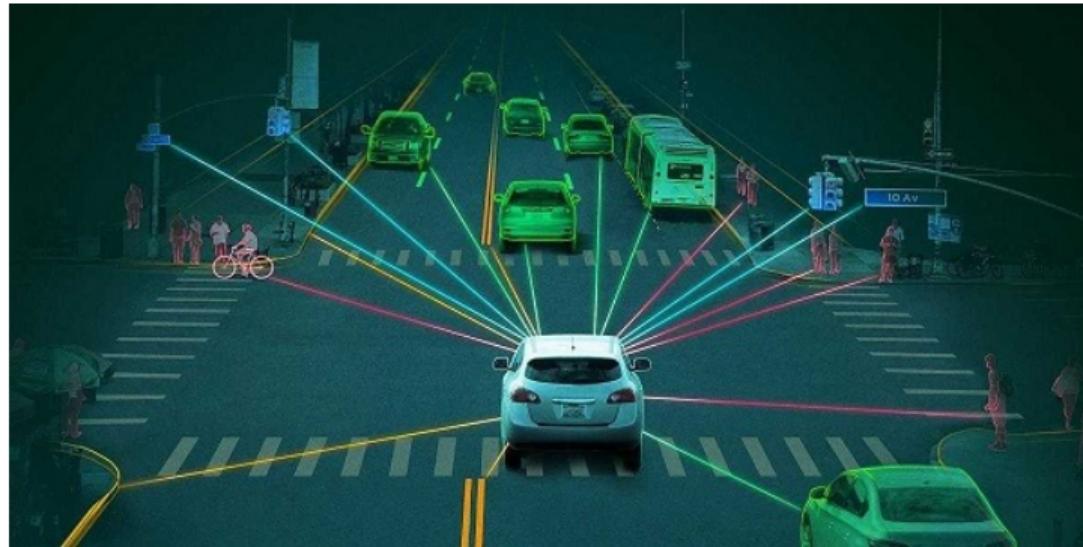


Figure: Sistema de conducción autónoma

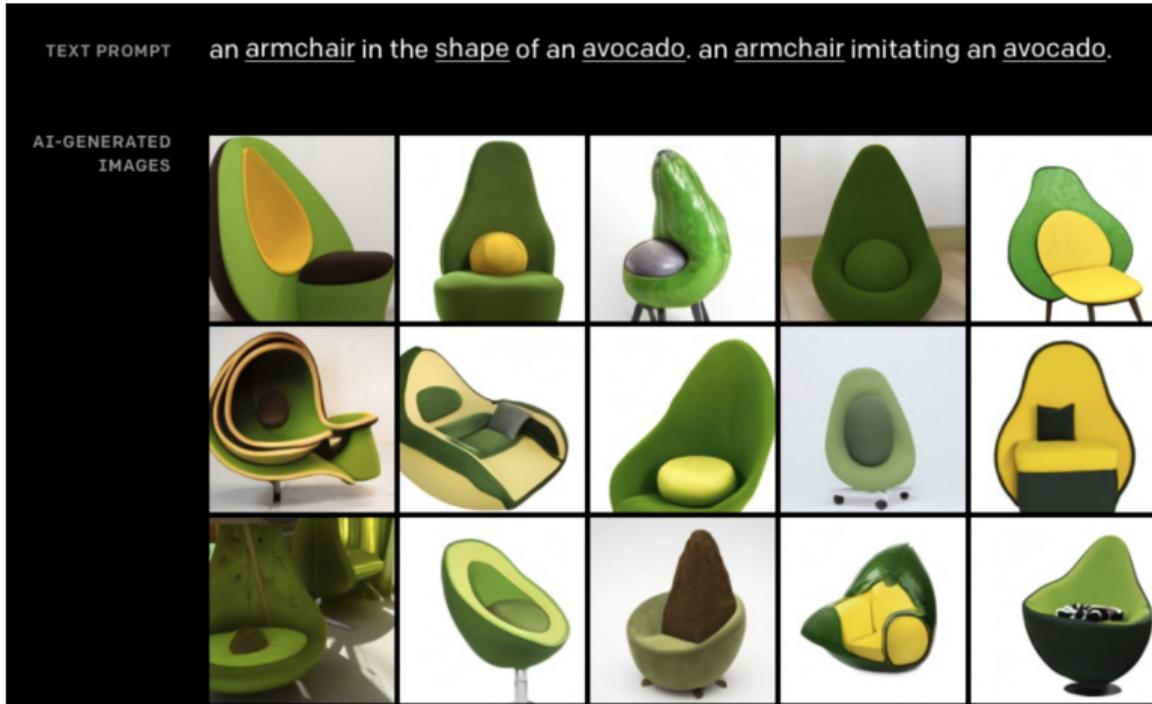


Figure: Dall-E

Índice

1 Introducción

- Motivación y características principales de las Redes Neuronales
- Funcionamiento y estructura
- Áreas de aplicación

2 Introducción a las Redes Neuronales

- Desde el Perceptrón hasta las Redes Neuronales
- Gradient descent
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

¿Qué mejoras presenta frente al Machine Learning?

El DL ha conseguido resolver problemas que el ML no había sido capaz, gracias a la estructura jerárquica de aprendizaje.

Por ello, actualmente hay cientos de proyectos de DL en investigación (DeepMind, OpenAI, Microsoft...), muy prometedores.

Motivación

¿Qué mejoras presenta frente al Machine Learning?

El DL ha conseguido resolver problemas que el ML no había sido capaz, gracias a la estructura jerárquica de aprendizaje.

Por ello, actualmente hay cientos de proyectos de DL en investigación (DeepMind, OpenAI, Microsoft...), muy prometedores. [es capaz de detectar patrones mucho mejor que los que se usaban antes](#)

¿Por qué?

Las Redes Neuronales tienen una gran capacidad para aprender patrones complejos (pueden computar casi cualquier función) y un proceso de aprendizaje incremental (pueden ser entrenados en datasets enormes).

¿Por qué ahora?

El DL se mantuvo apartado por el gran coste y dificultad de entrenar los modelos, por lo que fue eclipsado por los logros del ML. Esto cambió por:

- Aparición de las GPU's
- Grandes datasets públicos
- Optimización de los procesos de aprendizaje y mejoras en la arquitectura.

Empresas meten grandes inversiones para investigar en redes neuronales

Adiós al feature engineering

Una de las características principales de las NN es la capacidad de **extraer características representativas automáticamente** gracias a su **aprendizaje incremental** generado por la propia estructura de la red.

Se crea un proceso end-to-end de aprendizaje, necesario para problemas complejos.

- Machine Learning

$\text{data} \rightarrow \text{features} \rightarrow \text{model} \rightarrow \text{output}$

- Deep Learning

$\text{data} \rightarrow \text{model} \rightarrow \text{output}$

Adiós al feature engineering

Una de las características principales de las NN es la capacidad de **extraer características representativas automáticamente** gracias a su **aprendizaje incremental** generado por la propia estructura de la red.

Se crea un proceso end-to-end de aprendizaje, necesario para problemas complejos.

- Machine Learning

$$\textit{data} \rightarrow \textit{features} \rightarrow \textit{model} \rightarrow \textit{output}$$

- Deep Learning

saca características y saca un output, se generan automáticamente

$$\textit{data} \rightarrow \textit{model} \rightarrow \textit{output}$$

Motivación

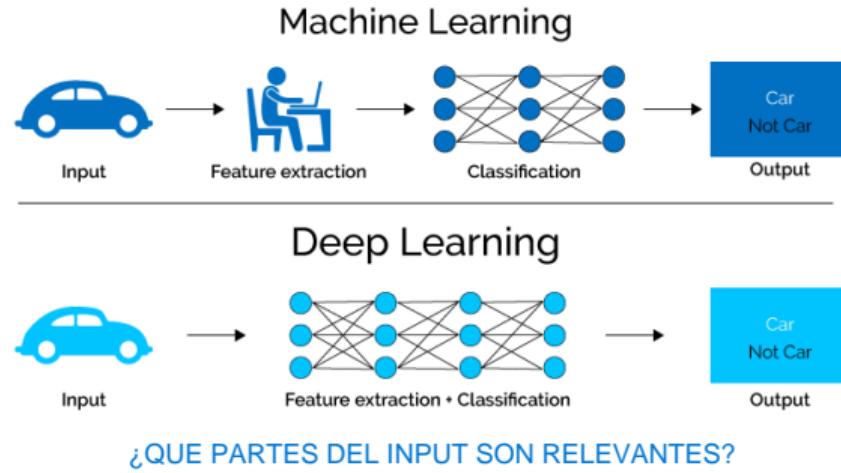


Figure: ML vs DL

Índice

1 Introducción

- Motivación y características principales de las Redes Neuronales
- **Funcionamiento y estructura**
- Áreas de aplicación

2 Introducción a las Redes Neuronales

- Desde el Perceptrón hasta las Redes Neuronales
- Gradient descent
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

input-output

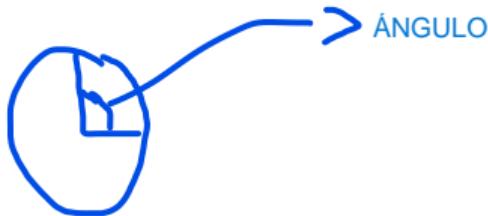
Los modelos de ML “aprenden” a transformar el input en el output principalmente en 2 etapas:

- ① Cambio del espacio de representación de los datos. (PCA, Normalización, Clustering, feature engineering...)
- ② Función para ir del espacio de representación al espacio del output. (SVM, árbol de decisión...)

Vamos a ver como el DL transforma el input en el output directamente usando neuronas artificiales.

Funcionamiento ML

Dado un ángulo que hora es con un rango de 0 a 24



input-output

Los modelos de ML “aprenden” a transformar el input en el output principalmente en 2 etapas:

- ① Cambio del espacio de representación de los datos. (PCA, Normalización, Clustering, feature engineering...) Primero tenemos un input en crudo sin transformación, tiene que pasar por las capas
- ② Función para ir del espacio de representación al espacio del output. (SVM, árbol de decisión...)

Vamos a ver como el DL transforma el input en el output directamente usando neuronas artificiales.

Funcionamiento ML

input-output

Los modelos de ML “aprenden” a transformar el input en el output principalmente en 2 etapas:

- ① Cambio del espacio de representación de los datos. (PCA, Normalización, Clustering, feature engineering...)
- ② Función para ir del espacio de representación al espacio del output. (SVM, árbol de decisión...) [Aplicar al modelo](#)

Vamos a ver como el DL transforma el input en el output directamente usando neuronas artificiales.

input-output

Los modelos de ML “aprenden” a transformar el input en el output principalmente en 2 etapas:

- ① Cambio del espacio de representación de los datos. (PCA, Normalización, Clustering, feature engineering...)
- ② Función para ir del espacio de representación al espacio del output. (SVM, árbol de decisión...)

Vamos a ver como el DL transforma el input en el output directamente usando neuronas artificiales.

Estructura

Neurona

Una neurona es una función de la forma $y = \varphi(\omega \cdot x + b)$. Es la unidad de computo más pequeña dentro de una NN. Los parámetros de la neurona son ω y b . La función de activación φ se encarga de 'apagar' o 'encender' la neurona.

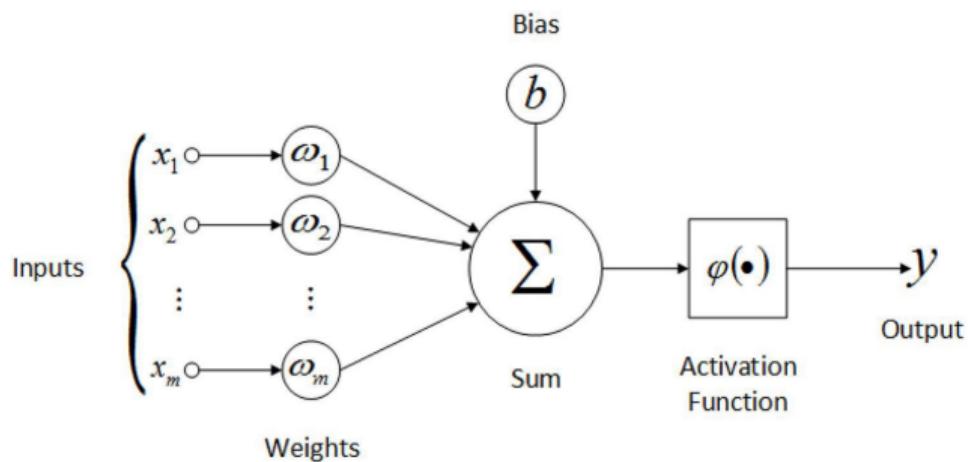


Figure: Neurona

Estructura

Las neuronas son funciones que resuelven preguntas muy sencillas

Neurona

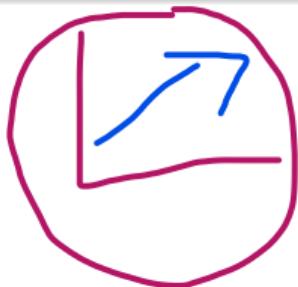
- ① Las neuronas son capaces de responder a preguntas muy sencillas, son como puertas lógicas ponderadas. Por ejemplo, aceptaría este trabajo si...
- ② Las neuronas también pueden verse como un detector de patrones (Producto escalar).

Cuantas más neuronas y capas tenga la red más complejos serán los patrones que detecte.

Neurona

- ① Las neuronas son capaces de responder a preguntas muy sencillas, son como puertas lógicas ponderadas. Por ejemplo, aceptaría este trabajo si... [Detector de patrones](#)
- ② Las neuronas también pueden verse como un detector de patrones (Producto escalar).

Cuantas más neuronas y capas tenga la red más complejos serán los patrones que detecte.



donde esta un vector en una franja

Neurona

- ① Las neuronas son capaces de responder a preguntas muy sencillas, son como puertas lógicas ponderadas. Por ejemplo, aceptaría este trabajo si...
- ② Las neuronas también pueden verse como un detector de patrones (Producto escalar).

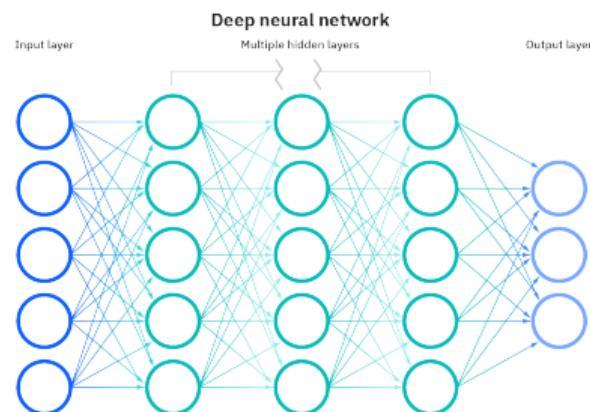
Cuantas más neuronas y capas tenga la red más complejos serán los patrones que detecte.

Estructura

Red Neuronal

Una Red Neuronal está formada por una serie de capas constituidas por neuronas. Estas capas procesan o transforman los datos para obtener la asociación $x \rightarrow y$. Cuantas más capas, más compleja puede ser dicha asociación. Normalmente las NN tienen decenas de capas y cada capa contiene centenas de neuronas.

Cada capa es un conjunto de neuronas, las neuronas de una misma capa no se conectan entre sí.



Cuanto más complejo sea el problema más capas son necesarias, para poder resolver problemas

Figure: Red Neuronal

Entrenamiento de una Red Neuronal

Como elegimos los parámetros y por ende como entrenamos al modelos

¿Cómo obtener una Red Neuronal útil?

El método general de entrenamiento es el supervisado. Modificaremos los pesos de la red θ ; de manera iterativa comparando las predicciones y' y los targets reales y . Así obtendremos una sucesión

$$\theta_0 \rightarrow \theta_1 \rightarrow \theta_2 \rightarrow \dots \rightarrow \theta_n$$

meter datos etiquetados y vamos modificando los pesos de la red y estamos mejor

Idealmente queremos que nuestra red f_{θ_i} converja a una función f tal que

$$f(x_i) = y_i \quad \forall x_i \in TestSet$$

Es un proceso incremental de aprendizaje no determinista.

Entrenamiento de una Red Neuronal

¿Cómo obtener una Red Neuronal útil?

El método general de entrenamiento es el supervisado. Modificaremos los pesos de la red θ ; de manera iterativa comparando las predicciones y' y los targets reales y . Así obtendremos una sucesión

$$\theta_0 \rightarrow \theta_1 \rightarrow \theta_2 \rightarrow \dots \rightarrow \theta_n \quad \text{mejorando pesos poco a poco}$$

Idealmente queremos que nuestra red f_{θ_i} converja a una función f tal que

$$f(x_i) = y_i \quad \forall x_i \in TestSet$$

Es un proceso incremental de aprendizaje no determinista.

Entrenamiento de una Red Neuronal

¿Cómo obtener una Red Neuronal útil?

El método general de entrenamiento es el supervisado. Modificaremos los pesos de la red θ ; de manera iterativa comparando las predicciones y' y los targets reales y . Así obtendremos una sucesión

$$\theta_0 \rightarrow \theta_1 \rightarrow \theta_2 \rightarrow \dots \rightarrow \theta_n$$

Idealmente queremos que nuestra red f_{θ_i} converja a una función f tal que

$$f(x_i) = y_i \quad \forall x_i \in TestSet$$

Dos personas que entrenen el mismo modelo con los mismos datos no tiene porque terminar con el mismo proyecto final, ya que hay muchos procesos aleatorios dentro de cada red neuronal. Es cierto que van a ser muy similares

Es un proceso incremental de aprendizaje no determinista.

Entrenamiento de una Red Neuronal

Si el error es pequeño no se deben de modificar los pesos

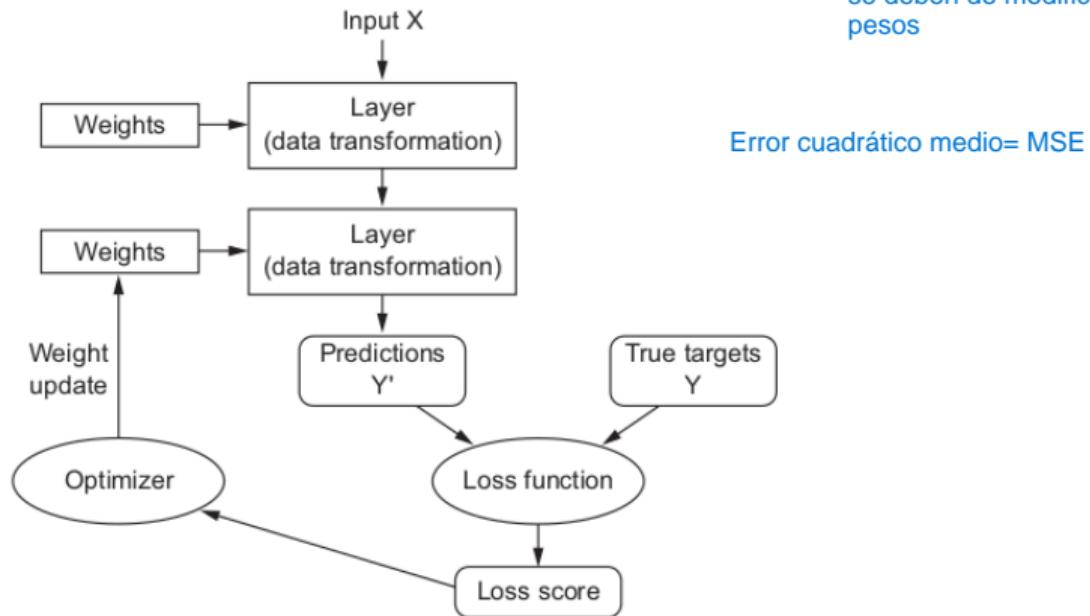


Figure: Fundamentos Deep Learning

Índice

1 Introducción

- Motivación y características principales de las Redes Neuronales
- Funcionamiento y estructura
- Áreas de aplicación

2 Introducción a las Redes Neuronales

- Desde el Perceptrón hasta las Redes Neuronales
- Gradient descent
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

Áreas de aplicación

Ventajas del DL

- ① Son capaces de resolver problemas o detectar patrones muy complejos, ya que pueden aproximar cualquier función.
- ② Son ideales para entornos con una cantidad elevada de datos etiquetados.
- ③ No hace falta un feature engineering complejo ni un gran conocimiento del problema.
- ④ El aprendizaje es incremental, por lo que se puede reanudar e ir mejorando.

Áreas de aplicación

Clasificación, regresión, NLP, segmentación, detección, reinforcement... ¿todas?

Áreas de aplicación

Ventajas del DL

- ① Son capaces de resolver problemas o detectar patrones muy complejos, ya que pueden aproximar cualquier función.
- ② Son ideales para entornos con una cantidad elevada de datos etiquetados.
- ③ No hace falta un feature engineering complejo ni un gran conocimiento del problema.
- ④ El aprendizaje es incremental, por lo que se puede reanudar e ir mejorando.

Áreas de aplicación

Clasificación, regresión, NLP, segmentación, detección, reinforcement... ¿todas?

PROCESAMIENTO DE LENGUAJE NATURAL

Áreas de aplicación

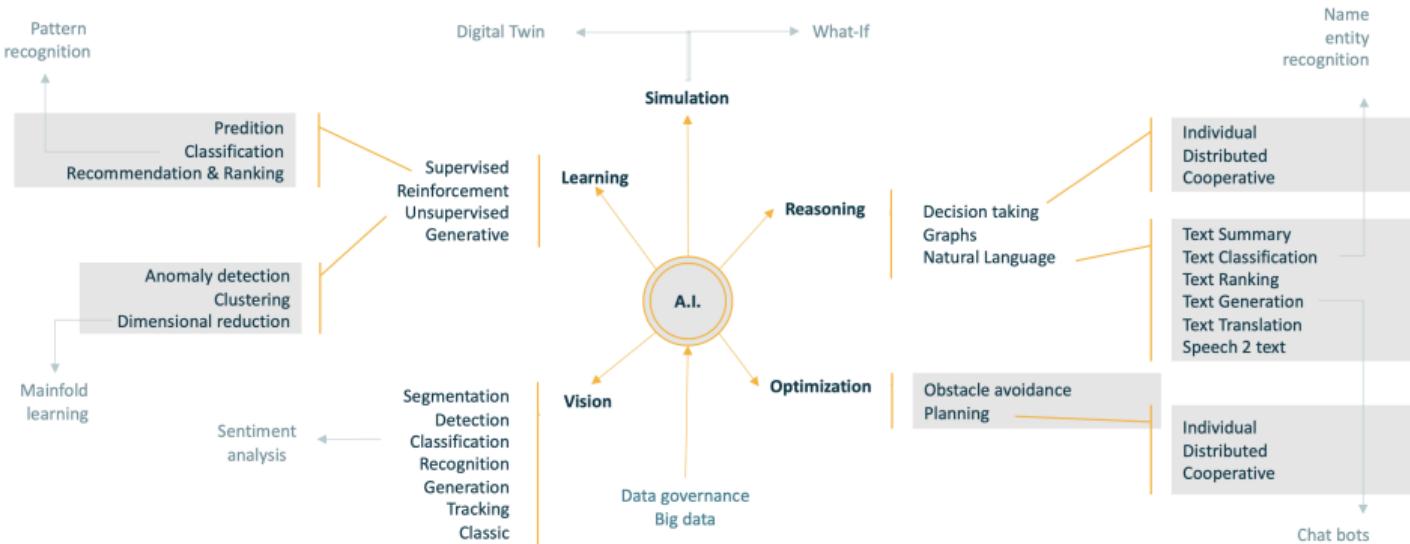


Figure: Taxonomía

Áreas de aplicación

Ventajas e inconvenientes

Ventajas prácticas del DL

- ① Gracias a los frameworks actuales como Keras, TensorFlow, Theano... es muy fácil implementar este tipo de soluciones.
- ② Gran variedad de enfoques, desde muy simples (clasificador) hasta muy complejas (detección de objetos)
- ③ Son totalmente escalables en predicción y entrenamiento!
- ④ Son reutilizables (transfer learning)

Inconvenientes del DL

- ① No se debe usar en entornos con pocos datos.
- ② Modelos más “simples” de ML obtienen un score similar. (stacking)
- ③ Son una “caja negra”. (relativo) son poco explicables, son un poco opacos
- ④ Coste computacional. SOBREAJUSTE, no es robusto, no generaliza, ESTAS SOBREENTRENANDO
- ⑤ Problema de dimensionalidad y overfitting. (relacionado con el tamaño del dataset)
- ⑥ Fine tuning más complejo. (es un problema práctico)

Un hiperparámetro, son una configuración que no puedes entrenar

Áreas de aplicación

Inconvenientes del DL

- ① No se debe usar en entornos con pocos datos.
- ② Modelos más “simples” de ML obtienen un score similar. (stacking)
- ③ Son una “caja negra”. (relativo)
- ④ Coste computacional.
- ⑤ Problema de dimensionalidad y overfitting. (relacionado con el tamaño del dataset)
- ⑥ Fine tuning más complejo. (es un problema práctico)

[Data Augmentation \(entrenar modelos con pocos datos\)](#)

Mitigación

Estos problemas se pueden mitigar, en cierta medida, con distintas técnicas como Data Augmentation, modelos optimizados...

Inconvenientes comunes con los modelos de ML

- ① Son soluciones muy concretas a un problema bien definido. **No son modelos universales.**
- ② Son difíciles de implementar en determinados entornos por la sensibilidad de los datos, como por ejemplo en medicina. También son difíciles de implementar en sectores muy críticos, como por ejemplo en procesos de una central nuclear o en aviones.

Los modelos siempre dan errores

Índice

1 Introducción

- Motivación y características principales de las Redes Neuronales
- Funcionamiento y estructura
- Áreas de aplicación

2 Introducción a las Redes Neuronales

- Desde el Perceptrón hasta las Redes Neuronales
- Gradient descent
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

Resumen

Vamos a ver un poco más en detalle la estructura de una NN, las funciones loss y el algoritmo de aprendizaje, esenciales para entender cómo se entrena una red neuronal.

Índice

1 Introducción

- Motivación y características principales de las Redes Neuronales
- Funcionamiento y estructura
- Áreas de aplicación

2 Introducción a las Redes Neuronales

- Desde el Perceptrón hasta las Redes Neuronales
- Gradient descent
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

Origen de las Neuronas

Neuronas biológicas vs artificiales

Las redes neuronales están basadas en la estructura cerebral. Las neuronas biológicas funcionan con impulsos eléctricos y se activan o desactivan en función de esos impulsos. Las neuronas artificiales emulan este comportamiento.

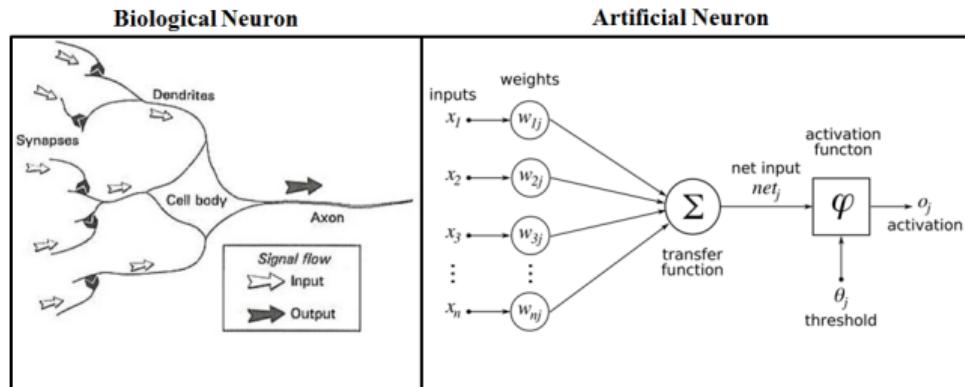


Figure: Neuronas biológicas vs artificiales

Perceptrón

El perceptrón es una neurona artificial cuya función de activación es la Step Function, por lo que su función asociada es:

$$f_{\theta}(x) = \begin{cases} 1 & \sum x_i \theta_i - b \geq 0 \\ 0 & \sum x_i \theta_i - b < 0 \end{cases}$$

Es la arquitectura de red neuronal más simple. Se puede usar como un SVM para clasificación binaria.

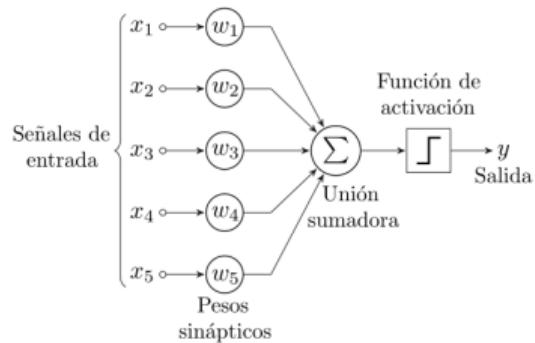


Figure: Perceptrón

Perceptrón

Entrenamiento del perceptrón

El entrenamiento del perceptrón es un proceso iterativo en el que se evalúa en cada paso un par (x, y) y se modifican los pesos según la ecuación:

$$\theta_{n+1} = \theta_n + \alpha(y - \tilde{y})x$$

El parámetro α mide el grado de convergencia.

Entrenamiento Perceptrón Link

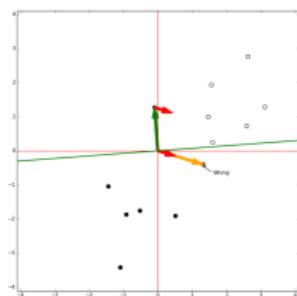


Figure: Entrenamiento perceptrón

Perceptrón multicapa

Perceptrón multicapa

Aunque el perceptrón se usaba en ciertos problemas simples, su utilidad estaba muy limitada. En cambio, si conectamos varios perceptrones y los apilamos, generamos un perceptrón multicapa, una arquitectura de NN más rica capaz de resolver problemas más complejos.

Este tipo de arquitectura tampoco llegó a proliferar por la dificultad de entrenar el modelo. Principalmente porque la función de activación es no continua y tiene derivada nula en casi todo punto. Para los algoritmos de optimización es muy difícil tratar con este tipo de funciones.

Perceptrón multicapa

Perceptrón multicapa

Aunque el perceptrón se usaba en ciertos problemas simples, su utilidad estaba muy limitada. En cambio, si conectamos varios perceptrones y los apilamos, generamos un perceptrón multicapa, una arquitectura de NN más rica capaz de resolver problemas más complejos.

Este tipo de arquitectura tampoco llegó a proliferar por la dificultad de entrenar el modelo. Principalmente porque la función de activación es no continua y tiene derivada nula en casi todo punto. Para los algoritmos de optimización es muy difícil tratar con este tipo de funciones.

Sigmoid

¿Cómo solucionar este problema?

Para resolver este problema podemos suavizar (hacerla diferenciable) la Step Function usando la función Sigmoid, que es C^∞ y es una buena aproximación. Su ecuación es:

$$s(x) = \frac{1}{1 + e^{-x}}$$

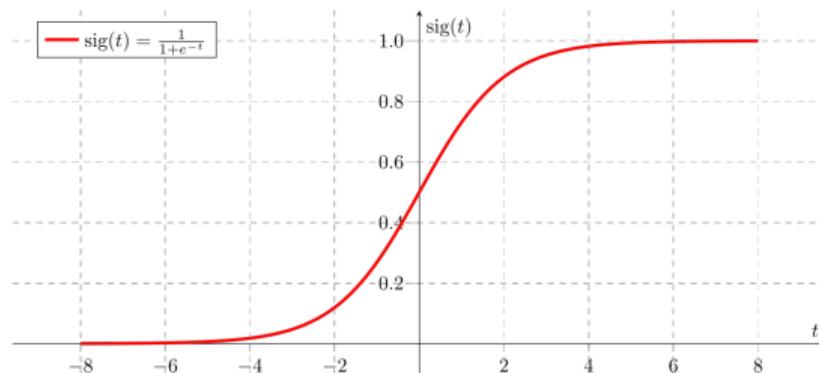


Figure: Sigmoid

Red Neuronal Totalmente Conectada

FCNN (Fully Connected Neural Network)

Una Red Neuronal con al menos una capa oculta, con todas las neuronas conectadas entre capas consecutivas y funciones de activación continuas y diferenciables en casi todo punto (como por ejemplo la sigmoid) se llama Red Neuronal Totalmente Conectada. Es el tipo de arquitectura funcional más básica.

Este tipo de arquitectura es totalmente funcional y se usa en muchos casos reales.

Red Neuronal Totalmente Conectada

FCNN (Fully Connected Neural Network)

Una Red Neuronal con al menos una capa oculta, con todas las neuronas conectadas entre capas consecutivas y funciones de activación continuas y diferenciables en casi todo punto (como por ejemplo la sigmoid) se llama Red Neuronal Totalmente Conectada. Es el tipo de arquitectura funcional más básica.

Este tipo de arquitectura es totalmente funcional y se usa en muchos casos reales.

Example

[Playground Link](#)

Índice

1 Introducción

- Motivación y características principales de las Redes Neuronales
- Funcionamiento y estructura
- Áreas de aplicación

2 Introducción a las Redes Neuronales

- Desde el Perceptrón hasta las Redes Neuronales
- **Gradient descent**
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

Entrenamiento de una NN

Estructura de la red

Cada capa en la red es de la forma $c_i = \sigma(A_i c_{i-1} + b_i) = r_i(c_{i-1})$, por tanto la red será composición de dichas funciones.

$$\tilde{y} = f_\theta(x) = r_n \circ r_{n-1} \circ \cdots \circ r_1(x)$$

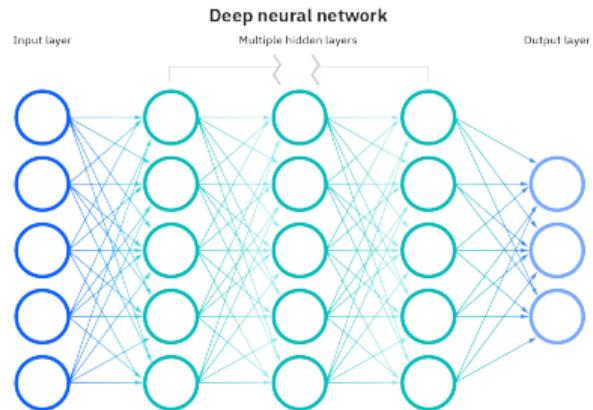


Figure: Red Neuronal

Entrenamiento de una NN

Relación x e y

Inicialmente la relación entre x e \tilde{y} es aleatoria (los pesos de la red son aleatorios), por lo que las predicciones \tilde{y} y los target reales y no son coherentes.

Necesitamos un proceso de entrenamiento para modificar los pesos para que el output \tilde{y} sea el deseado.

Entrenamiento de una NN

Relación x e y

Inicialmente la relación entre x e \hat{y} es aleatoria (los pesos de la red son aleatorios), por lo que las predicciones \hat{y} y los target reales y no son coherentes.

Necesitamos un proceso de entrenamiento para modificar los pesos para que el output \hat{y} sea el deseado.

Entrenamiento de NN

Entrenamiento

El proceso de entrenamiento es iterativo y sigue los siguientes pasos:

- ① Computar $\tilde{y} = f_{\theta}(x)$,
- ② Comparar \tilde{y} e y usando la función loss.
- ③ Modificar los pesos θ de la red para reducir el loss.

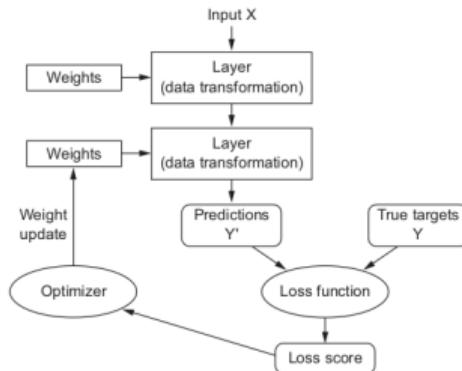


Figure: Fundamentos Deep Learning

Entrenamiento de NN

Funciones loss

Una función de perdida o loss function, es una función diferenciable en casi todo punto asociada a una métrica u objetivo. Las funciones loss más comunes son:

- Binary Cross Entropy, para clasificación:

$$BCE(y, \tilde{y}) = -(y \log \tilde{y} + (1 - y) \log(1 - \tilde{y}))$$

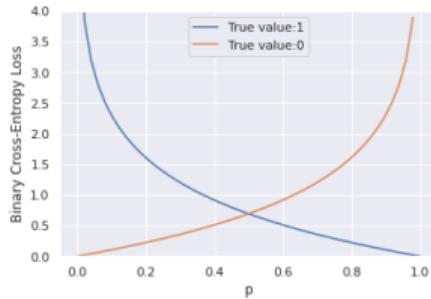


Figure: Binary Cross Entropy

Funciones loss

- Mean Squared Error, para regresión:

$$MSE(y, \tilde{y}) = (y - \tilde{y})^2$$

Keras Losses Link

Utilizamos la loss para entrenar el modelo y mejorar de manera indirecta la métrica asociada a nuestro objetivo. Por lo que debe existir una dependencia monótona entre ambas. En general es difícil encontrar funciones loss con buenas propiedades.

Funciones loss

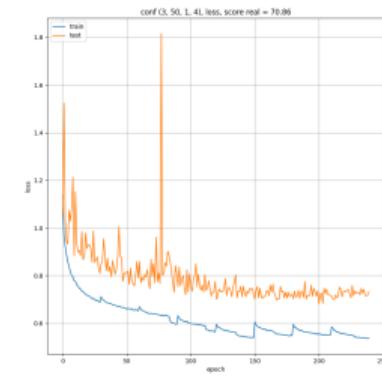
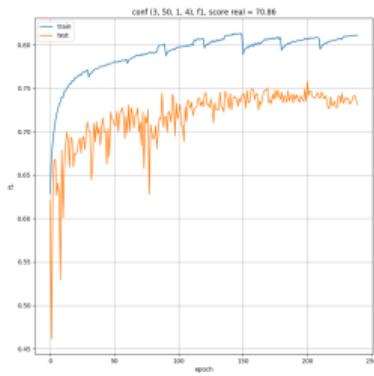
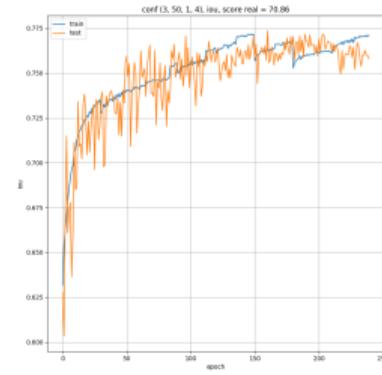
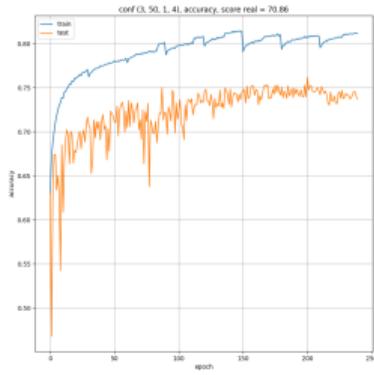
- Mean Squared Error, para regresión:

$$MSE(y, \tilde{y}) = (y - \tilde{y})^2$$

Keras Losses Link

Utilizamos la loss para entrenar el modelo y mejorar de manera indirecta la métrica asociada a nuestro objetivo. Por lo que debe existir una dependencia monótona entre ambas. En general es difícil encontrar funciones loss con buenas propiedades.

Entrenamiento de NN



Entrenamiento de NN

¿Cómo utilizar la loss function?

Con lo visto anteriormente, podemos expresar la función loss de la siguiente manera (simplificando la notación):

$$L(y, \tilde{y}) = L(y, f_{\theta}(x)) = L(y, x, \theta)$$

Fijado un punto (x_0, y_0) del dataset podemos ver L como función de θ .

$$L(y_0, f_{\theta}(x_0)) = L(y_0, x_0, \theta) = L_{y_0, x_0}(\theta)$$

Nuestro objetivo es minimizar el error cometido, i.e., minimizar L respecto de θ .

¿Cómo utilizar la loss function?

Con lo visto anteriormente, podemos expresar la función loss de la siguiente manera (simplificando la notación):

$$L(y, \tilde{y}) = L(y, f_\theta(x)) = L(y, x, \theta)$$

Fijado un punto (x_0, y_0) del dataset podemos ver L como función de θ .

$$L(y_0, f_\theta(x_0)) = L(y_0, x_0, \theta) = L_{y_0, x_0}(\theta)$$

Nuestro objetivo es minimizar el error cometido, i.e., minimizar L respecto de θ .

Entrenamiento de NN

¿Cómo utilizar la loss function?

Con lo visto anteriormente, podemos expresar la función loss de la siguiente manera (simplificando la notación):

$$L(y, \tilde{y}) = L(y, f_{\theta}(x)) = L(y, x, \theta)$$

Fijado un punto (x_0, y_0) del dataset podemos ver L como función de θ .

$$L(y_0, f_{\theta}(x_0)) = L(y_0, x_0, \theta) = L_{y_0, x_0}(\theta)$$

Nuestro objetivo es minimizar el error cometido, i.e., minimizar L respecto de θ .

Entrenamiento de NN

$$\nabla_{\theta} L$$

El Cálculo Diferencial nos asegura que el vector gradiente de la función L

$$\nabla_{\theta} L = \left(\frac{\partial L}{\partial \theta_i} \right)_i$$

es la dirección de máximo crecimiento local de L respecto de la variable θ . ¿Por qué?

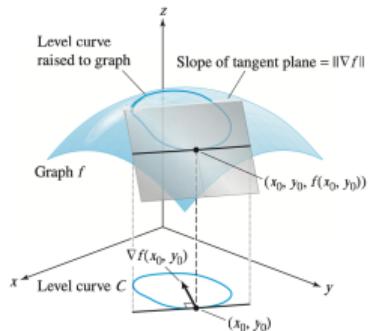


Figure: $\nabla_{\theta} L$

Entrenamiento de NN

$$\nabla_{\theta} L$$

Por tanto, si estamos en θ_0 y pasamos a

$$\theta_0 - \epsilon \nabla_{\theta} L|_{\theta_0}$$

el valor de L se reducirá, y esta reducción será óptima.

Algoritmo Gradient Descent

El algoritmo Gradient Descent es un algoritmo de optimización que usa el $\nabla_{\theta} L$ para minimizar L , siguiendo de manera iterativa los siguientes pasos:

- ① Calculamos (fijado un dataset (X, y))

$$\nabla_{\theta} L|_{\theta_i}$$

- ② Actualizamos los parámetros

$$\theta_{i+1} = \theta_i - \epsilon \nabla_{\theta} L|_{\theta_i}$$

Obtenemos una sucesión

$$\theta_0 \rightarrow \theta_1 \rightarrow \dots \rightarrow \theta_n$$

de tal manera que (fijado un dataset (X, y))

$$L_{\theta_0} \leq L_{\theta_1} \leq \dots \leq L_{\theta_n}$$

Entrenamiento de NN

Algoritmo Gradient Descent

El algoritmo Gradient Descent es un algoritmo de optimización que usa el $\nabla_{\theta} L$ para minimizar L , siguiendo de manera iterativa los siguientes pasos:

- ① Calculamos (fijado un dataset (X, y))

$$\nabla_{\theta} L|_{\theta_i}$$

- ② Actualizamos los parámetros

$$\theta_{i+1} = \theta_i - \epsilon \nabla_{\theta} L|_{\theta_i}$$

Obtenemos una sucesión

$$\theta_0 \rightarrow \theta_1 \rightarrow \dots \rightarrow \theta_n$$

de tal manera que (fijado un dataset (X, y))

$$L_{\theta_0} \leq L_{\theta_1} \leq \dots \leq L_{\theta_n}$$

Gradient Descent

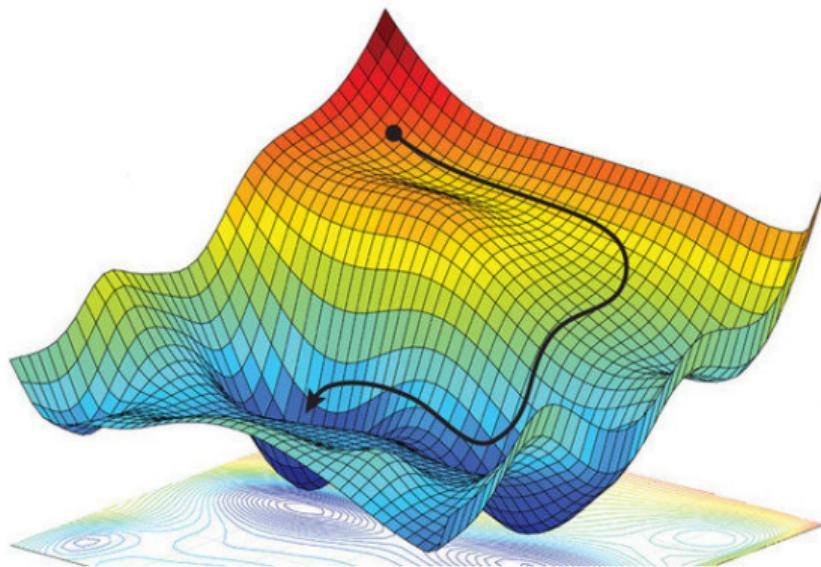


Figure: Gradient Descent

Requisitos

Para que el funcionamiento del algoritmo GD sea efectivo hay ciertos requisitos que se deben cumplir.

- ① La función loss será diferenciable en casi todo punto respecto de θ .
- ② Que tengan forma convexa para reducir los mínimos locales.
- ③ La elección del learning rate α debe ser correcta para que no se quede atascado en el entrenamiento o la dinámica de aprendizaje sea errática.
- ④ Tanto la dimensión del espacio de θ como la complejidad de la función $L(\theta)$ no deben de ser elevadas.

Requisitos

Para que el funcionamiento del algoritmo GD sea efectivo hay ciertos requisitos que se deben cumplir.

- ① La función loss será diferenciable en casi todo punto respecto de θ .
- ② Que tengan forma convexa para reducir los mínimos locales.
- ③ La elección del learning rate α debe ser correcta para que no se quede atascado en el entrenamiento o la dinámica de aprendizaje sea errática.
- ④ Tanto la dimensión del espacio de θ como la complejidad de la función $L(\theta)$ no deben de ser elevadas.

Requisitos

Para que el funcionamiento del algoritmo GD sea efectivo hay ciertos requisitos que se deben cumplir.

- ① La función loss será diferenciable en casi todo punto respecto de θ .
- ② Que tengan forma convexa para reducir los mínimos locales.
- ③ La elección del learning rate α debe ser correcta para que no se quede atascado en el entrenamiento o la dinámica de aprendizaje sea errática.
- ④ Tanto la dimensión del espacio de θ como la complejidad de la función $L(\theta)$ no deben de ser elevadas.

Requisitos

Para que el funcionamiento del algoritmo GD sea efectivo hay ciertos requisitos que se deben cumplir.

- ① La función loss será diferenciable en casi todo punto respecto de θ .
- ② Que tengan forma convexa para reducir los mínimos locales.
- ③ La elección del learning rate α debe ser correcta para que no se quede atascado en el entrenamiento o la dinámica de aprendizaje sea errática.
- ④ Tanto la dimensión del espacio de θ como la complejidad de la función $L(\theta)$ no deben de ser elevadas.

Requisitos

Para que el funcionamiento del algoritmo GD sea efectivo hay ciertos requisitos que se deben cumplir.

- ① La función loss será diferenciable en casi todo punto respecto de θ .
- ② Que tengan forma convexa para reducir los mínimos locales.
- ③ La elección del learning rate α debe ser correcta para que no se quede atascado en el entrenamiento o la dinámica de aprendizaje sea errática.
- ④ Tanto la dimensión del espacio de θ como la complejidad de la función $L(\theta)$ no deben de ser elevadas.

Entrenamiento de NN

Variantes del GD

Existen distintas variantes de este algoritmo que optimizan el paso de actualización de los pesos para que la dinámica de aprendizaje sea óptima. Las más simples son:

- 1 Batch Gradient Descent: Actualiza los parámetros según:

$$v = \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} L_{|\theta_i}(x_i, y_i)$$

- 2 Mini Batch Gradient Descent: Actualiza los parámetros según:

$$v = \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} L_{|\theta_i}(x_i, y_i) \text{ con } ((x_i, y_i))_{i=1}^m \subset ((x_j, y_j))_{j=1}^n$$

- 3 Stochastic Gradient Descent: Actualiza los parámetros según:

$$v = \nabla_{\theta} L_{|\theta_i}(x_i, y_i) \text{ con } 1 \leq i \leq n$$

Entrenamiento de NN

Variantes del GD

Existen distintas variantes de este algoritmo que optimizan el paso de actualización de los pesos para que la dinámica de aprendizaje sea óptima. Las más simples son:

- ① Batch Gradient Descent: Actualiza los parámetros según:

$$v = \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} L_{|\theta_i}(x_i, y_i)$$

- ② Mini Batch Gradient Descent: Actualiza los parámetros según:

$$v = \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} L_{|\theta_i}(x_i, y_i) \text{ con } ((x_i, y_i))_{i=1}^m \subset ((x_j, y_j))_{j=1}^n$$

- ③ Stochastic Gradient Descent: Actualiza los parámetros según:

$$v = \nabla_{\theta} L_{|\theta_i}(x_i, y_i) \text{ con } 1 \leq i \leq n$$

Entrenamiento de NN

Variantes del GD

Existen distintas variantes de este algoritmo que optimizan el paso de actualización de los pesos para que la dinámica de aprendizaje sea óptima. Las más simples son:

- ① Batch Gradient Descent: Actualiza los parámetros según:

$$v = \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} L_{|\theta_i}(x_i, y_i)$$

- ② Mini Batch Gradient Descent: Actualiza los parámetros según:

$$v = \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} L_{|\theta_i}(x_i, y_i) \text{ con } ((x_i, y_i))_{i=1}^m \subset ((x_j, y_j))_{j=1}^n$$

- ③ Stochastic Gradient Descent: Actualiza los parámetros según:

$$v = \nabla_{\theta} L_{|\theta_i}(x_i, y_i) \text{ con } 1 \leq i \leq n$$

Entrenamiento de NN

Variantes del GD

Existen distintas variantes de este algoritmo que optimizan el paso de actualización de los pesos para que la dinámica de aprendizaje sea óptima. Las más simples son:

- ① Batch Gradient Descent: Actualiza los parámetros según:

$$v = \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} L_{|\theta_i}(x_i, y_i)$$

- ② Mini Batch Gradient Descent: Actualiza los parámetros según:

$$v = \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} L_{|\theta_i}(x_i, y_i) \text{ con } ((x_i, y_i))_{i=1}^m \subset ((x_j, y_j))_{j=1}^n$$

- ③ Stochastic Gradient Descent: Actualiza los parámetros según:

$$v = \nabla_{\theta} L_{|\theta_i}(x_i, y_i) \text{ con } 1 \leq i \leq n$$

Entrenamiento de NN

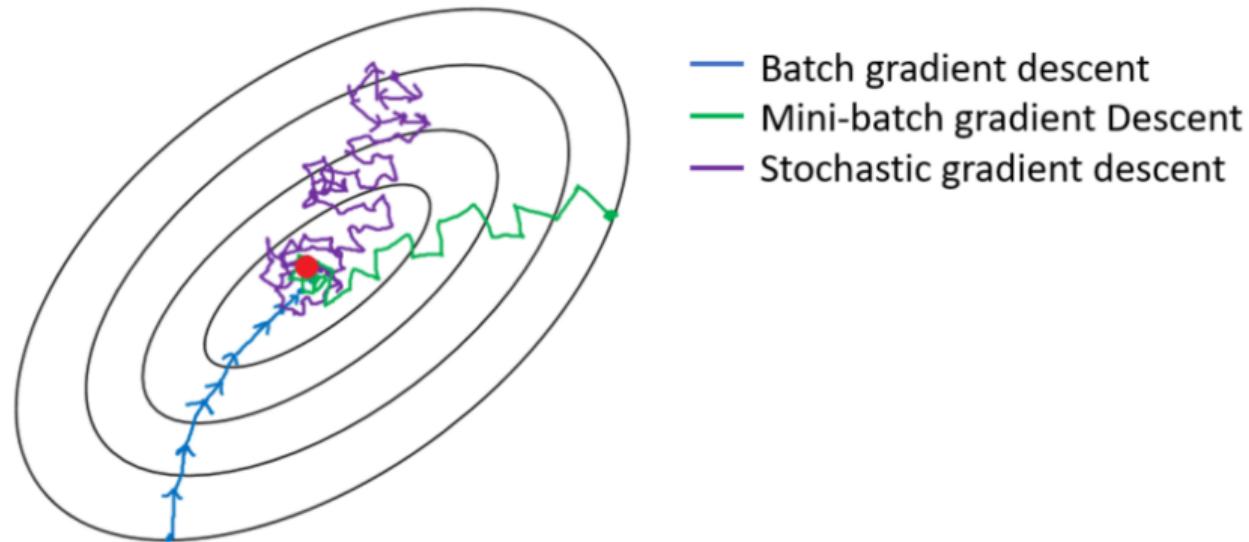


Figure: Variantes del GD

Índice

1 Introducción

- Motivación y características principales de las Redes Neuronales
- Funcionamiento y estructura
- Áreas de aplicación

2 Introducción a las Redes Neuronales

- Desde el Perceptrón hasta las Redes Neuronales
- Gradient descent
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

Cálculo del $\nabla_{\theta}L$

¿Cómo podemos calcular $\nabla_{\theta}L$?

Cuando el número de capas y de neuronas aumenta, la complejidad para calcular ∇L aumenta significativamente. El Backpropagation es un algoritmo que calcula de manera eficiente ∇L , de hecho fue uno de los avances que impulsó el DL.

Principalmente se basa en el uso de la regla de la cadena:

$$(f \circ g(x))' = f' \circ g(x) \cdot g'(x)$$

Ya que derivar directamente la expresión es intratable.

Cálculo del $\nabla_{\theta}L$

¿Cómo podemos calcular $\nabla_{\theta}L$?

Cuando el número de capas y de neuronas aumenta, la complejidad para calcular ∇L aumenta significativamente. El Backpropagation es un algoritmo que calcula de manera eficiente ∇L , de hecho fue uno de los avances que impulsó el DL.

Principalmente se basa en el uso de la regla de la cadena:

$$(f \circ g(x))' = f' \circ g(x) \cdot g'(x)$$

Ya que derivar directamente la expresión es intratable.

Cálculo del $\nabla_{\theta} L$

Backpropagation

Como ya hemos visto una NN puede representarse de la forma:

$$f_{\theta}(x) = r_n \circ r_{n-1} \circ \cdots \circ r_1(x) = \sigma_n(A_n(r_{n-1}(\dots)) + b_n)$$

donde

$$r_i = \sigma_i(A_i r_{i-1} + b_i)$$

con σ_i las funciones de activación y $\theta = \{\theta_i\}_i$ con $\theta_i = (A_i, b_i)$ los parámetros o pesos de la red.
Usando la regla de la cadena obtenemos que

$$\frac{\partial L}{\partial \theta_i} = \frac{\partial L}{\partial r_n} \frac{\partial r_n}{\partial r_{n-1}} \frac{\partial r_{n-1}}{\partial r_{n-2}} \cdots \frac{\partial r_{i+1}}{\partial r_i} \frac{\partial r_i}{\partial \theta_i}$$

Por tanto, podemos ver que aplicando la regla de la cadena en cada capa los términos se van repitiendo y podemos reutilizar dichos cálculos.

Cálculo del $\nabla_{\theta} L$

Backpropagation

Como ya hemos visto una NN puede representarse de la forma:

$$f_{\theta}(x) = r_n \circ r_{n-1} \circ \cdots \circ r_1(x) = \sigma_n(A_n(r_{n-1}(\dots)) + b_n)$$

donde

$$r_i = \sigma_i(A_i r_{i-1} + b_i)$$

con σ_i las funciones de activación y $\theta = \{\theta_i\}_i$ con $\theta_i = (A_i, b_i)$ los parámetros o pesos de la red. Usando la regla de la cadena obtenemos que

$$\frac{\partial L}{\partial \theta_i} = \frac{\partial L}{\partial r_n} \frac{\partial r_n}{\partial r_{n-1}} \frac{\partial r_{n-1}}{\partial r_{n-2}} \cdots \frac{\partial r_{i+1}}{\partial r_i} \frac{\partial r_i}{\partial \theta_i}$$

Por tanto, podemos ver que aplicando la regla de la cadena en cada capa los términos se van repitiendo y podemos reutilizar dichos cálculos.

Cálculo del $\nabla_{\theta} L$

Backpropagation

Como ya hemos visto una NN puede representarse de la forma:

$$f_{\theta}(x) = r_n \circ r_{n-1} \circ \cdots \circ r_1(x) = \sigma_n(A_n(r_{n-1}(\dots)) + b_n)$$

donde

$$r_i = \sigma_i(A_i r_{i-1} + b_i)$$

con σ_i las funciones de activación y $\theta = \{\theta_i\}_i$ con $\theta_i = (A_i, b_i)$ los parámetros o pesos de la red. Usando la regla de la cadena obtenemos que

$$\frac{\partial L}{\partial \theta_i} = \frac{\partial L}{\partial r_n} \frac{\partial r_n}{\partial r_{n-1}} \frac{\partial r_{n-1}}{\partial r_{n-2}} \cdots \frac{\partial r_{i+1}}{\partial r_i} \frac{\partial r_i}{\partial \theta_i}$$

Por tanto, podemos ver que aplicando la regla de la cadena en cada capa los términos se van repitiendo y podemos reutilizar dichos cálculos.

Pasos del Backpropagation

Principalmente el Algoritmo Backpropagation consta de dos etapas:

- ① Forward pass: Evaluamos el loss L y calculamos los outputs r_i en todas las capas. Propagamos los valores de atrás hacia delante en la red.
- ② Backward pass: Utilizando la regla de la cadena y los valores de r_i calculados previamente para computar de manera ordenada

$$\frac{\partial L}{\partial \theta_n} \rightarrow \frac{\partial L}{\partial \theta_{n-1}} \rightarrow \dots \rightarrow \frac{\partial L}{\partial \theta_1}$$

Propagamos el error de las capas finales a las iniciales.

Pasos del Backpropagation

Principalmente el Algoritmo Backpropagation consta de dos etapas:

- ① Forward pass: Evaluamos el loss L y calculamos los outputs r_i en todas las capas.
Propagamos los valores de atrás hacia delante en la red.
- ② Backward pass: Utilizando la regla de la cadena y los valores de r_i calculados previamente para computar de manera ordenada

$$\frac{\partial L}{\partial \theta_n} \rightarrow \frac{\partial L}{\partial \theta_{n-1}} \rightarrow \dots \rightarrow \frac{\partial L}{\partial \theta_1}$$

Propagamos el error de las capas finales a las iniciales.

Pasos del Backpropagation

Principalmente el Algoritmo Backpropagation consta de dos etapas:

- ① Forward pass: Evaluamos el loss L y calculamos los outputs r_i en todas las capas. Propagamos los valores de atrás hacia delante en la red.
- ② Backward pass: Utilizando la regla de la cadena y los valores de r_i calculados previamente para computar de manera ordenada

$$\frac{\partial L}{\partial \theta_n} \rightarrow \frac{\partial L}{\partial \theta_{n-1}} \rightarrow \cdots \rightarrow \frac{\partial L}{\partial \theta_1}$$

Propagamos el error de las capas finales a las iniciales.

Índice

1 Introducción

- Motivación y características principales de las Redes Neuronales
- Funcionamiento y estructura
- Áreas de aplicación

2 Introducción a las Redes Neuronales

- Desde el Perceptrón hasta las Redes Neuronales
- Gradient descent
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

Herramientas para Deep Learning

Vamos a ver distintas herramientas o recursos útiles para trabajar con redes neuronales.

Índice

1 Introducción

- Motivación y características principales de las Redes Neuronales
- Funcionamiento y estructura
- Áreas de aplicación

2 Introducción a las Redes Neuronales

- Desde el Perceptrón hasta las Redes Neuronales
- Gradient descent
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

Numpy

Numpy es una librería de python para trabajar con arrays o tensores. Las principales ventajas son:

- Muy fácil de usar y extremadamente compatible con muchas otras librerías (pandas, sk-learn...)
- Es la más rápida, está implementada en C.
- Tiene implementada una gran variedad de funciones matemáticas como operaciones lineales y algebraicas, generación de números aleatorios, funciones elementales...
- Tiene conceptos muy útiles como vectorization, indexing y broadcasting

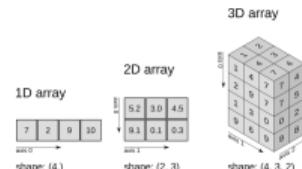


Figure: Numpy

Tensorflow/Keras

Tensorflow es la librería principal para la generación y entrenamiento de redes neuronales. Tiene todas las herramientas necesarias para crear cualquier tipo de red. También nos ofrece la posibilidad de trabajar a nivel de tensor, poder calcular los gradientes a mano y modificarlos como deseemos. Además dentro incluye Keras.

Keras es un intermediario, una capa sobre tensorflow que simplifica la creación de modelos y su entrenamiento. Actualmente se encuentra dentro de Tensorflow, lo cual simplifica las cosas y hace que los procesos sean más robustos. Trabajaremos con Keras functional API para la creación de modelos, ya que nos proporciona una mayor libertad sin añadir dificultad.

Tensorflow/Keras

Tensorflow es la librería principal para la generación y entrenamiento de redes neuronales. Tiene todas las herramientas necesarias para crear cualquier tipo de red. También nos ofrece la posibilidad de trabajar a nivel de tensor, poder calcular los gradientes a mano y modificarlos como deseemos. Además dentro incluye Keras.

Keras es un intermediario, una capa sobre tensorflow que simplifica la creación de modelos y su entrenamiento. Actualmente se encuentra dentro de Tensorflow, lo cual simplifica las cosas y hace que los procesos sean más robustos. Trabajaremos con Keras functional API para la creación de modelos, ya que nos proporciona una mayor libertad sin añadir dificultad.

Tensorflow/Keras

Las principales ventajas son:

- ① Muy fácil de usar y versátil.
- ② Tiene integrado Keras.
- ③ Muy popular y con una gran comunidad detrás, por lo que existe una gran variedad de recursos y proyectos reutilizables.
- ④ Está continuamente evolucionando.
- ⑤ Tiene la opción de usar una GPU de Nvidia a través de cuda. (Necesario Ubuntu)
- ⑥ Es usada por grandes compañías como Intel, Google, AMD, DeepMind, IBM...

Existen otras librerías como por ejemplo Pytorch, Theano, Caffe, MXNet...

Librerías

Otras librerías

Además, en determinadas ocasiones, será interesante utilizar otro tipo de librerías usadas en la Ciencia de Datos como por ejemplo Pandas, Matplotlib, Scikit-Learn, SciPy. Dichas librerías están diseñadas para la manipulación del dataset, dibujado de gráficas, cálculo de scores, optimización de hiperparámetros...

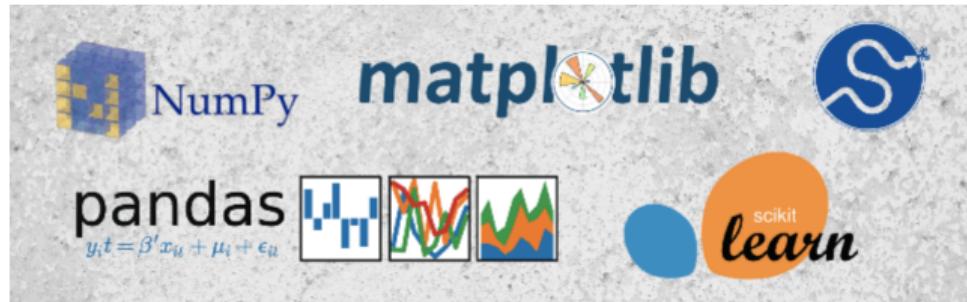


Figure: Otras Librerías

Índice

1 Introducción

- Motivación y características principales de las Redes Neuronales
- Funcionamiento y estructura
- Áreas de aplicación

2 Introducción a las Redes Neuronales

- Desde el Perceptrón hasta las Redes Neuronales
- Gradient descent
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

Sistema operativo

Preferiblemente se usará una distribución Debian, como Ubuntu 18, o en su defecto un sistema MacOS (sistemas Unix). Esto nos garantiza un rendimiento óptimo y una mejor compatibilidad entre librerías. Es recomendable saber instalar paquetes y tener conocimientos básicos del terminal.

Además podemos ejecutar código Python para crear y entrenar modelos en GPU sobre un entorno COLAB (Link Colab). Es un entorno en Cloud muy fácil de usar, gratis y potente. Podemos seleccionar un environment con GPU, sin necesidad de instalar o modificar nada.

Sistema operativo

Preferiblemente se usará una distribución Debian, como Ubuntu 18, o en su defecto un sistema MacOS (sistemas Unix). Esto nos garantiza un rendimiento óptimo y una mejor compatibilidad entre librerías. Es recomendable saber instalar paquetes y tener conocimientos básicos del terminal.

Además podemos ejecutar código Python para crear y entrenar modelos en GPU sobre un entorno COLAB (Link Colab). Es un entorno en Cloud muy fácil de usar, gratis y potente. Podemos seleccionar un environment con GPU, sin necesidad de instalar o modificar nada.

Entorno de desarrollo

IDE

Se recomienda el IDE (Entorno de Desarrollo Integrado) Pycharm Professional. Tiene una gran variedad de herramientas que nos facilitarán el desarrollo de aplicaciones:

- Gestión de proyectos y directorios
- Instalación de librerías
- Auto completar
- Refactor
- Debugger
- Integración con git
- Plugins
- Histórico local
- Profiler
- Remote Host

Entorno de desarrollo

IDE

Se recomienda el IDE (Entorno de Desarrollo Integrado) Pycharm Professional. Tiene una gran variedad de herramientas que nos facilitarán el desarrollo de aplicaciones:

- Gestión de proyectos y directorios
- Instalación de librerías
- Auto completar
- Refactor
- Debugger
- Integración con git
- Plugins
- Histórico local
- Profiler
- Remote Host

Índice

1 Introducción

- Motivación y características principales de las Redes Neuronales
- Funcionamiento y estructura
- Áreas de aplicación

2 Introducción a las Redes Neuronales

- Desde el Perceptrón hasta las Redes Neuronales
- Gradient descent
- Backpropagation

3 Herramientas para Deep Learning

- Librerías
- Entorno de desarrollo
- Otros recursos

Otros recursos

Netron

Herramienta para la visualización interactiva de redes neuronales.

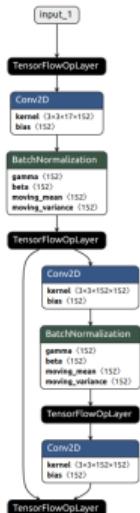


Figure: Netron

Guías y tutoriales de Tensorflow

En las páginas oficiales de Tensorflow y Keras podemos encontrar guías y tutoriales del uso de dichas librerías con ejemplos prácticos.

- [Keras Guide Link](#)
- [Tensorflow Guide Link](#)

Si queremos profundizar más en detalle en las API's, podemos hacerlo en las páginas webs:

- [Keras API](#)
- [Tensorflow API](#)

Guías y tutoriales de Tensorflow

En las páginas oficiales de Tensorflow y Keras podemos encontrar guías y tutoriales del uso de dichas librerías con ejemplos prácticos.

- Keras Guide Link
- Tensorflow Guide Link

Si queremos profundizar más en detalle en las API's, podemos hacerlo en las páginas webs:

- Keras API
- Tensorflow API

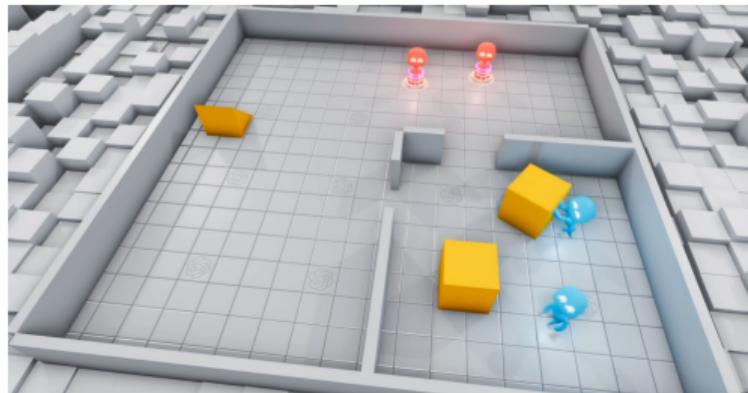
YouTube

- Deep Mind
 - ▶ AlphaGo - The Movie
 - ▶ AlphaFold
- Nvidia
 - ▶ GauGAN: Changing Sketches into Photorealistic Masterpieces
 - ▶ AI Reconstructs Photos with Realistic Results
 - ▶ The First Interactive AI Rendered Virtual World
 - ▶ Transforming Standard Video Into Slow Motion with AI
- Open AI
 - ▶ Multi-Agent Hide and Seek
 - ▶ Solving Rubik's Cube with a Robot Hand

Otros recursos

YouTube

- 3Blue1Brown
 - ▶ ¿Pero qué "es" una Red neuronal?
 - ▶ Decenso de gradiente, es como las redes neuronales aprenden
- Two Minute Papers
 - ▶ OpenAI juega a las escondidas
 - ▶ This AI Does Nothing In Games... And Still Wins!
 - ▶ 4 Experiments Where the AI Outsmarted Its Creators



Otros recursos

Playground

Playground es una página interactiva para visualizar de manera muy sencilla cómo son los filtros dentro de una red y cómo evoluciona el entrenamiento. Además podemos probar distintas configuraciones para entender mejor el funcionamiento de los hiperparámetros de una red.

[Playground Link](#)

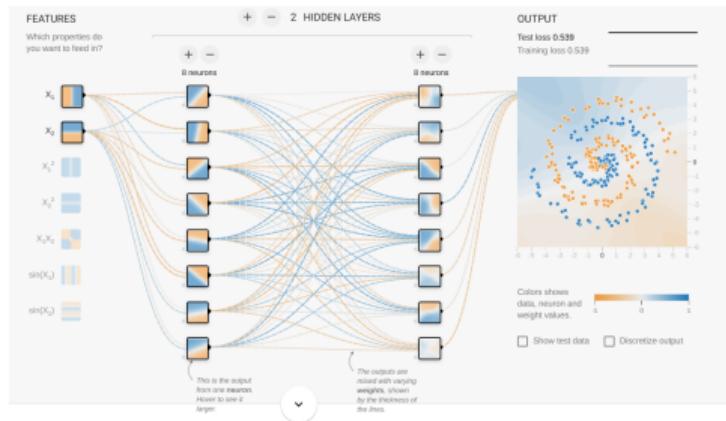


Figure: Playground

Recursos Web

Existen varios recursos web con ejemplos ilustrativos y material con el que poder interactuar.

- Open-AI
 - ▶ Generative Models
 - ▶ DALLE: Creating Images from Text
 - ▶ Microscope
- Distill
 - ▶ Visualizing Neural Networks
 - ▶ Differentiable Image Parameterizations
 - ▶ Feature Visualization
 - ▶ Why Momentum Really Works
- Blog Google
 - ▶ Grammar Correction as You Type, on Pixel 6
 - ▶ Self-Supervised Learning Advances Medical Image Classification
- Medium