

Redes Neuronales y Aprendizaje Profundo Parte 1

Jorge Linde Díaz

CUNEF

November 28, 2021



Redes Neuronales y Aprendizaje Profundo

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

Introducción

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

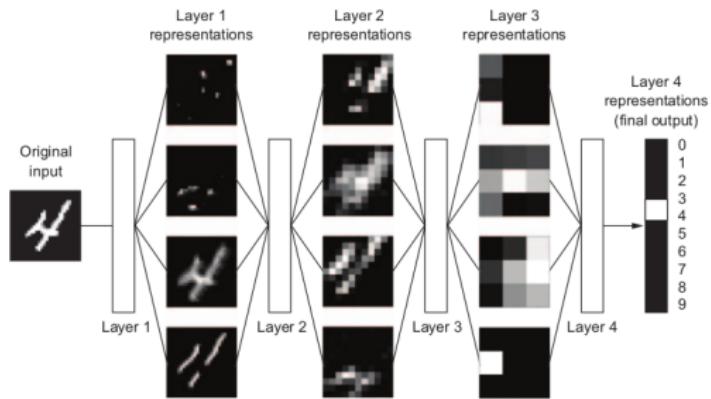


Figure: Red Neuronal

Introducción

Actualmente hay un gran interés en la Inteligencia Artificial, en particular en el Deep Learning, que podemos observar en conferencias y anuncios. Esto ha sido posible 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, que podemos observar en conferencias y anuncios. Esto ha sido posible 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

Introducción

Actualmente hay un gran interés en la Inteligencia Artificial, en particular en el Deep Learning, que podemos observar en conferencias y anuncios. Esto ha sido posible 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

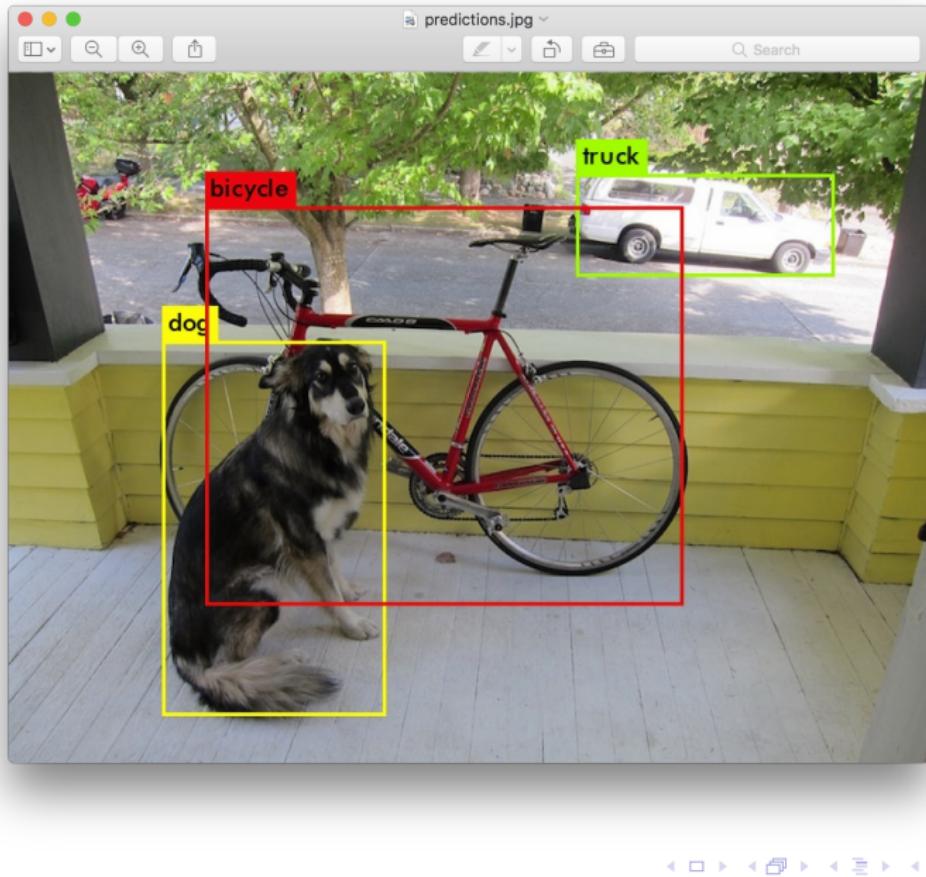
Introducción

Actualmente hay un gran interés en la Inteligencia Artificial, en particular en el Deep Learning, que podemos observar en conferencias y anuncios. Esto ha sido posible 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

Detección de objetos

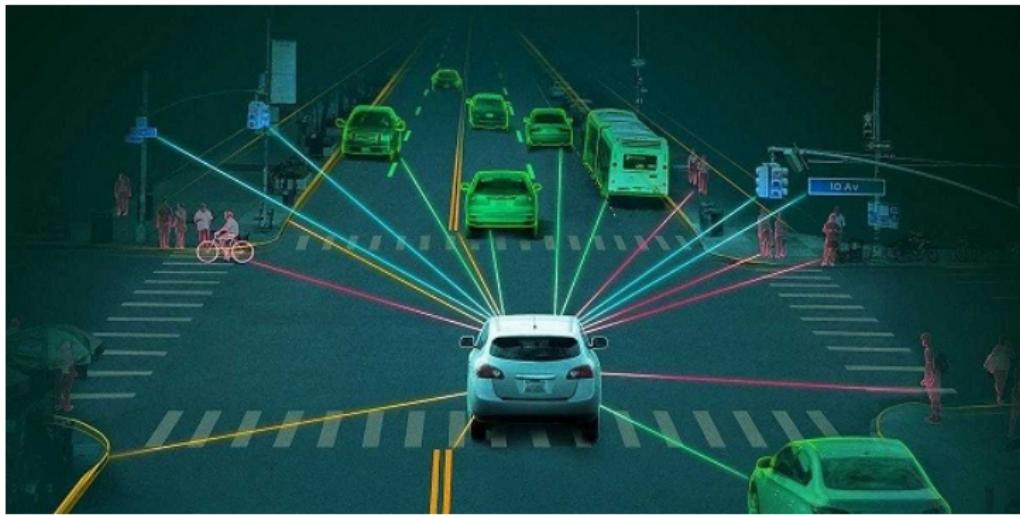


Vencer al campeón del mundo de GO

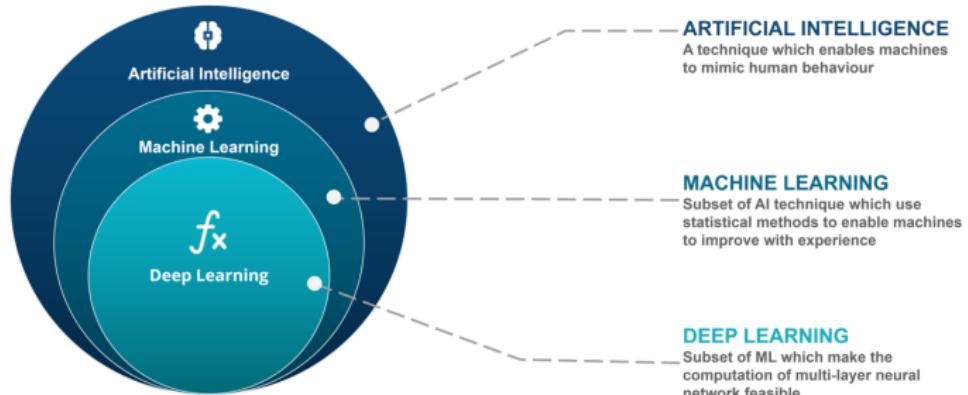


YouTube: AlphaGo - The movie

Conducción autónoma



IA vs 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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

Motivación

¿Qué mejoras presenta frente al Machine Learning?

El DL ha conseguido ciertos hitos, que el ML no ha sido capaz, gracias a la estructura jerárquica de aprendizaje.

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 ciertos hitos, que el ML no ha sido capaz, gracias a la estructura jerárquica de aprendizaje.

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

¿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).

Motivación

¿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.

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

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

Motivación

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

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

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

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...)

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...)

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...)

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...)

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.

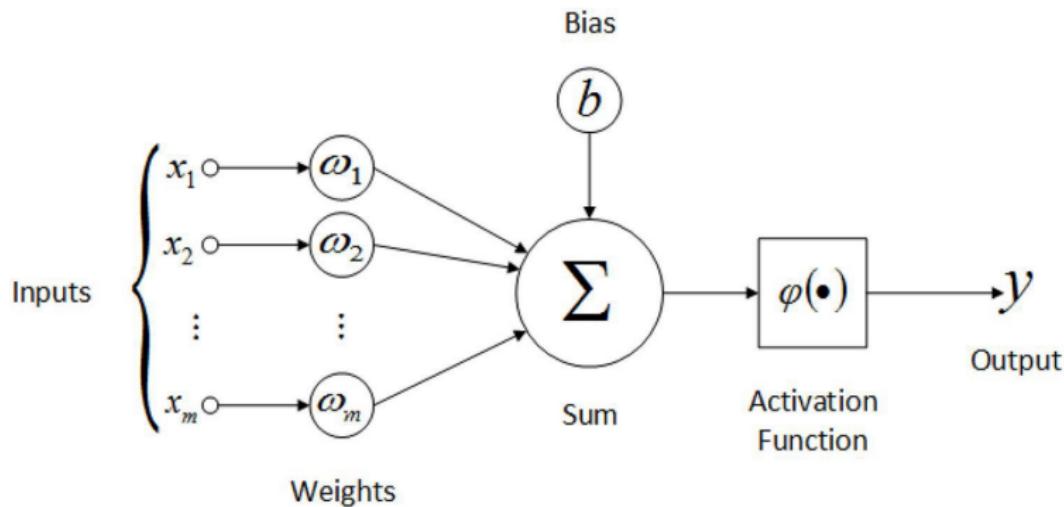


Figure: Neurona

Estructura

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...
- 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.

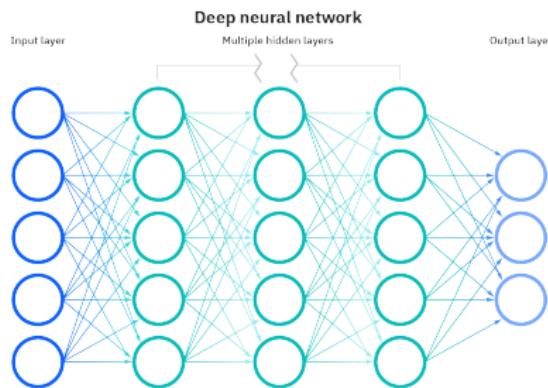


Figure: Red Neuronal

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_i 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

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

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_i de manera iterativa comparando las predicciones y' y los targets reales y . Así obtendremos una sucesión

$$\theta_0 \rightarrow \theta_1 \rightarrow \theta_2 \rightarrow \cdots \rightarrow \theta_n$$

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

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

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_i 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

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

Es un proceso incremental de aprendizaje no determinista.

Entrenamiento de una Red Neuronal

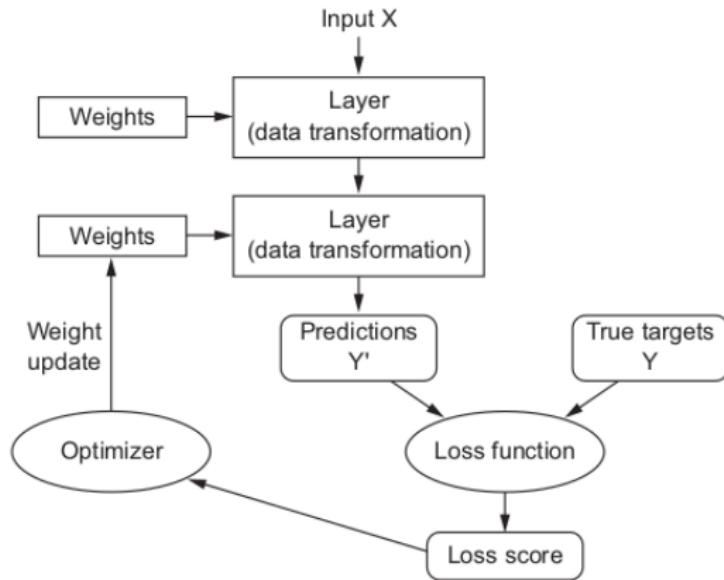


Figure: Fundamentos Deep Learning

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

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?

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)
- ④ Coste computacional.
- ⑤ Problema de dimensionalidad y overfitting. (relacionado con el tamaño del dataset)
- ⑥ Fine tuning más complejo. (es un problema práctico)

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)

Mitigación

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

Áreas de aplicación

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.

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

Introducción a las Redes Neuronales

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

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.

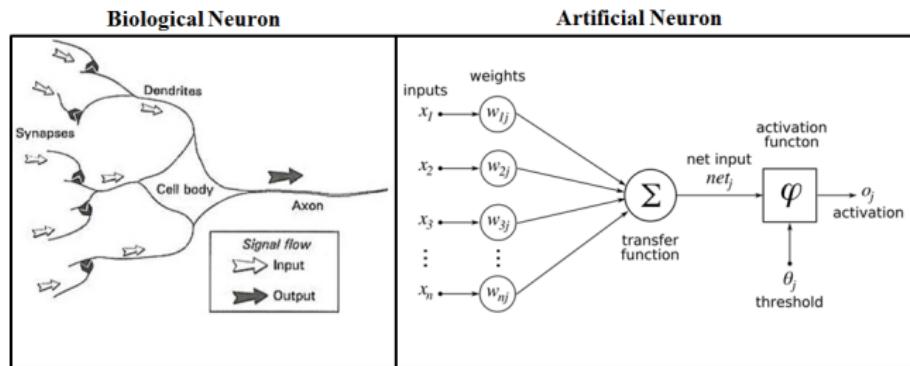


Figure: Neuronas biológicas vs artificiales

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.

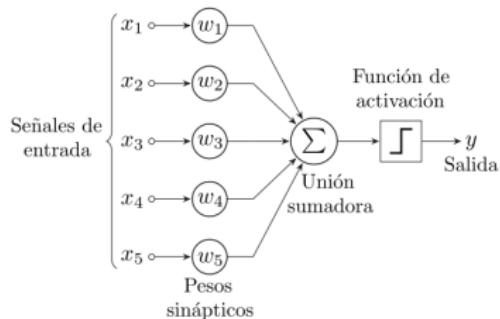


Figure: Perceptrón

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

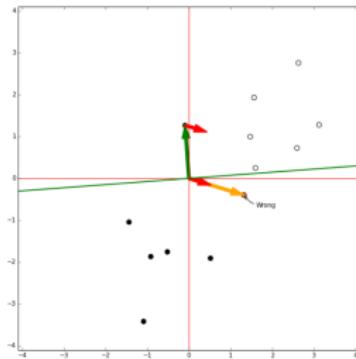


Figure: Entrenamiento perceptrón

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:

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

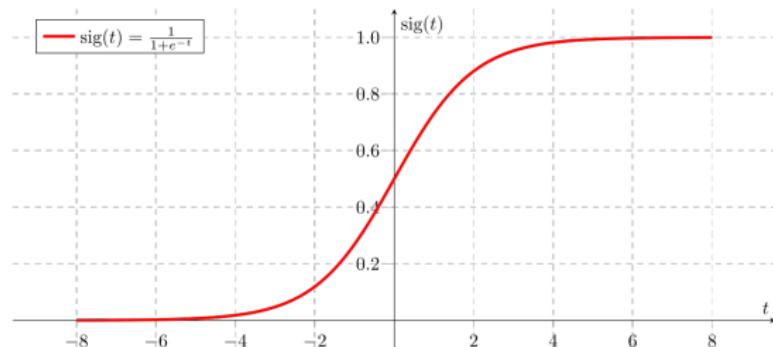


Figure: Sigmoid

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

Entrenamiento de una NN

Estructura de la red

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

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

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

Estructura de la red

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

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

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

Estructura de la red

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

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

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 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.

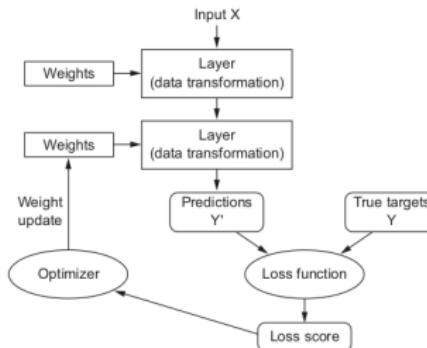


Figure: Fundamentos Deep Learning

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:

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

- 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

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:

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

- 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

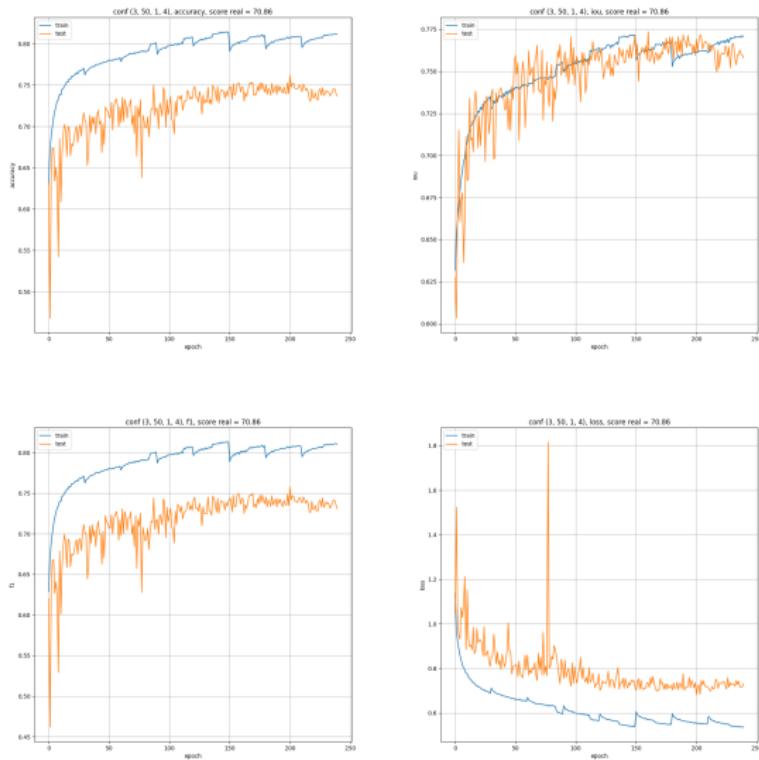


Figure: Comparación accuracy vs iou vs f1 vs BCE

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

¿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?

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.

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?

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.

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))

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

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

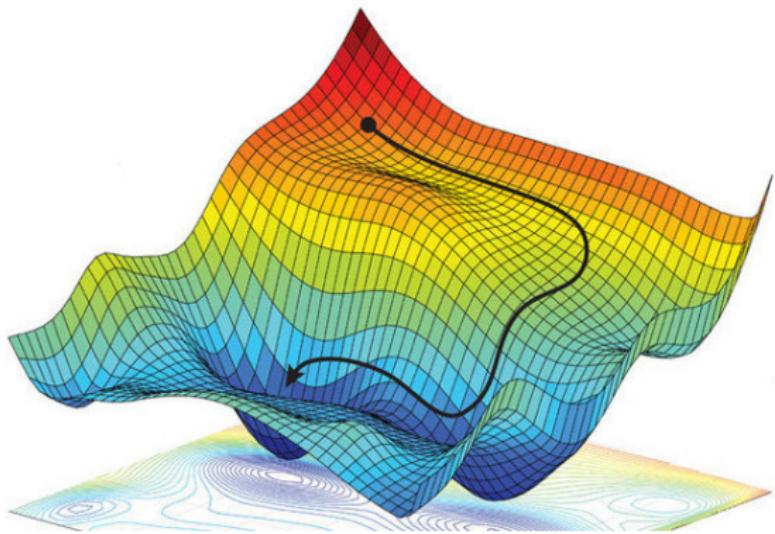


Figure: Gradient Descent

Entrenamiento de NN

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

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

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

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:

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

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:

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

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:

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

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:

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

Entrenamiento de NN

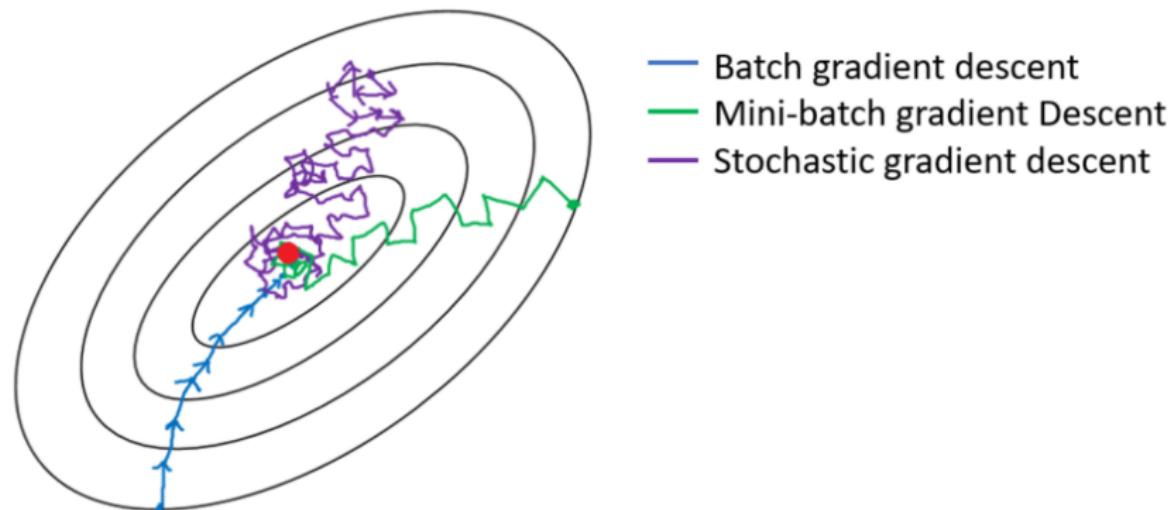


Figure: Variantes del GD

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

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 $\theta = \{\theta_i\}$; 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 $\theta = \{\theta_i\}$; 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 $\theta = \{\theta_i\}$; 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 $\nabla_{\theta}L$

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.

Cálculo del $\nabla_{\theta}L$

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.

Cálculo del $\nabla_{\theta} L$

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

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

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...

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...

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

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 uso del IDE (Entorno de Desarrollo Integrado) Pycharm Professional. Tiene una gran variedad de herramientas que nos facilitarán el desarrollo de aplicaciones Python, como por ejemplo:

- 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 uso del IDE (Entorno de Desarrollo Integrado) **Pycharm Professional**. Tiene una gran variedad de herramientas que nos facilitarán el desarrollo de aplicaciones Python, como por ejemplo:

- 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**

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

Otros recursos

Netron

Herramienta para la visualización interactiva de redes neuronales.

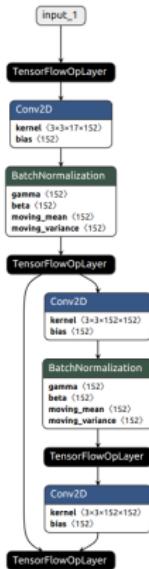


Figure: Netron

Otros recursos

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](#)

Otros recursos

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

Otros recursos

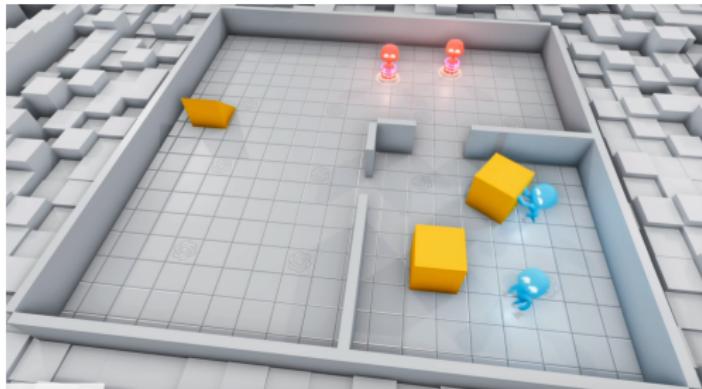
YouTube

- 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](#)

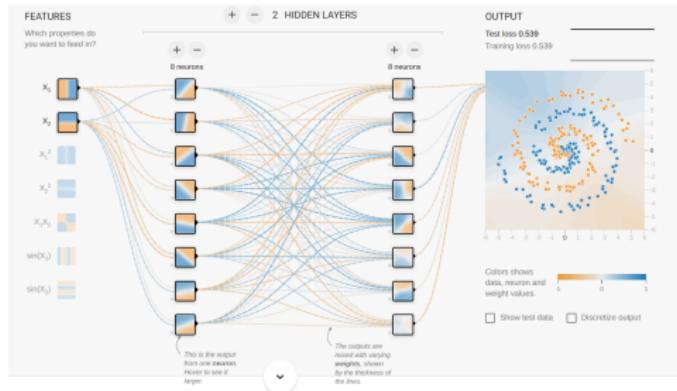


Figure: Playground

Otros recursos

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

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

Componentes de una Red Neuronal

Resumen

En este tema veremos más en detalle los componentes de una Red Neuronal, así como distintas modificaciones que podemos hacer en la arquitectura para mejorar su eficacia.

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

4 Componentes de una Red Neuronal

● Funciones de activación

- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

Funciones de activación

Sigmoid

Hasta ahora solo habíamos visto como funciones de activación la función step y la función sigmoid. A diferencia de la función step, la función sigmoid tiene buenas propiedades para entrenar una NN con ella.

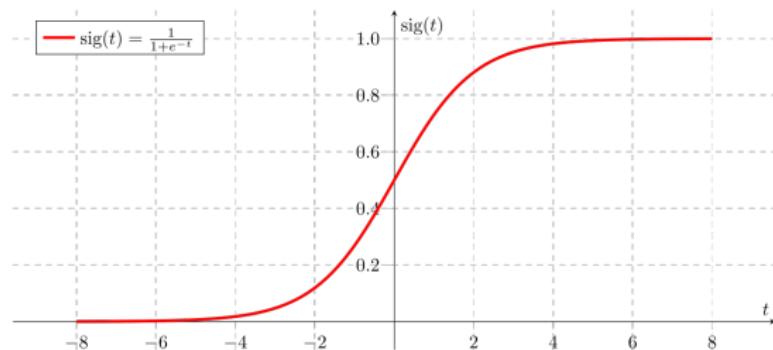


Figure: Sigmoid

Funciones de activación

Clasificación multiclas

Si tratamos un problema de clasificación multiclas no es aconsejable usar la función sigmoid en la capa de salida. Para este caso existe una función que es una generalización de la función sigmoid (?).

Softmax

La función softmax se usa en la última capa cuando tratamos problemas de clasificación multiclas. En este caso existe una dependencia entre los valores asociados a cada clase, ya que han de sumar entre todos 1. Se define como:

$$p = \text{softmax}(v) = \left(\frac{e^{v_i}}{\sum_j e^{v_j}} \right)_i$$

Al vector v se le conoce como Logit. La función softmax reparte las probabilidades de cada clase en función de su valor logit asociado. Se usa solo en la ultima capa, ya que esta dependencia solo aparece ahí.

Funciones de activación

Clasificación multiclas

Si tratamos un problema de clasificación multiclas no es aconsejable usar la función sigmoid en la capa de salida. Para este caso existe una función que es una generalización de la función sigmoid (¿?).

Softmax

La función softmax se usa en la última capa cuando tratamos problemas de clasificación multiclas. En este caso existe una dependencia entre los valores asociados a cada clase, ya que han de sumar entre todos 1. Se define como:

$$p = \text{softmax}(v) = \left(\frac{e^{v_i}}{\sum_j e^{v_j}} \right)_i$$

Al vector v se le conoce como Logit. La función softmax reparte las probabilidades de cada clase en función de su valor logit asociado.

Se usa solo en la ultima capa, ya que esta dependencia solo aparece ahí.

Funciones de activación

Clasificación multiclas

Si tratamos un problema de clasificación multiclas no es aconsejable usar la función sigmoid en la capa de salida. Para este caso existe una función que es una generalización de la función sigmoid (?).

Softmax

La función softmax se usa en la última capa cuando tratamos problemas de clasificación multiclas. En este caso existe una dependencia entre los valores asociados a cada clase, ya que han de sumar entre todos 1. Se define como:

$$p = \text{softmax}(v) = \left(\frac{e^{v_i}}{\sum_j e^{v_j}} \right)_i$$

Al vector v se le conoce como Logit. La función softmax reparte las probabilidades de cada clase en función de su valor logit asociado. Se usa solo en la ultima capa, ya que esta dependencia solo aparece ahí.

Funciones de activación

Tanh

Otra función de activación similar a la sigmoid, con buenas propiedades, es la función Tanh. Muy usada en problemas de regresión con rango $[-1, 1]$. Se define como:

$$\tanh(x) = \frac{2}{(1 + e^{-2x})} - 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Si sabemos que el rango del output es $[a, b]$, podemos escalar la salida con $(b - a)/2 \cdot \tanh(x) + (a + b)/2$ para que el rango sea el deseado. También podríamos no poner función de activación ($i?$).

Funciones de activación

Tanh

Otra función de activación similar a la sigmoid, con buenas propiedades, es la función Tanh. **Muy usada en problemas de regresión con rango $[-1, 1]$.**
Se define como:

$$\tanh(x) = \frac{2}{(1 + e^{-2x})} - 1 = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Si sabemos que el rango del output es $[a, b]$, podemos escalar la salida con $(b - a)/2 \cdot \tanh(x) + (a + b)/2$ para que el rango sea el deseado. También podríamos no poner función de activación (?).

Funciones de activación

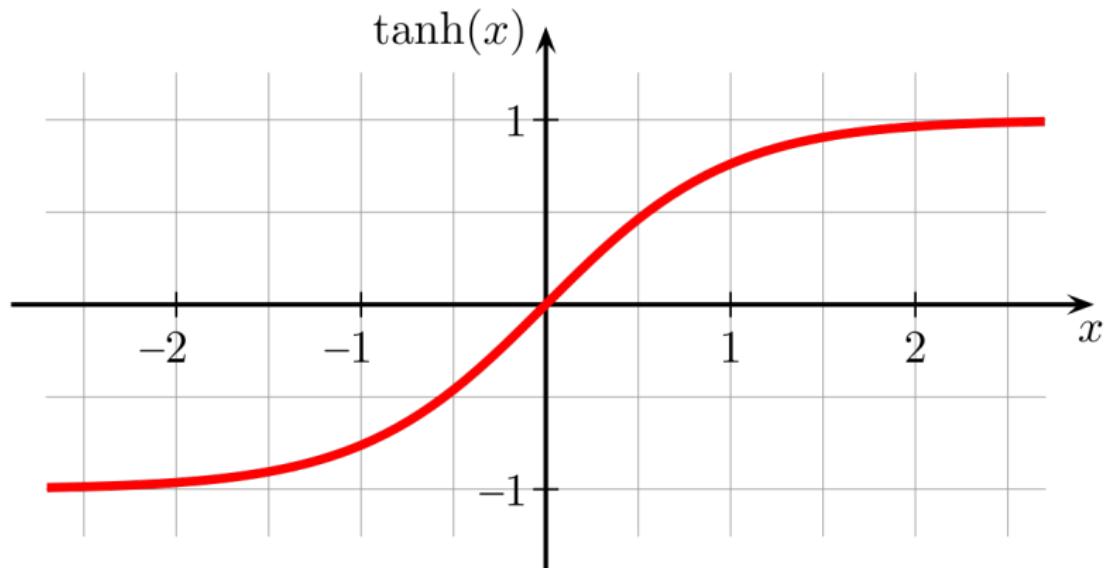


Figure: Tanh

Funciones de activación

Activaciones Lineales a trozos

Existe un conjunto de funciones que solventa ciertos problemas que aparecen durante el entrenamiento de redes profundas. Las más representativas son las funciones ReLU, ELU y LeakyReLU, definidas como:

$$relu(x) = \max(0, x)$$

$$elu_{\alpha}(x) = \begin{cases} \alpha(e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases}$$

$$leakyrelu_{\alpha}(x) = \max(\alpha x, x)$$

Este tipo de funciones son las que más se usan en la actualidad.

Funciones de activación

Activaciones Lineales a trozos

Existe un conjunto de funciones que solventa ciertos problemas que aparecen durante el entrenamiento de redes profundas. Las más representativas son las funciones ReLU, ELU y LeakyReLU, definidas como:

$$relu(x) = \max(0, x)$$

$$elu_{\alpha}(x) = \begin{cases} \alpha(e^x - 1) & x < 0 \\ x & x \geq 0 \end{cases}$$

$$leakyrelu_{\alpha}(x) = \max(\alpha x, x)$$

Este tipo de funciones son las que más se usan en la actualidad.

Funciones de activación

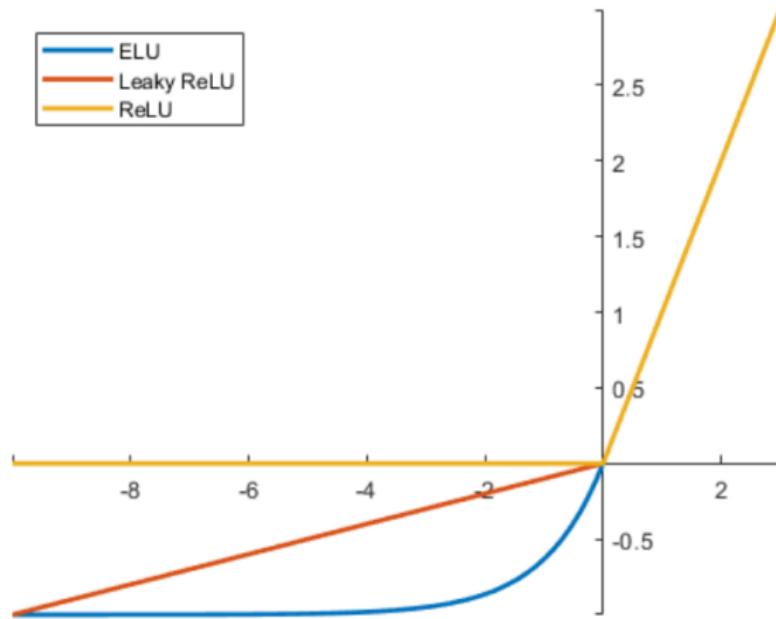


Figure: Activaciones Lineales a trozos

Funciones de activación

Propiedades de las Activaciones Lineales a trozos

Estas funciones solventan ciertos problemas durante el entrenamiento además de poseer otras ventajas:

- ① Son funciones menos costosas de computar.
- ② Solventan el problema del vanishing gradient problem.
- ③ La función ReLU puede 'apagar' neuronas.
- ④ Gran rendimiento en computer vision.

La elección de dichas funciones va a depender del problema, por lo que habrá que probarlas como un hiperparámetro más del modelo para obtener los mejores resultados.

Funciones de activación

Propiedades de las Activaciones Lineales a trozos

Estas funciones solventan ciertos problemas durante el entrenamiento además de poseer otras ventajas:

- ① Son funciones menos costosas de computar.
- ② Solventan el problema del vanishing gradient problem.
- ③ La función ReLU puede 'apagar' neuronas.
- ④ Gran rendimiento en computer vision.

La elección de dichas funciones va a depender del problema, por lo que habrá que probarlas como un hiperparámetro más del modelo para obtener los mejores resultados.

Funciones de activación

Propiedades de las Activaciones Lineales a trozos

Estas funciones solventan ciertos problemas durante el entrenamiento además de poseer otras ventajas:

- ① Son funciones menos costosas de computar.
- ② Solventan el problema del vanishing gradient problem.
- ③ La función ReLU puede 'apagar' neuronas.
- ④ Gran rendimiento en computer vision.

La elección de dichas funciones va a depender del problema, por lo que habrá que probarlas como un hiperparámetro más del modelo para obtener los mejores resultados.

Propiedades de las Activaciones Lineales a trozos

Estas funciones solventan ciertos problemas durante el entrenamiento además de poseer otras ventajas:

- ① Son funciones menos costosas de computar.
- ② Solventan el problema del vanishing gradient problem.
- ③ La función ReLU puede 'apagar' neuronas.
- ④ Gran rendimiento en computer vision.

La elección de dichas funciones va a depender del problema, por lo que habrá que probarlas como un hiperparámetro más del modelo para obtener los mejores resultados.

Propiedades de las Activaciones Lineales a trozos

Estas funciones solventan ciertos problemas durante el entrenamiento además de poseer otras ventajas:

- ① Son funciones menos costosas de computar.
- ② Solventan el problema del vanishing gradient problem.
- ③ La función ReLU puede 'apagar' neuronas.
- ④ Gran rendimiento en computer vision.

La elección de dichas funciones va a depender del problema, por lo que habrá que probarlas como un hiperparámetro más del modelo para obtener los mejores resultados.

Funciones de activación

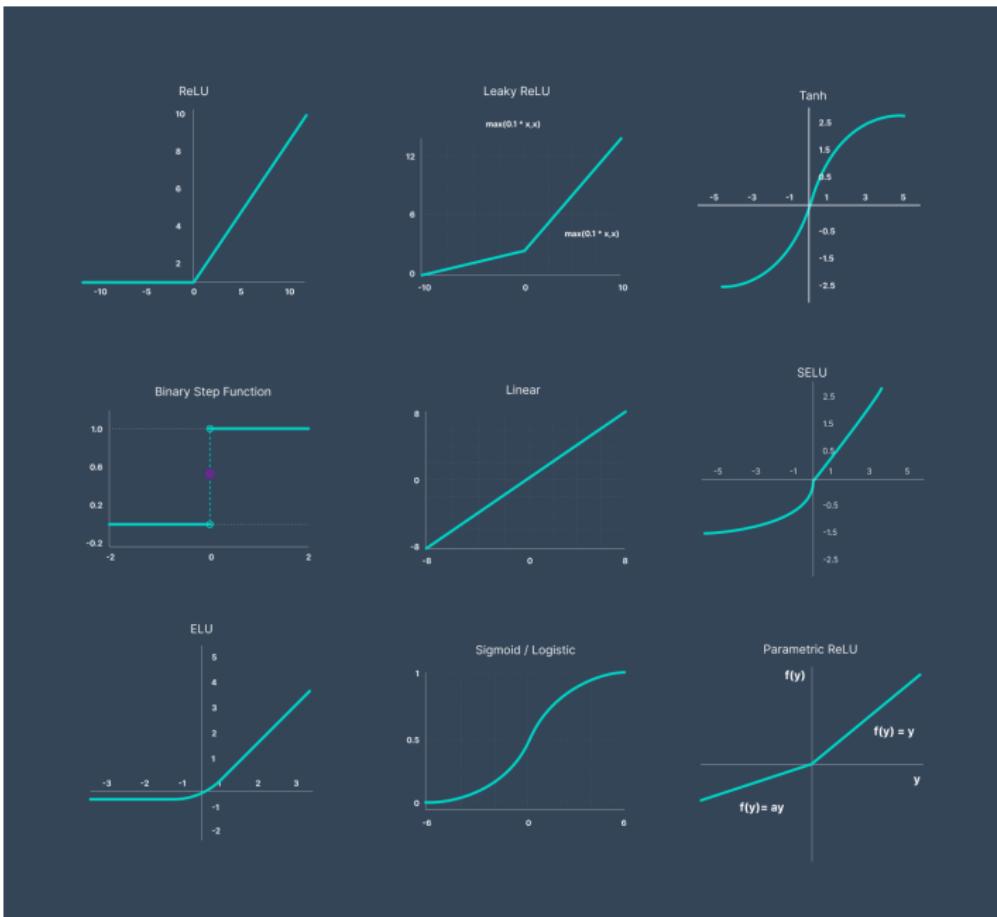
Propiedades de las Activaciones Lineales a trozos

Estas funciones solventan ciertos problemas durante el entrenamiento además de poseer otras ventajas:

- ① Son funciones menos costosas de computar.
- ② Solventan el problema del vanishing gradient problem.
- ③ La función ReLU puede 'apagar' neuronas.
- ④ Gran rendimiento en computer vision.

La elección de dichas funciones va a depender del problema, por lo que habrá que probarlas como un hiperparámetro más del modelo para obtener los mejores resultados.

Funciones de activación



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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales**
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- Hiperparámetros

Capas Convolucionales

Motivación Redes Convolucionales

En el área del computer vision es donde las NN han cobrado una mayor atención gracias a sus grandes resultados. Esto ha sido posible gracias a las capas Convolucionales, las cuales son capaces de extraer patrones complejos de las imágenes.

Este tipo de capas son usadas en Redes Convolucionales, las cuales son usadas a su vez en la gran mayoría de sistemas de computer vision. Como por ejemplo:

- ① Clasificación de imágenes
- ② Detección de objetos
- ③ Segmentación de objetos
- ④ Tracking
- ⑤ Autenticación biométrica
- ⑥ Generación sintética de imágenes
- ⑦ Conducción autónoma
- ⑧ Súper Resolución

Capas Convolucionales

Motivación Redes Convolucionales

En el área del computer vision es donde las NN han cobrado una mayor atención gracias a sus grandes resultados. Esto ha sido posible gracias a las capas Convolucionales, las cuales son capaces de extraer patrones complejos de las imágenes.

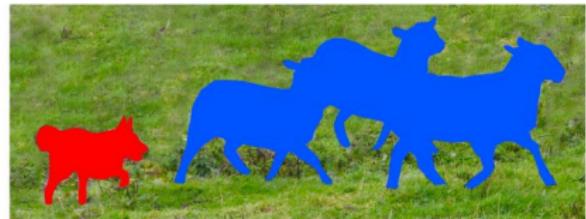
Este tipo de capas son usadas en Redes Convolucionales, las cuales son usadas a su vez en la gran mayoría de sistemas de computer vision. Como por ejemplo:

- ① Clasificación de imágenes
- ② Detección de objetos
- ③ Segmentación de objetos
- ④ Tracking
- ⑤ Autenticación biométrica
- ⑥ Generación sintética de imágenes
- ⑦ Conducción autónoma
- ⑧ Súper Resolución

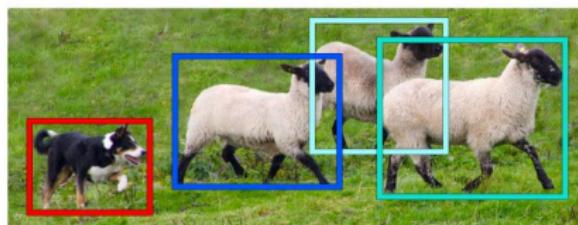
Capas Convolucionales



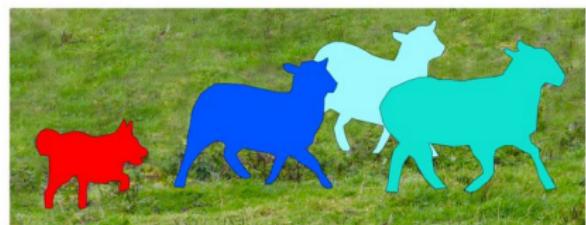
Image Recognition



Semantic Segmentation



Object Detection



Instance Segmentation

Figure: Ejemplos computer vision

Capas Convolucionales

Extracción de patrones

Cuando trabajamos con imágenes la dimensión del input es muy grande así como la complejidad de los patrones. En este caso con las NN totalmente conectadas no obtenemos los scores deseados. Principalmente porque tenemos que trabajar con redes muy grandes y porque las NN totalmente conectadas trabajan sobre toda la imagen.

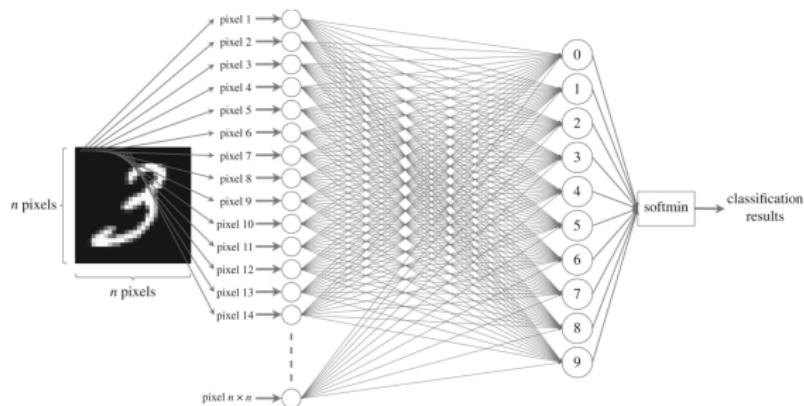


Figure: FCNN

Capas Convolucionales

Extracción de patrones

¿Como podemos resolver este problema?

Si tuviéramos que decidir si en una imagen aparece un perro no analizaríamos todos los pixeles, sino que intentaríamos detectar en la imagen unas orejas, cuatro patas, una cara de perro... Es decir, estamos buscando patrones bien definidos e invariantes por traslaciones.

Las capas convolucionales son capaces de detectar este tipo de patrones.

Pixeles vs Formas

Las capas convolucionales aprenden patrones locales, de esta manera son capaces de reconocer formas aunque no sean exactamente iguales. Esto hace que el dataset necesario para entrenar sea menor.

Capas Convolucionales

Extracción de patrones

¿Como podemos resolver este problema?

Si tuviéramos que decidir si en una imagen aparece un perro no analizaríamos todos los pixeles, sino que intentaríamos detectar en la imagen unas orejas, cuatro patas, una cara de perro... Es decir, estamos buscando patrones bien definidos e invariantes por traslaciones.

Las capas convolucionales son capaces de detectar este tipo de patrones.

Pixeles vs Formas

Las capas convolucionales aprenden patrones locales, de esta manera son capaces de reconocer formas aunque no sean exactamente iguales. Esto hace que el dataset necesario para entrenar sea menor.

Capas Convolucionales

Extracción de patrones

¿Como podemos resolver este problema?

Si tuviéramos que decidir si en una imagen aparece un perro no analizaríamos todos los pixeles, sino que intentaríamos detectar en la imagen unas orejas, cuatro patas, una cara de perro... Es decir, estamos buscando patrones bien definidos e invariantes por traslaciones.

Las capas convolucionales son capaces de detectar este tipo de patrones.

Pixeles vs Formas

Las capas convolucionales aprenden patrones locales, de esta manera son capaces de reconocer formas aunque no sean exactamente iguales. Esto hace que el dataset necesario para entrenar sea menor.

Capas Convolucionales

Extracción de patrones

¿Como podemos resolver este problema?

Si tuviéramos que decidir si en una imagen aparece un perro no analizaríamos todos los pixeles, sino que intentaríamos detectar en la imagen unas orejas, cuatro patas, una cara de perro... Es decir, estamos buscando patrones bien definidos e invariantes por traslaciones.

Las capas convolucionales son capaces de detectar este tipo de patrones.

Pixeles vs Formas

Las capas convolucionales aprenden patrones locales, de esta manera son capaces de reconocer formas aunque no sean exactamente iguales. Esto hace que el dataset necesario para entrenar sea menor.

Capas Convolucionales

Características de los patrones aprendidos

Las dos principales características de los patrones aprendidos son:

- ① Los patrones son invariantes respecto de traslaciones. Las capas convolucionales son capaces de detectarlos en cualquier parte de la imagen.
- ② Se aprenden patrones con una jerarquía espacial. Cada capa aprende distintos tipos de patrones. A medida que aumenta la profundidad de la capa, los patrones que aprende dicha capa son cada vez más complejos y aporta mucha más información abstracta de la imagen. Se consigue un aprendizaje incremental de abstracción visual.

Capas Convolucionales

Características de los patrones aprendidos

Las dos principales características de los patrones aprendidos son:

- ① Los patrones son invariantes respecto de traslaciones. Las capas convolucionales son capaces de detectarlos en cualquier parte de la imagen.
- ② Se aprenden patrones con una jerarquía espacial. Cada capa aprende distintos tipos de patrones. A medida que aumenta la profundidad de la capa, los patrones que aprende dicha capa son cada vez más complejos y aporta mucha más información abstracta de la imagen. Se consigue un aprendizaje incremental de abstracción visual.

Capas Convolucionales

Características de los patrones aprendidos

Las dos principales **características de los patrones** aprendidos son:

- ① Los patrones son invariantes respecto de traslaciones. **Las capas convolucionales son capaces de detectarlos en cualquier parte de la imagen.**
- ② Se aprenden patrones con una jerarquía espacial. Cada capa aprende distintos tipos de patrones. A medida que aumenta la profundidad de la capa, los patrones que aprende dicha capa son cada vez más complejos y aporta mucha más información abstracta de la imagen. **Se consigue un aprendizaje incremental de abstracción visual.**

Capas Convolucionales

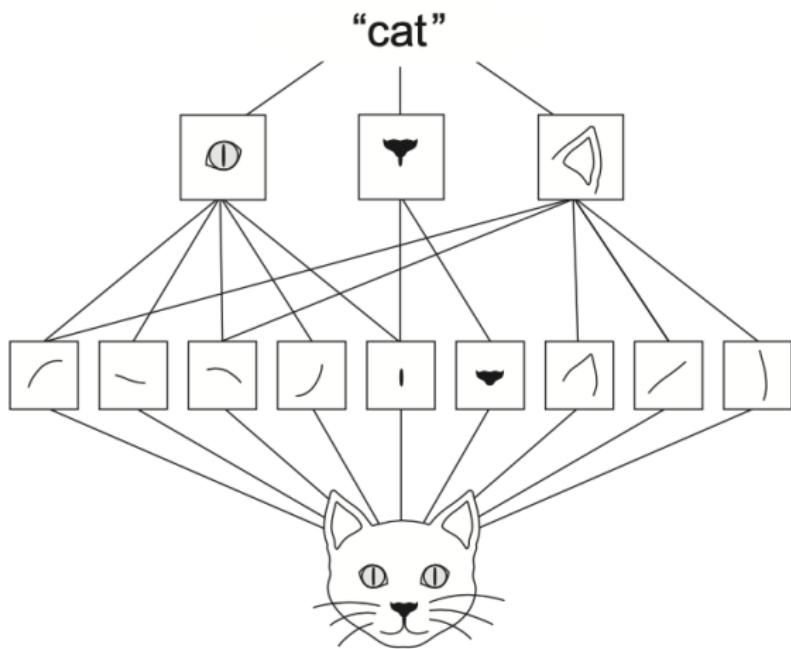
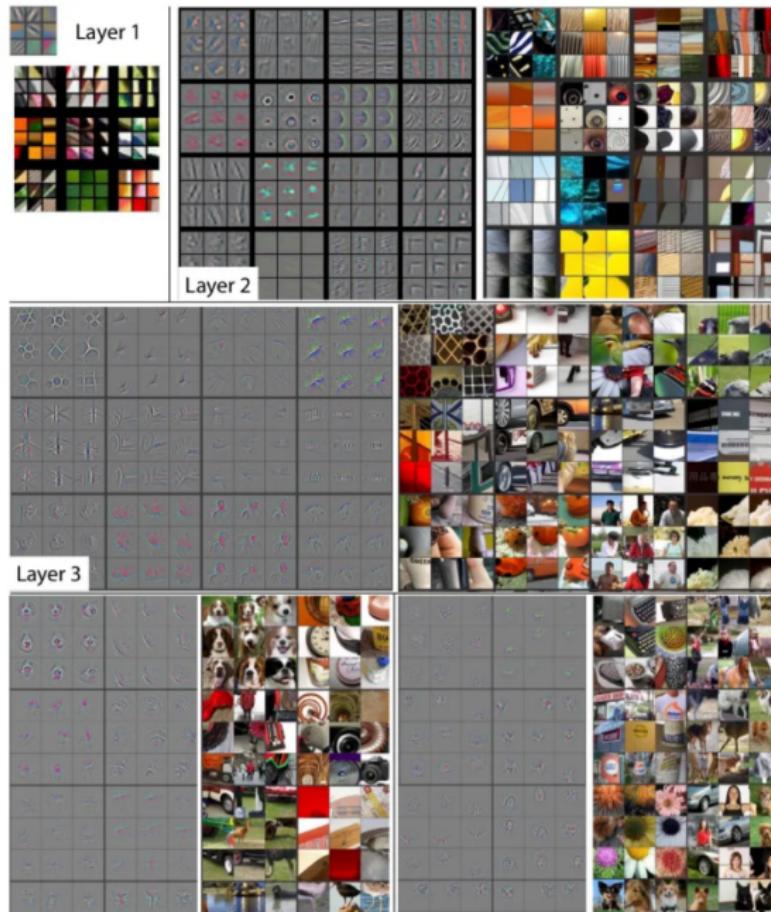


Figure: Patrones detectados por una CNN

Capas Convolucionales



Capas Convolucionales

Funcionamiento

El input y output de una capa convolucional es un tensor de 4 dimensiones (batch, width, height, channels). Las imágenes habituales tienen 3 canales RGB. Las capas convolucionales están compuestas por k filtros (pequeñas neuronas) de dimensión (n, n, m) que se desplazan a lo largo de los ejes x e y recorriendo la imagen.

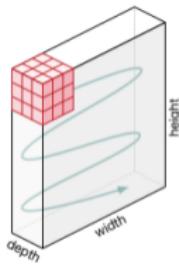


Figure: Recorrido filtro

Capas Convolucionales

Funcionamiento

A medida que el filtro se desplaza por la imagen se calcula la convolución entre ambas matrices, produciendo un único valor. El output final de la capa es un 2D-tensor (feature channel) por cada filtro.

Como el filtro se mueve sobre la imagen, el patrón asociado a dicho filtro será detectado en cualquier parte de la imagen.

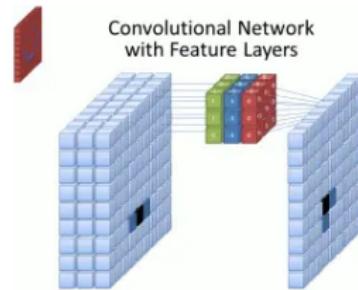


Figure: Convolución

Capas Convolucionales

Funcionamiento

A medida que el filtro se desplaza por la imagen se calcula la convolución entre ambas matrices, produciendo un único valor. El output final de la capa es un 2D-tensor (feature channel) por cada filtro.

Como el filtro se mueve sobre la imagen, el patrón asociado a dicho filtro será detectado en cualquier parte de la imagen.

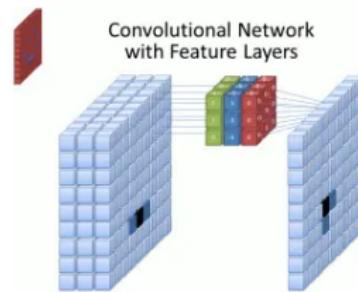


Figure: Convolución

Capas Convolucionales

Funcionamiento

En las zonas en las que el filtro detecte su patrón asociado producirá una respuesta alta, por lo que cada channel en el output de una capa convolucional se puede ver como un mapa de calor del patrón asociado.

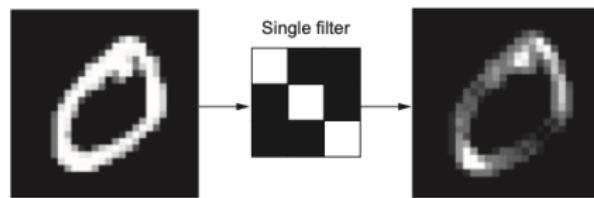


Figure: Detector de patrones

Capas Convolucionales

Parámetros principales

Los principales parámetros de una capa convolucional son:

- Número de filtros.
- Tamaño del filtro.
- Strides.
- Padding 'valid' o 'same'

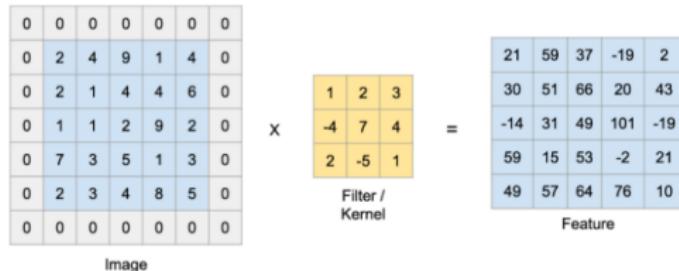


Figure: 1 filtro, kernel_size=3, strides=1, padding='same'

Capas Convolucionales

Parámetros principales

Los principales parámetros de una capa convolucional son:

- Número de filtros.
- Tamaño del filtro.
- Strides.
- Padding 'valid' o 'same'

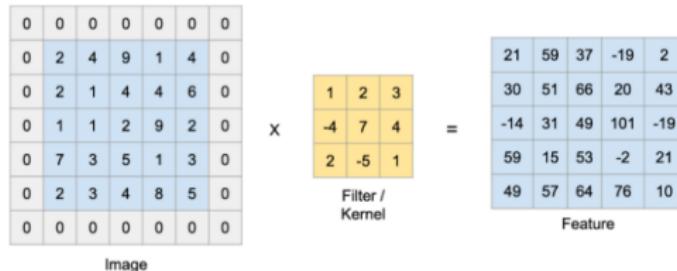


Figure: 1 filtro, kernel_size=3, strides=1, padding='same'

Capas Convolucionales

Parámetros principales

Los principales parámetros de una capa convolucional son:

- Número de filtros.
- Tamaño del filtro.
- Strides.
- Padding 'valid' o 'same'

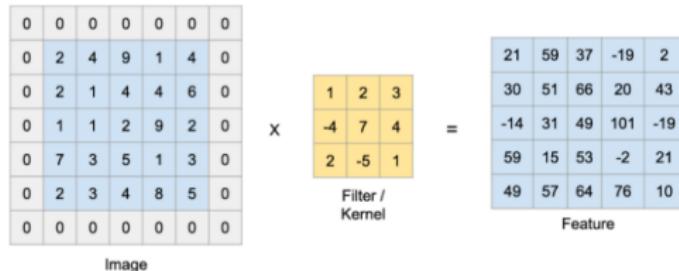


Figure: 1 filtro, kernel_size=3, strides=1, padding='same'

Capas Convolucionales

Parámetros principales

Los principales parámetros de una capa convolucional son:

- Número de filtros.
- Tamaño del filtro.
- Strides.
- Padding 'valid' o 'same'

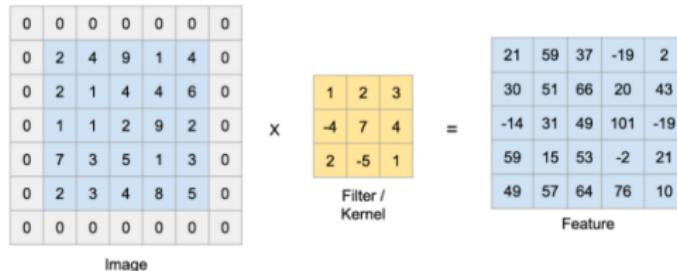


Figure: 1 filtro, kernel_size=3, strides=1, padding='same'

Capas Convolucionales

Parámetros principales

Los principales parámetros de una capa convolucional son:

- Número de filtros.
- Tamaño del filtro.
- Strides.
- Padding 'valid' o 'same'

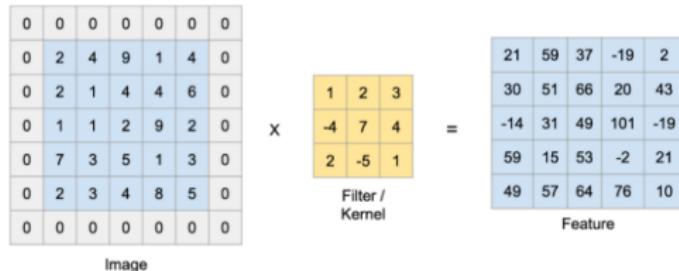


Figure: 1 filtro, kernel_size=3, strides=1, padding='same'

Capas Convolucionales

Redes convolucionales

Si apilamos distintas capas convolucionales obtendremos una red que detectará features o patrones de alto nivel, i.e, características abstractas de las imágenes.

Esta red aprenderá cuáles son los mejores filtros/kernel para detectar este tipo de features. Antes de las redes convolucionales estos filtros se definían ad-hoc para cada problema, limitando mucho su funcionamiento.

Lo bueno de este tipo de redes es que al aprender patrones locales y de alto nivel son muy robustas, por lo que podemos obtener grandes resultados.

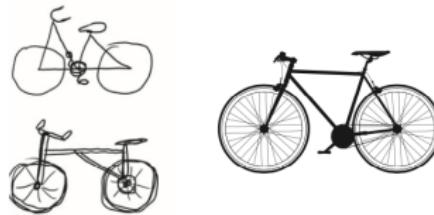


Figure: Las tres bicicletas tienen las mismas features de alto nivel

Capas Convolucionales

Redes convolucionales

Si apilamos distintas capas convolucionales obtendremos una red que detectará features o patrones de alto nivel, i.e, características abstractas de las imágenes.

Esta red aprenderá cuáles son los mejores filtros/kernel para detectar este tipo de features. Antes de las redes convolucionales estos filtros se definían ad-hoc para cada problema, limitando mucho su funcionamiento.

Lo bueno de este tipo de redes es que al aprender patrones locales y de alto nivel son muy robustas, por lo que podemos obtener grandes resultados.

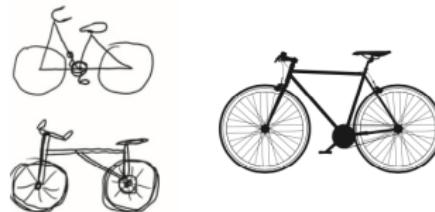


Figure: Las tres bicicletas tienen las mismas features de alto nivel

Capas Convolucionales

Redes convolucionales

Si apilamos distintas capas convolucionales obtendremos una red que detectará features o patrones de alto nivel, i.e, características abstractas de las imágenes.

Esta red aprenderá cuáles son los mejores filtros/kernel para detectar este tipo de features. Antes de las redes convolucionales estos filtros se definían ad-hoc para cada problema, limitando mucho su funcionamiento.

Lo bueno de este tipo de redes es que al aprender patrones locales y de alto nivel son muy robustas, por lo que podemos obtener grandes resultados.

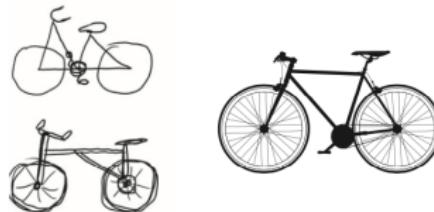


Figure: Las tres bicicletas tienen las mismas features de alto nivel

Capas Convolucionales

Redes convolucionales

Una vez que obtenemos estas features de alto nivel, es necesario analizarlas para poder dar una predicción. Esto puede ser realizado por una FCNN encima de nuestras capas convolucionales. Con ello obtendremos lo que se conoce como una Red Convolutacional.

Además, para reducir el coste computacional, normalmente se añaden capas maxpooling para dividir entre 2 el tamaño del input en cada capa.

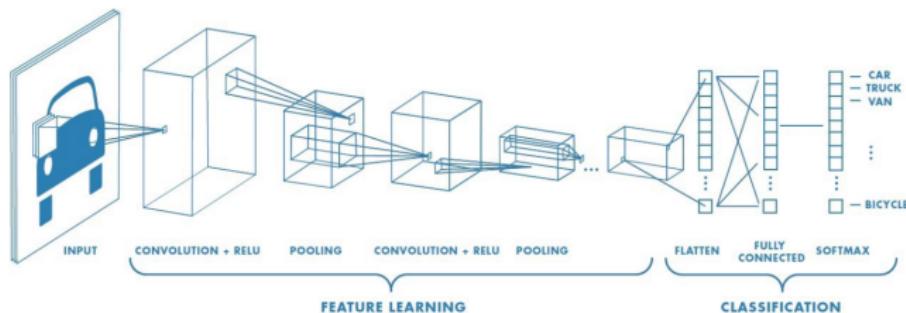


Figure: Arquitectura de Red Convolucional

Capas Convolucionales

Redes convolucionales

Una vez que obtenemos estas features de alto nivel, es necesario analizarlas para poder dar una predicción. Esto puede ser realizado por una FCNN encima de nuestras capas convolucionales. Con ello obtendremos lo que se conoce como una Red Convolutacional.

Además, para reducir el coste computacional, normalmente se añaden capas maxpooling para dividir entre 2 el tamaño del input en cada capa.

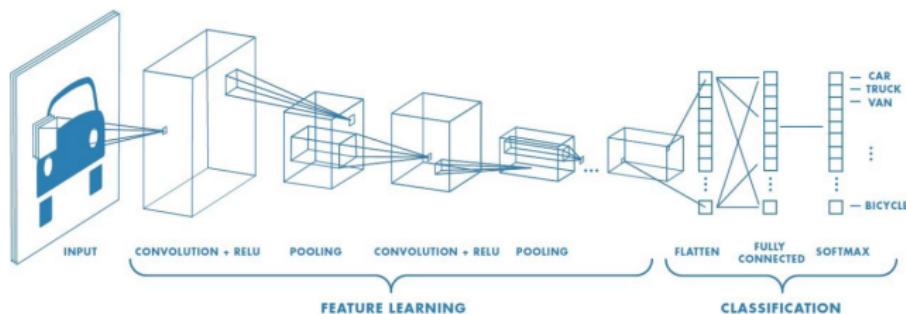


Figure: Arquitectura de Red Convolucional

Capas Convolucionales

Redes convolucionales

Normalmente los modelos de Deep Learning tienden a ser "cajas negras", pero en general las Redes Convolucionales no lo son. Podemos obtener bastante información que nos ayude a comprender su comportamiento, como por ejemplo:

- El output de cada capa: nos muestra cómo la red va transformando la imagen de partida. Obteniendo en cada capa un mapa de calor resaltando las partes del output donde el filtro tiene una respuesta alta con el input. A medida que profundizamos en la red el output se hace más disperso, ya que las features de alto nivel están solo presentes en las imágenes asociadas.

Capas Convolucionales

Redes convolucionales

Normalmente los modelos de Deep Learning tienden a ser "cajas negras", pero en general las Redes Convolucionales no lo son. Podemos obtener bastante información que nos ayude a comprender su comportamiento, como por ejemplo:

- El output de cada capa: nos muestra cómo la red va transformando la imagen de partida. Obteniendo en cada capa un mapa de calor resaltando las partes del output donde el filtro tiene una respuesta alta con el input. A medida que profundizamos en la red el output se hace más disperso, ya que las features de alto nivel están solo presentes en las imágenes asociadas.

Capas Convolucionales

Redes convolucionales

- Análisis de los filtros: con ello podemos ver cuáles son los patrones de la imagen que más los activan y por tanto entender cuáles son los patrones más representativos de nuestras imágenes.
- Mapa de calor con las partes más relevantes de la imagen: con ello podemos ver qué zonas de la imagen activan más la red y, por tanto, entender cuál es la parte más representativa de la imagen.

Capas Convolucionales

Redes convolucionales

- Análisis de los filtros: con ello podemos ver cuáles son los patrones de la imagen que más los activan y por tanto entender cuáles son los patrones más representativos de nuestras imágenes.
- Mapa de calor con las partes más relevantes de la imagen: con ello podemos ver qué zonas de la imagen activan más la red y, por tanto, entender cuál es la parte más representativa de la imagen.

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes**
- Optimizadores, métricas y funciones loss
- Hiperparámetros

Capas Recurrentes

Motivación Redes Recurrentes

Al igual que las Redes Convolucionales están pensadas para trabajar con imágenes, las Redes Recurrentes compuestas por Capas Recurrentes están pensadas para trabajar con secuencias de datos. Están diseñadas para trabajar en problemas donde:

- La longitud de la entrada de datos es variable. (Frases)
- Hay una dimensión en la que existe una componente temporal o una relación de orden. (Valor en Bolsa)
- Secuencias de datos muy largas que no pueden procesarse de golpe. (Histórico de temperatura en una ciudad)

Capas Recurrentes

Motivación Redes Recurrentes

Al igual que las Redes Convolucionales están pensadas para trabajar con imágenes, las Redes Recurrentes compuestas por Capas Recurrentes están pensadas para trabajar con secuencias de datos. Están diseñadas para trabajar en problemas donde:

- La longitud de la entrada de datos es variable. (Frases)
- Hay una dimensión en la que existe una componente temporal o una relación de orden. (Valor en Bolsa)
- Secuencias de datos muy largas que no pueden procesarse de golpe. (Histórico de temperatura en una ciudad)

Capas Recurrentes

Motivación Redes Recurrentes

Al igual que las Redes Convolucionales están pensadas para trabajar con imágenes, las Redes Recurrentes compuestas por Capas Recurrentes están pensadas para trabajar con secuencias de datos. Están diseñadas para trabajar en problemas donde:

- La longitud de la entrada de datos es variable. (Frases)
- Hay una dimensión en la que existe una componente temporal o una relación de orden. (Valor en Bolsa)
- Secuencias de datos muy largas que no pueden procesarse de golpe. (Histórico de temperatura en una ciudad)

Capas Recurrentes

Motivación Redes Recurrentes

Al igual que las Redes Convolucionales están pensadas para trabajar con imágenes, las Redes Recurrentes compuestas por Capas Recurrentes están pensadas para trabajar con secuencias de datos. Están diseñadas para trabajar en problemas donde:

- La longitud de la entrada de datos es variable. (Frases)
- Hay una dimensión en la que existe una componente temporal o una relación de orden. (Valor en Bolsa)
- Secuencias de datos muy largas que no pueden procesarse de golpe. (Histórico de temperatura en una ciudad)

Capas Recurrentes

Motivación Redes Recurrentes

Las redes recurrentes procesan la información en orden secuencial manteniendo un state o "resumen" relativo a la información procesada. Este proceso iterativo recorre la secuencia de datos y la procesa de manera incremental, a diferencia de una FCNN que la procesa de una sola vez.

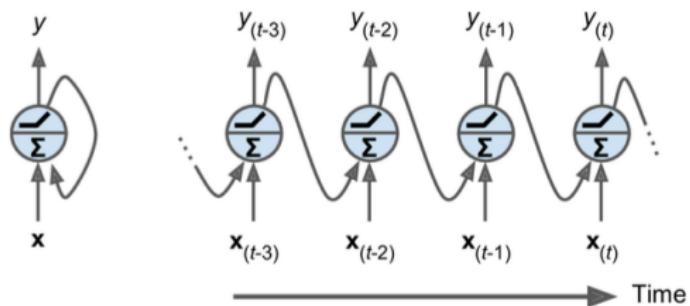


Figure: Red Recurrente

Capas Recurrentes

Funcionamiento

En el caso más simple podemos expresar el output de una red recurrente de la siguiente forma:

$$y_t = \sigma(W_x x_t + W_y y_{t-1} + b)$$

En este caso el estado latente o "resumen" $h_t = y_t$, pero en general no tiene por qué ser así.

Como podemos ver, la capa recurrente itera en un loop sin longitud predefinida procesando la secuencia. La longitud de x_t que se procesa en cada paso no tiene por qué ser 1, pueden ser bloques de m instantes de tiempo

$$x_t = (s_{m \cdot t}, s_{m \cdot t+1}, \dots, s_{m \cdot (t+1)-1})$$

Capas Recurrentes

Funcionamiento

En el caso más simple podemos expresar el output de una red recurrente de la siguiente forma:

$$y_t = \sigma(W_x x_t + W_y y_{t-1} + b)$$

En este caso el estado latente o "resumen" $h_t = y_t$, pero en general no tiene por qué ser así.

Como podemos ver, la capa recurrente itera en un loop sin longitud predefinida procesando la secuencia. La longitud de x_t que se procesa en cada paso no tiene por qué ser 1, pueden ser bloques de m instantes de tiempo

$$x_t = (s_{m \cdot t}, s_{m \cdot t+1}, \dots, s_{m \cdot (t+1)-1})$$

Capas Recurrentes

Funcionamiento

En función del problema con el que estemos tratando, el intervalo de tiempo a futuro que queremos predecir puede variar. Por ejemplo, dada una frase sin terminar podremos querer que nos prediga la siguiente palabra o que nos complete la frase. Las posibles configuraciones que podemos seleccionar para una Red Recurrente son:

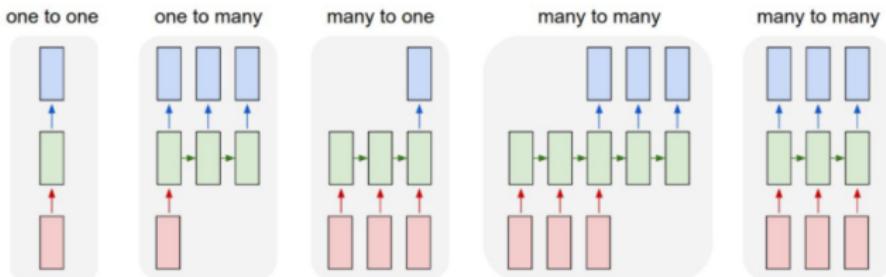


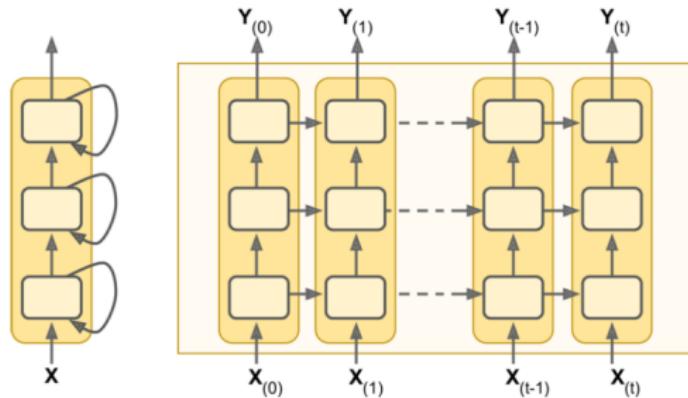
Figure: Configuraciones RNN

Capas Recurrentes

Redes Recurrentes Profundas

Al igual que con otras capas, las redes recurrentes se pueden apilar para formar una Red Recurrente Profunda. Con esto conseguimos un análisis profundo de la secuencia de datos.

Se debe tener cuidado tanto con la profundidad de la red como con la longitud de la secuencia de datos, ya que si estos valores son elevados la propagación del error a lo largo de red se "diluirá" y será muy difícil entrenar la red.

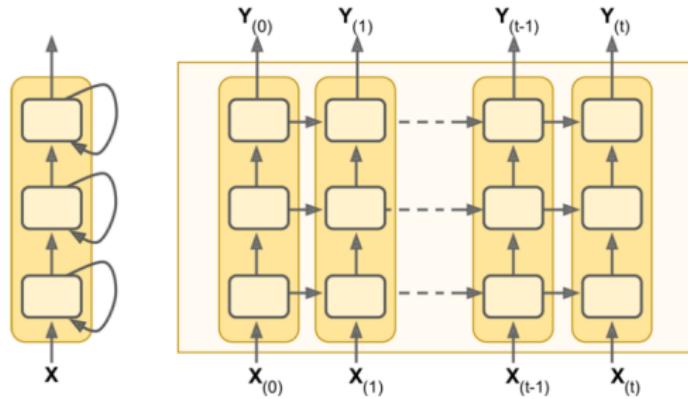


Capas Recurrentes

Redes Recurrentes Profundas

Al igual que con otras capas, las redes recurrentes se pueden apilar para formar una Red Recurrente Profunda. Con esto conseguimos un análisis profundo de la secuencia de datos.

Se debe tener cuidado tanto con la profundidad de la red como con la longitud de la secuencia de datos, ya que si estos valores son elevados la propagación del error a lo largo de red se "diluirá" y será muy difícil entrenar la red.



Capas Recurrentes

Mejoras

Podemos introducir un par de modificaciones a la arquitectura de la capa recurrente para obtener mejores resultados, por ejemplo:

- Introducir celdas recurrentes con memoria a largo plazo: las celdas LSTM guardan dos estados internamente, uno para la memoria a corto plazo y otro para la memoria a largo plazo. En general es recomendable usar este tipo de celdas, ya que se ha probado que consiguen mitigar el problema del vanishing gradient cuando entrenamos una RNN sobre una secuencia larga. (¿?)
- Introducir capas bidireccionales: este tipo de capas son útiles cuando en la secuencia de datos no existe una dependencia temporal monótona creciente (histórico temperatura vs frases).

Capas Recurrentes

Mejoras

Podemos introducir un par de modificaciones a la arquitectura de la capa recurrente para obtener mejores resultados, por ejemplo:

- Introducir celdas recurrentes con memoria a largo plazo: las celdas LSTM guardan dos estados internamente, uno para la memoria a corto plazo y otro para la memoria a largo plazo. En general es recomendable usar este tipo de celdas, ya que se ha probado que consiguen mitigar el problema del vanishing gradient cuando entrenamos una RNN sobre una secuencia larga. (¿?)
- Introducir capas bidireccionales: este tipo de capas son útiles cuando en la secuencia de datos no existe una dependencia temporal monótona creciente (histórico temperatura vs frases).

Capas Recurrentes

Mejoras

Podemos introducir un par de modificaciones a la arquitectura de la capa recurrente para obtener mejores resultados, por ejemplo:

- Introducir celdas recurrentes con memoria a largo plazo: las celdas LSTM guardan dos estados internamente, uno para la memoria a corto plazo y otro para la memoria a largo plazo. En general es recomendable usar este tipo de celdas, ya que se ha probado que consiguen mitigar el problema del vanishing gradient cuando entrenamos una RNN sobre una secuencia larga. (¿?)
- Introducir capas bidireccionales: este tipo de capas son útiles cuando en la secuencia de datos no existe una dependencia temporal monótona creciente (histórico temperatura vs frases).

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss**
- Hiperparámetros

Optimizadores

mini-batch

Los optimizadores son los que se encargan del proceso de aprendizaje, utilizando el gradiente de la función loss y el learning rate seleccionado para modificar los pesos de la red. Ya hemos visto el más simple con sus variantes, el Gradient Descent.

La variante mini-batch puede aplicarse a cualquier optimizador y, de hecho, es siempre recomendable. El tamaño del batch depende mucho del tamaño de la red, tamaño de la entrada y el problema a tratar, pero en general se usa un tamaño de entre 5 y 500. Hay que tener en cuenta que existe un trade-off entre velocidad de aprendizaje y calidad del aprendizaje.

Esta mejora se basa en el hecho de que

$$\sum_{j \in U} \frac{\partial f}{\partial \theta} \Big|_{x_j} \approx \sum_{i \in [n]} \frac{\partial f}{\partial \theta} \Big|_{x_i} \quad | \quad U \subseteq [n]$$

Con lo que obtenemos un aprendizaje similar a un coste mucho menor.

Optimizadores

mini-batch

Los optimizadores son los que se encargan del proceso de aprendizaje, utilizando el gradiente de la función loss y el learning rate seleccionado para modificar los pesos de la red. Ya hemos visto el más simple con sus variantes, el Gradient Descent.

La variante mini-batch puede aplicarse a cualquier optimizador y, de hecho, es siempre recomendable. El tamaño del batch depende mucho del tamaño de la red, tamaño de la entrada y el problema a tratar, pero en general se usa un tamaño de entre 5 y 500. Hay que tener en cuenta que existe un trade-off entre velocidad de aprendizaje y calidad del aprendizaje.

Esta mejora se basa en el hecho de que

$$\sum_{j \in U} \frac{\partial f}{\partial \theta} \Big|_{x_j} \approx \sum_{i \in [n]} \frac{\partial f}{\partial \theta} \Big|_{x_i} \quad | \ U \subseteq [n]$$

Con lo que obtenemos un aprendizaje similar a un coste mucho menor.

Optimizadores

mini-batch

Los optimizadores son los que se encargan del proceso de aprendizaje, utilizando el gradiente de la función loss y el learning rate seleccionado para modificar los pesos de la red. Ya hemos visto el más simple con sus variantes, el Gradient Descent.

La variante mini-batch puede aplicarse a cualquier optimizador y, de hecho, es siempre recomendable. El tamaño del batch depende mucho del tamaño de la red, tamaño de la entrada y el problema a tratar, pero en general se usa un tamaño de entre 5 y 500. Hay que tener en cuenta que existe un trade-off entre velocidad de aprendizaje y calidad del aprendizaje. Esta mejora se basa en el hecho de que

$$\sum_{j \in U} \frac{\partial f}{\partial \theta} \Big|_{x_j} \approx \sum_{i \in [n]} \frac{\partial f}{\partial \theta} \Big|_{x_i} \mid U \subseteq [n]$$

Con lo que obtenemos un aprendizaje similar a un coste mucho menor.

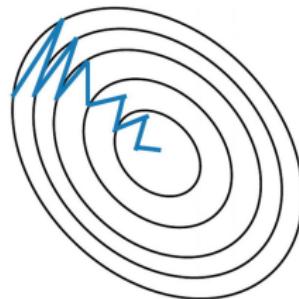
Optimizadores

Momentum

Además del mini-batch es recomendable usar el momentum. La optimización con momentum tiene en cuenta los gradientes de las etapas anteriores, con lo que conseguimos una inercia y aceleración en la convergencia del modelo. Es como bajar una cuesta ayudándonos de la gravedad (momentum) en vez de con velocidad constante (Gradient Descent).



Stochastic Gradient
Descent **without**
Momentum



Stochastic Gradient
Descent **with**
Momentum

Figure: Aprendizaje sin momentum vs con momentum

Optimizadores

Momentum

La ecuación para actualizar los pesos con momentum es la siguiente.

$$\begin{aligned}m &\leftarrow \beta m - \eta \nabla_{\theta} L \\ \theta &\leftarrow \theta + m\end{aligned}$$

Obteniendo en n pasos lo siguiente.

$$\theta - \beta^{n-1} \eta \nabla_{\theta} L_{x_1} - \cdots - \eta \nabla_{\theta} L_{x_n}$$

Selección del Optimizador

Existen distintos optimizadores, cada uno con alguna mejora para la optimización de la trayectoria. El que se ha probado que da mejor resultado es el optimizador Adam, aunque también es recomendable probar los optimizadores RMSprop, Adadelta y Adagrad.

Optimizadores

Momentum

La ecuación para actualizar los pesos con momentum es la siguiente.

$$\begin{aligned}m &\leftarrow \beta m - \eta \nabla_{\theta} L \\ \theta &\leftarrow \theta + m\end{aligned}$$

Obteniendo en n pasos lo siguiente.

$$\theta - \beta^{n-1} \eta \nabla_{\theta} L_{x_1} - \cdots - \eta \nabla_{\theta} L_{x_n}$$

Selección del Optimizador

Existen distintos optimizadores, cada uno con alguna mejora para la optimización de la trayectoria. El que se ha probado que da mejor resultado es el optimizador Adam, aunque también es recomendable probar los optimizadores RMSprop, Adadelta y Adagrad.

Optimizadores

Momentum

La ecuación para actualizar los pesos con momentum es la siguiente.

$$\begin{aligned}m &\leftarrow \beta m - \eta \nabla_{\theta} L \\ \theta &\leftarrow \theta + m\end{aligned}$$

Obteniendo en n pasos lo siguiente.

$$\theta - \beta^{n-1} \eta \nabla_{\theta} L_{x_1} - \cdots - \eta \nabla_{\theta} L_{x_n}$$

Selección del Optimizador

Existen distintos optimizadores, cada uno con alguna mejora para la optimización de la trayectoria. El que se ha probado que da mejor resultado es el optimizador Adam, aunque también es recomendable probar los optimizadores RMSprop, Adadelta y Adagrad.

Selección de la métrica

La métrica es una función que nos determina el rendimiento de nuestro modelo sobre un conjunto de datos. Esta función es totalmente dependiente del problema y su selección o construcción se tiene que hacer ad-hoc, aunque siempre hay ciertas recomendaciones o pautas a seguir. Existen distintas métricas muy utilizadas que cubren un gran abanico de posibilidades, como por ejemplo:

- Accuracy o Top-k accuracy
- Kullback-Leibler divergence
- Mean Squared Error, Mean Absolute Error
- AUC, Recall, Precisión, F1
- IoU
- R2

Selección de la métrica

La métrica es una función que nos determina el rendimiento de nuestro modelo sobre un conjunto de datos. Esta función es totalmente dependiente del problema y su selección o construcción se tiene que hacer ad-hoc, aunque siempre hay ciertas recomendaciones o pautas a seguir. Existen distintas métricas muy utilizadas que cubren un gran abanico de posibilidades, como por ejemplo:

- Accuracy o Top-k accuracy
- Kullback-Leibler divergence
- Mean Squared Error, Mean Absolute Error
- AUC, Recall, Precisión, F1
- IoU
- R2

Funciones Loss

Selección de la función loss

La función loss tiene que estar relacionada con la métrica seleccionada, además de poseer buenas propiedades para el entrenamiento. Esta relación ha de ser, en la medida de lo posible, una dependencia monótona creciente de manera que:

$$\text{metric}(\text{model}_1) > \text{metric}(\text{model}_2) \iff \text{loss}(\text{model}_1) > \text{loss}(\text{model}_2)$$

Esto es muy difícil de conseguir, pero en general existen varias parejas (métrica, loss) que funcionan bastante bien en la práctica. Incluso varias losses pueden ser útiles para una misma métrica.

Funciones Loss

Selección de la función loss

La función loss tiene que estar relacionada con la métrica seleccionada, además de poseer buenas propiedades para el entrenamiento. Esta relación ha de ser, en la medida de lo posible, una dependencia monótona creciente de manera que:

$$\text{metric}(\text{model}_1) > \text{metric}(\text{model}_2) \iff \text{loss}(\text{model}_1) > \text{loss}(\text{model}_2)$$

Esto es muy difícil de conseguir, pero en general existen varias parejas (métrica, loss) que funcionan bastante bien en la práctica. Incluso varias losses pueden ser útiles para una misma métrica.

Funciones Loss

Selección de la función loss

Al igual que pasa con las métricas existen distintas funciones loss muy usadas y útiles en la mayoría de los casos:

- BinaryCrossentropy
- Focal Loss
- Kullback-Leibler divergence
- Mean Squared Error, Mean Absolute Error
- Cosine Similarity
- IoU loss

Custom losses

Siempre se puede diseñar una loss para resolver un problema específico. Por ejemplo combinar BCE+IoU nos produce mejores siluetas en la segmentación, menor accuracy y predicciones polarizadas.

Funciones Loss

Selección de la función loss

Al igual que pasa con las métricas existen distintas funciones loss muy usadas y útiles en la mayoría de los casos:

- BinaryCrossentropy
- Focal Loss
- Kullback-Leibler divergence
- Mean Squared Error, Mean Absolute Error
- Cosine Similarity
- IoU loss

Custom losses

Siempre se puede diseñar una loss para resolver un problema específico. Por ejemplo combinar BCE+IoU nos produce mejores siluetas en la segmentación, menor accuracy y predicciones polarizadas.

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

4 Componentes de una Red Neuronal

- Funciones de activación
- Capas Convolucionales
- Capas Recurrentes
- Optimizadores, métricas y funciones loss
- **Hiperparámetros**

Hiperparámetros

Hiperparámetros del modelo

Como la mayoría de los modelos de Machine Learning las NN también tienen hiperparámetros que deberemos seleccionar. Siempre nos podemos ayudar de un optimizador como un random search para seleccionar los mejores.

A diferencia de los modelos de ML, el número de hiperparámetros que tiene una red es enorme, lo cual hace que muchas veces sea difícil hacer el fine tuning.

A grandes rasgos podemos distinguir dos tipos de hiperparámetros en una red, los referentes a la arquitectura y los referentes al entrenamiento.

Hiperparámetros

Hiperparámetros del modelo

Como la mayoría de los modelos de Machine Learning las NN también tienen hiperparámetros que deberemos seleccionar. Siempre nos podemos ayudar de un optimizador como un random search para seleccionar los mejores.

A diferencia de los modelos de ML, el número de hiperparámetros que tiene una red es enorme, lo cual hace que muchas veces sea difícil hacer el fine tuning.

A grandes rasgos podemos distinguir dos tipos de hiperparámetros en una red, los referentes a la arquitectura y los referentes al entrenamiento.

Hiperparámetros

Hiperparámetros del modelo

Como la mayoría de los modelos de Machine Learning las NN también tienen hiperparámetros que deberemos seleccionar. Siempre nos podemos ayudar de un optimizador como un random search para seleccionar los mejores.

A diferencia de los modelos de ML, el número de hiperparámetros que tiene una red es enorme, lo cual hace que muchas veces sea difícil hacer el fine tuning.

A grandes rasgos podemos distinguir dos tipos de hiperparámetros en una red, los referentes a la arquitectura y los referentes al entrenamiento.

Hiperparámetros

Hiperparámetros de la arquitectura

- Tipo de capas que se van a usar
- Número de capas
- Número de neuronas
- Funciones de activación
- Tamaño de los filtros, strides, padding, etc
- Parámetros de regularización

Hiperparámetros del entrenamiento

- Número de épocas y steps
- Learning Rate
- Optimizador
- Función Loss
- Batch size