Universidad Nacional de Córdoba

Facultad de Ingeniería Ciencias Exactas, Físicas y Naturales

Conceptos esenciales de inteligencia artificial - Machine Learning aplicado a problemas de ingeniería

Práctica profesional supervisada

Autores:
Pichetti, Augusto
Salse, Lucas Antonio

22 de septiembre de 2021

Índice

1. Introducción

Hoy en día como sabemos existe una importancia extremadamente alta sobre los datos que se recolectan en el día a día, ya sean datos obtenidos de carácter científico (mediciones climáticas, radiación, satelitales, médicos, etc.) o bien datos personales de cada individuo (gastos, ubicación, compras, pasatiempo, etc.). Todo esto generó que existiera la necesidad de poder emplear dichos datos y obtener un provecho de los mismos, con el objetivo de solucionar infinidades de problemas de carácter general (clima, enfermedades, etc.) como así también para beneficios económicos de las empresas (publicidad específica, segmentación, etc.). Es por esta razón para poder conseguir estos objetivos se emplea el concepto de inteligencia artificial.

1.1. Inteligencia Artificial

- La Inteligencia Artificial (Artificial Intelligence) se define como el estudio de los .ªgentes inteligentes", i.e. cualquier dispositivo que perciba su entorno y tome medidas que maximicen sus posibilidades de lograr con éxito sus objetivos. Poole et al.[1]

- La capacidad de un sistema para interpretar correctamente datos externos, para aprender de dichos datos y emplear esos conocimientos para lograr tareas y metas concretas a través de la adaptación flexible. Kaplan y Michael Haenlein. [2]

Esta definición nos da la idea de que la IA es un sistema reactivo, que reacciona a cambios externos y actúa en consecuencia.

1.2. Aprendizaje Automático

Una de las subramas de la inteligencia artificial es el aprendizaje automático que a su vez el subcampo de la ciencia de la computación, como se ve en la siguiente imagen:

El aprendizaje automático (Machine Learning) es el estudio científico de algoritmos y modelos estadísticos que los sistemas informáticos utilizan para realizar una tarea específica sin utilizar instrucciones explícitas, sino que se basan en patrones e inferencias. Es visto como un subcampo de inteligencia artificial. Los algoritmos de aprendizaje automático crean un modelo matemático basado en datos de muestra, conocidos como "datos de entrenamiento", para hacer predicciones o decisiones sin ser programado explícitamente para realizar la tarea. Bishop. [3]

Por tanto el Aprendizaje Automático es la generación de un modelo de predicción de salida a partir de grandes cantidades de datos de entrada, realizando un tratamiento de los mismos a través de diferentes etapas bien definidas, como se pueden apreciar en la Figura 1, las cuales iremos desarrollando en diferentes secciones.

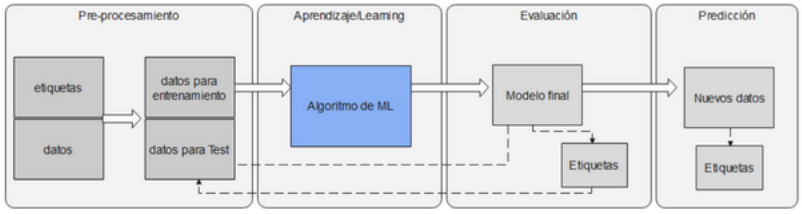


Figura 1: Diagrama de flujo de una aplicación de Machine Learning.

Es importante destacar la independencia del aprendizaje automático al momento de tomar decisiones a partir de los datos proporcionados sin intervención externa, es decir que no hay una especificación de reglas que dictan cómo deben ser tomadas estas decisiones. A su vez, los modelos obtenidos a partir de los algoritmos de Machine Learning deben tener la capacidad de predecir a partir de nuevos datos, nunca antes procesados por el modelo, a esto se lo conoce como generalización.

1.3. Preprocesamiento

A menudo, los resultados van a depender más de la calidad de los datos en relación al problema, que de la parte de aprendizaje automático.

Como vemos la importancia de un buen manejo de datos es vital para poder conseguir los objetivos propuestos es por esto que debemos ser capaces de realizar un preprocesamiento de los mismos.

- Los datos del mundo real están "sucios": incompletos (datos perdidos, ausencia de datos de alguna tipología), ruidosos (con errores o outliers, variables redundantes), inconsistentes (diferencias en codificación o nombres).
- Datos de calidad → Resultados de calidad.
- Es necesario analizar las características de los datos para conocer mejor el problema, detectar posibles datos erróneos, dependencias, outliers, valores perdidos, etc.
- Deberemos ser capaces de utilizar ciertas técnicas que nos permitan analizar y visualizar que los datos estén dentro de todo correctos.

Tareas típicas de esta etapa:

- Selección de datos.
  - Extracción de características.
  - Selección de características descartables para reducir el número de variables.
- Limpieza de datos:

- Recuperación de valores perdidos (imputación de datos).
- Tratamiento de valores anómalos (outliers).
- Suavizar el ruido.
- Eliminar inconsistencias.

- Transformación de datos. Por ej, convertir una variable nominal en varias variables binarias, escalado de datos, fecha de nacimiento -¿edad.
- Reducción de dimensionalidad. Consiste en emplear técnicas como PCA (análisis de componentes principales) para obtener combinaciones lineales de variables que reduzcan la dimensión de los datos.

Por lo tanto, una vez que se obtiene el dataset de entrada, es primordial realizar estas tareas de forma que presentemos un conjunto de datos que esté en condiciones de ser entrenado y luego el modelo resultante, al momento de ser probado con datos desconocidos, tenga un desempeño óptimo. Además, de ser necesario, se pueden utilizar algoritmos de Aprendizaje no supervisado los cuales resultan muy útiles cuando se cuentan con grandes cantidades de datos en contextos no conocidos.

2. Tipos de aprendizaje

Hay dos tipos de aprendizaje automático, aprendizaje supervisado y aprendizaje no supervisado.

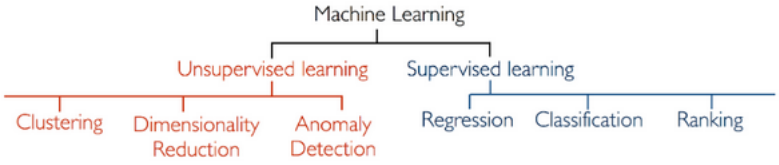


Figura 2: Clasificación Machine Learning.

2.1. Aprendizaje supervisado

Los modelos de **aprendizaje supervisado** son aquellos en los que se aprenden funciones, relaciones que asocian entradas con salidas, por lo que se ajustan a un conjunto de ejemplos de los que conocemos la relación entre la entrada y la salida deseada. Esto es que tenemos un conjunto de datos con los cuales se va a entrenar al modelo (training) que a su vez se encuentran etiquetados (label), es decir, que por cada dato de entrada sabemos a qué label pertenece. Luego lo aprendido se ve reflejado mediante una función que mapea las variables de entrada con las variables de salida, siempre intentando minimizar una

función de error. Este hecho incluso llega a proporcionar una de las clasificaciones más habituales en el tipo de algoritmos que se desarrollan, así, dependiendo del tipo de salida, suele darse una subcategoría que diferencia entre modelos de **clasificación**, si la salida es un valor categórico (por ejemplo, una enumeración, o un conjunto finito de clases) , y modelos de **regresión**, si la salida es un valor de un espacio continuo.

Otra parte importante de este aprendizaje es la validación de nuestra función. Esto se lleva a cabo utilizando un conjunto de datos para testeo (test) que por lo general resulta ser el 20 % [4] de los datos de entrenamiento. Con esto podremos evaluar y ver que tan eficiente es nuestro modelo. A continuación en la figura 3 se observa con un ejemplo como se encuentra estructurado este aprendizaje supervisado:

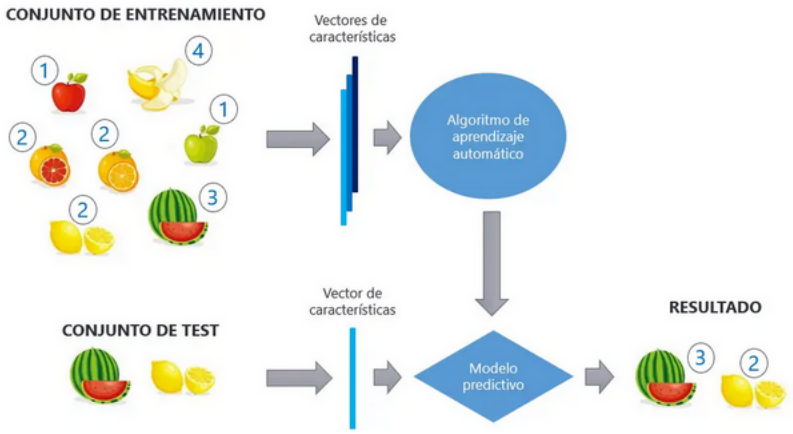


Figura 3: Etapas del aprendizaje supervisado.

En la imagen anterior de manera resumida tenemos:

- Conjunto de datos con su correspondiente etiqueta.
- Algoritmo de aprendizaje.
- Su transformación a vectores de características
- Conjunto de datos usados para test.
- Resultados.

2.1.1. Clasificación

En clasificación, la etiqueta es discreta, por ejemplo perro y no perro. En otras palabras, se proporciona una distinción clara entre las categorías. Es más, es importante indicar que estas categorías son nominales y no ordinales. Las variables nominales y ordinales son

ambas subcategorías de las variables categóricas. Las variables ordinales tienen asociado un orden, por ejemplo, las tallas de las camisetas "XL ¿L ¿M ¿S". Por el contrario, las variables nominales no implican un orden, por ejemplo, no podemos asumir (en general) "naranja ¿azul ¿verde".

Hay dos tipos principales de clasificaciones [5]:

- **Clasificación Binaria**: Es un tipo de clasificación en el que tan solo se pueden asignar dos clases diferentes (0 o 1). El ejemplo típico es la detección de email spam, en la que cada email es: spam → en cuyo caso será etiquetado con un 1 ; o no lo es → etiquetado con un 0.
- **Clasificación Multi-clase**: Se pueden asignar múltiples categorías a las observaciones. Como el reconocimiento de caracteres de escritura manual de números (en el que las clases van de 0 a 9).

El siguiente ejemplo es altamente representativo de una clasificación binaria:

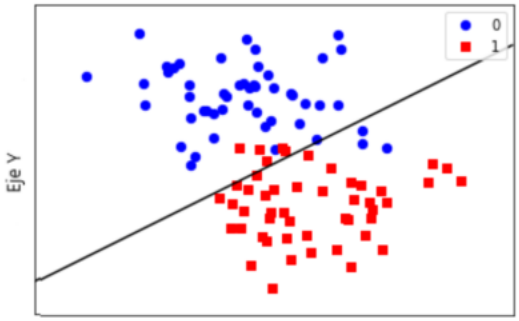


Figura 4: Ejemplo de clasificación binaria.

Para poder apreciar el concepto de clasificación binaria, en la Fig. 4 [6] se han graficado datos bidimensionales, es decir que cada dato tiene dos valores asociados de acuerdo a los ejes X e Y. El dataset cuenta con 100 muestras, las cuales están divididas en dos clases: ceros (círculos azules) y unos (cuadrados rojos).

El modelo a utilizar para la predicción será uno de los más conocidos y simples de utilizar en clasificación, regresión logística. Éste es un modelo lineal, lo que significa que creará una frontera de decisión que es lineal en el espacio de entrada, en 2D esto quiere decir que generará una línea recta para separar los puntos azules de los rojos. Como puede observarse, el modelo no es 100 % preciso ya que algunos puntos azules están en la categoría de los rojos y viceversa, por eso es que existen diversos modelos y debemos elegir según nuestro criterio cuál de ellos se adecúa mejor a nuestras necesidades.

Es importante indicar que no todos los modelos de clasificación serán útiles para separar adecuadamente las diferentes clases de un conjunto de datos. Algunos algoritmos, como

el "perceptron" (que se basa en redes neuronales artificiales básicas) no convergerán al aprender los pesos del modelo si las clases no pueden separarse por una frontera de decisión lineal. Algunos de los casos más típicos se representan en las siguientes figuras 5:

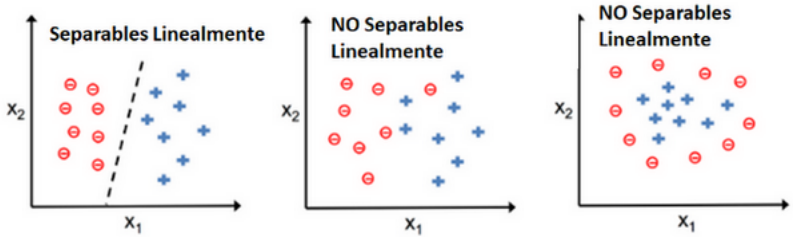


Figura 5: Clasificación lineal con diferentes datos.

Por tanto, seleccionar el algoritmo adecuado se convierte en un factor de importancia crucial en los problemas de clasificación. Es por esto que podemos ver sus ventajas y desventajas de cada uno en la siguiente tabla 6.

	Ventajas	Desventajas
Naive Bayes	• Simple de entender e implementar . • No necesita una gran cantidad de datos de entrenamiento. • Rápido.	• Supone que cada característica es independiente. • Sufre al tener características irrelevantes.
Regresion logistica	• Simple de entender e implementar. • Rara vez existe sobreajuste. • Rápido de entrenar.	• Es muy difícil lograr que se ajuste a datos no lineales. • Los valores atípicos alteran la precisión del modelo.
KNN	• Eficaz en datasets de varias clases. • Entrenamiento rápido.	• La dimensionalidad del dataset merma el rendimiento. • Lento en fase de predicción.
Arbol de decision	• Robusto a muestras con ruido. • Fácil de interpretar. • Resuelve problemas no lineales.	• Cuando hay muchas etiquetas de clase, los cálculos pueden ser complejos. • Puede sufrir sobreajuste.
Clasificador de bosque aleatorio	• No sufre sobreajuste como el árbol de decisión. • Funciona muy bien en grandes bases de datos. • Maneja automáticamente los valores faltantes.	• Consume mucho tiempo y recursos computacionales. • Difícil de interpretar. • Necesito elegir la cantidad adecuada de árboles.
Máquinas de vectores de soporte (SVC)	• Eficaz en espacios de gran dimensión. • Puede manejar soluciones no lineales. • Robusto al ruido.	• Lento para entrenarse con grandes conjuntos de datos. • Ineficaz si las clases se superponen. • Hay que elegir una buena función de kernel. • Difícil de interpretar al aplicar kernels no lineales.

Figura 6: Ventajas y desventajas de los algoritmos de clasificación.

2.1.2. Regresión

En regresión, la etiqueta es continua, es decir una salida real. Por ejemplo, en astronomía, la tarea de determinar si un objeto es una estrella, una galaxia o un cuásar es un problema de clasificación: la etiqueta viene de tres categorías distintas. Por otro la-

do, podríamos querer estimar la edad de un objeto basándonos en su imagen: esto sería regresión, porque la etiqueta (edad) es una cantidad continua [6].

En los problemas de regresión, tenemos como entradas las variables independientes o explicativas y las salidas o etiquetas son variables continuas. Por lo tanto, los modelos de regresión deben encontrar una relación (función lineal, polinomial, entre otras) que nos permitan predecir la salida.

En otras palabras, la regresión lineal es un método para predecir la variable dependiente (y) en función de los valores de las variables independientes (X). Se puede usar para los casos donde queremos predecir alguna cantidad continua.

Variables independientes (características): es una variable que se manipula para determinar el valor de una variable dependiente. Simplemente, son las características que queremos usar para predecir algún valor dado de y.

Variable dependiente (objetivo): la variable dependiente depende de los valores de la variable independiente. En pocas palabras, es la característica que estamos tratando de predecir. Esto también se puede conocer comúnmente como una variable de respuesta.

Este algoritmo **consiste en hallar una línea recta que mejor encaje en un conjunto de datos dados**, este conjunto de datos comprende las variables independientes y dependientes. Para esto podremos utilizar métodos matemáticos como el de los mínimos cuadrados, para buscar minimizar la distancia vertical de todos los puntos a la línea recta.

Una vez obtenida esta línea recta, seremos capaces de hacer predicciones hipotéticas sobre cuál será el valor de "y" dado "X".

Ventajas y desventajas sobre algoritmos de regresión:

	Ventajas	Desventajas
Ridge	• El costo computacional no es mayor que otros algoritmos. • Permite evitar el sobreajuste.	• Se necesita una excelente selección del hiperparámetro alpha. • Incrementa el sesgo.
LASSO	• Evita el sobreajuste. • Selecciona características tendiendo que sus coeficientes sean cero.	• Las características seleccionadas tienen demasiado sesgo. • Si tenemos n datos y p características, LASSO solo seleciona como máximo n características. • El rendimiento de la predicción es peor que para Ridge Regression.
Elastic Net	• Eficaz con muestras de gran dimensión.	• Alto costo computacional en comparación con LASSO o Ridge.
KNN Regresor	• No tiene período de entrenamiento. • Fácil de interpretar. • Permite agregar datos al modelo sin inconvenientes en la precisión del algoritmo.	• La dimensionalidad del conjunto de datos influye mucho en el rendimiento. • Es sensible a datos ruidosos, valores faltantes y outliers. • En grandes datasets se vuelve muy alto el costo computacional para calcular distancias.
Regresor de árbol de decisión	• No sufre sobreajuste como el árbol de decisión. • Funciona muy bien en grandes bases de datos. • Maneja automáticamente los valores faltantes.	• Al trabajar con variables continuas, se pierde mucha información al categorizar. • Necesita variables correlacionadas. • Alto tiempo de entrenamiento. • Se puede volver demasiado complejo.
Máquinas de vectores de soporte (SVC)	• Útil cuando las clases no son linealmente separables.	• Suelen ser ineficientes al momento del entrenamiento.

Figura 7: Ventajas y desventajas de los algoritmos de regresión.

2.2. Aprendizaje no supervisado

En el **aprendizaje no supervisado** no hay una salida deseada asociada a los datos. En su lugar, estamos interesados en extraer algún tipo de conocimiento o modelo a partir de los datos. En cierto sentido, se podría pensar que el aprendizaje no supervisado es una forma de descubrir etiquetas (label) a partir de los propios datos. Este tipo de aprendizaje suele ser más difícil de entender y evaluar.

Un ejemplo de aprendizaje no supervisado podría ser si nos encontramos con un texto extenso y queremos obtener una especie de resumen, de los temas o tópicos relevantes, probablemente de antemano no se sabe cuales son o su cantidad, por lo tanto nos enfrentamos a la situación de no conocer cuales serian las salidas esperadas del modelo. Otros ejemplos clásicos pueden ser agrupar fotografías similares o separación de diferentes fuentes que originan un determinado sonido.

Para poder llevar a cabo esto se puede realizar una agrupación de los datos según su similaridad (**clustering**), simplificando las estructura de los mismos manteniendo sus características fundamentales (como en los procesos de **reducción de la dimensionalidad**), o extrayendo la estructura interna con la que se distribuyen los datos en su espacio original (**aprendizaje topológico**) [7].

Algunas veces los dos tipos de aprendizaje se combinan: el **aprendizaje no supervisado** puede ser utilizado para encontrar características útiles en datos heterogéneos y luego usar estas características en un framework de **aprendizaje supervisado.**

En las próximas subsecciones explicaremos brevemente algunas de las tareas que comprenden el Aprendizaje no Supervisado.

2.2.1. Detección de anomalías

Uno de los primeros pasos a realizar cuando se nos presenta un conjunto de datos, es proceder con la tarea llamada detección de anomalías (anomaly detection, AD), o identificación de outliers o datos fuera de rango. Un outlier puede ser considerado como un dato atípico en un dataset. O bien un outlier es una observación en un dataset que parece ser inconsistente con el resto del conjunto. Johnson 1992.

2.2.2. Reducción de dimensionalidad

La reducción de dimensionalidad es otra de las grandes aplicaciones de los algoritmos no supervisados. El objetivo de este tipo de algoritmo es convertir un dataset de una cierta dimensión, digamos 1000 características, en otro con una menor dimensión, por ejemplo unas 300.

Las razones por las que nos interesa reducir la dimensionalidad son varias:

- Porque interesa **identificar y eliminar las variables irrelevantes.**
- Porque **no siempre el mejor modelo es el que más variables** tiene en cuenta.

AD Supervisada	**AD Semi-supervisada** (detección de novedades, *Novelty Detection*)	**AD No supervisada** (detección de outliers, *Outlier Detection*)
• Las etiquetas están disponibles, tanto para casos normales como para casos anómalos. • En cierto modo, similar a minería de clases poco comunes o clasificación no balanceada.	• Durante el entrenamiento, solo tenemos datos normales. • El algoritmo aprende únicamente usando los datos normales.	• No hay etiquetas y el conjunto de entrenamiento tiene datos normales y datos anómalos. • Asume que los datos anómalos son poco frecuentes. • Algunos ejemplos típicos de detección de anomalías pueden ser, cuando se quiere detectar intrusos en tráfico de red o bien detectar acciones fraudulentas en transacciones con tarjetas de crédito.

Figura 8: Tipos de entornos en los que se produce la detección de anomalías

- Porque **se mejora el rendimiento computacional**, lo que se traduce en un ahorro en coste y tiempo.
- Porque **se reduce la complejidad**, lo que lleva a facilitar la comprensión del modelo y sus resultados.
- Puede eliminar ruido presente en el dataset original.
- Los resultados pueden ser más fácilmente interpretables.

Uno de los modelos más conocidos y que requieren menor costo computacional es Principal Component Analysis (PCA). Es una técnica para reducir la dimensionalidad de los datos, creando una proyección lineal. Es decir, encontramos características nuevas para representar los datos que son una combinación lineal de los datos originales (lo cual es equivalente a rotar los datos). De esta forma, podemos pensar en el PCA como una proyección de nuestros datos en un nuevo espacio de características. La forma en que el PCA encuentra estas nuevas direcciones es buscando direcciones de máxima varianza. De lo contrario si lo que estamos buscando es una mejor visualización de los datos y además las características no son lineales, sería recomendable usar T-SNE.

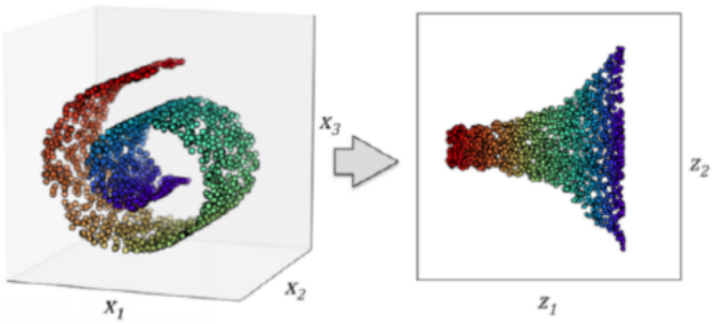


Figura 9: Reducción de la dimensionalidad de 3D a 2D.

En la Fig.9 [8] anterior se muestra un ejemplo de cómo la reducción de dimensionalidad facilita la visualización de un dataset de alta dimensionalidad en una proyección de 1, 2 o 3 dimensiones. Enumeramos en la Fig.10 las ventajas y desventajas de los algoritmos disponibles para esta tarea.

	Ventajas	Desventajas
PCA	• Simple de implementar. • Es uno de los algoritmos más rápidos de reducción de dimensionalidad.	• No puede detectar características no lineales. • Los datos necesitan ser normalizados.
Isomap	• Puede detectar características no lineales.	• Lento en grandes cantidades de datos.
T-SNE	• Puede detectar características no lineales. • Recomendable cuando se quiere obtener una mejor visualización de un dataset con alta dimensionalidad.	• Computacionalmente costoto y lento.

Figura 10: Pros y contras de algoritmos de reducción de dimensionalidad.

2.2.3. Clustering

El clustering es una técnica que conceptualmente es simple de comprender, consiste agrupar objetos con características similares. Por lo tanto, obtenemos diferentes grupos llamados clusters, donde en cada uno de ellos están contenidos los datos que son más similares entre ellos que con los que pertenecen a otros clusters, obteniendo de esta forma una útil subdivisión del dataset. El agrupamiento (clustering) es la tarea de reunir los ejemplos formando grupos de ejemplos similares de acuerdo a alguna medida de semejanza prefijada o disimilitud (distancia), como la distancia Euclídea.

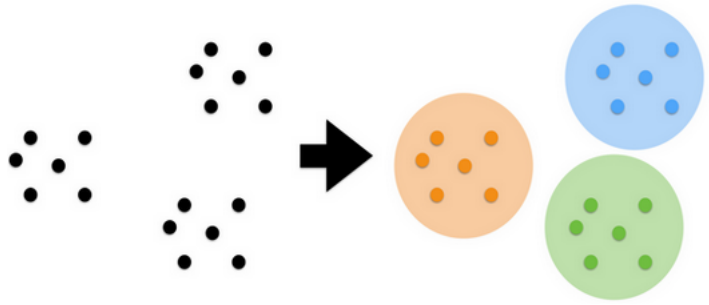


Figura 11: Ejemplo de clustering.

Estos algoritmos de aprendizaje no supervisados tienen una gama increíblemente amplia de aplicaciones y son muy útiles para resolver problemas del mundo real como la detección de anomalías, la recomendación de sistemas, la agrupación de documentos o la búsqueda de clientes con intereses comunes basados en sus compras.

El algoritmo más simple de clustering es K-means, el cual funciona para agrupar datos que se distribuyen en formas esféricas, si se usa la distancia euclídea. y a su vez hay que proporcionar la cantidad k de grupos en los cuales queremos distribuir el dataset, por ello se debe tener un conocimiento previo de cuantos clúster se espera tener. Otras alternativas pueden ser, realizar clustering jerárquico o clustering basados en densidades. En clustering jerárquico, vemos como resultado un dendograma, es decir un diagrama de árbol. A partir de esto, se decide un umbral de profundidad, donde se corta el árbol y de esta forma

se obtiene un agrupamientos, por lo tanto a diferencia con K-means, no necesitamos tener información para poder decidir la cantidad de grupos. En la Fig.12 podemos observar como quedan evidenciados a través del dendograma los diferentes clusters.

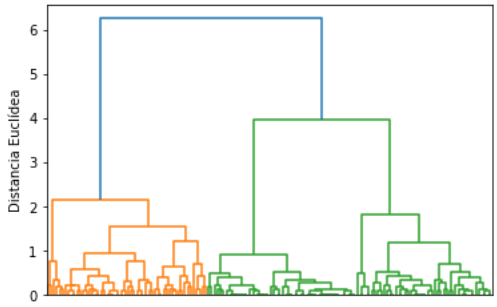


Figura 12: Dendograma generado por clustering jerárquico.

Además, podemos distinguir dos formas principales de clustering jerárquico: **divisivo** y **aglomerativo.**

- En el clustering **aglomerativo**, empezamos con un único patrón por clúster y vamos agrupando clusters (uniendo aquellos que están más cercanos), siguiendo una estrategia bottom-up para construir el dendrograma.
- En el clustering **divisivo**, sin embargo, empezamos incluyendo todos los puntos en un único grupo y luego vamos dividiendo ese grupo en subgrupos más pequeños, siguiendo una estrategia top-down.

En cambio el algoritmo DBSCAN (Density-based Spatial Clustering of Applications with Noise), divide el dataset buscando las regiones densas de puntos, como podemos observar claramente en Fig. 13. Con esta técnica, tampoco especificamos el número de parámetros a priori, sino que se establecen hiper parámetros adicionales, como lo son la cantidad mínima de puntos y un radio ϵ, para lograr un óptimo funcionamiento. Se basa en un número de puntos con un radio especificado ϵ y hay una etiqueta especial asignada a cada punto de datos. El proceso de asignación de esta etiqueta es el siguiente:

- Es un número especificado (MinPts) de puntos vecinos. Se asignará un punto central si existe este número de puntos de los MinPts que caen en el radio ϵ.
- Un punto fronterizo caerá en el radio de ϵ de un punto central, pero tendrá menos vecinos que el número de MinPts.
- Todos los demás puntos serán puntos de ruido.

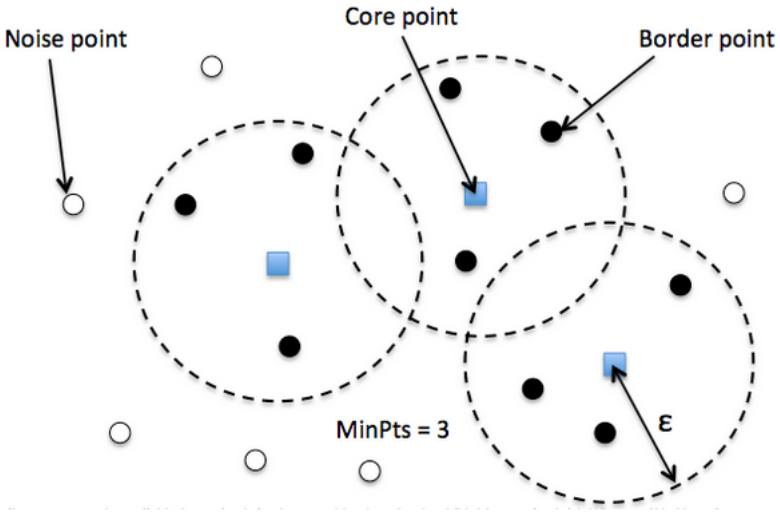


Figura 13: DBSCAN.

Pros y contras de los algoritmos de clustering:

	Ventajas	Desventajas
K-Means	• Fácil de implementar. • Rápido.	• Hay que conocer el número de grupos y asumir que los datos están normalizados. • Utiliza la distancia euclídea, por lo que debemos estar seguros de que las variables estén en la misma escala. • Sensible al ruido.
Agglomerative Clustering	• No es necesario indicar el número de grupos a priori. • No es sensible a la elección de la métrica de distancia.	• No es muy eficiente.
Mini Batch K-Means	• Puede agrupar conjuntos de datos masivos. • Reduce el tiempo de cómputo gracias a los mini-batches.	• La calidad de los resultados podría verse reducida respecto a K-means.
Mean Shift	• No es necesario indicar el número de grupos a priori.	• No es escalable para muchos datos.
DBSCAN	• No hay que especificar el número de clusters a priori. • Puede detectar grupos con formas irregulares. • Puede detectar outliers. • Robusto al ruido.	• Sensible a datos con alta dimensión. • No funciona bien cuando los clusters son de densidad variable.

Figura 14: Pros y contras de los algoritmos de clustering.

3. Redes neuronales

Los algoritmos de aprendizaje desarrollados durante siglos no pudieron abordar la complejidad de ciertos problemas reales. Es por esta razón que se buscó una comparación con el cerebro humano que se considera como la computadora más sostificada para resolver problemas extremadamente complejos. En esta comparación podemos ver que el sistema nervioso está constituido por unidades relativamente simples, las **neuronas**, por lo que copiar su comportamiento y funcionalidad puede ser la solución.

3.1. Relación con la biología

Como se explicó previamente vamos a relacionar el concepto de redes neuronales con las neuronas biológicas.

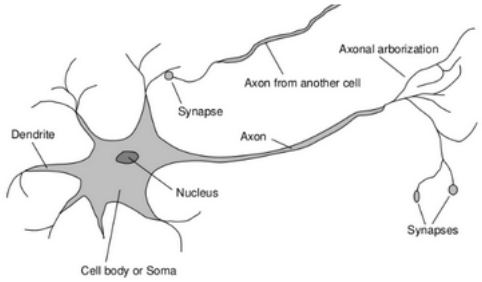


Figura 15: Neurona biológica.

- Una neurona no hace nada hasta que la influencia de sus entradas alcanza un determinado nivel.
- La neurona produce una salida en la forma de pulso que parte del núcleo y termina en sus ramificaciones.
- La salida causa la excitación o inhibición de otras neuronas.

En comparación con una neurona artificial llamada perceptón tenemos que:

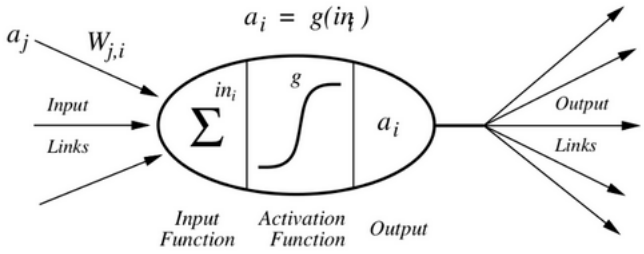


Figura 16: Neurona artificial perceptón.

- Las neuronas artificiales son nodos conectados a otros nodos mediante enlaces. A cada enlace se le asocia un peso.
- El peso determina la naturaleza (excitatoria + o inhibitoria -) y la fuerza (valor absoluto) de la influencia entre nodos.

- Si la influencia de todos los los enlaces de entrada es suficientemente alta, el nodo se activa.
- Cada nodo i tiene varias conexiones de entrada y de salida, cada una con sus pesos.
- La salida es una función de la suma ponderada de las entradas. (Devuelve un resultado).

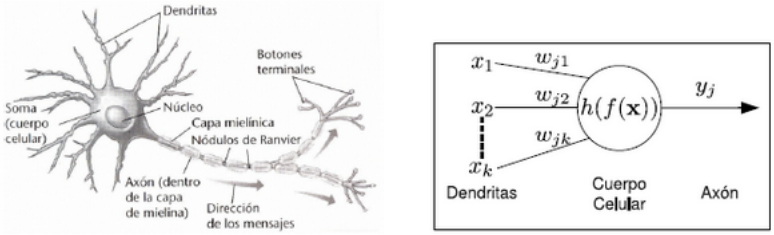


Figura 17: Comparación de neurona y perceptón.

El objetivo del aprendizaje profundo no es imitar cómo funcionan nuestros cerebros, sino tomar las piezas que entendemos y permitirnos trazar paralelos similares en nuestro propio trabajo.

3.2. Redes neuronales artificiales

Como vimos anteriormente este concepto surge de la comparación e inspiración de las neuronas biológicas y su funcionamiento, es por esto que una red neuronal artificial se define como un sistema de computación compuesto por un gran número de elementos simples, elementos de procesos muy interconectados, los cuales procesan información por medio de su estado dinámico como respuesta a entradas externas, es decir:

- Una Red Neuronal Artificial (RNA ó ANN) es un grafo de nodos (o neuronas) conectados por enlaces.
- Las neuronas se organizan por capas, de manera que las salidas de las neuronas de una capa sirven como entradas para las neuronas de la siguiente capa.
- También conocidas como MultiLayer Perceptron (MLP).

En la Fig.18 , podemos observar como los datos ingresan a la red por medio de una capa de entrada, luego son procesados por una o múltiples capas ocultas, y una vez que termina este proceso salen por la capa de salida. Las redes neuronales de igual manera que los algoritmos de Aprendizaje Automático se entrenan con un dataset de entrada proporcionando etiquetas correspondientes como salida. Sin embargo es importante mencionar que el foco

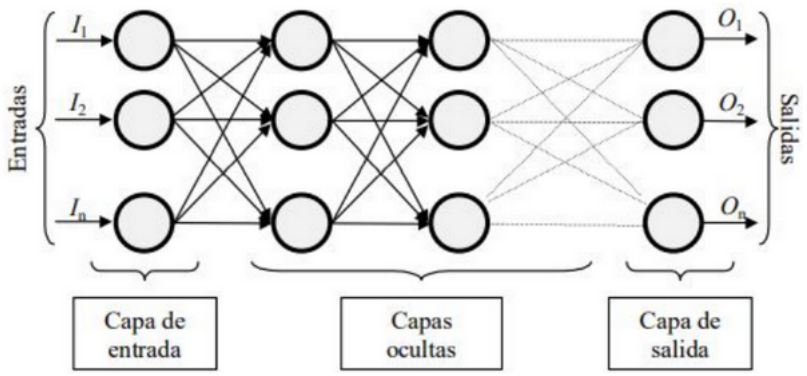


Figura 18: Ejemplo de una red neuronal totalmente conectada.

central en este caso se encuentra en la optimización de los pesos internos entre las capas ocultas, o también conocido como adaptación de los pesos. Por lo que el aprendizaje de una red neuronal se produce cuando se modifican los pesos de las conexiones a medida que varía la información de entrada. Un cambio en un peso puede producir, modificaciones, destrucción y hasta la creación de conexiones entre las neuronas. Si nos enfocamos en una neurona en particular, se pueden apreciar tres funciones que permiten el óptimo funcionamiento de las mismas en una red. Estas son:

- Función de entrada (*Input function*), toma la forma de una entrada global, considerando todos los valores de entrada a la red como uno solo. Por lo tanto tenemos un vector entrada que nos permite calcular la llamada función de entrada, y se describe de la siguiente manera:

$$input = (I_1 x w_1) * (I_2 x w_2) * ... * (I_n x w_n) \tag{1}$$

donde:

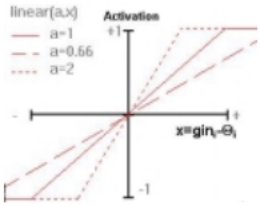
- * representa al operador apropiado. Puede ser, máximo, sumatoria, productoria, etc.
- **n** representa al número de entradas a la neurona Ni y wi al peso.

- Función de activación (activation function), permite calcular el estado de actividad de una neurona. Transformando la entrada global en un valor de activación, suele utilizarse un rango que comprende desde (0 a 1) o bien (-1 a 1). Debido a que una neurona puede estar completamente inactiva (0 o -1) o activa (1). La función de activación, es una función de la entrada global (gini) menos un umbral(i). Las más comunes son, función lineal, función sigmoid, función tangente hiperbólica.

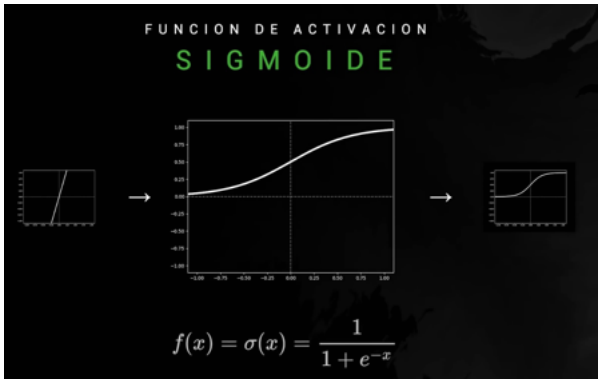
- Función lineal:

$$f(x) = \begin{cases} -1 & x \leq -1/a \\ a * x & -1/a < x < 1/a \\ 1 & x \geq 1/a \end{cases}$$

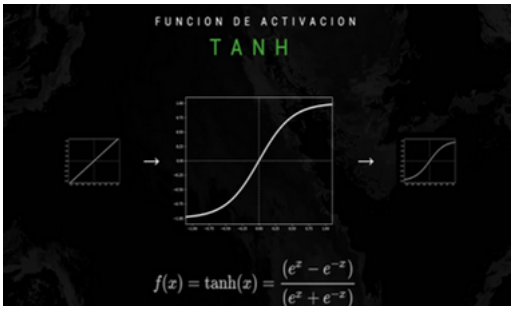
con $x = gin_i - \Theta_i$, y $a > 0$.



- Función sigmoidea:

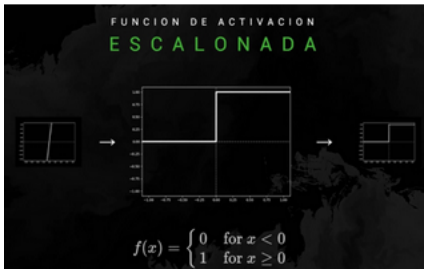


- Función tangente hiperbólica:

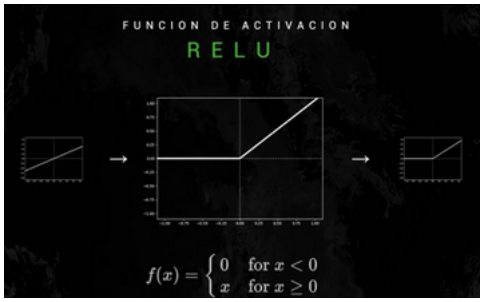


$$f(x) = \frac{e^{(gx)} - e^{(-gx)}}{e^{(gx)} + e^{(-gx)}}, conx = gin_i - \Theta_i$$

- Función escalonada:



- Función Relu:



- Función de salida (*output function*), el valor final de la función de salida es el resultado que produce la neurona i, el cual será transferido a las neuronas vinculadas. En el apartado anterior se especificó un umbral, el cual la función de activación debe superar, en caso de no hacerlo ninguna salida pasa a las neuronas posteriores. Los valores de salida normalmente se corresponden con los valores aceptados por las entradas de las neuronas, entonces suelen estar comprendidos entre [0,1] o [-1, 1]. En el caso de las funciones de salida también pueden ser binarios, 0,1 o -1,1. Las funciones de salida más comunes son ninguna (el valor de entrada pasa a la salida, se la conoce como identidad) o binaria (1 si el umbral es superado por el contrario adopta el valor 0).

3.3. Arquitectura de redes feedforward

Si bien hay muchas, muchas arquitecturas ANN diferentes, la arquitectura más común es la red hacia delante o feedforward, como se presenta en la Fig. 19.

En este tipo de arquitectura, solo se permite una conexión entre los nodos de los nodos en la capa i a los nodos en la capa i + 1 (de ahí el término feedforward). No hay conexiones hacia atrás o entre capas permitidas. Cuando las redes de retroalimentación incluyen conexiones de retroalimentación (conexiones de salida que retroalimentan las entradas) se denominan redes neuronales recurrentes. Una red neuronal feedforward es un tipo de red

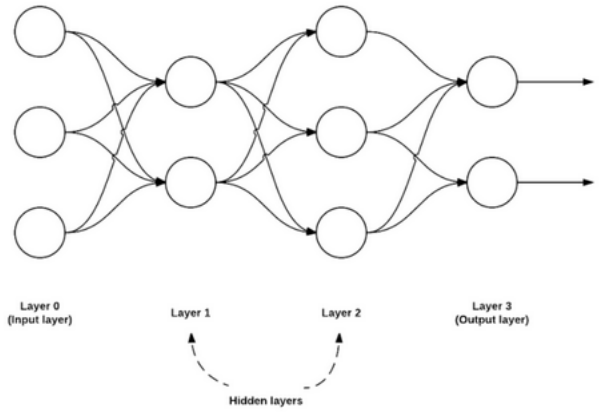


Figura 19: Un ejemplo de una red neuronal feedforward.

neuronal en la que las conexiones de las unidades no viajan en un bucle, sino en una única ruta dirigida. Esto difiere de una red neuronal recurrente , donde la información puede moverse hacia adelante y hacia atrás en todo el sistema. Una red neuronal feedforward es quizás el tipo más común de red neuronal, ya que es una de las más fáciles de entender y configurar. Estos tipos de redes neuronales se utilizan en minería de datos y otras áreas de estudio donde se requiere un comportamiento predictivo. [9]

Para describir mejor este tipo de red, normalmente usamos una secuencia de enteros para depositar rápida y concisamente el número de nodos en cada capa. Por ejemplo, la red en la Figura anterior es una red de alimentación directa 3-2-3-2:

- La capa 0 contiene 3 entradas, nuestros valores xi. Estos podrían ser intensidades de píxeles sin procesar de una imagen o un vector de características extraído de la imagen.
- Las capas 1 y 2 son capas ocultas que contienen 2 y 3 nodos, respectivamente.
- La capa 3 es la capa de salida o la capa visible: allí es donde obtenemos la clasificación de salida general de nuestra red. La capa de salida generalmente tiene tantos nodos como etiquetas de clase; un nodo para cada salida potencial.

3.4. Redes multicapas

Las limitaciones de las redes de una sola capa hicieron que se plantease la necesidad de implementar redes en las que se aumentase el número de capas, es decir, introducir capas intermediarias o capas ocultas entre la capa de entrada y la capa de salida de manera que se pudiese implementar cualquier función con el grado de precisión deseado, es decir, que las redes multicapa fuesen aproximadores universales.

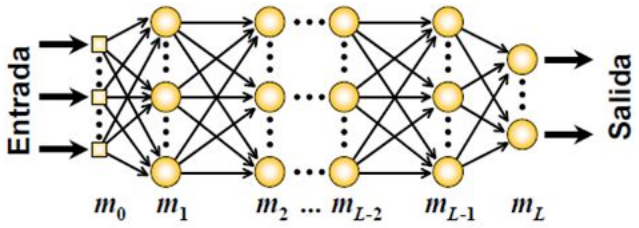


Figura 20: Se muestra una red multicapa.

Al tener estas redes una topología más complicada, también se complicó la forma para encontrar los pesos correctos, ya que el proceso de aprendizaje es el que decide qué características de los patrones de entrada son representadas por la capa oculta de neuronas.

El Perceptrón multicapa es una red de alimentación hacia adelante (feedforward) compuesta por una capa de unidades de entrada (sensores), otra capa de unidades de salida y un número determinado de capas intermedias de unidades de proceso, también llamadas capas ocultas porque no tienen conexiones con el exterior. Cada sensor de entrada está conectado con las unidades de la segunda capa, y cada unidad de proceso de la segunda capa está conectada con las unidades de la primera capa y con las unidades de la tercera capa, así sucesivamente. Las unidades de salida están conectadas solamente con las unidades de la última capa oculta.

Con esta red se pretende establecer una correspondencia entre un conjunto de entrada y un conjunto de salidas deseadas.

Supongamos W como la matriz de pesos y el vector b como el vector sesgo o bias. Consideremos:

$$Z(x) = W_x + b = \sum_{i=1}^{n} w_i x_i + b \tag{2}$$

Además cabe mencionar que la multiplicación punto a punto entre dos matrices de igual dimensión es lo que se conoce como el producto Hadamard.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

Por último definimos la salida de nuestro modelo como:

$$\hat{y} = \sigma(\sum_{i=1}^{n} w_i x_i + b) \tag{4}$$

3.5. Función de costo

La función de costo o pérdida, es una función que busca determinar el error entre el valor estimado y el valor real, con el fin de optimizar los parámetros de la red neuronal. Esencialmente modela la diferencia entre la predicción de un modelo y la salida real. Idealmente si estos valores están alejados, el valor de pérdida o error deben ser mayores. Del mismo modo, si estos dos valores están cercanos el valor del error debería ser bajo. Una posible función de costo a utilizar podría ser:

$$J(\Theta) = \frac{||\hat{y} - y||^2}{2} \tag{5}$$

siendo:

- $\hat{y}$ - el valor predicho.
- y - el valor obtenido.

Esta función se conoce como error al cuadrado. Simplemente se toma la diferencia entre la salida real y la salida predicha, elevamos al cuadrado ese valor y lo dividimos entre 2. Una de las principales razones para utilizar esta función a diferencia de otras como el error absoluto (J() = —ŷ-y—) es que el error cuadrático es diferenciable en su totalidad. Otros beneficios de la cuadratura incluyen:

- La cuadratura siempre dará un valor positivo, por lo que la suma no será cero.
- Hablamos de suma aquí porque posteriormente se sumarán los valores de error para cada dato en el dataset de entrenamiento y luego se hará un promedio para encontrar la pérdida para todo el lote de ejemplos de entrenamiento.
- La cuadratura enfatiza las diferencias más grandes.

Luego, una posible función de error a utilizar es la siguiente:

$$J(\theta) = \frac{1}{2n} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 \tag{6}$$

A esta función se la conoce como el error cuadrático medio. Se calcula el error al cuadrado para cada feature dentro del dataset y luego encontramos el promedio de estos valores, esto representa el error general del modelo en nuestro conjunto de entrenamiento. Consideremos un ejemplo de una sola feature con solo dos características. Dos características significa que tenemos dos valores de peso correspondientes y un valor de sesgo. En total, tenemos tres parámetros para nuestro modelo.

$$\hat{y} = w_1 x_1 + w_2 x_2 + b \tag{7}$$

$$J(\theta) = \frac{1}{2n}\sum_{i=1}^{m}(w_1 x_1^{(i)} + w_2 x_2^{(i)} + b - y^{(i)})^2 \tag{8}$$

Se busca entrar valores para nuestros pesos y sesgo que minimice el valor de nuestra función de coste. Dado que esta es una ecuación de múltiples variables, eso significa que tendríamos que tratar con derivadas parciales de esta función correspondiente.

$$\frac{\delta J}{\delta w_1}\frac{\delta J}{\delta w_2}\frac{\delta J}{\delta b} \tag{9}$$

a cada una de nuestras variables w_1, w_2 y b. Esto puede parecer lo suficientemente simple porque solo tenemos tres variables diferentes, sin embargo tenemos tantos pesos como características i.e. w_n pesos. Hacer una optimización multivariante con tantas variables es computacionalmente ineficiente y no es manejable. Por lo que se recurre a alternativas y aproximaciones.

3.6. Descenso de gradiente

Dado un conjunto de entrenamiento, queremos ajustar los pesos de las conexiones para que el error cometido en dicho conjunto sea mínimo. Para esto se utiliza el algoritmo del gradiente, para minimizar la función de costo. Es la capacidad de de aprendizaje que otorga el algoritmo de descenso de gradiente lo que hace que el aprendizaje automático y los modelos de aprendizaje profundo aprendan. Como se vio previamente, la forma más rápida de minimizar la función de coste sería encontrar derivadas de segundo orden de la misma con respecto a los parámetros del modelo. Sin embargo, esto es computacionalmente costoso.

La intuición básica detrás del descenso del gradiente puede ilustrarse mediante un escenario hipotético: una persona está atrapada en las montañas y está tratando de bajar (es decir, tratando de encontrar los mínimos). Hay mucha niebla de tal manera que la visibilidad es extremadamente baja. Por lo tanto, el camino hacia abajo de la montaña no es visible por lo que deberá de utilizar la información local para encontrar mínimos. Puede utilizar el método del descenso en gradiente, que consiste en mirar la inclinación de la colina en su posición actual, luego proceder en la dirección con el descenso más empinado. Usando este método, eventualmente encontrarán su camino. Sin embargo, supongo también que la pendiente de la colina no es inmediatamente obvia con una simple observación, sino que requiere un instrumento sofisticado para medir, que la persona tiene en ese momento. Se necesita bastante tiempo para medir la inclinación de la colina con dicho instrumento, por lo que , deben minimizar el uso del mismo si quieren bajar la montaña antes del atardecer. La dificultad es elegir la frecuencia con la que deben medir dicha inclinación para no desviarse. En esta analogía:

- La persona representa nuestro **algoritmo de aprendizaje**, y el camino que baja por la montaña representa la **secuencia de actualizaciones de parámetros** que nuestro modelo eventualmente explorara.
- La inclinación de la colina representa la **pendiente de la superficie de error en ese punto.**
- El instrumento utilizado para medir la inclinación es la diferenciación (la pendiente de la superficie de error se puede calcular tomando la derivada de la función de error al cuadrado en ese punto). Esta es la aproximación que hacemos cuando aplicamos el descenso de gradiente. Realmente no sabemos el punto mínimo, pero si sabemos la dirección que nos llevará a los mínimos (locales o globales) y damos un paso en esa dirección.
- La dirección en que la persona elige viajar se alinea con el gradiente de la superficie de error en ese punto.
- La cantidad de tiempo que viaja antes de tomar otra medida es la **velocidad de aprendizaje del algoritmo.**

Entonces el descenso del gradiente mide el gradiente local de la función de costo para un conjunto dado de parámetros () y da pasos en la dirección del gradiente descendente. Como ilustra la Figura 21 , una vez que el gradiente es cero, hemos alcanzado un mínimo.

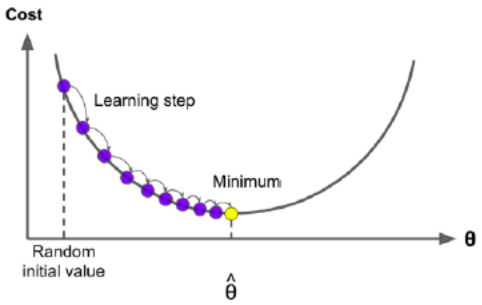


Figura 21: Representación gráfica del descenso de gradiente.

Como puede observarse en la figura 22 es importante ajustar apropiadamente el valor de la tasa de aprendizaje (learning rate). Si es demasiado pequeña, entonces el algoritmo tomará muchas iteraciones (steps) para encontrar el mínimo. Por otro lado, si es muy alta, es posible que se supere el mínimo y se termine más lejos que cuando comenzó, además esto puede llevar a que termine más lejos que cuando comenzó.

Luego, para actualizar la matriz de pesos y de sesgo se utilizarán las siguientes ecuaciones.

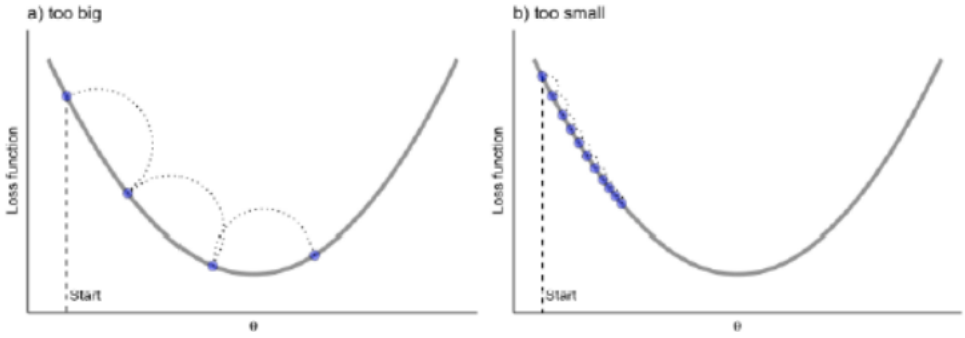


Figura 22: Tamaño de pasos del gradiente.

$$W' = W - \alpha \frac{\delta J}{\delta W} \tag{10}$$

$$b' = b - \alpha \frac{\delta J}{\delta b} \tag{11}$$

3.7. Backpropagation

En el punto anterior vimos como una red neuronal puede obtener sus pesos y sesgos utilizando el algoritmo de descenso de gradiente. Sin embargo, quedó un vacío en la explicación, el cual es como computar el gradiente de la función de costo. En este capítulo se procederá a explicar un algoritmo para el cómputo de dichos gradientes, conocido como retropropagación (backpropagation).

Ya sabemos cómo fluyen las activaciones en la dirección hacia adelante. Tomamos las features de entrada, las transformamos linealmente, aplicamos la activación sigmoidea en el valor resultante y finalmente tenemos nuestra activación que luego usamos para hacer una predicción [10].

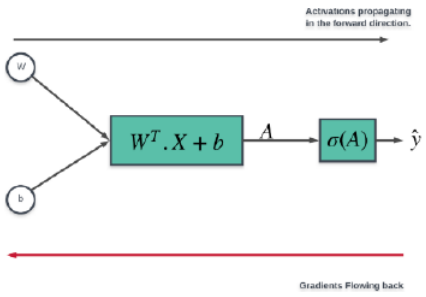


Figura 23: Las activaciones se propagan hacia adelante, pero los gradientes fluyen hacia atrás.

Como ejemplo, supongamos que queremos encontrar la derivada parcial de la variable y con respecto a x de la figura 24. No podemos obtenerlo de forma directa ya que hay otras tres variables involucradas en el gráfico computacional. Entonces, hacemos este proceso de forma iterativa, yendo hacia atrás en el gráfico de cálculo Primero, se obtiene la derivada parcial de la salida y con respecto a la variable C. Luego se utiliza la regla de la cadena de cálculo y determinamos la derivada parcial con respecto a la variable B y así sucesivamente hasta que obtengamos la derivada parcial que buscamos.

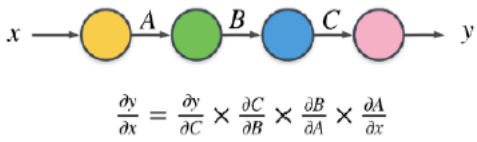


Figura 24: Representación gráfico simple.

Utilizando la función de costo definida en la ecuación y reescribiendo la misma en su forma vectorial.

$$J(\theta) = \frac{1}{2}||\hat{Y} - Y||^2 \tag{12}$$

La derivada parcial de la función de costo con respecto a la activación de nuestro modelo es:

$$\frac{\delta J}{\delta \hat{Y}} = \frac{1}{2}\frac{\delta J}{\delta \hat{Y}}||\hat{Y} - Y||^2 = \frac{1}{2}2\hat{Y} - Y\frac{\delta}{\delta \hat{Y}}(\hat{Y} - Y) = (\hat{Y} - Y)\frac{\delta}{\delta \hat{Y}}||\hat{Y} - Y|| = (\hat{Y} - Y) \tag{13}$$

Avancemos un paso hacia atrás y calculemos la próxima derivada parcial. Esto nos llevará un paso más cerca de los gradientes reales que buscamos calcular. Este es el punto donde aplicamos la regla de la cadena que fue mencionada previamente. Entonces, para calcular la derivada parcial de la función de costo con respecto a la salida transformada linealmente, es decir, la salida de nuestro modelo antes de aplicar la activación sigmoidea.

$$\frac{\delta J}{\delta D} = \frac{\delta J}{\delta \hat{Y}}\frac{\delta \hat{Y}}{\delta A} \tag{14}$$

La primera parte de esta ecuación es el valor que habíamos calculado en la ecuación 13. Lo esencial para calcular aquí es la derivada parcial de la predicción de nuestro modelo con respecto a la salida transformada linealmente. Observamos la ecuación para la predicción de nuestro modelo, la función activación sigmoidea.

$$\hat{Y} = \sigma(A) = \frac{1}{1 + e^{-A}} \tag{15}$$

Derivada de la salida final de nuestro modelo, i.e. significa la derivada parcial de la función sigmoide con respecto a su entrada.

$$\frac{\delta}{\delta A}\sigma(A) = \frac{\delta}{\delta A}\frac{1}{1+e^{-A}} = \frac{\delta}{\delta A}(1+e^{-A})^{-1} \tag{16}$$

$$-1(1+e^{-A})^{-2}\frac{\delta}{\delta A}(1+e^{-A}) = -1(1+e^{-A})^{-2}(-e-A) = \frac{e^{-A}}{(1+e^{-A})^2} \tag{17}$$

Continuando, podemos simplificar aún más esta ecuación.

$$\sigma(A) = \frac{1}{1+e^{-A}} \tag{18}$$

$$e^{-A} = \frac{1}{\sigma(A)} - 1 = \frac{1-\sigma(A)}{\sigma(A)} \tag{19}$$

Sustituyendo este valor en la ecuación 14 obtenemos:

$$\frac{\delta J}{\delta A} = \frac{\delta J}{\delta \hat{Y}}\frac{e^{-A}}{(1+e^{-A})^2} \tag{20}$$

$$\frac{\delta J}{\delta A} = \frac{\delta J}{\delta \hat{Y}}\frac{1-\sigma(A)}{\sigma(A)}\sigma(A)\sigma(A) \tag{21}$$

$$\frac{\delta J}{\delta A} = \frac{\delta J}{\delta \hat{Y}}\sigma(A)(1-\sigma(A)) \tag{22}$$

Necesitamos la derivada parcial de la función de coste correspondiente a cada uno de los pesos. Al estar recurriendo a la vectorización, es posible obtener todo esto de una vez. Es por eso que hemos estado utilizando la notación mayúscula W en lugar de $w_1, w_2, \ldots, w_n$.

$$\frac{\delta J}{\delta W} = \frac{\delta J}{\delta A}\frac{\delta A}{\delta W} \tag{23}$$

$$\frac{\delta J}{\delta b} = \frac{\delta J}{\delta A}\frac{\delta A}{\delta b} \tag{24}$$

La derivación de los pesos partiendo de la ecuación 23:

$$\frac{\delta J}{\delta W} = \frac{\delta J}{\delta A}\frac{\delta}{\delta W}(W^T X + b) = \frac{\delta J}{\delta A}X \tag{25}$$

Y de la ecuación 24:

$$\frac{\delta J}{\delta b} = \frac{\delta J}{\delta A}\frac{\delta}{\delta b}(W^T X + b) = \frac{\delta J}{\delta A}1 = \frac{\delta J}{\delta A} \tag{26}$$

Finalmente, se demostró desde un punto de vista matemático el concepto de backpropagation, como se realiza la actualización de los pesos y sesgos utilizando descenso por gradiente.

3.8. Versiones del descenso del gradiente

El descenso de gradiente en casos donde se trabaja con datasets muy grandes puede llegar a ser excepcionalmente lento debido a que en cada iteración requiere calcular una predicción por cada punto de entrenamiento en nuestros datos antes que actualicemos nuestra matriz de pesos. Para solucionar esto se introdujeron modificaciones a este algoritmo, derivando así en distintas versiones [11]:

- Descenso del gradiente en lotes (o batch): todos los datos disponibles se introducen de una vez. Esto supondrá problemas de estancamiento, ya que el gradiente se calculará usando siempre todas las muestras, y llegará un momento en que las variaciones serán mínimas. Como regla general siempre conviene que la entrada a una red neuronal tenga algo de aleatoriedad.

- Descenso del gradiente estocástico: se introduce una única muestra aleatoria en cada iteración. El gradiente se calculará para esa muestra concreta, lo que supone la introducción de la deseada aleatoriedad, dificultando así el estancamiento. El problema de esta versión es su lentitud, ya que necesita de muchas más iteraciones, y además no aprovecha los recursos disponibles.

- Descenso del gradiente estocástico en mini-lotes: en lugar de alimentar la red con una única muestra, se introducen N muestras en cada iteración; conservando las ventajas de la segunda versión y consiguiendo además que el entrenamiento sea más rápido debido a la paralelización de las operaciones. Se elige un valor de N que nos aporte un buen balance entre aleatoriedad y tiempo de entrenamiento (de tal forma que este no sea demasiado grande para la memoria disponible).

3.9. Descenso de gradiente estocástico (SGD)

Como se vio en el punto previo, el descenso de gradiente puede tener ciertos inconvenientes al tratar con datasets de gran tamaño. Para solucionar estos problemas, se utiliza la variante del gradiente estocástico trabajando con mini-lotes. Este es una simple modificación del algoritmo de descenso de gradiente estándar que computa el gradiente y actualiza la matriz de pesos W en pequeños lotes o batches de datos de entrenamiento, en vez del dataset entero. Mientras esta modificación nos lleva a actualizaciones más "ruidosas", también nos permite tomar más pasos a lo largo del gradiente, llevando en última instancia a una convergencia más rápida sin afectar negativamente al coste y precisión del modelo. Desde una perspectiva de implementación, también se trata de aleatorizar las muestras de entrenamiento antes de aplicar SGD ya que el algoritmo es sensible a lotes. En una implementación "purista" de SGD, el tamaño de su mini lote seria de uno, lo que implica que se muestrearía aleatoriamente un punto de datos del conjunto de entrenamiento, se calcularía el gradiente y se actualizarían nuestros parámetros. Sin embargo, a menudo se

utilizan mini lotes de un tamaño mayor. Los tamaños de lote típicos incluyen 32, 63, 128 y 256 puntos de datos. A continuación se listan las justificaciones de esta decisión:

- Ayuda a reducir la variación en la actualización de parámetros, lo que conduce a una convergencia más estable.
- Las potencias de dos a menudo son deseables para los tamaños de lote, ya que permiten que las bibliotecas de optimización de álgebra lineal interna sean más eficientes.

En general, el tamaño del mini lote no es un hiperparámetro por el uno que debería preocuparse demasiado. Si se está usando una GPU para entrenar su red neuronal, uno mismo determina cuantos ejemplos de entrenamiento entraran en su GPU y luego usa la potencia más cercana de dos, ya que el tamaño de lote se ajustará en la GPU. Para el entrenamiento con CPU, normalmente se utiliza uno de los tamaños enumerados anteriormente para asegurarse de obtener los beneficios de las bibliotecas de optimización de álgebra lineal.

3.10. Sobreajuste y bajo-ajuste

Uno de los problemas más frecuentes cuando entrenamos redes neuronales, sean convolucionales o de cualquier otra familia, se conoce como overfitting (sobreajuste), como así también existe su antónimo underfitting (bajo ajuste).

Como nosotros sabemos nuestro objetivo en el aprendizaje supervisado es aproximar de la mejor forma posible la función subyacente que "mapea" entradas a salidas, es decir, que el modelo sea capaz de poder generalizar. La generalización se refiere a qué tan bien los conceptos aprendidos por un modelo de aprendizaje automático se aplican a ejemplos específicos que el modelo no vio cuando estaba aprendiendo. El objetivo de un buen modelo de aprendizaje automático es generalizar bien los datos de entrenamiento a cualquier dato del dominio del problema. Esto nos permite hacer predicciones en el futuro sobre datos que el modelo nunca ha visto.

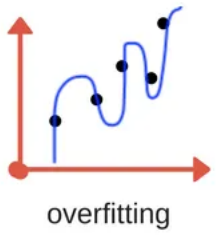


Figura 25: Overfitting.

En el overfitting, representado en la figura 25, nuestro modelo se ajustará/aprenderá los casos particulares que le enseñamos y será incapaz de reconocer nuevos datos de entrada.

Por lo tanto nuestro algoritmo estará considerando como válidos sólo los datos idénticos a los de nuestro conjunto de entrenamiento y siendo incapaz de distinguir entradas buenas como fiables si se salen un poco de los rangos ya preestablecidos. De esta forma, el modelo lo que hace es memorizar, no aprender.

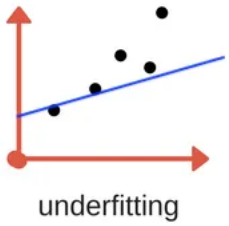


Figura 26: Underfitting.

En la figura 26 se puede observar el caso en el que tenemos underfitting lo que ocurre cuando nuestro modelo es muy simplista, insuficiente para capturar los matices, particularidades y complejidades en los datos. Esto nos lleva a que el modelo no predice los objetivos en los conjuntos de datos de entrenamiento con mucha precisión.

3.11. Regularización

Para disminuir los efectos del sobreajuste se utiliza la regularización que después de la tasa de aprendizaje, es el parámetro más importante de su modelo que puede ajustar. Es una forma de penalizar la complejidad en los modelos, típicamente a nivel de parámetros.

Existen varios tipos de técnicas de regularización, como la regularización **L1**, la regularización **L2** (comúnmente llamada pérdida de peso) y **Elastic Net**, que se utilizan al actualizar la función de pérdida en sí, agregando un parámetro adicional para restringir la capacidad del modelo.

La regularización nos ayuda a controlar la capacidad de nuestro modelo, asegurando que nuestros modelos sean mejores para hacer clasificaciones (correctas) en los puntos de datos en los que no fueron entrenados, lo que llamamos la capacidad de generalizar. Si no aplicamos la regularización, nuestros clasificadores pueden volverse demasiado complejos y ajustarse fácilmente a nuestros datos de entrenamiento, en cuyo caso perdemos la capacidad de generalizar, generando así *overfitting*.

3.12. Los cuatros componentes de una red neuronal

Hay cuatro componentes principales que se necesitan para armar una red neuronal y algoritmo de aprendizaje profundo: un conjunto de datos, un modelo/arquitectura, una función de pérdida y una optimización.

3.12.1. Conjuntos de datos

También llamado dataset, es el primer componente en el entrenamiento de una red neuronal: los datos en sí mismos junto con el problema que estamos tratando de resolver definen nuestros objetivos finales.

La combinación del conjunto de datos y el problema que está tratando de resolver influye en la elección de la función de pérdida, la arquitectura de red y el método de optimización utilizado para entrenar el modelo. Por lo tanto mientras tengamos una mayor cantidad de datos y que estos sean de una alta calidad, se tendrá mayor posibilidades de tener un buen modelo predictivo.

3.12.2. Función de costo

Dado nuestro conjunto de datos y objetivo objetivo, necesitamos definir una función de costo que se alinee con el problema que estamos tratando de resolver.

- Herramienta matemática utilizada por las neuronas para medir el éxito en la predicción. Se busca minimizar la misma.
- Toma los valores predichos y los compara con los originales para ver el rendimiento de la red.

3.12.3. Modelo de arquitectura

La arquitectura de una red es la conexión entre neuronas o capas, el tipo de neuronas presentes e incluso la forma en que son entrenadas.

Existen algunas configuraciones como ser:

- Feed forward network: Consiste en varias capas de neuronas totalmente conectadas (full connected), es decir que una capa se encuentra conectada con toda la capa siguiente.
- Redes neuronales convolucionales: Son principalmente usadas en procesamiento de imágenes. Las capas que las componen (no todas formadas por neuronas).

3.12.4. Método de optimización

El último componente es definir un método de optimización, que es una herramienta matemática utilizada para actualizar la red neuronal según el resultado obtenido en la función de pérdida.

4. Redes neuronales convolucionales

Podemos definir una red neuronal convolucional (CNN en Inglés) como una red neuronal que cambia una capa totalmente conectada (fully-connected) por una convolucional para al menos una de las capas de la red. En estas redes las neuronas corresponden a campos receptivos de una manera muy similar a las de las neuronas de la corteza visual primaria de un cerebro biológico, esto quiere decir que las primeras capas pueden detectar líneas, curvas y se van especializando hasta llegar a capas más profundas que reconocen formas complejas (como ser un rostro o una silueta). Este tipo de red es una variación de un perceptrón multicapa, sin embargo, debido a que su aplicación es realizada en matrices bidimensionales, son muy efectivas para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones. Permite el desarrollo de modelos supervisados y no supervisados. Cada capa en una CCN aplica un conjunto de filtros, usualmente cientos o miles de ellos y combinan los resultados, alimentando la entrada de la siguiente capa de la red. Durante el entrenamiento, una CNN automáticamente aprende los valores para esos filtros.

En el contexto de la clasificación de imágenes, una CNN puede aprender a:

- Detectar bordes a partir de datos de píxeles sin procesar en la primera capa.
- Usar esos bordes para detectar formas (i.e. blobs) en la segunda capa.
- Usar esas formas para detectar características de alto nivel tales como estructuras faciales, partes de un auto, entre otras, en las capas de más alto nivel.

La última capa en una CNN utiliza estas características de alto nivel para realizar predicciones considerando los contenidos de una imagen. Las CNNs nos proveen beneficios claves con respecto al reconocimiento de imágenes:

- **Invariancia local:** nos permite clasificar una imagen que contiene un objeto particular sin importar donde aparece éste en la imagen.
- **Composicionalidad:** cada filtro compone un parche local de características de nivel inferior en una representación de nivel superior, similar a como podemos componer un conjunto de funciones matemáticas que se basan en la salida de funciones anteriores. Esta composición permite que nuestra red aprenda características más importantes de forma más profunda.

Existen distintos tipo de convoluciones, las convoluciones **bi-dimensionales (2D)** son usadas para tratar las imágenes, mientras que las convoluciones **unidimensionales (1D)** nos permiten analizar entradas secuenciales, obteniendo la información con dependencias temporales. Entonces al combinar estas dos técnicas se puede apreciar cómo evolucionan en el tiempo las imágenes capturadas y así hacer predicciones a futuro.

4.1. Convolución 1D

Sean f y g funciones discretas [12], entonces f * g es la convolución de f y g, y está definida como:

$$(f * g)(x) = \sum_{i=\infty}^{\infty} f(\mu)g(x - \mu) \tag{27}$$

Intuitivamente, la convolución de dos funciones representa la magnitud en la que se superponen f y una versión trasladada e invertida de g. Se puede definir a la función g como la entrada y a f como el kernel o núcleo de la convolución. Sin embargo, en los algoritmos de machine learning lo que se maneja usualmente son vectores o arreglos de tal forma que nos resultará más provechoso analizar la convolución entre ellos. Si la función f varía sobre un conjunto finito de valores a = a1, a2, ..., an entonces puede ser representado como el vector $[a_1 a_2 \dots a_n]$. Si las funciones f y g son representadas como vectores $a = [a_1 a_2 \dots a_m]$ y $b = [b_1 b_2 \dots b_n]$, entonces $f * g$ es un vector $c = [c_1 c_2 \dots c_{m+n-1}]$ definido de la siguiente forma:

$$c = \sum_{\mu} a_\mu b_{x-\mu+1} \tag{28}$$

Donde u abarca todos los subíndices legales para au y bx-u+1, específicamente $u = max(1, x - 1 + 1)...min(x, m)$. Lo que puede parecer complicado en la teoría no lo es en la práctica, observemos la Fig. 27 [13]. El vector input también se denomina vector de características y el vector output mapa de características. Lo que sucede es que si el kernel tiene un único valor solo es necesario multiplicarlo por cada valor del vector input y guardarlo en el índice correspondiente del vector output.

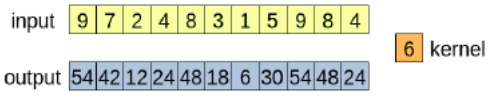


Figura 27: Vectores de convolución unidimensional con kernel simple.

En cambio si tenemos un kernel de dimensiones 2 x 1 como en la Fig. 28 para obtener el valor de salida i debemos usar los valores de entrada i y su vecino $i + 1$.

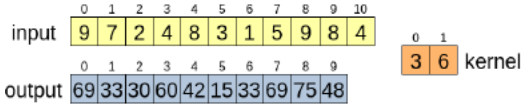


Figura 28: Vectores de convolución unidimensional con kernel doble.

Para obtener el primer valor del vector de salida se realiza la operación o[0] = i[0]k[0] + i[1]k[1] = 69. De esta forma iteramos a lo largo de todo el vector de entrada hasta obtener todos los valores de salida. Podemos notar que el tamaño del vector de salida es menos ahora, a medida que aumentamos el tamaño del kernel disminuye el del vector de salida. Con objeto de dejar totalmente en claro el algoritmo, observamos la Fig. 29. Para obtener el valor del índice 4 del vector de salida, operamos $o[4] = i[3]k[0] + i[4]k[1] + i[5]k[2] = 23$.

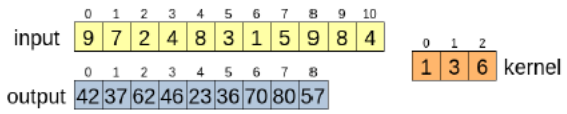


Figura 29: Vectores de convolución unidimensional con kernel triple.

El tamaño del vector de salida es determinado por la siguiente formula [14] :

$$output_{size} = \frac{W - F + 2P}{S + 1} \tag{29}$$

donde $w = imput_{size}, F = kernel_{size}, P = paddingyS = stride.$

4.2. Convolución 2D

Extendiendo la convolución a funciones de dos variables se tiene lo siguiente. Si f y g son funciones discretas de dos variables, entonces $f * g$ es la convolución de f y g y se define:

$$(f * g)(x, y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u, v)g(x - u, y - v) \tag{30}$$

Podemos considerar estas funciones de dos variables como matrices con $A_xy = f(x, y)$ y obtener una definición matricial de la convolución. Si las funciones f y g son representadas como las matrices A y B con dimensión de nxm y kxi respectivamente, entonces f * g da como resultado una matriz C de dimensiones $(n + k - 1)x(m + i - 1)$ definida:

$$c_{xy} = \sum_{u} \sum_{v} a_{uv} b_{x-u+1,y-v+1} \tag{31}$$

donde u y v abarcan todos los subíndices posibles para $a_{uv}yb_{x-u+q,y-v+1}$. Así como notamos que el algoritmo para la convolución 1D no era tan complejo como su definición formal, lo mismo sucede para la convolución 2D pero extrapolando el mecanismo a una dimensión mas. En la figura 30 [15] se analiza el procedimiento. Se debe centrar el kernel K sobre el primer valora a calcular, para luego realizar las respectivas multiplicaciones y posteriormente guardarlas en la matriz de salida O, de esta manera se ira iterando de derecha a izquierda y de arriba hacia abajo toda la matriz I.

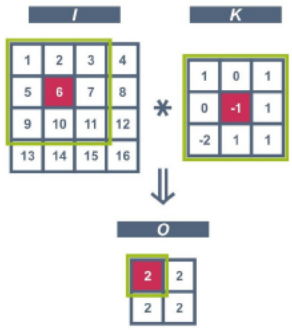


Figura 30: Vectores de convolución bidimensional con kernel 3x3.

Consideramos la Fig. 31 [16] , tenemos una imagen RGB que ha sido separa por sus tres canales de color: rojo, verde y azul. Hay varios espacios de color en los que existen las imágenes: escala de grises, RGB, HSV, CMYK, entre otros.

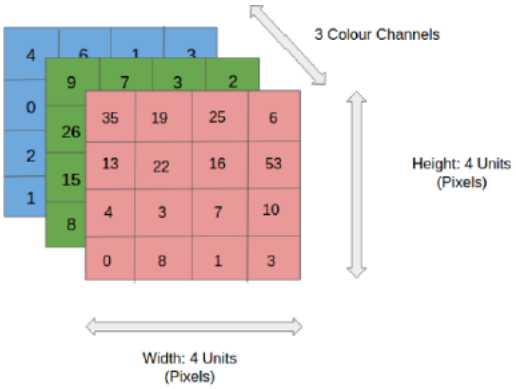


Figura 31: Imagen RGB 4x4x3.

Si consideramos la totalidad de la imagen como un prisma donde la profundidad corresponde a cada canal de color, podemos ver en la Figura 32 el movimiento que realiza el kernel (con forma de cubo) a través del volumen del prisma.

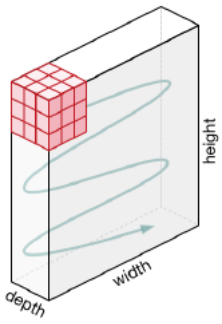


Figura 32: Movimiento del kernel.

4.2.1. Padding

Un problema a abordar al aplicar la convolución es que se tiene a perder píxeles en el perímetro de nuestra imagen (o vector). Dado que normalmente se utilizan núcleos pequeños, para cualquier convolución dada, es posible que solo se pierdan unos pocos píxeles, pero esto puede sumarse a medida que se aplican muchas capaz convolucionales sucesivas. Una solución sencilla a este problema es agregar píxeles adicionales de relleno (padding) alrededor del limite de nuestra imagen de entrada, aumentando así el tamaño efectivo de la imagen. Normalmente, establecemos los valores de los píxeles adicionales en cero [17]. Existen otros tipos de relleno, como continuar el borde, condiciones de contorno cíclicas o rellenar con la media de la imagen, entre otros posibles. En la figura 33, se rellena una entrada de 3 x 3, aumentando su tamaño a 5 x 5. La salida correspondiente aumenta entonces a una matriz de 4 x 4.

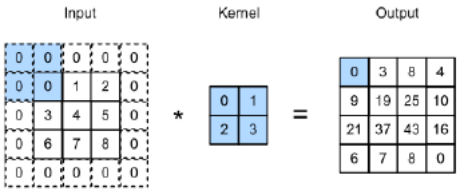


Figura 33: Relleno o Padding.

4.2.2. Stride

Al realizar la convolucion, se comienza con la ventana en la esquina superior izquierda del tensor de entrada y luego la deslizamos sobre todas las ubicaciones, tanto hacia abajo como hacia la derecha. Usualmente deslizamos un elemento a la vez. Sin embargo, a veces, ya sea por eficiencia computacional o porque se desea reducir la resolución, movemos nuestra ventana mas de un elemento a la vez, omitiendo las ubicaciones intermedias. Nos

referimos al numero de filas y columnas atravesadas por diapositiva como la zancada o stride. Hasta ahora se ha ejemplificado utilizando un stride de 1, tanto para la altura como para el ancho. A veces es posible que se desee usar un paso mas grande, como en la figura 34 [16].

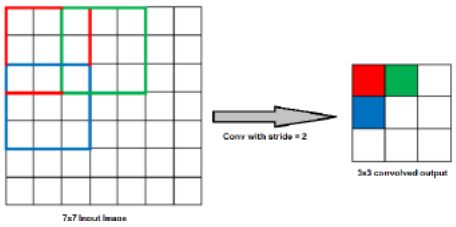


Figura 34: Convolución de *stride* de 2.

De esta forma así como se vio que la operación de padding permite aumentar la dimensión de salida, el stride permite reducir la misma mas allá que la reducción generada por el kernel.

4.3. Tipos de capas

Existen varias tipos de capas usadas [6] para construir CNNs pero las más comunes incluyen:

- Convolucional (CONV)
- Activación (ACT)
- Pooling (POOL)
- Fully-connected (FC)
- Dropout (DO)

Apilando estas capas de una manera específica producimos una CNN. De estos tipos de capas, CONV y FC (y en menor medida, BN (*Batch normalization*) son las únicas capas que contienen parámetros que se aprenden durante el proceso de entrenamiento.

Las capas ACT y DO no se consideran verdaderas capas en sí mismas, pero a menudo se incluyen en los diagramas de red para que la arquitectura sea explícitamente clara.

Las capas (POOL), de igual importancia que CONV y FC, también se incluyen en los diagramas de red, ya que tienen un impacto sustancial en las dimensiones espaciales de una imagen mientras se mueve a través de una CNN.

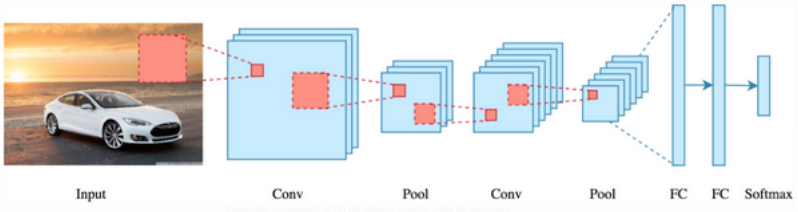


Figura 35: Capas de una red neuronal convolucional.

4.3.1. Convolución

La capa convolucional (CONV) es el componente básico de una red neuronal convolucional. Los parámetros de la capa CONV consisten en un conjunto de K kernels entrenables, donde cada uno tiene un ancho y un alto, y casi siempre son cuadrados.

Se puede considerar al kernel como una pequeña matriz que se aplica a una imagen mediante un producto escalar para extraer ciertas características importantes o patrones de esta.

Una capa o filtro puede tener varios kernels que al convolucionar con el vector de entrada (que podría ser una imagen) producen mapas de características (features map).

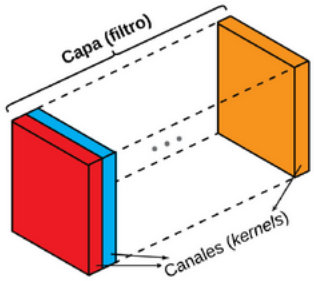


Figura 36: Capa compuesta de una colección de kernels.

La diferencia entre filtro y kernel es un poco complicada. A veces, se usan indistintamente, lo que podría crear confusiones. Esencialmente, estos dos términos tienen una sutil diferencia. Un "kernel"se refiere a una matriz 2D de pesos. El término "filtro"se refiere a estructuras 3D de varios kernels apilados juntos. Para un filtro 2D, el filtro es igual que el kernel. Pero para un filtro 3D y la mayoría de las convoluciones en el aprendizaje profundo, un filtro es una colección de kernels (fig. "anterior"). Cada kernel es único, enfatizando diferentes aspectos del canal de entrada.

El funcionamiento de la capa convolucional se resume en pensar en cada uno de los K kernels deslizándose a través del vector de entrada, computando un producto de Hadamard,

sumando cada uno de sus valores y luego almacenando el valor generado en un mapa 2D de activación. En las siguientes figuras se puede observar una visualización de la secuencia.

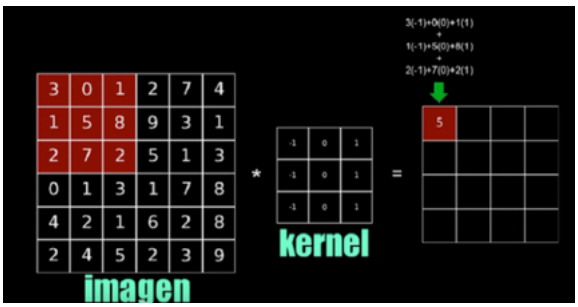


Figura 37: Operación de un kernel contra la imagen.

Figura 38: Se puede observar a la izq. un conjunto de kernels que actúan sobre una imagen dando como resultado un mapa de activación de menor dimensión que la imagen.

Luego de aplicar los K filtros al vector de entrada, ahora tenemos K×2D mapas de activación. Luego apilamos nuestro K mapas de activación a través de la dimensión de profundidad de nuestra matriz para formar el volumen final de salida.

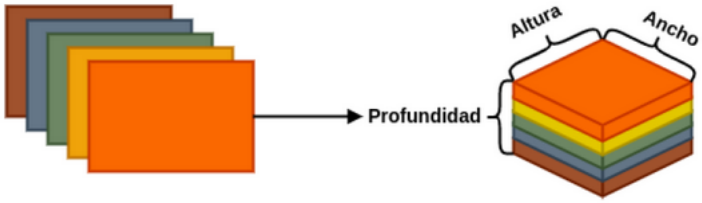


Figura 39: Los k mapas obtenidos se apilan para formar un volumen de entrada de la siguiente capa.

Cada entrada en el volumen de salida es, por tanto, una salida de una neurona que "mira"sólo una pequeña región de la entrada. De esta manera, la red .ªprende"los filtros

que se activan cuando ven un tipo específico de característica en una ubicación espacial determinada en el volumen de entrada.

En las capas inferiores de la red, los filtros pueden activarse cuando ven regiones con forma de borde o de esquina. Luego, en las capas más profundas de la red, los filtros pueden activarse en presencia de características de alto nivel, como partes de la cara, la pata de un perro, el capó de un automóvil, etc.

4.3.2. Activación

Luego de cada capa CONV en una CNN, normalmente aplicamos una función no lineal, como ReLU, ELU, etc (se ejemplifica en la figura). Se utiliza la función de activación ReLU por la linealidad generada por la convolución. Las capas ACT no son técnicamente çapas"(debido al hecho de que no se aprenden parámetros/pesos dentro de una capa de activación) y, a veces, se omiten en los diagramas de arquitectura de red, ya que se supone que una activación sigue inmediatamente a una convolución.

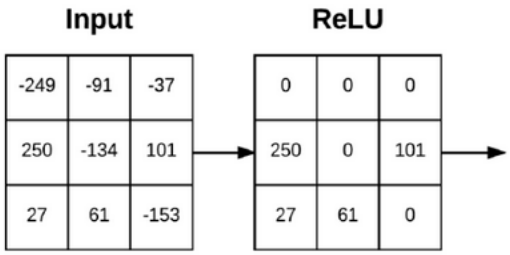


Figura 40: Se muestra un ejemplo de volumen de entrada desplazándose a través de un ACT ReLU.

4.3.3. Fully-connected

Las neuronas en las capas FC están totalmente conectadas a todas las activaciones de la capa anterior, como en una red neuronal feed-forward. Siempre se ubican al final de la red.

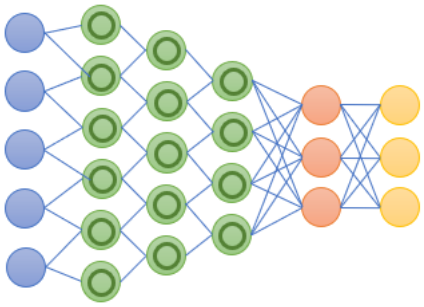


Figura 41: Neuronas fully-connected.

4.3.4. Pooling

De manera similar a la capa convolucional, la capa de POOL es responsable de reducir el tamaño espacial de la entidad convolucionada. Esto es para disminuir la potencia computacional requerida para procesar los datos a través de la reducción de dimensionalidad. Además, es útil para extraer características dominantes que son invariantes rotacionales y posicionales, manteniendo así el proceso de entrenamiento efectivo del modelo.

La operación de pooling es una operación no paramétrica (no se aprenden pesos) que sirve para reducir la dimensión espacial de una imagen o activación. La operación tiene en común con la convolución en que opera a través de un kernel de una dimensión dada, tiene un stride definido y permite la operación de padding. Las formas más usuales de aplicar un pooling, dado un tamaño del kernel k, son las siguientes:

- **Max pooling (MPOOL):** dada una ventana k $\times$ k de la entrada, tomó el máximo de los valores de esa ventana como salida.
- **Average pooling (APOOL):** devuelve el promedio de todos los valores de la parte de la imagen cubierta por el kernel.

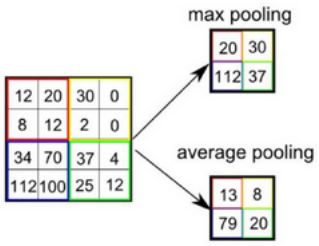


Figura 42: Tipos de pooling.

Cabe destacar que MAXPOOL también actúa como supresor de ruido ya que descarta las activaciones ruidosas por completo además de la reducción de dimensionalidad. Por otro lado, el AVGPOOL simplemente realiza la reducción de dimensionalidad como un mecanismo de supresión de ruido.

Ambas técnicas agrupan de forma efectiva los datos, por lo tanto nos preguntamos cuándo usar cada una. Cuando se elije una capa uno debe estar versado con los datos y como este proceso funciona y porque es beneficioso para el dataset. Average pooling retiene una gran cantidad de datos, mientras que max pooling rechaza una gran cantidad de los mismos [18]. Los objetivos detrás de esto son:

- Max pooling extrae solo las características que mas destacan de los datos.
- Average pooling extrae características de una forma mas simple.

Por lo tanto, average pooling a veces no puede extraer las características más importantes porque tiene en cuenta todo, y da un valor promedio el cual puede o no ser importante. Pero esto también significa que average pooling impulsa a la red a identificar la extensión completa del objeto, mientras que max pooling restringe esto a solo las características más importantes, pudiendo perder así algunos detalles. Por lo tanto, la elección del método de pooling es **dependiente de lo que se busque en la capa de pooling y en la CNN**. En la figura 43 se muestran dos casos con el mismo tipo de imagen, pero intercambiando el tono del fondo con el del primer plano. Como puede observarse esto conlleva un impacto drástico en la efectividad de la salida de la capa de max pooling, mientras que en la de average pooling mantiene su carácter simple y promedio.

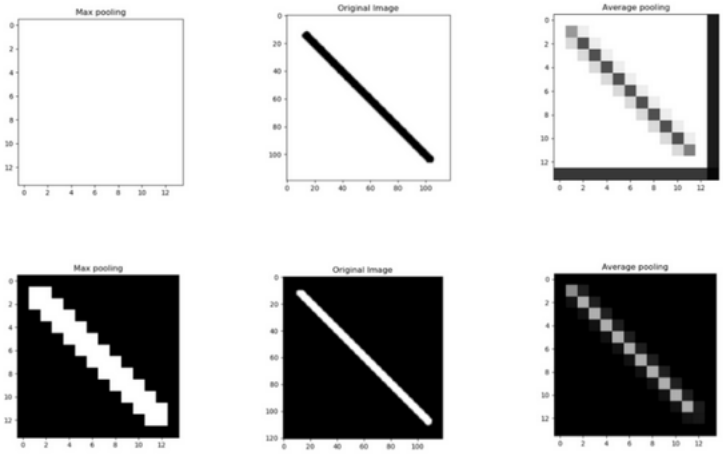


Figura 43: Dos casos de pooling, avgpool vs maxpool.

Max pooling funciono realmente bien para generalizar la linea con el fondo negro, pero la linea con el fondo blanco desapareció totalmente. Average pooling nos puede ayudar en

los casos en los que haya efectos tan drásticos Max pooling funciona mejor para fondos más oscuros y, por lo tanto, puede ahorrar mucho en costos de cálculo computacional mientras que average pooling muestra un efecto similar independientemente del fondo. Por lo tanto se puede concluir que, **las capas deben ser escogidas de acuerdo a los datos y los resultados requeridos**, teniendo en cuenta la importancia y prominencia de las características en el mapa, y entendiendo como ambas funcionan e impactan en la CNN, se puede elegir que capa se utilizará.

4.3.5. Batch normalization

Por lo general, para entrenar una red neuronal, realizamos un preprocesamiento de los datos de entrada, por ejemplo, normalizar todos los datos para que se parezcan a una distribución normal (es decir, media cero y una varianza unitaria). Algunas razones para realizar esto puede ser prevenir la saturación temprana de funciones de activación no lineales como la función sigmoidea, asegurar que todos los datos de entrada estén en el mismo rango de valores, etc.

La operación toma una entrada del tamaño del batch elegido, y la normaliza tal que la media de los valores sobre el batch sea 0 y su varianza sea 1.

Pero el problema aparece en las capas intermedias porque la distribución de las activaciones cambia constantemente durante el entrenamiento. Esto ralentiza el proceso de entrenamiento porque cada capa debe aprender a adaptarse a una nueva distribución en cada paso del entrenamiento. Este problema se conoce como cambio de covariables interno.

Podemos utilizar la normalización por lotes o *Batch Normalization* (BN) como un método para normalizar las entradas de cada capa para así forzarlo a tener aproximadamente la misma distribución en cada paso de entrenamiento, con el fin de combatir el problema expresado anteriormente. Durante el tiempo de entrenamiento, una capa de normalización por lotes se obtiene el promedio y la varianza del lote:

$$\mu_\beta = \frac{1}{m}\sum_{i=1}^{m} x_i \tag{32}$$

$$\sigma_\beta^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\beta)^2 \tag{33}$$

Normalizamos las entradas de la capa usando las estadísticas del lote calculados previamente:

$$\bar{x}_i = \frac{x_i - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}} \tag{34}$$

Establecemos $1e70$ para evitar sacar la raíz cuadrada de cero. Aplicar esta ecuación implica que las activaciones que salen de una capa BN tendrán una media y una varianza unitaria aproximadamente cero (es decir, centrada en cero). Reemplazamos el mini-lote

y con promedios de y calculados durante el proceso de entrenamiento. Esto asegura que podemos pasar vectores a través de nuestra red y aún así obtener predicciones precisas sin ser sesgados por y del mini-lote final pasado a través de la red en el momento del entrenamiento.

La BN también tiene el beneficio adicional de ayudar a .[es]tabilizar.[el] entrenamiento, lo que permite una mayor variedad de tasas de aprendizaje y fortalezas de regularización. Esto no alivia la necesidad de ajustar estos parámetros, por supuesto, pero le facilitará la vida al hacer que la tasa de aprendizaje y la regularización sean menos volátiles y más fáciles de ajustar. También tenderá a notar pérdidas finales más bajas y una curva de pérdida más estable en sus redes.

4.3.6. Dropout

El dropout (DO) es en realidad una forma de regularización que tiene como objetivo ayudar a prevenir el sobreajuste aumentando la precisión de las pruebas, quizás a expensas de la precisión del entrenamiento.

La razón está en reducir el sobreajuste alterando de forma explícita la arquitectura de la red en tiempo de entrenamiento. La desconexión aleatoria de las conexiones garantiza que ningún nodo de la red sea responsable de la activación cuando se le presenta un patrón determinado. El DO garantiza que haya múltiples nodos redundantes que se activarán cuando se les presenten entradas similares (lo que a su vez ayuda a que nuestro modelo a generalizar).

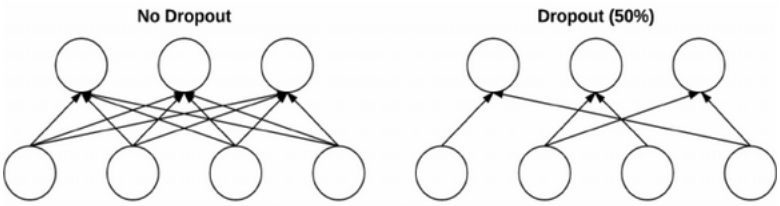


Figura 44: Izq.:Dos capas FC sin DO. Derecha: Las mismas dos capas luego de realizar dropout sobre la mitad de la conexiones

4.4. WaveNet y capas convolucionales casuales dilatadas

WaveNet es una red neuronal profunda para generar audio muestra a muestra. La arquitectura de este modelo permite aprovechar las eficiencias de las capas de convolución al mismo tiempo que alivia el desafío de aprender las dependencias a largo plazo en una gran cantidad de pasos de tiempo (más de 1000). [19] El modelo es totalmente probabilístico y auto regresivo En este modelo, cada muestra de audio está condicionada por la

muestra de audio anterior. La probabilidad condicional es modelada por una pila de capas convolucionales.

En el núcleo de WaveNet se encuentra la capa de convolución causal dilatada (figura 45), que le permite tratar adecuadamente el orden temporal y manejar las dependencias a largo plazo sin una explosión en la complejidad del modelo. [20]

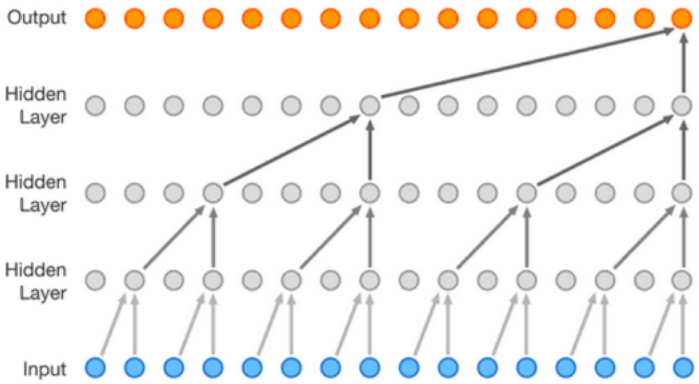


Figura 45: Paso de la información a través de la convolución casual dilatada

En una capa de convolución unidimensional tradicional, deslizamos un filtro de pesos a través de una serie de entrada, aplicándolo secuencialmente a las regiones (generalmente superpuestas) de la serie. Pero cuando utilizamos el historial de una serie temporal para predecir su futuro, debemos tener cuidado. A medida que formamos capas que eventualmente conectan los pasos de entrada a las salidas, debemos asegurarnos de que las entradas no influyan en los pasos de salida que los siguen a tiempo. De lo contrario, estaríamos usando el futuro para predecir el pasado, lo que sería hacer trampa.

Para asegurarnos de no hacer trampa de esta manera, ajustamos nuestro diseño de convolución para prohibir explícitamente que el futuro influya en el pasado. En otras palabras, sólo permitimos que las entradas se conecten a salidas de pasos de tiempo futuros en una estructura causal, como se muestra a continuación en una visualización del documento WaveNet. En la práctica, esta estructura 1D causal es fácil de implementar shifteando las salidas convolucionales tradicionales en varios pasos de tiempo.

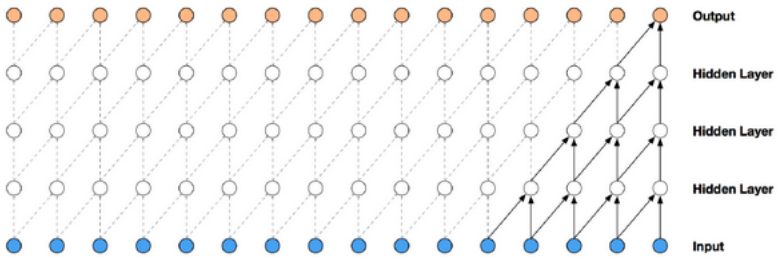


Figura 46: Visualización de una pila de capas causales convoluciones

Las convoluciones causales proporcionan la herramienta adecuada para manejar el flujo temporal, pero necesitamos una modificación adicional para manejar adecuadamente las dependencias a largo plazo. En la figura de convolución causal simple, puede ver que solo los 5 pasos de tiempo más recientes pueden influir en la salida resaltada. De hecho, necesitaríamos una capa adicional por paso de tiempo para llegar más atrás en la serie (para usar la terminología adecuada, para aumentar el campo receptivo de la salida). Con una serie de tiempo que tiene una gran cantidad de pasos, el uso de convoluciones causales simples para aprender de toda la historia rápidamente haría un modelo demasiado complejo computacional y estadísticamente.

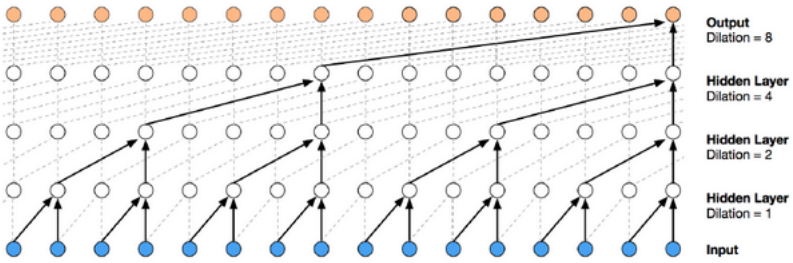


Figura 47: Visualización de una pila de capas convolucionales dilatadas

En lugar de cometer ese error, WaveNet utiliza convoluciones dilatadas, que permiten que el campo receptivo aumente exponencialmente en función de la profundidad de la capa de convolución.

En una capa de convolución dilatada, los filtros no se aplican a las entradas de una manera secuencial simple, sino que omiten una entrada de tasa de dilatación constante entre cada una de las entradas que procesan, como en el diagrama WaveNet a continuación. Al aumentar la tasa de dilatación multiplicativamente en cada capa (por ejemplo, 1, 2, 4, 8,), podemos lograr la relación exponencial entre la profundidad de la capa y el tamaño del campo receptivo que deseamos.

En la figura anterior, puede ver cómo ahora solo necesitamos 4 capas para conectar los 16 valores de la serie de entrada a la salida resaltada (digamos, el valor de paso de tiempo 17). Por extensión, cuando se trabaja con una serie de tiempo diaria, se puede capturar más de un año de historia con solo 9 capas de convolución dilatadas de esta forma.

5. Redes neuronales recurrentes

Anteriormente hemos visto redes neuronales tales como las feedforward o convolucionales nos permiten clasificar un dato, por ejemplo una palabra, un sonido o una imagen, pero tienen un inconveniente, y es que cuando se tiene una secuencia de datos, por ejemplo una secuencia de palabras, una conversación o una secuencia de imágenes, es decir un video, este tipo de arquitecturas no pueden procesar ese tipo de datos. Las Redes Neuronales Recurrentes (RNN) [31] resuelven este inconveniente al ser capaces de procesar diferentes tipos de secuencias, como textos, conversaciones, videos, musica y además de eso no solo clasifican los datos como lo hacen las redes previamente vistas, sino que también están en capacidad de generar nuevas secuencias. Si a una red neuronal feedforward o convolucional se le presenta una imagen o una palabra, con el entrenamiento adecuado estas arquitecturas lograran clasificar sin número de datos, logrando a la vez una alta precisión. Pero qué sucede si en lugar de una única imagen o palabra se introduce a la red una secuencia de imágenes o una secuencia de palabras. En este caso ninguna de estas redes será capaz de procesar los datos por dos motivos:

- Estas arquitecturas están diseñadas para que los datos de entrada y salida siempre tengan el mismo tamaño; sin embargo un video o una conversación se caracterizan por ser un tipo de datos con un tamaño variable, una cantidad variable de “frames” en el caso del video o una cantidad variable de palabras en el caso de la conversación.
- En un video o en una conversación, los datos están **correlacionados**, esto quiere decir que la siguiente palabra pronunciada o la siguiente imagen en la secuencia de video dependerá de la palabra o imagen anterior respectivamente. Incluso, estas palabras e imágenes estarán relacionadas con aquellas que se presenten más adelante en la secuencia y una NN o CNN no está en capacidad de analizar esta relación.

Una secuencia es una serie de datos que siguen un orden específico y tienen únicamente significado cuando se analizan en conjunto y no de manera individual. Es evidente que una secuencia no tiene un tamaño predefinido pues no se puede saber con antelación el número de datos. Las RNN resuelven los inconvenientes expresados anteriormente, pues pueden procesar tanto a la entrada como a la salida secuencias sin importar su tamaño, y además teniendo en cuenta la correlación existente entre los diferentes elementos de esa secuencia. Para ello este tipo de redes usan el concepto de recurrencia: para generar la salida, que también se conoce como activación, la red usa no solo la entrada actual sino la activación

generada en la iteración previa. En pocas palabras, las redes neuronales recurrentes usan un cierto tipo de memoria para generar la salida deseada.

5.1. Funciones (Neurona recurrente)

Hasta ahora hemos visto redes cuya función de activación solo actúa en una dirección, hacia delante, desde la capa de entrada hacia la capa de salida, es decir, que no recuerdan valores previos. Una red RNN es parecida, pero incluye conexiones que apuntan "hacia atrás", una especie de retroalimentaciones entre las neuronas dentro de las capas. Imaginemos la RNN más simple posible, compuesta por una sola neurona que recibe una entrada, produciendo una salida, y enviando esa salida a sí misma, como se muestra en la figura 48:

Figura 48: Neurona recurrente

El loop de A permite que la información pase de un paso de la red al siguiente. Una red neuronal recurrente se puede considerar como múltiples copias de la misma red, cada una de las cuales pasa un mensaje a un sucesor, es decir, en cada instante de tiempo (también llamado timestep en este contexto), esta neurona recurrente recibe la entrada x de la capa anterior, así como su propia salida del instante de tiempo anterior para generar su salida y. Si desenrollamos el ciclo podemos representar la RNN a través del eje del tiempo, como se muestra en la Figura 49.

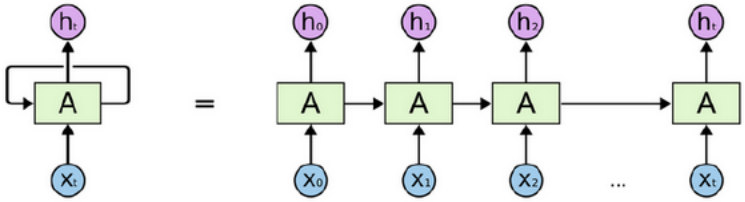


Figura 49: Una RNN desenrollada

Siguiendo esta misma idea, una capa de neuronas recurrentes se puede implementar de

tal manera que, en cada instante de tiempo, cada neurona recibe dos entradas, la entrada correspondiente de la capa anterior y a su vez la salida del instante anterior de la misma capa. Ahora cada neurona recurrente tiene dos conjuntos de parámetros, uno que lo aplica a la entrada de datos que recibe de la capa anterior y otro conjunto que lo aplica a la entrada de datos correspondiente al vector salida del instante anterior.

Esta naturaleza en cadena revela que las redes neuronales recurrentes están íntimamente relacionadas con secuencias y listas. En su esencia una RNN se parece demasiado a una FFNN, excepto que también tiene conexiones hacia atrás. La RNN más simple posible es la que mostramos en la figura 50:

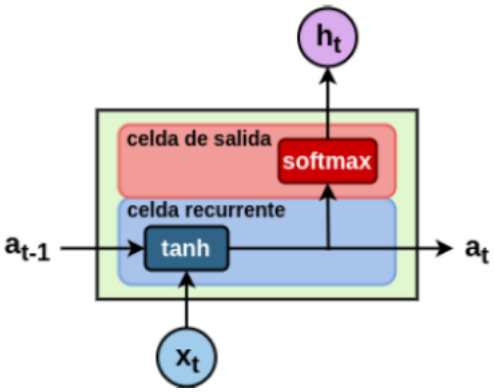


Figura 50: Unidad funcional RNN detallada

Se observa que las entradas son el dato actual, x_t y la activación anterior, at1, mientras que las salidas son la predicción actual, ht, y la activación actual, at. Esta activación también recibe el nombre de hidden state o estado oculto.

Se define:

$$a_t = tanh(W_{aa}a_{t-1} + W_{ax}x_t + b_a) \tag{35}$$

$$h_t = sofmax(W_{ya}a_t + b_y) \tag{36}$$

Donde:

- W_{ax}: matriz de pesos multiplicando la entrada.
- W_{aa}: matriz de pesos multiplicando el estado oculto.
- W_{ya}: matriz de pesos que relacionan el estado oculto de salida.
- b_a: bias.

- b_y: bias que relaciona el estado oculto a la salida.

Es posible entrenar una RNN con una gran cantidad de texto y le pediremos que modele la distribución de probabilidad del siguiente carácter en la secuencia dada una secuencia de caracteres anteriores. Esto nos permitirá generar texto nuevo, de a un carácter a la vez.

Como ejemplo práctico, suponga que solo tenemos un vocabulario de cuatro letras posibles helo y queremos entrenar a un RNN en la secuencia de entrenamiento hello. Esta secuencia de entrenamiento es de hecho una fuente de 4 ejemplos de entrenamiento separados:

1. La probabilidad de e probablemente debería estar dado el contexto de **h**.
2. **l** Debería estar probablemente en el contexto de he.
3. Probablemente también debería ser dado el contexto de hel.
4. Y finalmente **o** debería ser probablemente dado el contexto de hell.

Concretamente, codificamos cada carácter en un vector usando la codificación 1-of-k (es decir, todo cero excepto uno en el índice del carácter en el vocabulario) y los introduciremos en el RNN uno a la vez con la función step. Luego observaremos una secuencia de vectores de salida de 4 dimensiones (una dimensión por carácter), que interpretamos como la confianza que el RNN asigna actualmente a cada carácter que sigue en la secuencia.

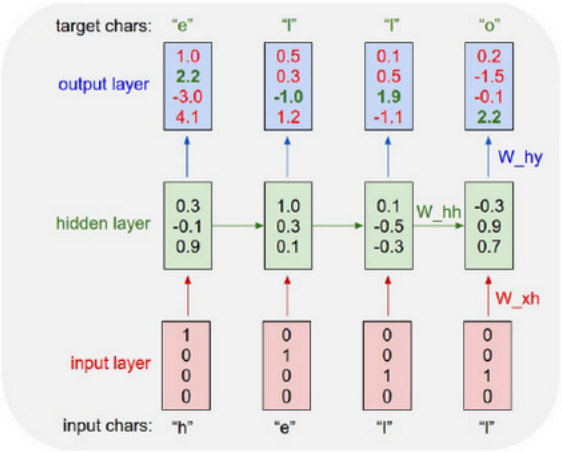


Figura 51: Actualización de pesos RNN

En la Fig. 51 observamos un ejemplo de RNN con capas de entrada y salida de 4 dimensiones y una capa oculta de 3 unidades (neuronas). Las activaciones en el pase hacia adelante cuando el RNN recibe los caracteres "hellçomo entrada. La capa de salida contiene

los pesos que el RNN asigna al siguiente carácter (el vocabulario es "h, e, l, o"); Queremos que los números verdes sean altos y los números rojos bajos.

Por ejemplo, vemos que en el primer paso de tiempo cuando el RNN vio el carácter "h.ªsignó un peso de 1.0 a la siguiente letra que era "h", 2.2 a la letra .e, 3.0 a "lz 4.1 a .o". Dado que en nuestros datos de entrenamiento (la cadena "hello") el siguiente carácter correcto es .e, nos gustaría aumentar su peso (verde) y disminuir los pesos de todas las demás letras (rojo).

De manera similar, tenemos un carácter objetivo deseado en cada uno de los 4 pasos de tiempo a los que nos gustaría que la red le asignara una mayor confianza. Dado que el RNN consta completamente de operaciones diferenciables, podemos ejecutar el algoritmo de back-propagaton para averiguar en qué dirección debemos ajustar cada uno de sus pesos para aumentar los pesos de los objetivos correctos.

Luego podemos realizar una actualización de parámetros, que empuja cada peso una pequeña cantidad en esta dirección de gradiente. Si tuviéramos que alimentar las mismas entradas al RNN después de la actualización del parámetro, encontraríamos que las puntuaciones de los caracteres correctos (por ejemplo, .e en el primer paso de tiempo) serían ligeramente más altas (por ejemplo, 2.3 en lugar de 2.2), y los pesos de los caracteres incorrectos serían ligeramente inferiores.

Luego, repetimos este proceso una y otra vez hasta que la red converge y sus predicciones son finalmente consistentes con los datos de entrenamiento en el sentido de que los caracteres correctos siempre se predicen a continuación.

5.2. Entrenamiento

Recordemos que en las redes neuronales presentadas anteriormente, básicamente se hace Forward-Propagation para obtener el resultado de aplicar el modelo y verificar si este resultado es correcto o incorrecto para obtener la Loss. Después se hace Backward-Propagation(o Backpropagation) que recordemos que no es otra cosa que ir hacia atrás a través de la red neuronal para encontrar las derivadas parciales del error con respecto a los pesos de las neuronas. Estas derivadas son utilizadas por el algoritmo Gradient Descent Para minimizar iterativamente una función dada, ajustando los pesos hacia arriba o hacia abajo, dependiendo de cómo se disminuye la Loss.

Para entrenar una RNN, el truco simplemente es desenrrollarla a través del tiempo y simplemente usar backpropagation (estrategia que recibe el nombre de backpropagation a través del tiempo (BPTT)) como observamos en la Fig. 52

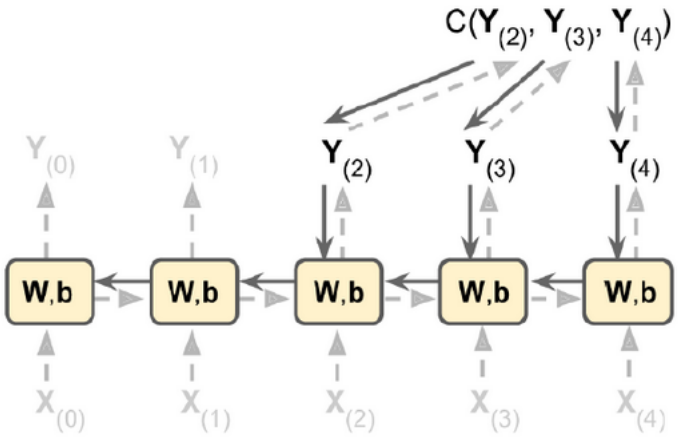


Figura 52: Backpropagation a través del tiempo

Primero realizamos una pasada hacía adelante a través de la red desenrollada (representada en la figura por las flechas punteadas).

Luego la secuencia de salida es evaluada utilizando una función de costo C(Y(0), Y(1), . . . , Y(T)) (donde T es el paso máximo de tiempo). Notemos que la función de coste puede ignorar algunas salidas en función de lo que necesitemos como se muestra en la Fig. 50. Los gradientes de esa función de costo luego son propagados hacía atrás a través de la red desenrollada (representada a través de las líneas sólidas).

Finalmente los parámetros del modelo son actualizados usando los gradientes calculados por BPTT. Notar que los gradientes fluyen hacía atrás a través de todas las salidas utilizadas por la función de costo, no solamente a través de la salida final.

Al realizar el proceso de BPTT, se requiere a nivel matemático incluir la conceptualización de desenrollar, ya que la Loss de un determinado instante de tiempo depende del instante (timestep) anterior. Dentro de BPTT, el error es propagado hacia atrás desde el último hasta el primer instante de tiempo, mientras se desenrollan todos los instantes de tiempo. Esto permite calcular la Loss para cada instante de tiempo, lo que permite actualizar los pesos. Pero el lector ya intuye que el grafo no cíclico que resulta del desplegado en el tiempo es enorme y poder realizar el BPTT es computacionalmente costoso.

5.3. Desvanecimiento del gradiente

Otra forma de alimentar una RNN podría ser a través de las palabras individuales de una oración, dado que esto se realiza de forma secuencial, debemos proveer de una palabra a la vez.

En el ejemplo de la Fig.53 intentaremos predecir la intención del usuario tomando como entrada la oración "What time is it?". [21]

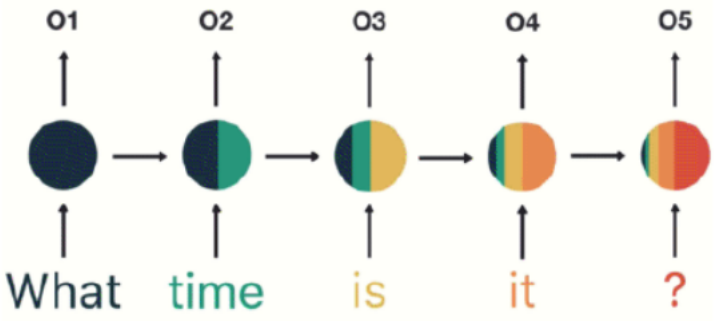


Figura 53: RNN siendo cargada con las palabras de la oración

1. Inicializa sus capas de red y el estado oculto inicial. La forma y dimensión del estado oculto dependerá de la forma y dimensión de su RNN.
2. Luego recorre sus entradas, pasa la palabra y el estado oculto al RNN.
3. El RNN devuelve la salida y un estado oculto modificado.
4. Continúas repitiendo hasta que te quedas sin palabras.
5. Por último, pasa la salida a la capa de feedforward y devuelve una predicción (Fig. 54).

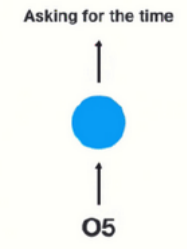


Figura 54: Predicción de la RNN

Pero prestemos atención a la Fig.55. Es posible que haya notado la extraña distribución de colores en los estados ocultos. Eso es para ilustrar un problema con los RNN conocido como memoria a corto plazo.

Figura 55: Estado oculto de la RNN

La memoria a corto plazo es causada por el infame problema del desvanecimiento del gradiente, que también prevalece en otras arquitecturas de redes neuronales. A medida que el RNN procesa más pasos, tiene problemas para retener información de los pasos anteriores.

Como puede ver, la información de la palabra **"What"** y **"time"** es casi inexistente en el último paso. La memoria a corto plazo y el desvanecimiento del gradiente se deben a la naturaleza del algoritmo de back-propagation.

Al hacer *back-propagation*, cada nodo de una capa calcula su gradiente con respecto a los efectos de los gradientes, en la capa anterior. Entonces, si los ajustes a las capas anteriores son pequeños, los ajustes a la capa actual serán aún más pequeños.

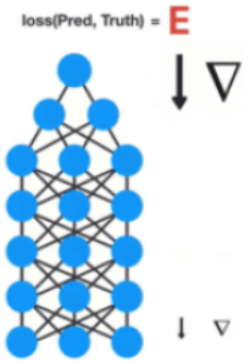


Figura 56: Desvanecimiento del gradiente desde las capas superiores a las inferiores

Es posible pensar en cada paso de tiempo en una RNN como una capa y para entrenarla se usa *backpropagation* a través del tiempo. Los valores del gradiente se reducirán exponencialmente a medida que se propaga a través de cada paso de tiempo.

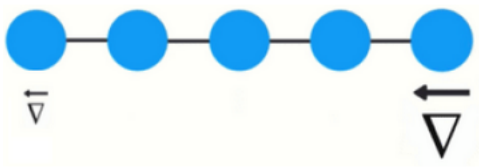


Figura 57: El gradiente se achica a medida que se propaga hacia atrás en el tiempo.

Nuevamente, el gradiente se utiliza para realizar ajustes en los pesos de las redes neuronales, lo que le permite aprender. Pequeños gradientes significan pequeños ajustes. Eso hace que las capas tempranas no aprendan.

Debido a los gradientes que desaparecen, la RNN no aprende las dependencias de largo alcance en los pasos de tiempo. Eso significa que existe la posibilidad de que las palabras **"What"** y **"time"** no se consideren al intentar predecir la intención del usuario. Entonces, la red tiene que hacer la mejor suposición con ïs it?". Eso es bastante ambiguo y sería difícil incluso para un humano. Por lo tanto, no poder aprender en pasos de tiempo anteriores hace que la red tenga una memoria a corto plazo.

5.4. Tipos de arquitecturas

Existen diversas arquitecturas disponibles para estas redes como observamos en la Figura 58 [31], donde cada rectángulo es un vector y cada flecha representa funciones. Los vectores de entrada están en rojo, los vectores de salida están en azul y los vectores verdes mantienen el estado de la RNN.

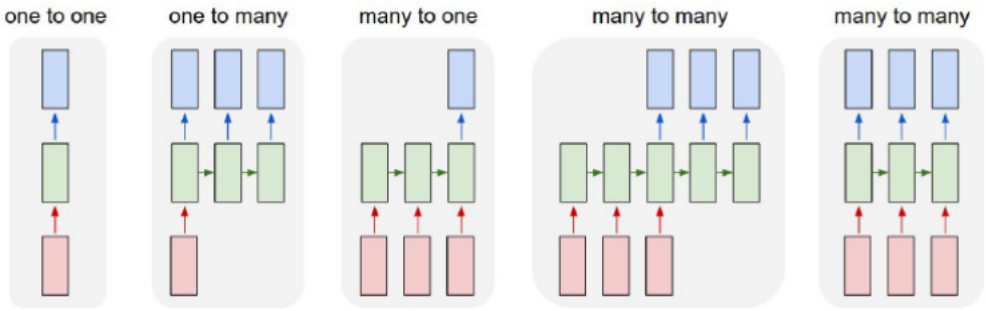


Figura 58: Tipos de arquitecturas para una RNN.

One-to-one: Modo de procesamiento vanilla i.e. sin RNN, desde una entrada de tamaño fijo a una salida de tamaño fijo, por ejemplo la clasificación de imágenes.

One-to-many: La entrada es un único dato y la salida es una secuencia. Un ejemplo de esta arquitectura es el ïmage captioning.[en] donde la entrada es una y la salida es una secuencia de caracteres, un texto, que describe el contenido de la imagen.

Many-to-one: La entrada es una secuencia y la salida es por ejemplo una categoría. Un ejemplo de esto es la clasificación de sentimientos, en donde por ejemplo la entrada es un texto que contiene una crítica a una película y la salida es una categoría indicando si la película le gustó a la persona o no.

Many-to-many: Tanto la entrada como a la salida se tienen secuencias. La primer figura se refiere a RNN utilizadas en traductores automáticos: en este caso la secuencia de salida no se genera al mismo tiempo que la secuencia de entrada pues para poder traducir por ejemplo una frase al español se requiere primero conocer la totalidad del texto en inglés. Y desde luego, en esta misma arquitectura podemos encontrar los conversores de voz a texto o texto a voz. La segunda figura se refiere a secuencias sincronizadas de entrada y salida, por ejemplo clasificación de vídeo donde deseamos etiquetar cada fotograma.

Como era de esperar, el régimen secuencial de operación es mucho más poderoso en comparación con las redes fijas que están condenadas desde el principio por un número fijo de pasos computacionales y, por lo tanto, también es más provechoso a la hora de construir sistemas más inteligentes.

Además, las RNN combinan el vector de entrada con su vector de estado con una función fija (pero aprendida) para producir un nuevo vector de estado. En términos de programación, esto puede interpretarse como ejecutar un programa fijo con ciertas entradas y algunas variables internas. Visto de esta manera, los RNN esencialmente describen programas.

Si entrenar redes neuronales es optimización sobre funciones, entrenar redes recurrentes es optimización sobre programas.

5.5. Tipos de RNNs

Para mitigar la problemática del gradiente desvaneciente, surgieron dos redes neuronales recurrentes especializadas. Las redes denominadas *Long Short-Term Memory o LSTM* para abreviar. Las otras se denominan *Gated Recurrent Units o GRU*.

5.5.1. LSTM

Las LSTM y GRU funcionan esencialmente como las RNN, pero son capaces de aprender las dependencias a largo plazo mediante mecanismos llamados "puertas". Estas puertas son diferentes operaciones de tensor que pueden aprender que información agregar o quitar al estado oculto. Debida a esta capacidad, la memoria a corto plazo es un problema menor para ellas. [22] Si consideramos la celda LSTM como una caja negra, aparenta ser idéntica a una RNN excepto que su estado se divide en dos vectores; $h_{(t)}$ y $c_{(t)}$ ("c" se mantiene por celda). Es posible pensar en $h_{(t)}$ como un estado de corto plazo y a $c_{(t)}$ como un estado de largo plazo.

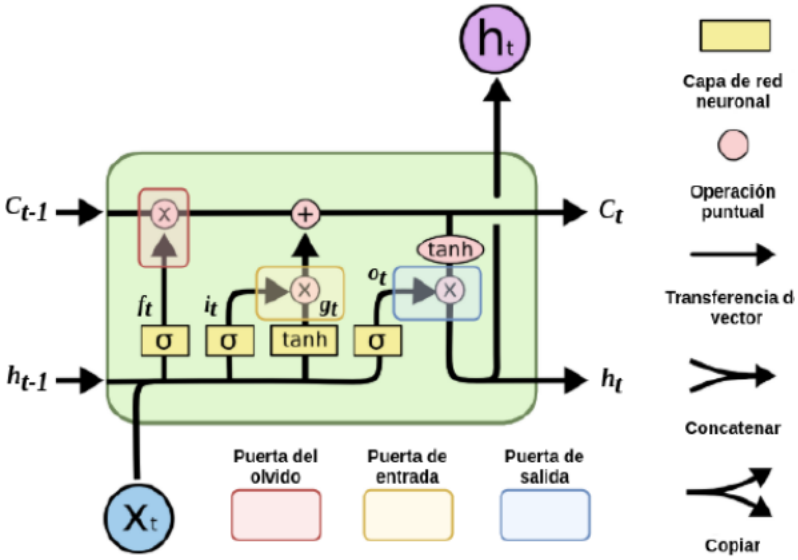


Figura 59: Celda LSTM.

El diagrama completo del LSTM lo podemos observar en la Fig.59, pero vamos a ir paso a paso analizando cada una de las partes que lo componen. La clave de los LSTM es el estado de la celda, la linea horizontal que atraviesa la parte superior de la Fig. 60. El estado de la celda es como una cinta transportadora. Corre directamente a lo largo de toda la cadena, con solo algunas interacciones lineales menores. Es muy fácil que la información fluya sin cambios.

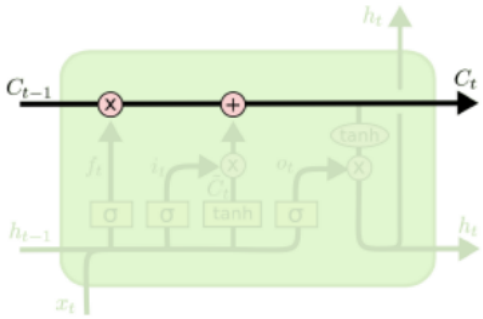


Figura 60: Celda de estado.

El LSTM tiene la capacidad de eliminar o agregar información al estado de la celda, regulada cuidadosamente por estructuras llamadas puertas. Las puertas son una forma de dejar pasar información opcionalmente. Están compuestos por una capa de red neuronal

sigmoidea y una operación de multiplicación. La capa sigmoidea genera números entre 0 y 1, que describen cuánto de cada componente debe dejarse pasar. Un valor de 0 significa "no dejar pasar nada", mientras que un valor de 1 significa "dejar pasar todo". Un LSTM tiene tres de estas puertas para proteger y controlar el estado de la celda. El primer paso es decidir que información vamos a eliminar del estado de la celda. Esta decisión la toma una capa sigmoidea llamada puerta del olvido (Fig.61). Examina h_{t-1} y x_t, y genera un numero entre 0 y 1 para cada numero en el estado de celda C_{t-1}. Un 1 representa "mantener esto completamente", mientras que un 0 representa "deshacerse de esto por completo".

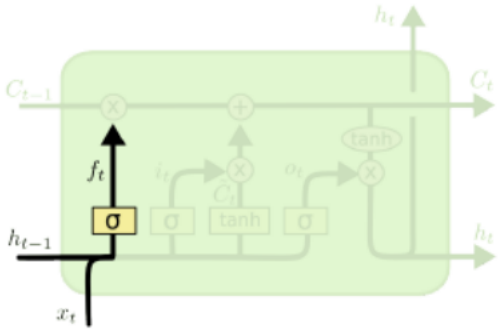


Figura 61: Puerta del olvido.

Por lo tanto, la puerta del olvido queda representada por la siguiente ecuación:

$$f(t) = \sigma(W_{xfx(t)} + W_{hf}h_{t-1} + b_f) \tag{37}$$

El siguiente paso es decidir que nueva información almacenaremos en el estado de la celda. Esto tiene dos partes.

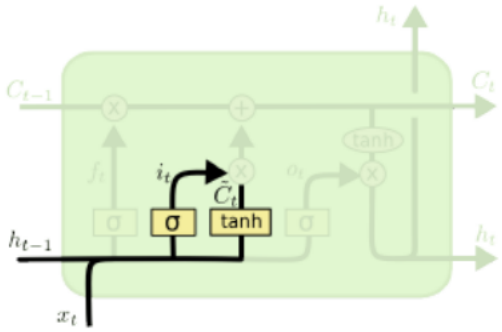


Figura 62: Puerta de entrada.

Una capa sigmoidea llamada puerta de entrada (Fig. 62) decide que valores actualizaremos.

$$i(t) = \sigma(W_{xi}x_{(t)} + W_{hi}h_{t-1}b_i) \tag{38}$$

A continuación, una capa tanh genera un vector de nuevos valores candidatos, que podrían agregarse al estado.

$$g(t) = tanh(W_{xg}x - (t) + W_{hg}h_{(t-1)} + b_g) \tag{39}$$

En el siguiente paso, combinaremos estos estos dos para crear una actualización del estado. Ahora es el momento de actualizar el estado de la celda anterior, $C_(t-1)$, al nuevo estado de la celda $C_{(t)}$. Los pasos anteriores ya decidieron que hacer, solo tenemos que realizar esto.

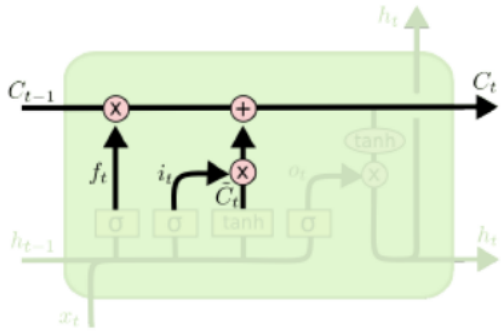


Figura 63: Actualización de la celda.

Multiplicamos el estado anterior por $f_{(t)}$, olvidando las cosas que decidimos olvidar antes. Luego le sumamos $i_{(t)}g_{(t)}$. Estos son los nuevos valores candidatos, escalados según cuanto decidimos actualizar cada valor de estado..

$$c_t = f_{(t)} \bigotimes C_{(t-1)} + i_{(t)} \bigotimes g_{(t)} \tag{40}$$

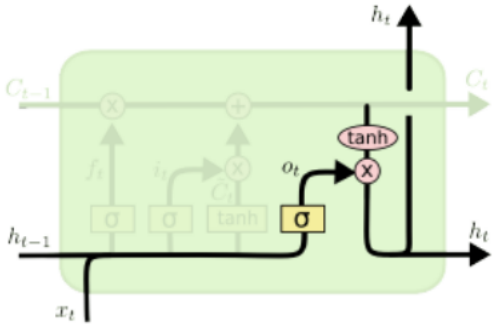


Figura 64: Puerta de salida.

Finalmente, tenemos que decidir que vamos a producir. Esta salida se basara en el estado de nuestra celda, pero sera una versión filtrada. Primero, se ejecuta una capa sigmoidea que se denomina puerta de salida que decide que partes del estado de la celda vamos a generar. Luego, colocamos el estado de la celda a través de tanh (para obtener los valores entre -1 y 1) y lo multiplicaremos por la salida de la puerta, de modo que solo produzcamos las partes que decidimos.

$$o_{(t)} = \sigma(W_{xo}x_t + W_{ho}h_{(t-1)} + b_o) \tag{41}$$

$$h_{(t)} = o_{(t)} \bigotimes tanh(C_{(t)}) \tag{42}$$

Pasando en limpio y observando nuevamente la Fig.59, el vector de entrada $x_{(t)}$ y el estado anterior de corto plazo $h_{(t-1)}$ que alimenta cuatro capas FC. Cada una de ellas sirve a un propósito diferente [23]:

- La capa principal es la que genera $g_{(t)}$. Tiene la función habitual de analizar las entradas actuales $x_{(t)}$ y el estado anterior, a corto plazo, $h_{(t-1)}$. En una celda básica RNN, no hay nada mas que esta capa, y su salida va directamente hacia $y_{(t)}$ y $h_{(t)}$. Por el contrario, en una celda LSTM, la salida de esta capa no sale directamente, sino que se almacena parcialmente en el estado a largo plazo.
- Las otras tres capas son controladores de puerta. Dado que usan la función de activación sigmoidea, sus salidas se ubican entre 0 y 1. Sus salidas alimentan a operaciones de multiplicación elemento a elemento, también conocido como producto de Hadamard [24], por lo que si generar ceros, cierran la puerta, y si generan 1 la abren. Específicamente:
  - La puerta de olvido, controlada por $f_{(t)}$, controla que partes del estado a largo plazo $C_{(t-1)}$ deben borrarse.

- La puerta de entrada, controlada por $i_{(t)}$, controla que partes de $g_{(t)}$ deben agregarse al estado a largo plazo.
- La puerta de salida, controlada por $o_{(t)}$, controla que partes del estado a largo plazo deben leerse y generarse en esta paso de tiempo, tanto en $h_{(t)}$ como en $y_{(t)}$.

En resumen, una celda LSTM puede aprender a reconocer una entrada importante (papel de la puerta de entrada), almacenarla en el estado a largo plazo, aprender a preservarla durante el tiempo que sea necesario (papel de la puerta de olvido) y aprender a extraerla siempre que sea necesario.

5.5.2. GRU

Las Unidades Recurrentes Bloqueadas o Gated Recurrent Unit (GRU) [25] fueron propuestas en 2014 por Kyunghyun et al. [26]. Es una versión simplificada de la celda LSTM, pero se ha demostrado que pueden trabajar con el mismo rendimiento en general [27] [28]. El primer paso [29] que realiza la red GRU es la activación $h_{(t)}$ de la GRU al tiempo t, esta es una interpolación lineal entre la activación previa $h_{(t-1)}$ y el candidato de activación $g_{(t)}$.

$$h_{(t)} = (1 - z_{(t)} \bigotimes h_{(t-1)} + z_{(t)} \bigotimes g_{(t)}) \tag{43}$$

Como se ve en la Figura 65, en este caso la celda solo trabaja con este único vector de estado h(t) y con dos puertas que modificaran el flujo de dicho vector. La primera de ellas, la puerta de actualización, cuya salida es z(t), sirve como controladora de las puertas de entrada y de olvido, de modo que su rango de valores de salida [0 - 1[controla si se elimina información del estado, o se añade. Su calculo puede verse en la ecuación siguiente.

$$z(t) = \sigma(W_{xz}^T x_{(t)} + W_{hz}^T h_{(t-1)} + b_z) \tag{44}$$

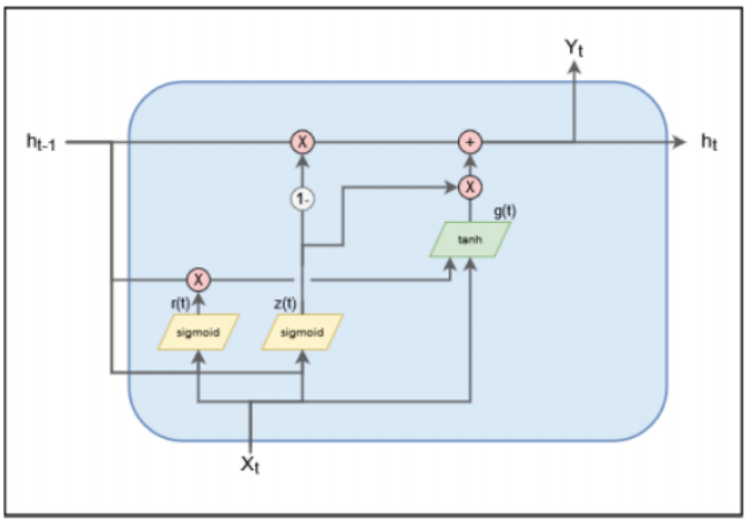


Figura 65: Estructura de una celda GRU.

La segunda, la puerta de reinicio, cuya salida es $r_{(t)}$, controla que parte del estado anterior se le mostrara a la capa principal con función tanh $g_{(t)}$, que sera la que de lugar a la salida de la celda. El calculo de $r_{(t)}$ puede verse en la ecuación 45 y el de $g_{(t)}$ en la ecuación 46.

$$r(t) = \sigma(W_{xr}^T x_{(t)} + W_{hr}^T h_{t-1} + b_r) \tag{45}$$

$$g(t) = tanh(W_{xg}^T x_{(t)} + W_{hg}^T (r(t) \bigotimes h_{(t-1)}) + b_g) \tag{46}$$

5.6. Secuencias de entradas y salidas

Si bien en la Sección 7.4 ya desglosamos los diferentes tipos de arquitecturas que puede tener una RNN, seria de utilidad ahora que ya se posee con un marco teórico aceptable, describirlas un poco mas y ejemplificar en que casos seria provechosa su utilización. Se tomara como referencia la Figura 66 [23].

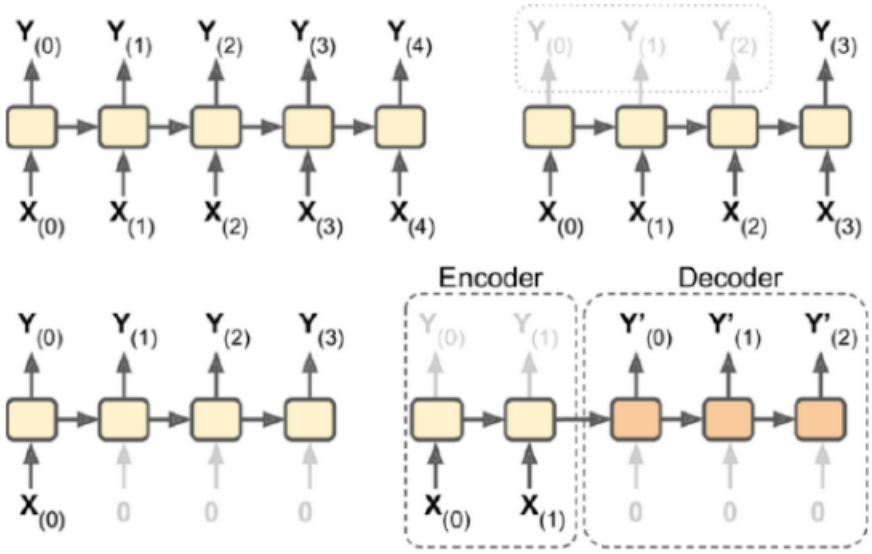


Figura 66: Superior-izquierda: seq-to-seq. Superior derecha: seq-to-vector Inferior-izquierda: vector-to-seq. Inferior-derecha: encoder-decoder.

Secuencia a secuencia Una RNN puede tomar simultáneamente una secuencia de entradas y producir una secuencia de salidas. Por ejemplo, este tipo de red es útil para predecir series de tiempo como los precios de las acciones: se le brindan los precios de los últimos N días y debe generar precios desplazados un día en el futuro, es decir, N - 1 días hasta mañana.

Secuencia a vector Alternativamente, se puede alimentar a la red con una secuencia de entradas e ignorar todas las salidas excepto la ultima. En otras palabras, esta es una red de secuencias a vector. Por ejemplo, podría alimentar a la red con una secuencia de palabras correspondientes a una critica de una película y la red generaría una puntuación de sentimiento (e.g, de 0 [odio] a 1 [amor]).

Vector a secuencia Por el contrario, se puede alimentar la red con una sola entrada en el primer paso de tiempo (y ceros para todos los demás pasos de tiempo) y dejar que genere una secuencia. Por ejemplo la entrada podría ser una imagen y la salida podría ser un titulo para esa imagen.

Secuencia a secuencia con retardo Por ultimo, podría tener una red de secuencia a vector, llamada encoder, seguida de una red de vector a secuencia, llamada decoder. Esto se puede utilizar para traducir una oración de un idioma a otro. Alimentaria la red con una oración en un idioma, el encoder convertiría esta oración en una representación de vector único, y luego el decoder decodificaría este vector en una oración en otro idioma. Este modelo de dos pasos, llamado encoder-decoder, funciona mucho mejor que intentar traducir sobre la marcha con una única RNN secuencia a secuencia, ya que las últimas palabras de una oración pueden afectar las primeras palabras de la traducción, por lo que

debe esperar hasta que haya escuchado la oración completa antes de traducirla.

6. Series temporales

6.1. Definición

Una serie temporal se define como una colección de observaciones de una variable recogidas secuencialmente en el tiempo. Estas observaciones se suelen obtener en instantes de tiempo equiespaciados. Si los datos se obtienen en instantes temporales de forma continua, se debe o bien digitalizar la serie, es decir, recoger sólo los valores en instantes de tiempo equiespaciados, o bien acumular los valores sobre intervalos de tiempo.

Como objetivo se busca estudiar los cambios en esa variable con respecto al tiempo, para poder predecir sus valores futuros.

Ejemplos de series temporales podemos encontrarlos en muchos campos de conocimiento:

Economía y Marketing

- Precio del alquiler de pisos durante una serie de meses.
- Evolución del índice del precio del trigo con mediciones anuales.
- Beneficios netos mensuales de cierta entidad bancaria.
- Índices del precio del petróleo.

Demografía

- Número de habitantes en cierto país por año.
- Tasa de mortalidad infantil por año.

Medio ambiente

- Evolución horaria de niveles de óxido de azufre y de niveles de óxido de nitrógeno en una ciudad durante una serie de años.
- Lluvia recogida diariamente en una ciudad.
- Temperatura media mensual.
- Medición diaria del contenido en residuos tóxicos en un río.

La característica fundamental de las series temporales es que las observaciones sucesivas no son independientes entre sí, y el análisis debe llevarse a cabo teniendo en cuenta el orden temporal de las observaciones. Los métodos estadísticos basados en la independencia de las observaciones no son válidos para el análisis de series temporales porque las observaciones en un instante de tiempo dependen de los valores de la serie en el pasado. [30]

6.2. Objetivos de series temporales

Se pueden considerar varios posibles objetivos:

1. **Descripción** Cuando se estudia una serie temporal, lo primero que se tiene que hacer es dibujarla y considerar las medidas descriptivas básicas. Así, se tiene que considerar:
   - Si los datos presentan forma creciente (tendencia).
   - Si existe influencia de ciertos periodos de cualquier unidad de tiempo (estacionalidad).
   - Si aparecen outliers (observaciones extrañas o discordantes)
2. **Predicción** Cuando se observan los valores de una serie, se pretende normalmente no sólo explicar el pasado, sino también predecir el futuro.

6.3. Componentes de una serie temporal

El estudio descriptivo de series temporales se basa en la idea de descomponer la variación de una serie en varias componentes básicas. Este enfoque no siempre resulta ser el más adecuado, pero es interesante cuando en la serie se observa cierta tendencia o cierta periodicidad. Hay que resaltar que esta descomposición no es en general única.

Este enfoque descriptivo consiste en encontrar componentes que correspondan a una tendencia a largo plazo, un comportamiento estacional y una parte aleatoria.

Las componentes o fuentes de variación que se consideran habitualmente son las siguientes:

- **Tendencia**: Se puede definir como un cambio a largo plazo que se produce en relación al nivel medio, o el cambio a largo plazo de la media. La tendencia se identifica con un movimiento suave de la serie a largo plazo.
- **Efecto Estacional**: Muchas series temporales presentan cierta periodicidad o dicho de otro modo, variación de cierto periodo (anual, mensual, etc.). Por ejemplo, el paro laboral aumenta en general en invierno y disminuye en verano. Estos tipos de efectos son fáciles de entender y se pueden medir explícitamente o incluso se pueden eliminar del conjunto de los datos, desestacionalizando la serie original.
- **Componente Aleatoria**: Una vez identificados los componentes anteriores y después de haberlos eliminado, persisten unos valores que son aleatorios. Se pretende estudiar qué tipo de comportamiento aleatorio presentan estos residuos, utilizando algún tipo de modelo probabilístico que los describa.

De las anteriores las dos primeras son componentes determinísticas, mientras que la última es aleatoria. Así, se puede denotar que:

$$X_t = T_t + E_t + I_t$$

donde T_t es la tendencia, E_t es la componente estacional, que constituyen la señal o parte determinística, e I_t es el ruido o parte aleatoria.

Es necesario aislar de alguna manera la componente aleatoria y estudiar qué modelo probabilístico es el más adecuado. Conocido éste, podremos conocer el comportamiento de la serie a largo plazo.

Este aislamiento de la componente aleatoria se suele abordar de dos maneras:

Enfoque descriptivo: Se estima T_t y Et y se obtiene I_t como:

$$I_t = X_t - T_t - E_t$$

Enfoque de Box-Jenkins: Se elimina de Xt la tendencia y la parte estacional (mediante transformaciones o filtros) y queda sólo la parte probabilística. A esta última parte se le ajustan modelos paramétricos.

6.4. Clasificación descriptiva de la series temporales

Estacionarias: Una serie es estacionaria cuando es estable, es decir, cuando la media y la variabilidad son constantes a lo largo del tiempo. Esto se refleja gráficamente en que los valores de la serie tienden a oscilar alrededor de una media constante y la variabilidad con respecto a esa media también permanece constante en el tiempo. Es una serie básicamente estable a lo largo del tiempo, sin que se aprecian aumentos o disminuciones sistemáticos de sus valores. Para este tipo de series tiene sentido conceptos como la media y la varianza. Sin embargo, también es posible aplicar los mismos métodos a series no estacionarias si se transforman previamente en estacionarias.

Por ejemplo:

Se presenta una serie estacionaria discreta. La serie es estable alrededor de un valor central.

Figura 67: Numero de reclamaciones vs reclamaciones semanales.

Si representamos un histograma de esta serie, podemos describir adecuadamente la información:

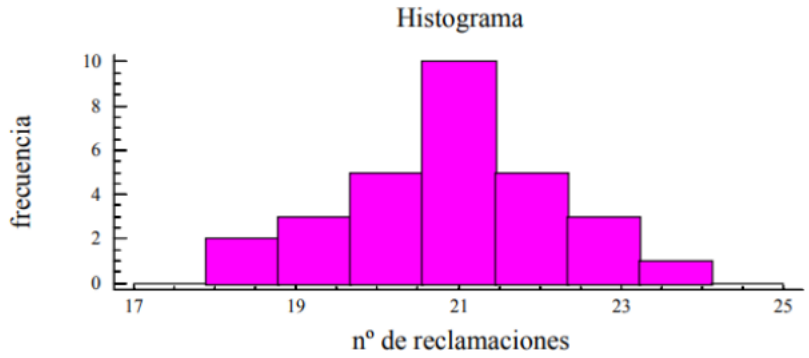


Figura 68: Histograma de la serie de reclamos.

En promedio se reciben unas 21 reclamaciones semanales. Este número es bastante estable y la distribución de la variable es aproximadamente simétrica. La mejor predicción para el próximo valor de la serie es la media, aunque lo ideal sería aplicar los modelos de Inferencia para series estacionarias.

No Estacionarias: Son series en las cuales la media y/o variabilidad cambian en el tiempo. Los cambios en la media determinan una tendencia a crecer o decrecer a largo plazo, por lo que la serie no oscila alrededor de un valor constante. Es decir tenemos:

- Pueden mostrar cambios de varianza.
- Pueden mostrar una tendencia, es decir que la media crece o baja a lo largo del tiempo.

- Además, pueden presentar efectos estacionales, es decir que el comportamiento de la serie es parecido en ciertos tiempos periódicos en el tiempo.

Por ejemplo:

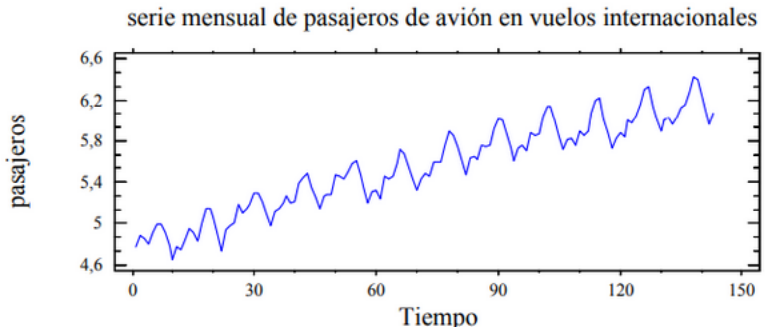


Figura 69: Serie no estacionaria.

La serie presenta además de una tendencia creciente, una pauta estacional debido a que los pasajeros transportados en los meses de verano es mayor que en el resto del año.

6.5. Clasificación del modelo

Desde un punto de vista enfocado al modelado predictivo, los datos de entrada se pueden subdividir de acuerdo a sus características para comprender mejor su relación con la variable de salida.

6.5.1. Tipo de variables de entrada

Una variable de entrada puede ser:

- **Endógena**: si se ve afectada por otras variables del sistema y la variable de salida depende de ella.
- **Exógena**: si es independiente de otras variables del sistema y la variable de salida depende de ella.

En pocas palabras, las variables endógenas están influenciadas por otras variables del sistema, incluidas ellas mismas, mientras que las variables exógenas no lo están y se consideran fuera del sistema. Por lo general, un problema de pronostico de series temporales tiene variables endógenas, por ejemplo el resultado de una función de cierto numero de pasos de tiempo anteriores, y puede o no tener variables exógenas. A menudo, las variables exógenas se ignoran debido al fuerte enfoque en la series de tiempo. Pensar explícitamente en ambos tipos de variables puede ayudar a identificar datos exógenos que se pasan por alto fácilmente o incluso características de ingeniería que pueden mejorar el modelo.

6.5.2. Objetivos

Los problemas de modelado predictivo pueden ser de:

- **Regresión**: son aquellos en los que se predice una cantidad.
- **Clasificación**: son aquellos en los que se predice una categoría.

6.5.3. Estructura

Es útil trazar cada variable en una serie temporal e inspeccionar la trama en busca de posibles patrones. Este tipo de patrones puede ser:

- **No estructurado**: No hay patrón dependiente del tiempo sistemático obvio o discernible en una variable de serie temporal.
- **Estructurado**: Patrones sistemáticos dependientes del tiempo en una variable de serie temporal, por ejemplo, como se mencionó anteriormente la tendencia y/o estacionalidad.

Como se vio previamente en el inciso 8.3, a menudo podemos simplificar el proceso de modelado identificando y eliminando las estructuras obvias de los datos, como una tendencia creciente o un ciclo repetido. Algunos métodos clásicos incluso permiten especificar parámetros para el manejo de estas estructuras sistemáticas directamente.

6.5.4. Cantidad de variables

Una serie temporal puede se puede clasificar de acuerdo a la cantidad de variables medidas que serán utilizadas como entrada al modelo. Estas pueden ser univariadas, es decir una única variable, o multivariadas, múltiples variables. De acuerdo a esta clasificación, los modelos a aplicar difieren de forma considerable en cuanto a su complejidad (multivariadas).

6.5.5. Horizonte de pronóstico

El horizonte de pronostico es el periodo de tiempo en el futuro para el cual se pretende predecir. Este generalmente varia desde un horizonte de pronostico a corto plazo, menos de tres meses, hasta horizontes a largo plazo, mas de dos años. Sin embargo eso es una cuestión relativa a como fueron medidos los datos del dataset (segundos, minutos, horas, días, etc). Dado que nuestro dataset esta indexado según una marca de tiempo (timestamp) determinada, podemos decidir cuantos pasos hacia adelante vamos a realizar en nuestra predicción.

Cantos mas pasos de tiempo se proyecten a futuro, más desafiante será el problema dada la naturaleza agravada de la incertidumbre en cada paso de tiempo previsto.

6.5.6. Estático vs Dinámico

Esto hace referencia a la actualización del modelo para dar nuevos pronósticos.

- **Estático**: El modelo de pronóstico se ajusta una vez y se usa para hacer predicciones.
- **Dinámica**: El modelo de pronóstico se ajusta a los nuevos datos disponibles antes de cada predicción.

6.5.7. Uniformidad de tiempo

- **Contiguo**: Las observaciones se hacen uniformes a lo largo del tiempo. Muchos problemas de series temporales tienen observaciones contiguas, como una observación cada hora, día, mes o año.
- **Discontinuo**: Las observaciones no son uniformes a lo largo del tiempo. La falta de uniformidad de las observaciones pueden deberse a valores perdidos o corruptos. También puede ser una característica del problema cuando las observaciones solo están disponibles esporádicamente o en intervalos de tiempo irregulares.

En el caso de observaciones no uniformes, es posible que se requiera un formato de datos específico al ajustar algunos modelos para que las observaciones sean uniformes a lo largo del tiempo.

Referencias

[1] David I Poole, Randy G Goebel, and Alan K Mackworth. *Computational intelligence.* Oxford University Press New York, 1998.

[2] Kaplan y Michael Haenlein. *Siri, Siri in my Hand, who's the Fairest in the Land? On the Interpretations* 2018.

[3] Christopher M Bishop. *Pattern recognition and machine learning. Springer,* 2006.

[4] *The 80/20 Split Intuition and an Alternative Split Method* URL `https://towardsdatascience.com/finally-why-we-use-an-80-20-split-for-training-and-test-` 2020.

[5] Victor Roman *Aprendizaje Supervisado: Introducción a la Clasificación y Principales Algoritmos* URL `https://medium.com/datos-y-ciencia/aprendizaje-supervisado-introducci%C3%B3n-a-la-clasificaci%C3%B3n-y-principales-algoritmos-dadee99c9407` 2019.

[6] Pedro Antonio Gutiérrez. Github - pagutierrez/tutorial-sklearn: Tutorial sobre scikit-learn completo. URL https://github.com/pagutierrez/tutorial-sklearn,2020. (Accessed on 12/28/2020).

[7] Fernando Sancho Caparrini. *Aprendizaje Supervisado y No Supervisado.* URL http://www.cs.us.es/~fsancho/?e=77 2020.

[8] B. G. Trejo. *Selección de herramientas de Machine Learning aplica- do a problemas de ingeniería.* Facultad de Ciencias Exactas, Físicas y Naturales, Universidad Nacional de Córdoba 2019.

[9] ¿Qué es una red neuronal Feedforward? URL https://quesignificado.org/que-es-una-red-neuronal-feedforward/

[10] Sachin Malhotra. *Demystifying Gradient Descent and Backpropagation via Logistic Regression based Image...* URL https://www.freecodecamp.org/news/demystifying-gradient-descent-and-backpropagation-via-logistic-regression-based-image-c

[11] Jaime Durán. *Todo lo que Necesitas Saber sobre el Descenso del Gradiente Aplicado a Redes Neuronales* URL https://medium.com/metadatos/todo-lo-que-necesitas-saber-sobre-el-descenso-del-gradiente-aplicado-a-redes-neuronales~:text=El%20algoritmo%20m%C3%A1s%20utilizado%20para,es%20el%20descenso%20del%20gradiente.&text=Lo%20definiremos%20m%C3%A1s%20adelante%2C%20pero,su%20desviaci%C3%B3n%20a%20la%20salida 2019.

[12] Frank Keller. *Convolutions and Kernels.* School of Informatics, Universityof Edinburgh Feb 2010.

[13] Cogneethi. C 4.1 — 1D Convolution — CNN — Object Detection — Machine Learning — EvODN, Aug 2019. URL https://www.youtube.com/watch?v=yd_j_zdLDWs 2019.

[14] StackOverflow. How did they calculate the output volume for this convnet example in Caffe?, Jan 2021. URL https://stackoverflow.com/questions/32979683/how-did-they-calculate-the-output-volume-for-this-convnt-example-in-caffe 2015.

[15] Louis N Andrianaivo, Roberto D'Autilia, and Valerio Palma. *Architecture recognition by means of convolutional neural networks. International Archives of the Photogrammetry, Remote Sensing Spatial Information Sciences,* 2019.

[16] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. Medium,* Oct 2020. URL https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[17] Dive into Deep Learning 0.16.0 documentation. URL https://d2l.ai/chapter_convolutional-neural-networks/padding-and-strides.html 2020.

[18] Priyanshi Sharma. *MaxPool vs AvgPool.* URL https://iq.opengenus.org/maxpool-vs-avgpool/

[19] Deepmind. WaveNet: A Generative Model for Raw Audio. Sep 2016 URL https://deepmind.com/blog/article/wavenet-generative-model-raw-audio/

[20] Joseph Eddy. *Time Series Forecasting with Convolutional Neural Networks - a Look at WaveNet. Feb 2019* URL https://jeddy92.github.io/JEddy92.github.io/ts_seq2seq_conv

[21] Michael Phi. *Illustrated Guide to Recurrent Neural Networks - Towards Data Science. Medium, Sep 2019 ISSN 7958-0499* URL https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb809c9

[22] Christopher Olah. *Understanding LSTM Networks,* Aug 2015. URL https://colah.github.io/posts/2015-08-Understanding-LSTMs

[23] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, 2019. ISBN 9781492032595*

[24] Producto de Hadamard (matrices) - Hadamard product (matrices) - qaz.wiki, Jan 2021. URL https://es.qaz.wiki/wiki/Hadamard_product_(matrices)

[25] Juan Julian Cea Moran. *Redes Neuronales Recurrentes para la generacion automatica de musica.* URL http://oa.upm.es/63687/1/TFM_JUAN_JULIAN_CEA_MORAN.pdf

[26] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. *Learning phrase representations using rnn encoder-decoder for statistical machine translation.*

[27] Mirco Ravanelli, Philemon Brakel, Maurizio Omologo, and Yoshua Bengio. *gated recurrent units for speech recognition. IEEE Transactions on Emerging Topics in Computational Intelligence.* 2018.

[28] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. *Lstm: A search space odyssey. IEEE transactions on neural networks and learning systems.* 2016.

[29] José Francisco Núñez Castro. *Aprendizaje automático en fusión nuclear con Deep Learning.* 2017. URL http://opac.pucv.cl/pucv_txt/txt-3500/UCC3994_01.pdf

[30] Series Temporales. URL http://halweb.uc3m.es/esp/Personal/personas/jmmarin/esp/EDescrip/tema7.pdf

[31] Andrej Karpathy. *The Unreasonable Effectiveness of Recurrent Neural Networks.* Jun 2020 URL https://karpathy.github.io/2015/05/21/rnn-effectiveness/