

# Síntesis de texto usando Modelos de Lenguaje

Alumno: Alejo Torres

Profesor Guia: Maria Lorena Talame

Carrera: Ingeniería en Informática

Unidad Académica: Facultad de Ingeniería

Año: 2023

### **Abstract**

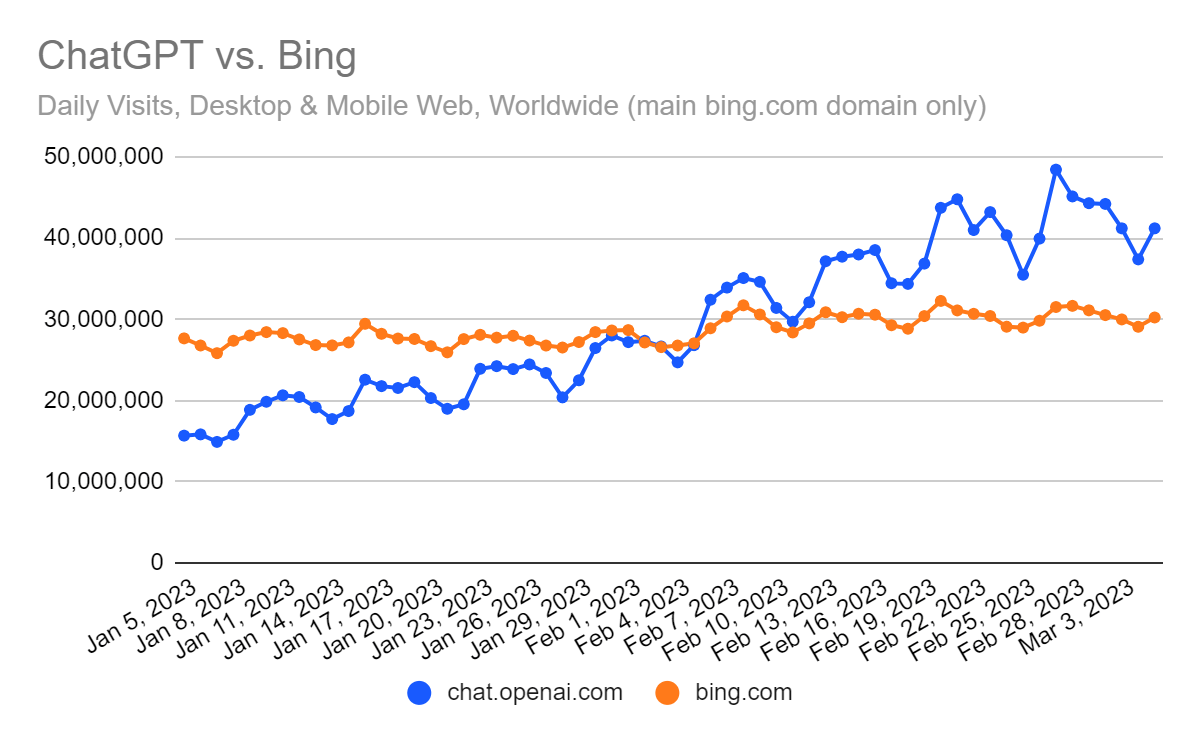
Los modelos de inteligencia artificial enfocados a la generación de contenido, también llamadas Inteligencias Artificiales generativas, sin duda alguna han revolucionado el paradigma de trabajo de sectores completos. La tecnología es un hecho y existen numerosos productos y servicios que integran su potencial. Los *Large Language Models (LLM)* que generan contenido del tipo *text-to-text*, son el motor que impulsa numerosas aplicaciones de tendencia actual. ChatBots, asistentes virtuales y motores de búsqueda personalizados son algunos ejemplos de estos. Sin embargo los *LLMs* comprenden una potente herramienta para el tratamiento de la información, al ser capaces de procesar lenguaje natural es posible usar estos modelos generalistas en tareas de extracción y síntesis de contenido. El propósito de este Proyecto de Grado es el de desarrollar una aplicación que permita resumir el contenido de un archivo en formato PDF, sin importar su tamaño, aprovechando el poder de inferencia de los *LLMs*. Presentando este servicio de inferencia a través de una interfaz intuitiva y amigable para un usuario final.

**Palabras claves:** *Machine Learning,Síntesis de texto, Large Language Models,Transformer*

### **Introducción**

#### **Descripción del Problema**

Inteligencia artificial, Modelos generativos, *ChatGPT*. Son palabras tendencia que están en la boca de todo el mundo hoy en día. Tanto en usuarios con formación técnica, como aquellos quienes aprovechan las ventajas de esta tecnología sin preocuparse realmente por su trasfondo. Podemos observar en el gráfico como en el mes de Febrero del año 2023 la página de *ChatGPT*(*ChatGPT*, s. f.), el asistente impulsado por *IA* más popular del momento,sobrepasa la cantidad de visitas diarias de Bing(*Bing.com*, s. f.), un motor de búsqueda muy popular.

*Ilustración I: visitas diarias en el periodo de Enero 2023 a Marzo 2023 de ChatGPT y Bing(ChatGPT Topped 1 Billion Visits in February, s. f.)*

Hemos visto el surgimiento de startups tecnológicas basadas completamente en esta esfera de conocimiento generando aún más especulación y entusiasmo por el alcance que pueden lograr los productos y servicios que integran exitosamente las virtudes de la inteligencia artificial. (*Top 10*, 2022)

Sin embargo, un área interesante para explotar es la del tratamiento de información de manera automática aprovechando las capacidades de procesamiento del lenguaje natural que brindan los Grandes Modelos de Lenguaje, que en adelante llamaremos *LLMs*. El poder condensar la información relevante es fundamental para un correcto proceso de estudio, sobre cualquier tema. En la era de la información en la que vivimos actualmente, donde contamos con acceso casi instantáneo a grandes volúmenes de información, la capacidad de síntesis se vuelve aún más importante. El problema surge cuando la cantidad de texto a procesar demanda más tiempo del que contamos para esa tarea.

#### **Motivación**

El interés por basar mi proyecto de grado alrededor de esta temática surge gracias a la oportunidad que me fue brindada por la *Universidad Católica de Salta* en conjunto con la *Universidad Politécnica de Madrid*. Donde pasé un semestre cursando las asignaturas de *Sistemas Inteligentes, Aprendizaje automático II* y *Minería de datos* de dicha universidad, las cuales fueron dictadas por los profesores *Alberto Diaz Alvarez* y el *Dr. Francisco Serradilla García*. Sus excelentes cátedras me sirvieron a modo de introducción y pude adquirir las bases necesarias para continuar mi aprendizaje de forma autónoma, donde decidí inclinarme por las IAs generativas, en específico aquellos modelos basados en la arquitectura *Transformer*.

Fue durante este proceso de investigación y de asimilación de conocimientos que se despertó en mí la curiosidad por el desarrollo de aplicaciones que utilicen estas tecnologías a su favor. Así fue como pude tomar noción del alcance real que se puede lograr si utilizamos estas técnicas para agilizar, y modernizar. los procesos de enseñanza y estudio. Mi intención es generar una potente herramienta de estudio que permita facilitar la asimilación de conocimiento, tanto para personas involucradas en tareas de investigación como estudiantes de cualquier nivel.

La comunidad *Open Source*, quién innumerables veces ha aportado en mis desarrollos personales es otro gran factor por el que he decido atacar este tema. Cuento con la intención de liberar todo el material resultante de este proyecto con la esperanza de poder aportar significativamente al proceso de formación de cualquiera que así lo desee.

***“Si hemos podido ver más lejos, es porque nos paramos sobre hombros de gigantes” - Isaac newton***

#### **Pasos a seguir**

El trabajo implica, en primera instancia, una Investigación y familiarización con las técnicas y tecnologías involucradas, como lo son la *Arquitectura* *Transformer,*la cual en pocas palabras puede definirse como una red neuronal que aprende contexto y, por lo tanto, significado mediante el seguimiento de relaciones en datos secuenciales como las palabras de esta oración (Merritt, 2022)*.* que da sustento a los *LLMs*. La revisión de librerías, proyectos y estudios relacionados con el Procesamiento del Lenguaje Natural,el cual abreviamos como (*NLP*), técnicas de *clustering* y *Machine Learning*. desglosando los conceptos hasta el punto que puedan ser comprendidos y utilizados para la construcción del software de síntesis de texto.

Lo anterior mencionado se complementará con el diseño de la arquitectura de los servicios de inferencia del sistema donde se integrarán todas estas herramientas para lograr el fin último del Software. Sobre este diseño se evaluará el rendimiento del mismo a través de la generación de resúmenes de temas específicos que luego serán evaluados por profesionales en la materia. Gracias a esto podremos perfeccionar las directivas y los parámetros de los modelos.

Teniendo en cuenta estos resultados se producirán las modificaciones pertinentes a la arquitectura. Acompañando del desarrollo de una interfaz web amigable e intuitiva con la que el usuario podrá interactuar con los servicios de inteligencia artificial mencionados anteriormente.

### **Estado de la cuestión**

#### **¿Qué es la inteligencia artificial?**

Podemos definir la inteligencia artificial,de manera general, como la capacidad de artilugios artificiales de realizar tareas propias de una inteligencia humana.(Derivando, 2021)

Cuestiones como la capacidad de cálculo y la memoria nos resultan atributos fáciles de asignar tanto a un computador como a una persona. Sin embargo cualidades como la capacidad de aprendizaje, la creatividad e incluso la autoconciencia son cualidades humanas sobre las que poco a poco la inteligencia artificial ha avanzado.

La Inteligencia Artificial,la cual a partir de ahora abreviaremos como *IA*  es un campo de la informática y la ciencia de la computación que se enfoca en desarrollar sistemas y programas capaces de realizar tareas que requieran inteligencia humana. En esencia, la *IA* busca crear máquinas que puedan emular y simular procesos de pensamiento, razonamiento y toma de decisiones propios de los seres humanos. (*Qué es la Inteligencia Artificial*, s. f.)

Es un campo multidisciplinario y a lo largo de las décadas, la *IA* ha evolucionado desde conceptos teóricos hasta aplicaciones prácticas que están transformando rápidamente diversas industrias.

En el núcleo de la *IA* se encuentran los algoritmos y modelos que permiten a las máquinas aprender de los datos y adaptarse para resolver problemas complejos. El aprendizaje automático (*Machine Learning*) es una subárea fundamental de la *IA* que implica el desarrollo de técnicas que permiten a las máquinas mejorar su rendimiento a medida que se exponen a más datos.

Podemos definir *Machine Learning* como un conjunto de técnicas mediante las cuales un algoritmo que debe realizar una tarea en específico puede modificar su comportamiento basándose en los datos que dispone o en una medida de que tan bien o mal ha realizado su tarea o en que tan bien o mal ha actuado según lo que le informe un tercero.Es decir pueden aprender de sus errores. Este es el concepto fundamental que da origen a todas las técnicas, algoritmos y arquitecturas. Respecto a cómo aprenden estos algoritmos existen tres enfoques fundamentales:(Gonzalez, 2020)

* *Aprendizaje supervisado*: útil cuando contamos con datos etiquetados, es decir, conocemos de antemano la solución inherente al conjunto de datos, por lo que podemos optimizar el algoritmo comparando su rendimiento con lo que debería resultar.
* *Aprendizaje no supervisado*: Sirven cuando no tenemos etiquetas en los datos, es decir no conocemos el resultado a priori para el conjunto de datos, están relacionados con tareas de agrupamiento,
* *Aprendizaje por refuerzo*: Actúa mucho por prueba y error.Generalmente utilizado para entrenar Agentes, usando las mediciones tomadas del entorno y su rendimiento para calcular un puntaje.

Una vez puestos en contexto, las redes neuronales son una de estas técnicas encerradas dentro de la esfera del machine learning, presumiblemente la más potente de todas.

Una concepcion erronea que se tiene sobre la redes de neuronas es que está completamente inspirada en el cerebro humano, cuando en realidad lo único en lo que se asemejan estas dos es en la forma como los nodos o Neuronas de una red se interconectan entre sí, como también lo hacen las Células de nuestro cerebro. Además podríamos asemejar la sinapsis que ocurre en el mismo con los cálculos que se producen dentro de cada uno de estos pequeños nodos. Aparte de eso, no tienen nada de similar.

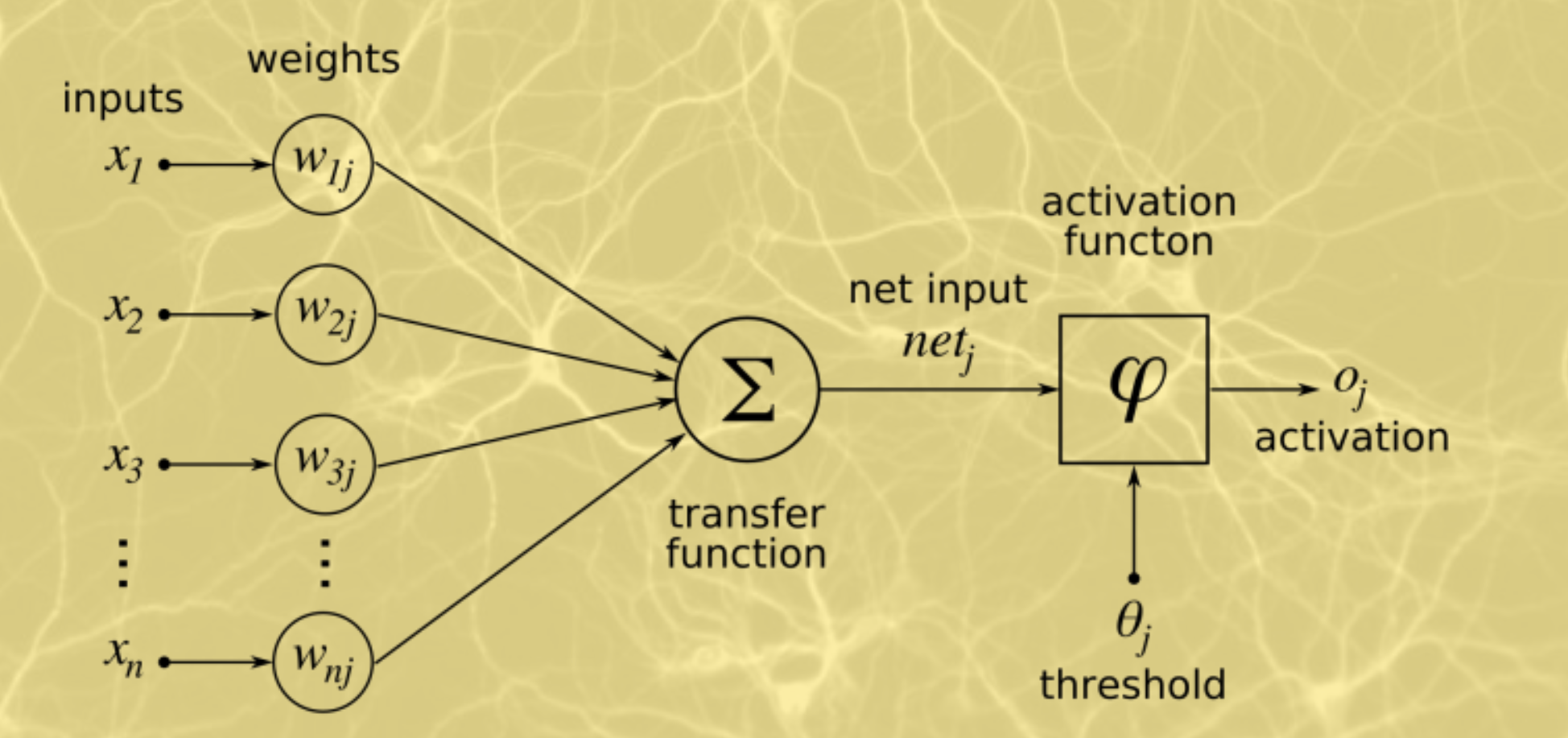
#### **¿Qué es una red neuronal?**

Una red neuronal es un sistema computacional que involucra Hardware y Software para tratar de imitar las capacidades computacionales de los sistemas biológicos usando elementos simples interconectados, llamados neuronas las cuales efectúan una serie de cálculos simples sobre los datos de entrada para luego comunicar el resultado a las demás neuronas relacionadas

Algunas ventajas que presentan frente a otras técnicas de machine learning son:

* Aprendizaje adaptativo: Esto quiere decir que aprende de un conjunto de experiencias, es decir los datos de entrada, sin necesidad de que un experto en la materia deba estar indicando cómo debe ser el proceso de aprendizaje
* Autoorganización: La propia red, a través de sus parámetros genera una interpretación de la información que recibe durante el entrenamiento
* Tolerancia a fallos: puede trabajar con información parcial,dañada o ruidosa.Si bien es cierto que un modelo será tan bueno como los datos sobre los cuales fue entrenado, esto no quiere decir que no tenga cierto nivel de tolerancia al ruido y a la información faltante
* Flexibilidad: pueden adaptarse a nuevos entornos sin ser entrenadas debido a la capacidad de aprendizaje
* Tiempo real: Debido a la naturaleza matricial de los cálculos en una red podemos aprovechar el poder de cómputo especializado de las unidades de procesamiento de gráficos junto con técnicas de loteo para poder usar máquinas paralelas alcanzando más velocidad y reduciendo los tiempos de entrenamiento e inferencia
* Generalización: una vez entrenada puede clasificar elementos desconocidos y también podremos determinar qué cosa la red todavía no ha aprendido a hacer.

Por muy extraño que parezca las redes de neuronas no son más que expresiones matemáticas.Tomamos las entradas de datos como *inputs* de la función, aplicamos lo pesos de la red y las desviaciones, y nuestra salida son las predicciones o inferencias de la red. Las redes de neuronas son una clase de función matemática. Podemos estudiarla a partir del siguiente diagrama. *(Notas de clases,Alberto Diaz Alvarez ,UPM)*



*Ilustración II: Representación formal de una neurona (Notas de clases,Alberto Diaz Alvarez ,UPM)*

Una explicación intuitiva sería la siguiente: Cada uno de los datos de entrada *Xi* sufrirán un escalado, dictado por un parámetro entrañable llamado peso *Wij*. Este parámetro indica la relevancia de esa entrada para la tarea en cuestión. Que un parámetro sea entrenable significa que el mismo será optimizado durante el proceso de entrenamiento. Luego a cada uno de estos resultados, vamos sumarlos para obtener el resultado parcial dentro de la red.Sin embargo lo que tenemos hasta ahora es una suma de términos escalados respecto a un valor arbitrario. Existe otro parámetro entrenable llamado desviación o *bias*, el cual constituye un término independiente permitiendo así una traslación en el eje vertical. Para comprender mejor esta idea pensemos en una serie de observaciones graficadas en un par de ejes cartesianos, nuestro trabajo es realizar un trabajo de regresión sobre ellos, ajustando una recta lo mejor que se pueda. Sin un término independiente no podríamos ajustar bien la recta para estos datos particulares, por lo que su función es agregar libertad al modelo.

Siguiendo el mismo concepto, sabemos que al sumar terminos lineales el resultado siempre será lineal, Sin embargo es muy complejo encontrar datos del mundo real que presenten una naturaleza lineal (Obviando fenómenos físicos puntuales), por lo que nos interesa hallar una forma de realizar regresión o clasificaciones con un modelo más libre,que permita ajustar una curva arbitraria y desprenderse de una simple regresión lineal. Que por si no quedó claro, es justo lo que sucede en cada una de las neuronas, una pequeña *regresión lineal[[1]](#footnote-0)* sobre el dato en cuestión. Para lograr esto se introduce el concepto de  *Función de Activación*.

##### **Función de Activación**

Una función de activación es un componente fundamental en las redes neuronales. Se utiliza para determinar si y en qué medida una neurona debe activarse (enviar una señal) en función de la suma ponderada de las entradas que recibe. Las funciones de activación introducen no linealidades en el modelo, lo que permite a las redes neuronales aprender relaciones complejas en los datos.

Como vimos las neuronas reciben entradas y las multiplican por sus correspondientes pesos,suman las desviaciones y luego se aplica una función de activación al resultado. Existen diversas funciones de activacion, pensadas para tareas especificas.La elección de la función de activación depende del problema y las características del modelo.Sin embargo, las más comunes y que luego se usarán luego para explicar el resto de temas son las siguientes:

* *Función Rectified Linear Unit (ReLU)*: La función ReLU es más popular en la actualidad debido a su eficiencia computacional y la prevención del problema de gradientes que afecta otras funciones. Simplemente devuelve cero si el valor de entrada es negativo y el mismo valor si es positivo. Su fórmula es:
* *Función Softmax*: Usada especialmente en la capa de salida de redes neuronales para convertir las salidas en probabilidades. Es útil en problemas de clasificación multiclase. Su fórmula es: donde *Zi* es la entrada de la función softmax para *i*
* *Función Sigmoide:* La función sigmoide tiene una forma de "S" y transforma cualquier valor en un rango entre 0 y 1. Fue ampliamente utilizada en redes neuronales debido a la facilidad de poder expresar la derivada en términos de la misma función pero está siendo reemplazada en muchas aplicaciones debido a problemas de *desvanecimiento* y *explotación* del gradiente del cual hablaremos más adelante. Su fórmula es:

Mediante la *no linealidad* que brinda este artilugio matemático podemos construir modelos que se ajuste a cualquier comportamiento que presentan los datos. Pero sigue quedando sin resolver algo crucial, la forma en la que estos sistemas aprenden. (Apuntes de clase, Sistemas Inteligentes,*Dr. Francisco Serradilla García* )

##### **La regla de la cadena**

Antes de hablar sobre el mecanismo utilizado para el aprendizaje dentro de las redes neuronales es necesario comentar el artilugio matemático que se aplica en este algoritmo. La *regla de la cadena* (también conocida como el teorema de las funciones compuestas) es una fórmula explícita de la derivada de una función compuesta por dos funciones derivables.Esta regla permite conocer la j-ésima derivada parcial de la i-ésima aplicación parcial de la composición de dos funciones de varias variables. (*Regla de la cadena - Wikipedia, la enciclopedia libre*, s. f.)

Un ejemplo conceptual sería el siguiente. Si un auto puede viajar el doble de rápido que una bicicleta y una bicicleta es cuatro veces mas rapida que un ser humano entonces un auto es 8 veces más rápido que un ser humano

Como sabemos una derivada es una razón de cambio, una proporción que indica cuánto cambia una variable en función de un cambio producido en otra. esta es la idea fundamental detrás del entrenamiento de una red neuronal, en definitiva queremos ver que tanto ha influido cada parámetro en el resultado obtenido para así ajustarlo adecuadamente

##### **Descenso del Gradiente**

La utilidad de la regla de la cadena en este campo viene acompañada por el algoritmo de descenso del gradiente (*DG*). Intrínsecamente, la inteligencia artificial tiene sus bases en un problema de optimización, El entrenamiento de una red de neuronas es un problema de este estilo pero en este caso no optimizamos costos ni tratamos de maximizar ganancias, sino que buscamos reducir un error el cual resulta de comparar nuestra inferencia y las etiquetas originales. (Notas de clase, Sistemas Inteligentes,*Dr. Francisco Serradilla García*)

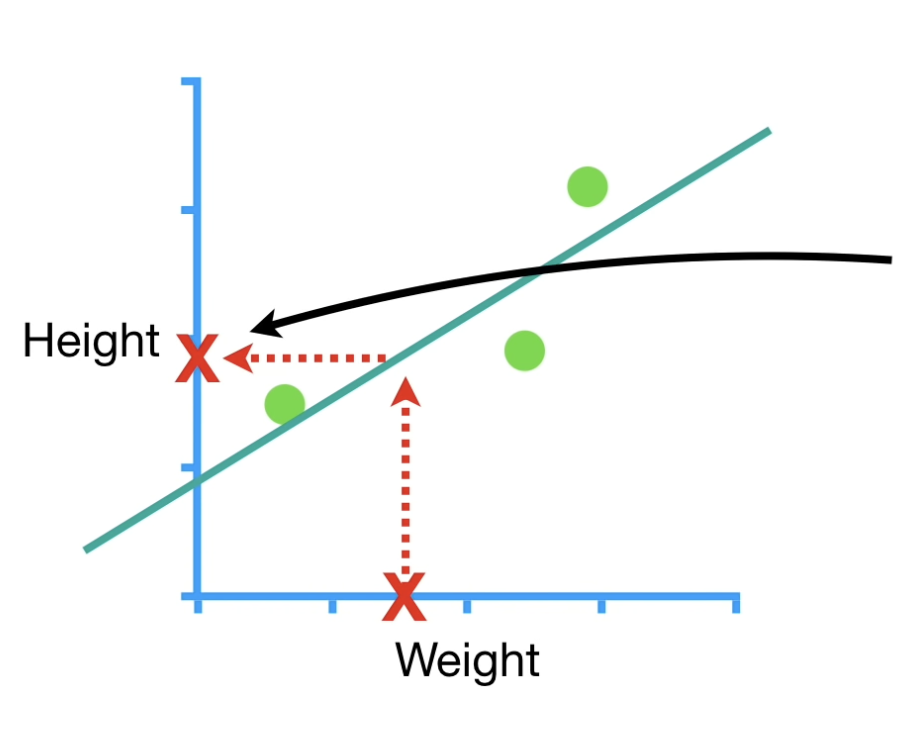
El error producido es el resultado de la diferencia entre lo que realmente es y lo que nuestro modelo infiere. En definitiva buscamos que esta función de pérdida o costo sea lo menor posible.

Las variables independientes que intervienen en esta función de error tendrán origen en todos los parámetros entrenables de nuestra arquitectura, los pesos, los cuales formarán matrices en cada capa y nuestras desviaciones o bias, términos independientes que se agrupan vectorialmente los cuales trasladan a la función.

Dicho esto, la pregunta ahora es de qué manera podemos optimizar estos parámetros, la más común, o en la que la mayoría de las técnicas se basan en el método del descenso del gradiente. aunque también pueden ajustarse los pesos de la red mediante técnicas que no utilicen *DG* como por ejemplo un *algoritmo de computación evolutiva*.donde se conviertan los pesos de la red en un cromosoma y viceversa para poder aplicar los operadores propios del algoritmo, produciendo así la nueva generación de cromosomas.Esta técnica es muy útil par el aprendizaje no supervisado. Si quiere saber más sobre el tema puede referirse al Notebook Jupyter del Anexo I

Para dejar en claro a qué nos referimos cuando hablamos de parámetros, hay que tener en cuenta que cuando realizamos regresiones lineales optimizamos los parámetros de pendiente e intersección, cuando hacemos regresiones logísticas optimizamos la curva, si usamos algoritmos de clustering optimizamos agrupación etc. Estos son algunos ejemplos de parámetros que pueden optimizarse en Machine Learning.

Pensemos en el siguiente ejemplo, con las siguientes observaciones. donde tratamos de aproximar una recta a los datos para tratar de predecir diferentes valores de peso



*Ilustración III: gráfico ejemplo de una recta que aproxima una serie de mediciones, StatQuest*

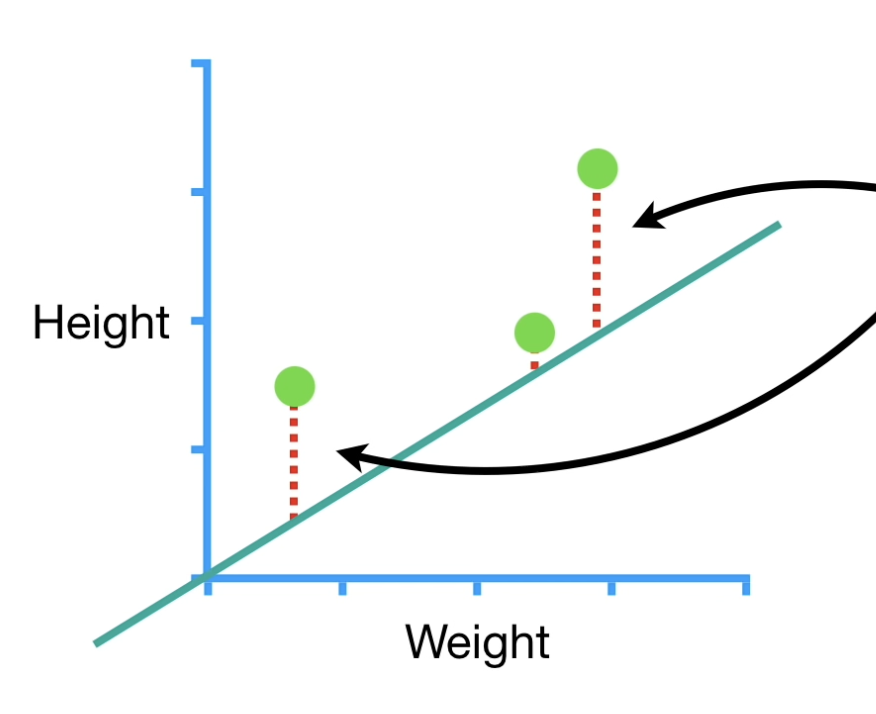
Donde la recta tiene la siguiente ecuación



Vamos a optimizar únicamente la pendiente de la recta, usando el método del descenso del gradiente y vamos a introducir un valor numérico a la intersección obtenido por el método de mínimos cuadrados. Mínimos cuadrados se refiere a un método utilizado en estadísticas y matemáticas para encontrar la mejor aproximación a través de una línea o curva que minimiza la suma de los cuadrados de las diferencias entre los valores observados y los valores predichos por la línea o curva. Este método es comúnmente utilizado en el análisis de regresión y se utiliza para ajustar modelos matemáticos a conjuntos de datos. (StatQuest with Josh Starmer, 2019)

Puede surgir la siguiente pregunta *¿porque no usamos el método de mínimos cuadrados para optimizar los parámetros en vez del descenso del gradiente?* El método de mínimos cuadrados resuelve este problema encontrando la solución más óptima o la solución perfecta, Lo que sucede es que este método se basa en resolver la derivada igualando la ecuación a 0, para buscar los puntos mínimos. Esto vuelve al metodo de minimos cuadrados poco flexible, el método del descenso del gradiente estima muy bien esta solución perfecta sin la necesidad de este paso limitante, volviéndolo mucho más flexible en escenarios donde no se puede resolver dicha ecuación de manera analitica,compo veremos a continuacion el metodo del descenso del gradiente es un método numérico e iterativo

Nuestro Error en este caso surgirá de la aproximación generada por la recta en el punto en cuestión y la medición tomada que forma parte del conjunto de datos



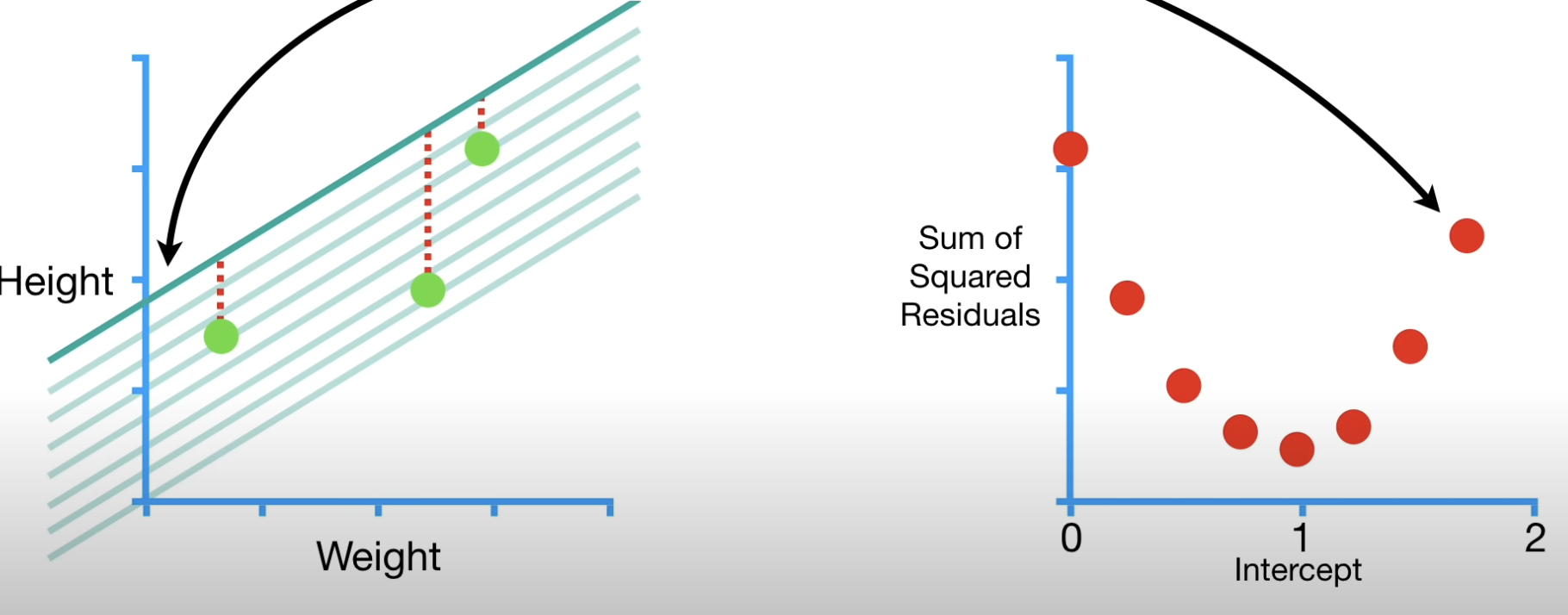
*Ilustración IV: gráfico que indica el error entre la aproximación y las mediciones Fuente: StatQuest*

Es decir el error será la diferencia entre lo observado y lo predicho, para obtener la prediccion.Si luego a este error le tomamos su cuadrado obtenemos el *error cuadrático.*

Algunas de las razones por las que se opta por utilizar el cuadrado del error son:

* *Propiedades Matemáticas:* El error cuadrático produce una función de pérdida que es continua, diferenciable y convexa en la mayoría de los casos, lo que es esencial para el proceso de optimización usando el descenso de gradiente, que requiere de las derivadas como veremos más adelante.
* *Sensibilidad a Diferencias Grandes:* El error cuadrático penaliza de manera más significativa las diferencias grandes entre las predicciones y los valores reales. Esto puede ser útil cuando se desea que el modelo sea más sensible a los casos en los que se cometen errores significativos.
* *Solución Analítica:* En algunos casos, el error cuadrático puede tener soluciones analíticas para los valores óptimos de los parámetros del modelo. Esto facilita el cálculo de los valores que minimizan el error cuadrático.
* *Derivadas Claras*: Las derivadas del error cuadrático en relación con los parámetros del modelo son simples y pueden calcularse directamente. Esto facilita el cálculo de los gradientes necesarios para los métodos de optimización. (Frost, 2021)

Siguiendo el ejemplo, dijimos que íbamos a optimizar el parámetro de la intersección de la recta, si repetimos el proceso de calcular los errores de cada recta para un valor de intersección distinto obtendremos los siguientes gráficos donde vemos que desplazando la intersección de la recta produce un valor distinto para el error. Gráficamente podemos ver en el gráfico de puntos donde se mide el error respecto de la interacción que existe un mínimo valor. Vamos a ver como aplicando el algoritmo de descenso del gradiente podemos encontrarlo



*Ilustración V: comparación de la gráfica de Error contra la grafica de la recta de aproximación (StatQuest)*

Lo que estamos buscando es:

Es decir buscamos saber cuánto influye la intersección en el error de nuestro modelo, que en este caso es la recta. Por esto sabemos que *predictedValue* está dado por la ecuación de la recta Por lo que es posible aplicar la regla de la cadena y obtener la derivada respecto del parámetro, para luego sustituir las variables necesarias por las mediciones y así obtener un número que representa el error para ese modelo en concreto.

Esta información nos indica la pendiente de la curva de error, como nuestro objetivo es minimizarlo nos interesa recorrerla de manera descendente, además como buscamos aproximar esta derivad a 0 , mientras mas lejos estemos del 0 tomaremos incrementos más grandes en dirección descendente, mientras más cerca estemos del 0 nuestros pasos se volverán más pequeños,esto lo logramos con otro parámetro intrínseco del algoritmo llamado factor de aprendizaje o *LearningRate.* Gracias a la derivada que nos brindara la pendiente de la curva, sabiendo que cuando la misma es 0 (o muy aproximado) nos encontraremos en un punto de inflexión, en este caso uno mínimo. Este se trata de un mínimo local, ya que en el problema de optimización de parámetros de una red neuronal no existen los mínimos absolutos. Todo mínimo que encontremos será local esto está relacionado con la topología compleja y no lineal del espacio de búsqueda de parámetros en el que se encuentran inmersas las redes neuronales. (*Understanding Local Minima in Neural-Network Training - Technical Articles*, s. f.)

Dado que las redes neuronales suelen tener múltiples capas y una gran cantidad de parámetros, el espacio de búsqueda es extremadamente complejo y suele contener muchas curvas y valles. Esto da lugar a la presencia de mínimos locales, que son puntos donde la función de pérdida es menor que en sus alrededores inmediatos, pero no necesariamente el valor más bajo en todo el espacio.

El término "mínimo absoluto" se refiere al punto de la función de error que tiene el valor más bajo en todo el espacio. Debido a la complejidad y no linealidad del espacio de búsqueda en las redes neuronales, es difícil asegurar que el proceso de descenso del gradiente convergerá exactamente al mínimo absoluto. En cambio, es más común que el descenso del gradiente pueda quedar atrapado en mínimos locales, donde el gradiente se vuelve cero en esa ubicación, pero que no son los valores más bajos en comparación con otros mínimos locales o regiones más bajas en la función de pérdida. (*Understanding Local Minima in Neural-Network Training - Technical Articles*, s. f.)

Sin embargo, es importante destacar que aunque los mínimos locales son un desafío en el proceso de optimización, en la práctica, las redes neuronales a menudo logran un rendimiento aceptable mediante la exploración y la búsqueda de soluciones en el espacio de parámetros. Además, hay técnicas avanzadas, como el uso de tasas de aprendizaje adaptativas, la inicializaciones adecuadas de los pesos y la exploración de arquitecturas de redes diferentes, que pueden ayudar a mitigar el problema de los mínimos locales en el entrenamiento.

A modo de resumen, podemos decir que el algoritmo del descenso del gradiente está compuesto por los siguientes pasos:

1. Calcular la derivada de la función de error para cada parámetro involucrado en ella, Cuando hablamos de un número arbitrario de parámetros decimos que tomamos el gradiente de la función
2. inicializamos los parámetros con valores aleatorios
3. utilizar estos valores en el gradiente calculado en el paso 1
4. calcular el tamaño del paso
5. Calcular el nuevo valor del parámetro

Las condiciones de cierre de este algoritmo, en teoría son 2: Puede suceder que el que el tamaño de paso sea muy cercano a 0, o que hayamos alcanzado un número fijo de iteraciones (StatQuest with Josh Starmer, 2019)

En la práctica puede detenerse el entrenamiento luego de una serie de iteraciones(*epochs*) o cuando alcancemos un valor de error aceptable o luego de un tiempo fijo, de manera manual.

##### **Descenso del Gradiente estocástico**

Como pudimos observar el descenso del gradiente es un método que requiere de muchos cálculos Por ejemplo,si tuviéramos un datasets con 23000 atributos que representan genes con el objetivo de predecir una enfermedad y queremos realizar una regresión logística sobre esos datos con un total de 1000000 muestras, deberíamos calcular 1000000 de términos por cada una de las 23000 derivadas correspondientes a cada atributo del conjunto.Es decir 23000000 terminos por cada paso. Y si en el algoritmo se pueden producir un total de 1000 pasos entonces hay que calcular 23000000000 términos.Para grandes volúmenes de datos el descenso del gradiente convencional es muy lento

El término "estocástico" se refiere a la naturaleza aleatoria del proceso de optimización. En el descenso estocástico del gradiente, en lugar de calcular el gradiente completo de la función de pérdida en cada paso utilizando todo el conjunto de datos de entrenamiento, se utiliza solo una muestra aleatoria (mini-lote o batch) de los datos en cada iteración. (Notas de Clase, Aprendizaje Automatico II)

El proceso estocástico implica una selección al azar de un subconjunto pequeño de datos para calcular el gradiente en lugar de utilizar todo el conjunto de datos. Esto introduce un elemento de aleatoriedad en el proceso de optimización. A medida que el *SGD* actualiza los pesos de la red neuronal en función de cada mini-lote, la dirección del gradiente puede variar considerablemente entre los mini-lotes debido a la variabilidad de los datos seleccionados. Esto puede hacer que el proceso de optimización sea menos predecible pero también más rápido en términos de cálculo, ya que se evita el costo computacional de calcular el gradiente completo para todo el conjunto de datos. (Mehta, 2020)

Este enfoque estocástico puede ayudar a salir de mínimos locales y alcanzar regiones más amplias del espacio de parámetros, lo que puede ser beneficioso en algunos casos. Sin embargo, también puede hacer que la convergencia hacia el mínimo global sea menos suave debido a la naturaleza aleatoria de las actualizaciones. Para abordar esto, a menudo se utilizan técnicas de control de la tasa de aprendizaje, programación de tasas de aprendizaje adaptativas (*schedulers*) y otros ajustes para equilibrar la variabilidad estocástica con la convergencia estable.Este método es especialmente útil cuando tenemos redundancia de datos. (Apuntes y Notas de clase, Sistemas I)

##### **El algoritmo de retropropagación**

Sabemos que la unidad básica de las redes neuronales es el *perceptrón* y que la unión de varias de ellas forman una red que permite resolver problemas de clasificación y regresión mayores. Generalmente, una red neuronal está formada por capas de varias neuronas. La primera, la capa de entrada, recibe los datos. Por ejemplo los síntomas de una posible enfermedad.Luego, una serie de capas, llamadas *capas ocultas*, procesarán los datos de entrada y una última, la *capa de salida*, devolverá los datos de salida, en el caso del ejemplo planteado si el paciente tiene la enfermedad o no.

El deber del creador de una red neuronal es encontrar cuántas capas y qué cantidad de neuronas serán necesarias para que un algoritmo cambie los pesos y pueda transformar la entrada que haya recibido en la salida que esperamos.

Existen muchos algoritmos que cumplen esta tarea, pero el más popular es *Backpropagation* siendo este uno de los más usados y fáciles de entender. Hay muchas formas de medir el error. En el ejemplo anterior usamos el cuadrado de los errores. En específico Backpropagation busca minimizar el error cuadrático medio producido por la capa de salida (Andrej Karpathy, 2022)

Donde J es el número de neuronas de salida

Por otra parte, el algoritmo necesita el conjunto de datos que la red quiera aprender, es decir, las entradas y las salidas, a las que llamaremos,*conjunto de entrenamiento*. El conjunto de entrenamiento consiste en pares de datos de entrada y salida. No está demás decir que la cantidad de neuronas de la capa de entrada y de salida, deberá coincidir con el conjunto de números.En el siguiente diagrama podemos ver una arquitectura del perceptrón multicapa con 2 neuronas en la capa de entrada, 2 capas de 2 y 3 neuronas respectivamente y otra neurona en la salida

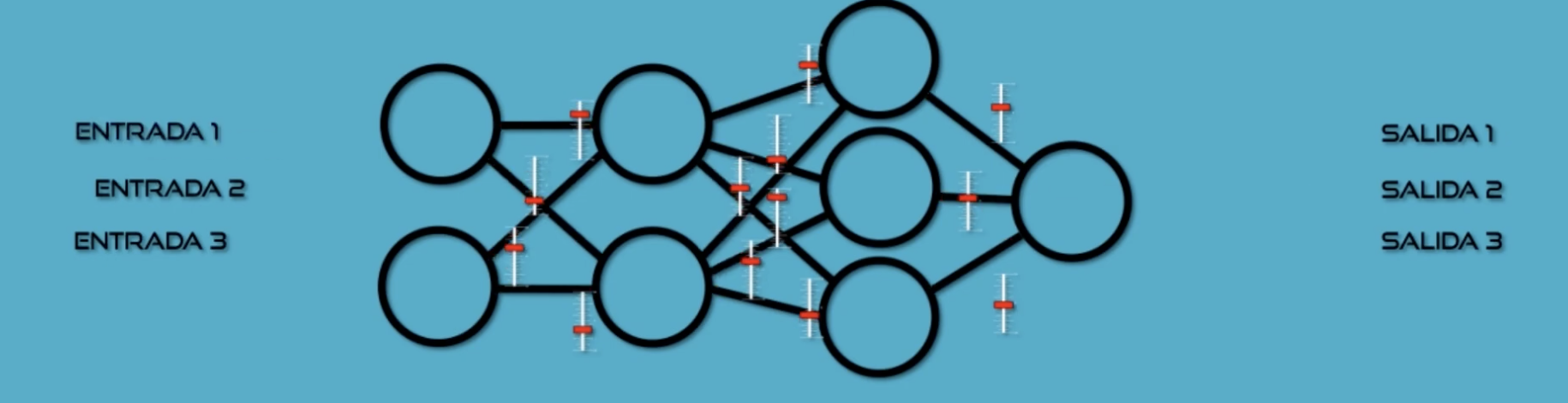


Ilustración VI: esquema de un perceptrón multicapa, Fuente: (BitBoss)

Los pasos de este algoritmo son los siguientes:(BitBoss, 2020)

1. **Inicialización de pesos:** En el inicio, todos los pesos de las conexiones entre las neuronas se establecen aleatoriamente o usando alguna técnica de inicialización. Son estos los que determinan la influencia que tiene cada conexión en la salida de una neurona.
2. **Paso hacia adelante (Forward Pass):** Cuando se presenta una entrada a la red, esta se pasa a través de la red capa por capa. Cada neurona toma una combinación ponderada de las salidas de las neuronas en la capa anterior, las suma y pasa el resultado a través de una función de activación. Esto finalmente produce una predicción en la salida de la última capa
3. **Cálculo del error:** Comparar la predicción realizada por la red con el valor real que debería ser. Esto es lo que llamamos error o "pérdida" ya que mide cuán incorrecta es la predicción de la red para esa entrada en particular.
4. **Paso hacia atrás (Backward Pass):** Aquí es donde comienza el algoritmo de backpropagation propiamente dicho. El objetivo es entender cómo los pesos de las conexiones entre las neuronas deben ajustarse para reducir el error. Para hacerlo, calculamos la contribución de cada peso al error total.
5. **Cálculo de gradientes:** El algoritmo calcula la derivada parcial del error con respecto a cada peso. Esto indica cómo cambiar cada peso para disminuir el error. Utiliza la regla de la cadena para propagar el error desde la capa de salida hasta las capas anteriores.
6. **Actualización de parámetros:** Una vez que se calculan las derivadas, podemos ajustar los parámetros de la red. La idea es mover los parámetros en la dirección que reduzca el error. Esto se hace mediante un proceso de optimización como el visto en el descenso de gradiente. Los pesos se ajustan proporcionalmente a las derivadas calculadas, utilizando una tasa de aprendizaje, que puede o no variar. Esto también se conoce como la *regla Delta*
7. **Iteración:** Repites los pasos 2 al 6 para muchas instancias del conjunto de datos de entrenamiento, una y otra vez, para que la red se ajuste gradualmente a los datos. Cada iteración mejora la red haciendo que los pesos se ajusten en la dirección que minimiza el error total.

Este proceso de ajustar los pesos en función del error se repite iterativamente hasta que la red neuronal pueda hacer predicciones con un error aceptablemente bajo en nuevos datos que no ha visto durante el entrenamiento. En resumen, el algoritmo de backpropagation es la base matemática que permite a las redes neuronales aprender y mejorar su rendimiento a través de la optimización de sus conexiones ponderadas.

##### **A tener en cuenta**

Tanto el algoritmo de retropropagación (*Backpropagation*) como el algoritmo de *descenso del gradiente* son conceptos fundamentales en el campo del aprendizaje profundo y el aprendizaje automático en general. Aunque están relacionados y a menudo se usan juntos en el entrenamiento de redes neuronales, se refieren a aspectos diferentes del proceso de optimización.

El descenso del gradiente es un método de optimización que busca encontrar el valor mínimo de una función mediante la iteración de ajustar los parámetros en la dirección opuesta al gradiente de la función en un punto dado. Recordemos que el gradiente es el vector que apunta en la dirección de mayor aumento de la función, por lo que moverse en la dirección opuesta al gradiente con pasos pequeños lleva al valor mínimo de la función.El algoritmo de descenso del gradiente se utiliza para ajustar los parámetros de un modelo de manera que se minimice la función de pérdida, la cual mide qué tan bien se ajusta el modelo a los datos. Por último, la tasa de aprendizaje controla el tamaño de los pasos tomados en la dirección del gradiente.

Cuando hablamos de retropropagación nos referimos al proceso específico utilizado en el entrenamiento de redes neuronales para calcular los gradientes de la función de pérdida con respecto a los pesos de las conexiones de la red. Implica el cálculo eficiente de las derivadas parciales de la función de pérdida en función de los pesos y las entradas de la red.La retropropagación se basa en la regla de la cadena del cálculo diferencial, que permite descomponer el gradiente de la función de pérdida en términos de gradientes más pequeños de las capas de la red. Esto permite calcular los gradientes para ajustar los pesos de cada capa de la red utilizando el algoritmo de descenso del gradiente.

La retropropagación es el proceso que permite calcular los gradientes necesarios para el descenso del gradiente en el contexto de redes neuronales. La retropropagación calcula cómo cada peso de la red contribuye al error final y cómo esos errores se propagan hacia atrás en la red. Una vez calculados estos gradientes, el descenso del gradiente se aplica para ajustar los pesos y minimizar la función de pérdida

El algoritmo de retropropagación es una técnica específica dentro del entrenamiento de redes neuronales que calcula gradientes, mientras que el algoritmo de descenso del gradiente es un enfoque general de optimización utilizado en diversos problemas de aprendizaje automático, incluyendo el entrenamiento de redes neuronales. (Notas de Clase, Sistemas Inteligentes)

Para revisar una implementación ad-hoc de un perceptrón multicapa y los algoritmos mencionados en esta sección, puede visitar los *NoteBook Jupyter del Anexo I*. Si desea conocer más en profundidad el algoritmo de Backpropagation puede revisar el Notebook Jupyter del Anexo II donde se implementa un motor Autograd, es decir una implementación de este algoritmo, junto con un una estructura DAG para representar una red neuronal

sencilla.

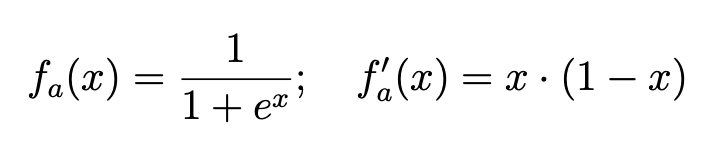
##### **Un ejemplo práctico**

Supongamos que tenemos una red de L capas donde sus neuronas tienen una función de activación sigmoidal. La función sigmoidal es otro tipo de función de activación como las vistas anteriormente, es muy sencilla debido a que su derivada puede expresarse en términos de la propia función, aunque en la práctica no es tan utilizada ya que aporta al problema de *explosion* del gradiente y/o *desvanecimiento* del gradiente.

El desvanecimiento del gradiente ocurre cuando los gradientes (derivadas parciales) de la función de error con respecto a los parámetros de la red disminuyen drásticamente a medida que se retro propagan hacia las capas anteriores. En otras palabras, a medida que los gradientes se propagan hacia las capas iniciales de la red, su magnitud disminuye exponencialmente. Esto puede resultar en que las capas iniciales no se actualicen de manera efectiva y, en última instancia, en un entrenamiento lento y problemas de *convergencia,* es decir poder llegar a un valor que minimice la función de pérdida.

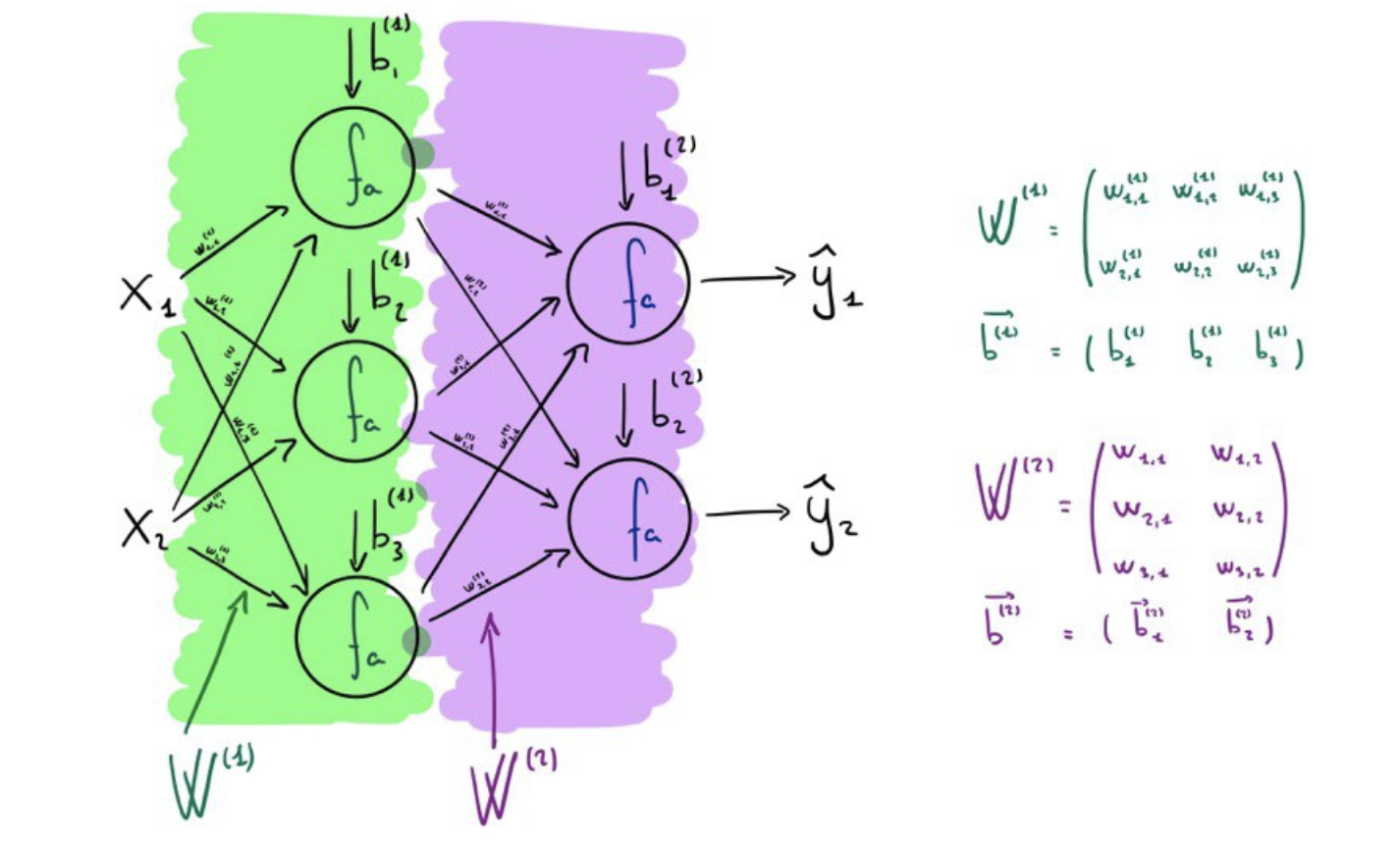
La explosión del gradiente es el fenómeno opuesto. Ocurre cuando los gradientes aumentan exponencialmente a medida que se retro propagan hacia las capas anteriores. Esto puede llevar a actualizaciones de parámetros extremadamente grandes, lo que resulta en inestabilidad numérica y problemas de convergencia en el proceso de optimización.

Ambos problemas pueden causar que el modelo sea difícil de entrenar y puede llevar a resultados inesperados o a una convergencia lenta, lo que dificulta la obtención de un modelo óptimo.

Aclarado esto se describe abajo junto con su derivada, la fórmula de la función sigmoide:

En el siguiente diagrama vemos un perceptrón multicapa que cuenta con 2 capas ocultas. Si bien a las entradas y salidas también se las llaman capas, técnicamente, las capas son aquellas que exclusivamente están compuestas por neuronas artificiales. Podemos ver que cada capa está representada por un color, al costado se encuentran la matriz de pesos y el vector de desviaciones correspondiente a cada capa. Estos son los parámetros que optimizaremos más adelante.

Agrupamos de manera matricial los parámetros debido a que resulta más como su representación matemática. Además en la práctica, los cálculos vectoriales permiten aprovechar técnicas de *batching* o loteo para paralelizar y agilizar el cómputo. Cada Wij se lee como el peso de la conexión entre la entrada *I* con respecto de la neurona *J*. Las desviaciones corresponden una a cada neurona.

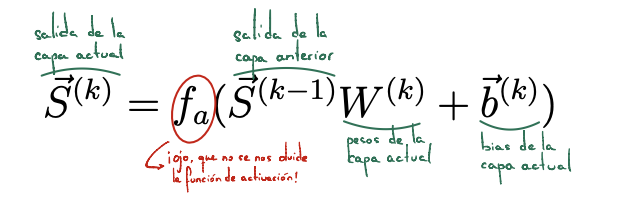


*Ilustración VII: perceptron multicapa representación matricial de sus parámetros, Fuente: Entrenamiento de un MLP, Alberto Alvarez Diaz,UPM*

También vamos a suponer que tenemos un ejemplo *E* con el que vamos a entrenar. Este ejemplo *E* estará compuesto por unos datos de entrada *X* y unos datos esperados de salida Y.Por último, estableceremos un parámetro α denominado factor de aprendizaje o tasa de aprendizaje que servirá como factor de ajuste en la velocidad de entrenamiento.

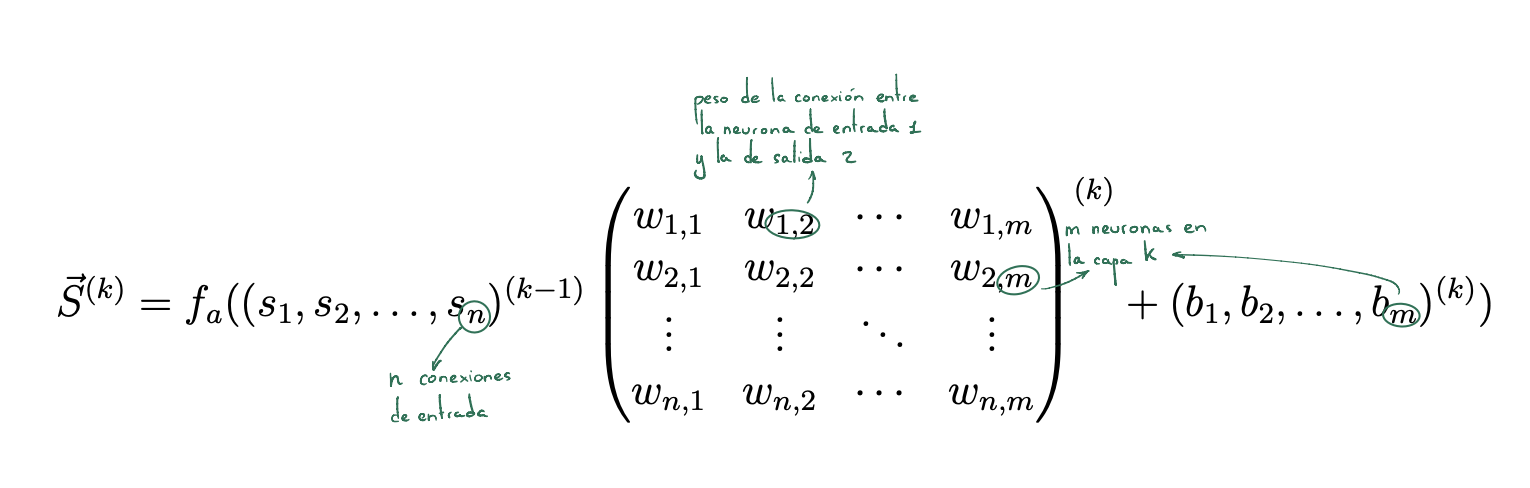
###### **Propagación**

Como ya vimos, la propagación o *inferencia* consiste en obtener un resultado de la red a partir de unos datos de entrada. Durante el entrenamiento nosotros le iremos dando los datos de entrada del conjunto de entrenamiento, en nuestro caso X.Como un perceptrón multicapa está compuesto de varias capas, el proceso será ir aplicando la siguiente ecuacion a cada una de las capas:



*Ilustración VIII:Ecuación que modela la propagación de los datos en un MLP, Fuente: Entrenamiento de un MLP, Alberto Alvarez Diaz,UPM*

Si la desarrollamos usando la matriz de pesos y vector de desviaciones anteriores nos quedaría:



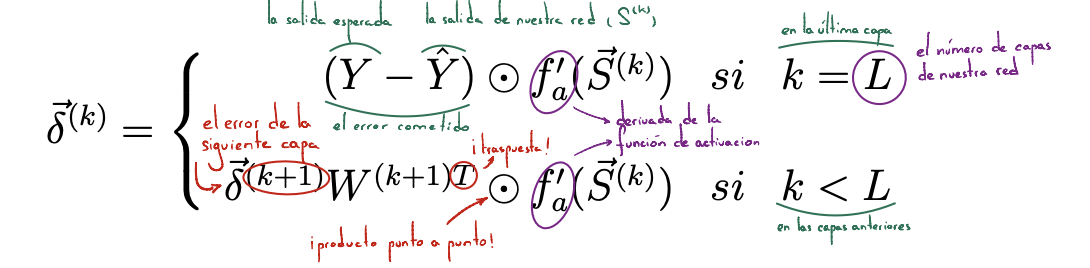
*Ilustración IX:Ecuación que modela la propagación de los datos en un MLP, en su forma desplegada, Fuente: Entrenamiento de un MLP, Alberto Alvarez Diaz,UPM*

Como se puede ver, el orden de aplicación es de la primera a la última capa, ya que para calcular la salida de una capa se usa como entrada la salida de la capa anterior, así hasta llegar a S(0) que sería equivalente a nuestro vector de entrada X. Asimismo, al valor de salida de la última capa S(L) le denominaremos Yˆ.

###### **Retropropagación**

Consiste en propagar desde la última capa hasta la primera el error existente entre el valor Yˆ que devuelve la red para la entrada X y el valor de salida esperado Y .

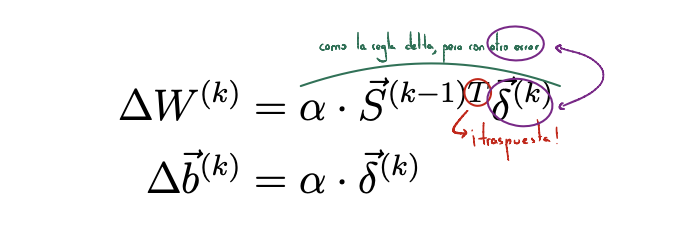
Lo primero que hay que hacer es calcular el error cometido tanto en cada una de las capas para actualizar sus correspondientes pesos. La clave aquí es que hay que calcular primero el error de la última capa, y luego los anteriores, usando el error de su correspondiente capa siguiente. Se denota como ⃗δ(k) y se define como se indica a continuación:



*Ilustración X:Ecuación que modela la Retropropagación de los datos en un MLP, Fuente: Entrenamiento de un MLP, Alberto Alvarez Diaz,UPM*

Donde se define un comportamiento especial únicamente en caso de tratarse de la última capa, donde podemos calcular el error mediante comparar la salida con el valor observado, teniendo en cuenta revertir (derivar) el impacto de la función de activación en la salida. Nótese que, como estamos trabajando con estructuras vectoriales, realizamos un producto punto a punto, o componente a componente.

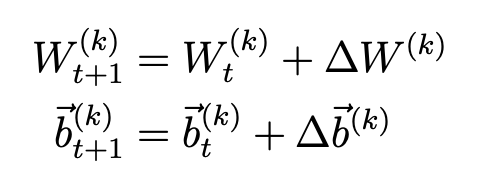
En el resto de capas (capas anteriores) podemos ver cómo se calcula el error teniendo un comportamiento recursivo en la definición de la fórmula, es decir tomando el error de la capa siguiente.Después, una vez calculados los errores de todas las capas, pasamos a calcular las matrices de cambio, esto es, las matrices que contendrán los valores con los que se van a modificar las matrices de pesos de cada capa.Este es un paso intermedio que facilita los cálculos. Estas se calcularán de la siguiente manera:



*Ilustración XI:Ecuación que la matriz de cambios de los parámetros, Fuente: Entrenamiento de un MLP, Alberto Alvarez Diaz,UPM.*

Esto sirve para agrupar matricialmente cada uno de los pequeños pasos que debe

dar cada uno de los parámetros en dirección opuesta al gradiente,finalmente, sumamos estas matrices de cambio a las matrices de parámetros originales.



*Ilustración XII:Suma de los cambios a los parámetros originales, Fuente: Entrenamiento de un MLP, Alberto Alvarez Diaz,UPM*

Si desea revisar el ejemplo con datos numéricos para los parámetros y cada uno de los cálculos, puede revisar el Anexo III

#### **¿Qué es Deep Learning ?**

Es importante aclarar que cuando las redes de neuronas tienen más de una capa se las conoce como redes profundas que dan lugar a toda una rama del machine learning llamada Deep learning, Hoy en día, hemos evolucionado al punto de pensar las arquitecturas de estos sistemas en capas y no en neuronas individuales, por lo que es su gran medida estamos trabajando con varias de ellas, entrando en la categoría de Deep Learning. Las IAs generativas y en específico los LLMs entran en esta categoría.(«Introduction to Deep Learning», 2018)

Al trabajar con Deep Learning algunos inconvenientes son:

* *Entrenar se vuelve un proceso arduo* y que requiere de mucha prueba y error al no existir una única forma de hacerlo, existen diversos enfoques que pueden encajar mejor o peor en el problema a trabajar y un conjunto de buenas prácticas pero no existe un “libro dorado” que de la solución exacta.
* *La elección de la arquitectura* se basa plenamente en prueba y error, depende netamente del problema a resolver y la cantidad de datos.
* *Perdemos la explicabilidad* del modelo,es decir no sabemos con certeza cómo toma las decisiones y en base a que lo hace, es este el mayor impedimento a la hora de utilizar el aprendizaje profundo y las redes de neuronas en campos como la medicina, justicia y seguridad por ejemplo, donde técnicas como *Random Forest* continúan siendo ampliamente usadas debido a que estos modelos cuentan con buenas características de explicabilidad.
* *El volumen de datos necesario* aumenta en gran medida, para entrenar las arquitecturas de deep learning es necesario un número considerable de registros, mientras más profunda la red mayor será la cantidad de datos necesarios

(Apuntes de Clase)

#### **Las IAs Generativas**

La Inteligencia Artificial Generativa (*IAG*) es una rama de la inteligencia artificial que se enfoca en la generación de contenido original a partir de datos existentes. Nombradas por *Yann LeCun en 2014* como *“La idea más interesante en Machine Learning en los últimos diez años”.* Estas técnicas son capaces de generar imágenes, música, texto y demás contenido con una potente componente creativa en ellos, que simulan ser creados por humanos.Se basan en modelos y algoritmos que pueden aprender a capturar y modelar la distribución subyacente de un conjunto de datos de entrenamiento para luego generar instancias nuevas y auténticas que se asemejen a esos datos.

La forma más popular de construir y entrenar una Inteligencia artificial de generación de contenido es a través de las Redes Neuronales Generativas Adversariales (*GAN*). (*¿Qué es la Inteligencia Artificial Generativa?*, 2023)

Una GAN consiste en dos redes neuronales: el *generador* y el *discriminador*. El generador toma ruido aleatorio como entrada y genera datos sintéticos, mientras que el discriminador toma datos reales y datos generados por el generador y trata de distinguir entre ellos. Ambas redes son entrenadas en un proceso competitivo: el generador busca generar datos que engañen al discriminador, mientras que el discriminador intenta mejorar en su capacidad de discernir entre datos reales y generados. Antes de poder introducir una explicación de esta arquitectura, debemos conocer el concepto de *espacios latentes*, la forma de crearlos y sus variantes.

##### **Autoencoders**

Un autoencoder es una arquitectura de red neuronal que pertenece a la categoría de modelos no supervisados. Su objetivo principal es la extracción y compresión de características, así como la posterior reconstrucción de los datos originales a partir de estas representaciones latentes, todo ello en un proceso de *entrenamiento auto-supervisado*.

Son utilizados como una técnica de *reducción de la dimensionalidad* y compresión de datos.

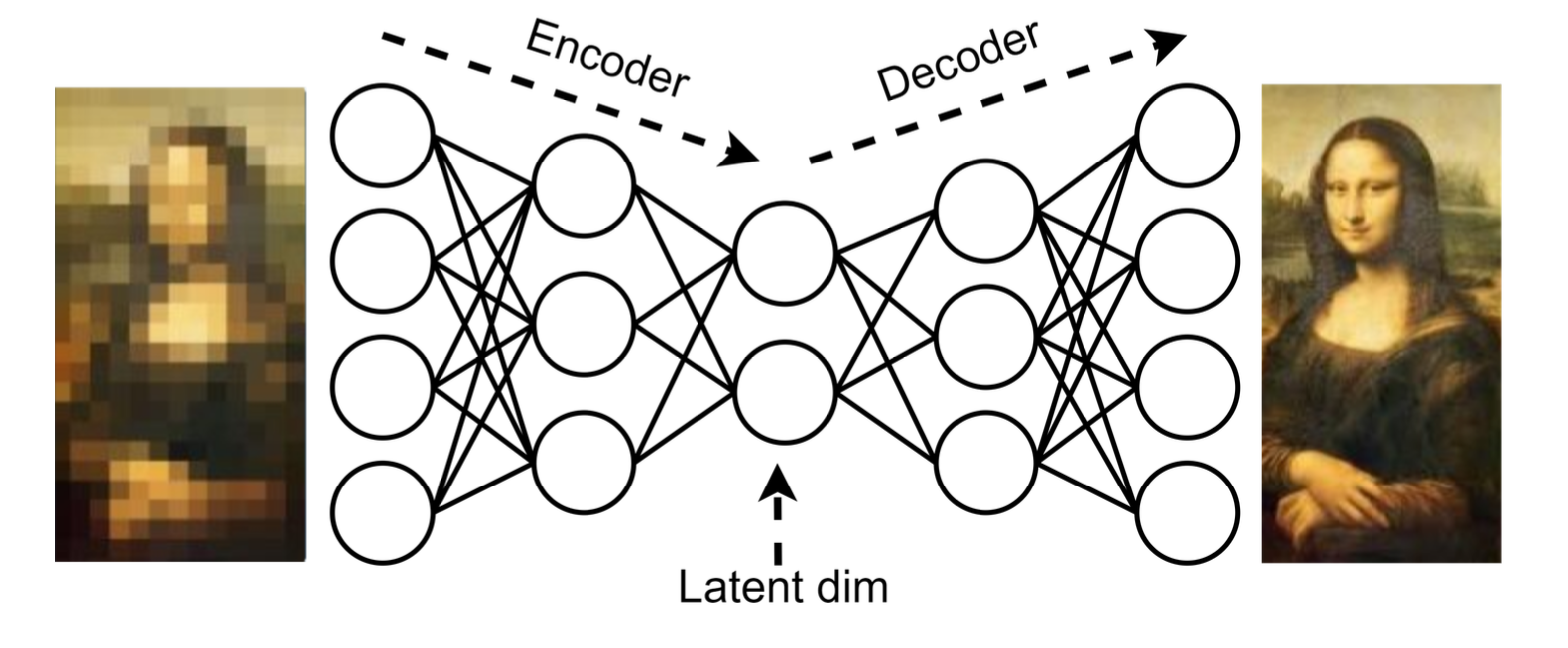
A nivel técnico, un autoencoder consta de dos componentes principales: el codificador (*encoder*) y el decodificador (*decoder*). Estos dos componentes trabajan juntos para aprender una representación más compacta de los datos de entrada sin etiquetas, permitiendo que el modelo capture las características más significativas y subyacentes de los datos.

Podemos resumir el comportamiento de un autoencoder en un alto nivel de la siguiente forma: Primero,el codificador toma los datos de entrada y los transforma en un espacio de características de menor dimensión, conocido como *espacio latente*. Este proceso de codificación implica aplicar transformaciones lineales y no lineales a los datos originales para capturar sus patrones más relevantes. La reducción de la dimensión en el espacio latente implica una compresión de los datos, cabe mencionar que se trata de una compresión con pérdida de información, Sin embargo se entiende que el espacio latente es capaz de capturar los atributos más relevantes de la información.El decodificador, por su parte, toma la representación en el espacio latente generada por el codificador y la reconstruye de nuevo en el espacio original de alta dimensión. Este proceso implica transformaciones inversas de las aplicadas por el codificador. El objetivo es que la reconstrucción sea lo más cercana posible a los datos originales.

Durante el entrenamiento, la función de pérdida del autoencoder compara los datos de entrada con las reconstrucciones generadas por el decodificador. El objetivo es minimizar la diferencia entre los datos originales y las reconstrucciones, lo que conduce a que el autoencoder aprenda a capturar las características más relevantes en su espacio latente. Son muy útiles en una gran variedad de aplicaciones: Reducción de la dimensionalidad de datos de alta dimensión,Eliminación de ruido de señales y Generación de datos nuevos y similares a los de entrada, es por esto último que se conocen también como la primer arquitectura de red de generación de contenido (Bank et al., 2021)

###### **Funcionamiento**

Un Autoencoder no se diferencia mucho de un perceptrón multicapa como los que ya hemos explicado anteriormente.Se entrenan a través del método de descenso del gradiente usando un algoritmo de Backpropagation para ajustar los pesos. (Autoencoders Notas de clase)



*Ilustración XIII:Esquema simplificado de un Autoencoder, Alberto Alvarez Diaz,UPM*

El entrenamiento de un autoencoder involucra el aprendizaje simultáneo del codificador y del decodificador en función de un objetivo común el cual es *comprimir los datos de entrada en una representación latente y luego reconstruirlos con la mayor fidelidad posible*. Los pasos involucrados son los siguientes:

* *Codificación*: Durante el entrenamiento, se presentan datos de entrada al codificador, que es la parte inicial del autoencoder. El codificador aplica transformaciones lineales y no lineales a los datos para mapearlos a una representación en el espacio latente de menor dimensión.El término espacio latente se utiliza para describir los parámetros (pesos y desviaciones) que resultan de esta etapa, se entiende, que los mismos están cargados con las características más relevante de los datos a los cuales fueron expuestos.Como las Redes neuronales tienden a tener una forma de embudo, supone además una reducción de las dimensiones en comparación con los datos originales
* *Cálculo de la Pérdida de Reconstrucción*: Luego, la representación latente generada por el codificador se pasa al decodificador. El decodificador intenta generar una reconstrucción que sea lo más similar posible a los datos originales. Se calcula la pérdida de reconstrucción, que mide la discrepancia entre los datos originales y las reconstrucciones generadas por el decodificador. El objetivo es minimizar esta pérdida.
* *Retropropagación y Actualización*: El gradiente de la pérdida de reconstrucción se propaga hacia atrás a través del decodificador y el codificador, lo que permite ajustar los pesos y sesgos de ambas partes de la red. El codificador aprende a generar representaciones latentes que sean útiles para la reconstrucción precisa en el decodificador.

Gracias a que forzamos el aprendizaje de las características más relevantes, elementos como el ruido o las distorsiones tienden a ser eliminadas automáticamente.

(Autoencoders notas de clase)

###### **Limitación de los Autoencoders**

Los autoencoders tienen mucho potencial, pero en la generación de *datos sintéticos* no son muy eficientes. El principal problema se debe a que el espacio latente generado no es continuo,sino que se encuentra formado por regiones separadas unas de otras que agrupan las características de ejemplos similares. Entre estas regiones no hay un espacio continuo de representaciones intermedias coherentes.Técnicamente si existe una continuidad entre regiones, pero la misma carece de sentido.

Para comprender mejor esto pensemos en que asignamos la tarea de reconocer las características más relevantes de una serie de números aún autoencoder, El mismo agrupara dichas características en zonas específicas del espacio latente,Preguntémonos ahora *¿De verdad no hay un espacio intermedio entre un 1 y un 7? ¿o entre un 3 y un 8?*, intuitivamente podemos reconocer características y similitudes entre los números,el problema está en que los autoencoders generan un espacio latente continuo sin una estructura específica en términos de distribuciones probabilísticas. Esto significa que las interpolaciones entre puntos en el espacio latente (como pensar en la relación entre el 1 y el 7 o el 3 y el 8) no tienen una interpretación semántica directa y no se pueden realizar operaciones probabilísticas confiables, como la generación de nuevas muestras de manera coherente. Esto hace que la interpolación entre ejemplos sea imposible. En definitiva, si el espacio intermedio no es continuo, las salidas del decoder no son realistas.

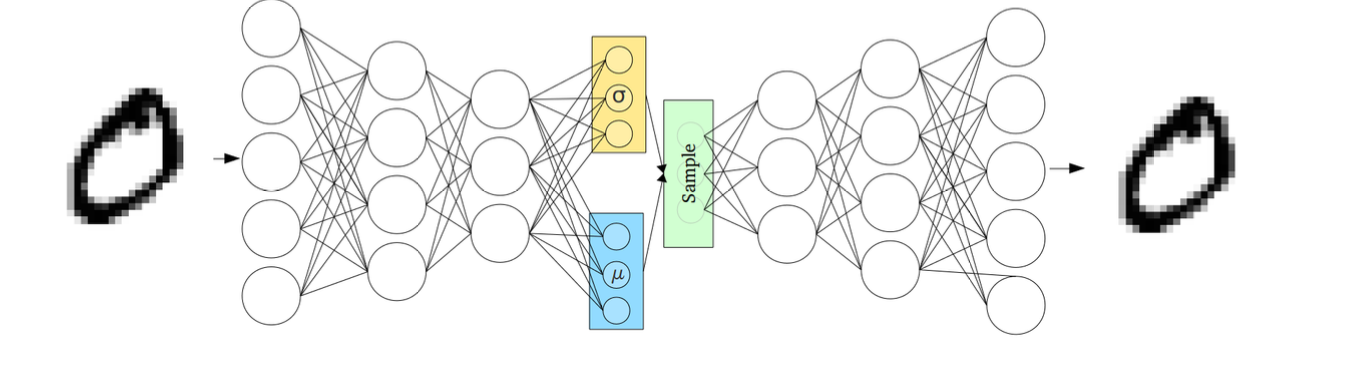
Veremos a continuación una variante de esta técnica que resuelve el problema, especialmente formada para la generación de contenido (Autoencoders Notas de clase)

###### **Autoencoders Variacionales**

Los Autoencoders Variacionales (*VAEs*) abordan esta limitación al introducir un componente probabilístico en el espacio latente. Estos son una extensión de los autoencoders tradicionales donde utilizamos conceptos de inferencia probabilística para modelar las distribuciones en el espacio latente. Esto hace que el espacio latente en *VAEs* tenga una estructura probabilística, lo que permite realizar interpolaciones suaves y operaciones generativas más flexibles.En definitiva, contamos con un *espacio latente continuo*. En los VAEs el espacio latente está definido por dos vectores, uno de *medias* y el otro de *desviaciones típicas,* formando un vector de distribuciones normales



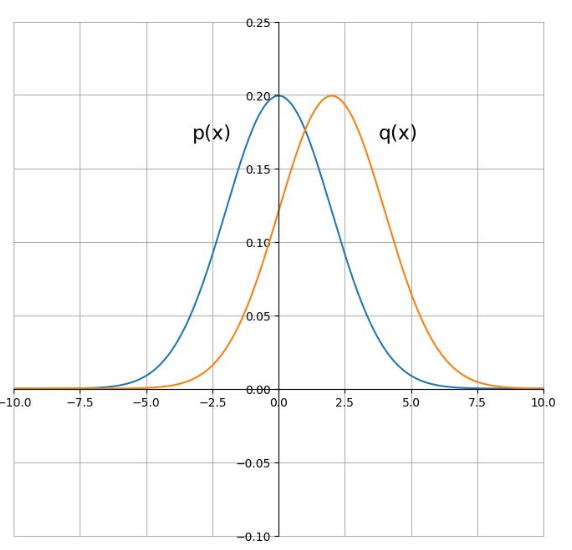
Con este artilugio matemático logramos que el Decoder asocie áreas enteras y no solo puntos concretos a ligeras variantes de la misma salida, esto es lo que en definitiva genera un espacio mucho más interpolado y suave



*Ilustración XIV: esquema de un VAE, Fuente: (Autoencoders,Alberto Díaz Álvarez, Edgar Talavera Muñoz y Guillermo Iglesias Hernández,2022,UPM)*

Al hacer esto necesitaremos ajustar las funciones de pérdida individualmente. Utilizaremos una función de pérdida o error convencional para medir la distancia del objeto generado y una divergencia KL o divergencia de Kullback-leibler entre la distancia latente aprendida y la distribución previa, que actúa como término de regularización  *(Autoencoders,Alberto Díaz Álvarez, Edgar Talavera Muñoz y Guillermo Iglesias Hernández,2022,UPM)*

La divergencia KL es una medida utilizada en teoría de la información para medir la diferencia entre distribuciones de probabilidad. Dicho de otra forma es el mecanismo que debemos utilizar para poder cuantificar una medida de error sobre esta nuevo espacio latente continuo es decir una medida de cuán bien se ajusta la distribución latente aprendida a la distribución latente real de los datos.

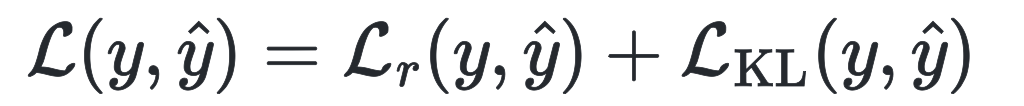


##### 

*Ilustración XV: Dos distribuciones Normales comparadas en el mismo sistema de ejes.Fuente (Autoencoders,Alberto Díaz Álvarez, Edgar Talavera Muñoz y Guillermo Iglesias Hernández,2022,UPM)*

Podemos pensar que P(x) es la distribución normal conocida y Q(x) es la distribución,también normal pero aprendida por el *VAE*

Pasaremos a dividir nuestra función de pérdida en dos términos, el término de reconstrucción y el término de regularización, donde entra a jugar la divergencia KL.Se le llama término de regularización porque trata de evitar que el modelo colapse en un espacio latente poco informativo.



A modo de resumen podemos distinguir las capacidades y limitaciones de los VAE:

* Son capaces de generalizar gracias al uso de una función de densidad Pueden
* generar datos indistinguibles de los de entrada
* Pueden generar nuevos datos sin copiar directamente los ejemplos

las principales desventajas son

* Poca calidad en los datos generados, lo que limita su uso en aplicaciones reales
* Poca diversidad en los datos generados por depender, entre otros, sólo de la función de densidad aprendida.

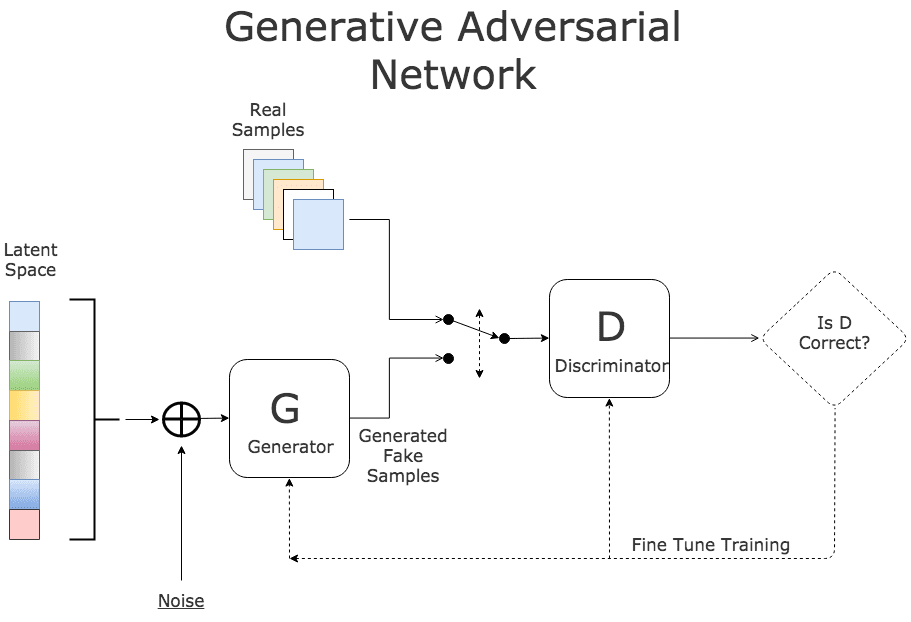
Esta última es la que impulsa el surgimiento de la arquitectura GAN, que veremos a continuación

##### **Redes Generativas Adversariales**

La arquitectura Generative Adversarial Network (GAN) es una técnica fundamental en el campo de la inteligencia artificial generativa. Las GANs son un tipo de modelo generativo que se utiliza para generar datos realistas y coherentes.

Ahora estamos en condiciones de explicar la arquitectura GAN. Fue necesario comprender los espacios latentes, y en especial los VAE ya que son el primer componente de la arquitectura, es desde estos que se obtienen las características que serán “moldeadas” gracias al ruido y que juntos conformarán el input de la parte Generadora de la red.

(«Generative Adversarial Networks - Hot Topic in Machine Learning», s. f.)



Las GANs constan de dos componentes principales: el *generador* y el *discriminador*. Estos dos componentes se entrenan en un proceso de competencia mutua.

El generador toma una señal de ruido o un vector latente como entrada y genera datos sintéticos, como imágenes, a partir de ese ruido. Inicialmente, estos datos sintéticos son de baja calidad y no se parecen a los datos reales que se están modelando.Por otro lado el discriminador toma tanto datos reales como datos generados por el generador como entrada y trata de distinguir entre los dos tipos de datos. Su objetivo es aprender a diferenciar los datos generados de los datos reales.

El proceso de entrenamiento de una GAN implica una lucha entre el generador y el discriminador. A medida que el generador mejora su capacidad para generar datos más realistas, el discriminador también mejora su capacidad para distinguir entre datos reales y generados. Esto da lugar a un equilibrio en el que el generador aprende a generar datos que son cada vez más difíciles de distinguir de los datos reales.

Si bien su entrenamiento es muy complicado debido a que es relativamente sencillo que suceda un colapso modal, este concepto refiere a una situación en la que el generador de la GAN produce consistentemente solo un subconjunto limitado de datos generados, ignorando gran parte de la variabilidad en los datos reales que se están modelando. En otras palabras, el generador se "colapsa" en la generación de sólo un puñado de patrones o ejemplos, en lugar de capturar toda la diversidad de datos que debería generar. Además puede suceder que el generador se sobreajuste al sesgo de la discriminadora, produciendo siempre un tipo específico de contenido, que no necesariamente puede considerarse válido,sino simplemente ser ruido, pero que logre engañar a la red discriminadora. Puede suceder también que la red discriminadora se vuelva demasiado buena para la red generadora, evitando el correcto entrenamiento de la misma.

Con esto hemos cubierto las bases teóricas que dan origen a la generación de contenido usando inteligencia artificial. Ahora hablaremos de una rama de la IA y la lingüística computacional que se encarga de trabajar la interacción entre las computadoras y el lenguaje humano. Los avances en IAs generativas, junto con las técnicas de modelado sequential que fueron evolucionando con los años resultaron en la arquitectura que dará sustento al producto en torno al cual gira este trabajo. los *LLMs* y la arquitectura *Transformer.*

#### **¿Que es el NLP?**

#### **Historia del modelado secuencial**

##### **Redes neuronales recurrentes**

##### **Neuronas LSTM**

#### **¿Que es un LLM?**

#### **La Arquitectura Transformer**

##### **Embeddings de palabras**

##### **codificación posicional**

##### **El mecanismo de Self-Attention**

##### **Encoders**

##### **Decoders**

#### **¿Qué es el Clustering?**

##### **Algoritmo K-means**

#### **Antecedentes relacionados con el problema**

#### **Temas pendientes de resolución**

# Dudas

Que tan tecnico tengo que ir con la teoria

deberia poner todas las formulas y hacer un desarrollo matematico de cada una?

Como hago con el tema de las imagenes,como doy credito a los autores

Fuente: lugar de donde salio “revista autor”

Es mejor si me hago “mis propias” imagenes.

Fuente: elaboracion propia

Puedo poner animaciones?

si puede entrar en un PDF puede ser.

Como hago para referirme a una pieza de codigo o notebook de jupyter?

elaboracion propia o el repositorio

primero (Srivastava et al., s. f.)

Referencias

Andrej Karpathy (Director). (2022, agosto 17). *The spelled-out intro to neural networks and backpropagation: Building micrograd*. https://www.youtube.com/watch?v=VMj-3S1tku0

Bank, D., Koenigstein, N., & Giryes, R. (2021). *Autoencoders* (arXiv:2003.05991). arXiv. http://arxiv.org/abs/2003.05991

*Bing.com*. (s. f.). Bing. Recuperado 17 de agosto de 2023, de https://www.bing.com/?form=HPFBBK&ssd=20230816\_2200&mkt=es-ES

BitBoss (Director). (2020, mayo 27). *Redes Neuronales—Backpropagation*. https://www.youtube.com/watch?v=boP3O89rErA

*ChatGPT*. (s. f.). Recuperado 17 de agosto de 2023, de https://chat.openai.com

*ChatGPT Topped 1 Billion Visits in February*. (s. f.). Similarweb. Recuperado 17 de agosto de 2023, de https://www.similarweb.com/blog/insights/ai-news/chatgpt-1-billion/

Derivando (Director). (2021, agosto 4). *¿Qué es y cómo funciona la INTELIGENCIA ARTIFICIAL?* https://www.youtube.com/watch?v=\_tA5cinv0U8

Frost, J. (2021, noviembre 12). *Mean Squared Error (MSE)*. Statistics By Jim. https://statisticsbyjim.com/regression/mean-squared-error-mse/

Generative Adversarial Networks—Hot Topic in Machine Learning. (s. f.). *KDnuggets*. Recuperado 17 de agosto de 2023, de https://www.kdnuggets.com/generative-adversarial-networks-hot-topic-in-machine-learning.html

Gonzalez, J. L. (2020, julio 13). Tipos de aprendizaje automático. *SoldAI*. https://medium.com/soldai/tipos-de-aprendizaje-autom%C3%A1tico-6413e3c615e2

Introduction to Deep Learning. (2018, junio 1). *GeeksforGeeks*. https://www.geeksforgeeks.org/introduction-deep-learning/

Mehta, A. (2020, julio 17). Why is Stochastic Gradient Descent? *Bayshore Intelligence Solutions*. https://medium.com/bayshore-intelligence-solutions/why-is-stochastic-gradient-descent-2c17baf016de

Merritt, R. (2022, abril 19). *¿Qué Es un Modelo Transformer? | Blog de NVIDIA*. Blog oficial de NVIDIA Latino América. https://la.blogs.nvidia.com/2022/04/19/que-es-un-modelo-transformer/

*Qué es la Inteligencia Artificial*. (s. f.). Recuperado 17 de agosto de 2023, de https://planderecuperacion.gob.es/noticias/que-es-inteligencia-artificial-ia-prtr

*¿Qué es la Inteligencia Artificial Generativa?* (2023, marzo 5). OBS Business School. https://www.obsbusiness.school/blog/que-es-la-inteligencia-artificial-generativa

*Regla de la cadena—Wikipedia, la enciclopedia libre*. (s. f.). Recuperado 17 de agosto de 2023, de https://es.wikipedia.org/wiki/Regla\_de\_la\_cadena

Regresión lineal. (2023). En *Wikipedia, la enciclopedia libre*. https://es.wikipedia.org/w/index.php?title=Regresi%C3%B3n\_lineal&oldid=151063050

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (s. f.). *Dropout: A Simple Way to Prevent Neural Networks from Overﬁtting*.

StatQuest with Josh Starmer (Director). (2019, febrero 5). *Gradient Descent, Step-by-Step*. https://www.youtube.com/watch?v=sDv4f4s2SB8

*Top 10: Promising AI startups in 2022*. (2022, noviembre 29). https://aimagazine.com/articles/top-10-promising-ai-startups-in-2022

*Understanding Local Minima in Neural-Network Training—Technical Articles*. (s. f.). Recuperado 17 de agosto de 2023, de https://www.allaboutcircuits.com/technical-articles/understanding-local-minima-in-neural-network-training/

1. es un modelo matemático usado para aproximar la relación de dependencia entre una variable dependiente y una variable independiente (cita de wikipedia) («Regresión lineal», 2023) [↑](#footnote-ref-0)